Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

**Бакалаврська кваліфікаційна робота**

на тему: Розробка програмного забезпечення аудіоплеєра для смартфонів на платформі Андроїд

Виконав: студент IV курсу, групи 4ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Гавриш Євгеній Олегович
(прізвище та ініціали)
Керівник: д.т.н., проф. каф. ПЗ Ліщинська Л.Б.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент: доцент каф. ЗІ Баришев Ю.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Допущено до захисту
Зав. кафедри ПЗ О.Н. Романюк
«09» червня 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – «Інформаційні технології»
Спеціальність 121 – «Інженерія програмного забезпечення»
Освітньо-професійна програма – «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Зав. кафедри ПЗ, проф., д.т.н.
_____ О.Н. Романюк
"21" березня 2025 р.

**ЗАВДАННЯ**
**НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ**

Гавришу Євгенію Олеговичу

1. Тема роботи – розробка програмного забезпечення аудіоплеєра для смартфонів на платформі Андроїд.

Керівник роботи: Ліщинська Людмила Броніславівна, д.т.н., професор кафедри програмного забезпечення, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи 2 червня 2025.

3. Вихідні дані до роботи: аудіоплеєр для Android, реалізовані модулі відтворення та керування музикою, користувацький інтерфейс, структура даних для зберігання інформації про аудіофайли, інтеграція з медіабібліотекою пристрою, тестування роботи застосунку, інструкція користувача, проаналізовані результати роботи системи.

4. Зміст текстової частини: зміст, вступ, аналіз стану систем аудіоплеєра для смартфонів на платформі Андроїд, порівняльний аналіз аналогів, аналіз методів розв'язання задачі, постановка задачі дослідження, розробка інтерфейсу та функціоналу програмного продукту, аналіз вхідних даних системи, розробка інтерфейсу програмного додатку, розробка моделі системи, розробка алгоритмів роботи системи, розробка програмних модулів та методів реалізації системи, аудіоплеєра для смартфонів на платформі андроїд, варіантний аналіз і обгрунтування вибору засобів для реалізації програмного забезпечення, розробка модулів застосунку, тестування системи, розробка інструкції користувача.

5. Перелік ілюстративного матеріалу: титульний слайд, актуальність дослідження, мета, об'єкт та предмет дослідження, цілі дослідження, аналоги, порівняльний аналіз аналогів, діаграма взаємодії користувача, вибір середовища розробки, інтерфейс застосунку, вимоги до пристрою.

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | Завдання видала | Завдання прийняв |
| 1-4 | Ліщинська Л.Б., д.т.н., професор кафедри ПЗ | 24.03.25 | 01.06.25 |

7. Дата видачі завдання 24 березня 2025 року

## КАЛЕНДАРНИЙ ПЛАН

| № з\п | Назви етапів бакалаврської кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Аналіз розвитку систем аудіоплеєра для смартфонів на платформі андроїд | 25.03.25 – 11.04.25 | вик. |
| 2 | Розробка інтерфейсу програмного забезпечення | 11.04.25 – 21.04.25 | вик. |
| 3 | Розробка алгоритмів роботи програмного забезпечення | 21.04.25 – 28.04.25 | вик. |
| 4 | Розробка програмних модулів та методів реалізації системи аудіоплеєра | 28.04.25 – 19.05.25 | вик. |
| 5 | Тестування програмного забезпечення | 19.05.25 – 26.05.25 | вик. |
| 6 | Оформлення матеріалів до захисту БКР | 26.05.25 – 30.05.25 | вик. |

Завдання отримав ст. гр. 4ПІ-21б _____ (підпис) Є.О. Гавриш (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) Л.Б. Ліщинська (ініціали та прізвище)

# АНОТАЦІЯ

УДК 004.91

Гавриш Є.О. Розробка програмного забезпечення аудіоплеєра для смартфонів на платформі Андроїд: бакалаврська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 55 ст.

На укр. мові. Бібліогр. : 21 назв; рис. : 42; табл. : 3.

У бакалаврській кваліфікаційній роботі розглянуто процес розробки програмних аудіоплеєра для мобільних пристроїв, що працюють під управлінням операційної системи Android. Проведено аналіз вимог до функціоналу програми, проєктування архітектури, реалізації основних модулів та тестування додатку.

Для розробки використовуються Android Studio як основне середовище розробки та мова програмування Kotlin. Особливу увагу приділено використанню інструментів Android SDK, а також дотриманню принципів ефективного і зручного інтерфейсу користувача. Результатом роботи є працездатний прототип аудіоплеєра з базовим функціоналом, який може бути використаний як основа для подальшого розвитку або інтеграції в більші програмні продукти.

На основі досліджень, проведених у БКР, розроблено алгоритми та програмний засіб для роботи з музичними файлами.

Ключові слова: мобільний застосунок, медіафайли, аудіоплеєр, платформа Android, UI/UX дизайн.

# ABSTRACT

UDC 004.91

Havrysh Y. O. Development of audio player software for smartphones on the Android platform: bachelor's thesis in specialty 121 – software engineering, educational program – software engineering. Vinnytsia: VNTU, 2025. 55 p.

In Ukrainian. Bibliography : 21 titles; Figures: 42; Tables: 3.

The bachelor's thesis considers the process of developing software audio players for mobile devices running the Android operating system. An analysis of the requirements for the program's functionality, architecture design, implementation of the main modules, and application testing was conducted.

Android Studio is used as the main development environment and the Kotlin programming language for development. Particular attention is paid to the use of Android SDK tools, as well as adherence to the principles of an effective and convenient user interface. The result of the work is a working prototype of an audio player with basic functionality, which can be used as a basis for further development or integration into larger software products.

Based on research conducted for the thesis, algorithms and a software tool for working with music files were developed.

Keywords: mobile application, media files, audio player, Android platform, UI/UX design.

# ЗМІСТ

# ВСТУП

**Обґрунтування вибору теми дослідження.**

У сучасному світі мобільні пристрої стали невід’ємною частиною повсякденного життя користувачів та стали майже символом статусу на додаток до зручності та безпеки, що походить від їхнього володіння [2]. Смартфони виконують не лише функції зв’язку, а й слугують універсальними мультимедійними інструментами, зокрема для прослуховування музики.

Аудіоплеєри [3] є одними з найпоширеніших типів мобільних застосунків, які забезпечують доступ до особистих колекцій музичних файлів та, в деяких випадках, онлайн-контенту. Водночас, сучасні аудіоплеєри часто мають низку вад, що знижують зручність їх використання. Багато з існуючих застосунків орієнтовані на потокову передачу даних, вимагають постійного підключення до інтернету або нав'язують платні підписки, що обмежує доступ до функціоналу для звичайного користувача. Крім того, деякі додатки є надто ресурсоємними, що призводить до швидкого розрядження акумулятора або уповільнення роботи пристрою. Іноді інтерфейси перевантажені рекламою, складні в навігації або мають обмежену підтримку локальних медіафайлів. Усе це створює потребу в альтернативних продуктах, які були б легкими, швидкими та орієнтованими на базові потреби користувачів. У зв’язку з цим розробка сучасного, функціонального та інтуїтивно зрозумілого аудіоплеєра є актуальним завданням.

**Актуальність.** Актуальність потреби в розробці аудіоплеєрів для смартфонів пояснюється тим, що в сучасності велика кількість людей прослуховує з використанням смартфонів, а використання окремих пристроїв для відтворення музики падає [4]. В той же час велика кількість застосунків, що мають функціонал для програвання аудіофайлів мають недоліки, що не задовольняють користувачів функціоналом, або ж мають ускладнений інтерфейс та не використовуються.

**Зв’язок роботи з науковими програмами, планами, темами.** Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

**Мета та завдання дослідження.** Метою дослідження є спрощення управління користувачами прослуховування аудіофайлів.

Відповідно до поставленої мети в бакалаврській кваліфікаційній роботі потрібно вирішити такі задачі:

– аналіз предметної галузі;

– аналіз аналогів;

– аналіз методів розв’язання задачі;

– постановка задачі дослідження;

– визначити вхідні дані системи;

– розробити алгоритм взаємодії користувача із застосунком;

– розробити алгоритми роботи основних функцій застосунку;

– покращити метод сканування аудіофайлів на пристрої;

– розробити інтерфейс на основі даних, отриманих після аналізу існуючих застосунків;

– розробити функціонал програмного продукту, необхідного для роботи;

– протестувати розроблений програмний продукт;

– створити інструкцію користувача та визначити мінімальні та оптимальні характеристики пристрою для роботи програмного продукту.

**Об'єкт дослідження** – процеси управління аудіофайлами та робота з їх метаданими.

**Предмет дослідження** – методи та засоби процесів управління аудіофайлами та роботи з їх метаданими.

**Наукова новизна отриманих результатів.** Удосконалено метод пошуку аудіофайлів за рахунок використання контент-провайдера в якості посередника між застосунком та файловою системою пристрою, що дозволяє зменшити використання ресурсів та підвищує швидкодію.

**Методи дослідження.** У процесі дослідження застосовувався теоретичний аналіз архітектур програмних систем, для визначення оптимальних підходів до побудови структури аудіоплеєра, методи об’єктно-орієнтованого програмування, для реалізації логіки відтворення аудіофайлів, методи моделювання користувацького інтерфейсу з урахуванням принципів UI/UX-дизайну для забезпечення зручності у взаємодії з додатком, аналіз і тестування програмного коду для перевірки працездатності окремих функціональних модулів, засоби комп’ютерного моделювання та емуляції в середовищі Android Studio для перевірки роботи аудіоплеєра на віртуальних пристроях та в реальних умовах.

**Практична цінність отриманих результатів** полягає в тому, що на основі теоретичних досліджень розроблено алгоритми та програму для відтворення аудіофайлів на смартфонах на платформі Android .

Структура програми дозволятиме легку адаптацію та розширення її функціоналу, шляхом інтегрування нових модулів – наприклад, рекомендаційні системи чи хмарне сховище. Отримані результати можуть бути використані у навчальному процесі, для демонстрації принципів побудови Android-застосунків у реальних умовах, а також як основа для стартап-проєктів у сфері мобільного програмного забезпечення.

**Особистий внесок здобувача.** Усі наукові результати отримано здобувачем самостійно. У праці, котра була опублікована у співавторстві, студенту належить: [1] – типи мобільних застосунків, їх особливості та інструменти їх реалізації.

**Апробація матеріалів бакалаврської кваліфікаційної роботи.** Основні положення БКР доповідалися та обговорювалися на LIV всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (м. Вінниця, 2025).

# 1 АНАЛІЗ РОЗВИТКУ СИСТЕМ АУДІОПЛЕЄРА ДЛЯ СМАРТФОНІВ НА ПЛАТФОРМІ АНДРОЇД

## 1.1 Аналіз стану систем аудіоплеєра для смартфонів

З розвитком мобільних технологій аудіоплеєри стали невід'ємною складовою екосистеми смартфонів. Вони задовольняють потреби мільйонів користувачів [3], які щодня слухають музику у транспорті, під час занять спортом, роботи або відпочинку. Аудіоплеєри виконують не лише базову функцію відтворення аудіофайлів, але й стали комплексними мультимедійними платформами з розширеним функціоналом і високим рівнем інтеграції з системою пристрою.

Певна частина аудіоплеєрів поступово інтегрується з хмарними сервісами [5], що дає можливість слухати музику в потоковому режимі без необхідності зберігати файли на пристрої. Найпопулярніші платформи надають доступ до величезних бібліотек музики, натомість такий функціонал вимагає постійного підключення до Інтернету. Ще однією з поширених тенденцій розвитку плеєрів є поглиблена персоналізація. Користувачі таких платформ очікують, що плеєр буде аналізувати використання такої платформи, пропонуватиме рекомендовані треки, підтримуватиме плейлисти, теми оформлення, та голосове керування. Всі ці можливості потребують ретельно продуманої архітектури, гнучкого інтерфейсу, потужних серверів, що здатні працювати з великими обсягами даних та величезних витрат для підтримки роботи такої системи. Такі платформи вимагають реєстрації чи підключення до облікових записів, оскільки їх функціонал вимагає зберігати та обробляти дані всіх користувачів [6].

Інтерфейс є також важливим аспектом для користувача та взаємодії з пристроєм. Аудіоплеєр повинен мати інтуїтивно зрозумілий, мінімалістичний і функціональний інтерфейс, створений для роботи на пристроях з різними розмірами екранів та режимів роботи. Важливою є підтримка фонової роботи,

керування з панелі сповіщень, взаємодія з гарнітурою, а також безперервність відтворення навіть при зміні активностей у системі.

Не менш актуальними є вимоги до продуктивності та ресурсоефективності, в той час як багато сучасних застосунків перенасичені функціями, це може негативно впливати на швидкість роботи, час автономної роботи пристрою та споживання пам’яті. Висока енергоефективність, покращення процесів і плавність інтерфейсу – важливі переваги у боротьбі за лояльність користувачів.

З технічного погляду, розробка якісного аудіоплеєра потребує глибокого розуміння роботи з мультимедійними API Android, правильного використання потоків, підтримки різних форматів аудіо, обробки метаданих, створення стійкої архітектури додатку та налаштування компонентів інтерфейсу відповідно до вимог UI/UX.

Ще одним важливим трендом є відкритість до кастомізації. Користувачі все частіше бажають змінювати вигляд програми, налаштовувати жести, переходи між треками, поведінку при підключенні Bluetooth-пристроїв, а також мати доступ до статистики прослуховування.

Водночас, незважаючи на широкий вибір плеєрів на ринку, багато з них мають спільні недоліки, зокрема: агресивну рекламу, платний доступ до базових функцій, незручний або застарілий інтерфейс, обмеження у підтримці форматів або нестабільну роботу у фоновому режимі. Це відкриває нішу для створення оптимального аудіоплеєра, що буде поєднувати простоту, зручність, автономність і сучасний вигляд.

Отже, аналіз ринку показує, що попри велику кількість аудіоплеєрів, потреба в нових, легких, зручних і функціональних аудіоплеєрах з базовими можливостями все ще залишається актуальною. Такий застосунок повинен бути орієнтований на користувача, забезпечувати стабільну роботу, мати зрозумілий інтерфейс і можливість для подальшого розширення функціоналу.

## 1.2 Порівняльний аналіз аналогів

Сучасний ринок мобільних застосунків насичений великою кількістю аудіоплеєрів, кожен з яких має свої особливості, переваги та недоліки. Більшість популярних плеєрів орієнтовані на зручність користування, високу продуктивність, розширені функції відтворення та інтеграцію з онлайн-сервісами.

Нижче наведено огляд найпоширеніших мобільних аудіоплеєрів для Android, що дозволяє виявити особливості і сформулювати вимоги до власного застосунку.

VLC – це кросплатформовий медіаплеєр з відкритим кодом, який підтримує відтворення практично всіх типів медіафайлів, включаючи аудіо та відео [7]. Додаток має зручний інтерфейс (див. рисунок 1.1), підтримує списки відтворення, еквалайзер, а також відтворення з мережі. Його перевага – у стабільності роботи та широкому спектрі підтримуваних форматів. Однак інтерфейс не завжди виглядає сучасно, а кількість налаштувань може бути надмірною для звичайного користувача.

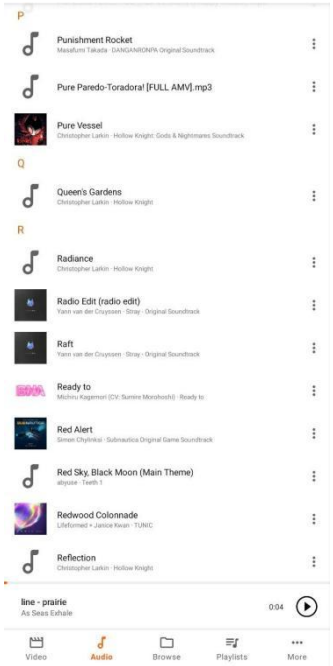


Рисунок 1.1 – Інтерфейс додатку VLC

Poweramp є одним з найпопулярніших аудіоплеєрів на Android завдяки своєму потужному еквалайзеру (див. рисунок 1.2), розширеному функціоналу та високій якості відтворення [8]. Програма підтримує більшість популярних форматів аудіо, має налаштовуваний інтерфейс, обкладинки альбомів, підтримку текстів пісень і багато параметрів для меломанів. Основним недоліком є наявність платної версії, без якої функціонал обмежений, а також дещо складне меню для нових користувачів.

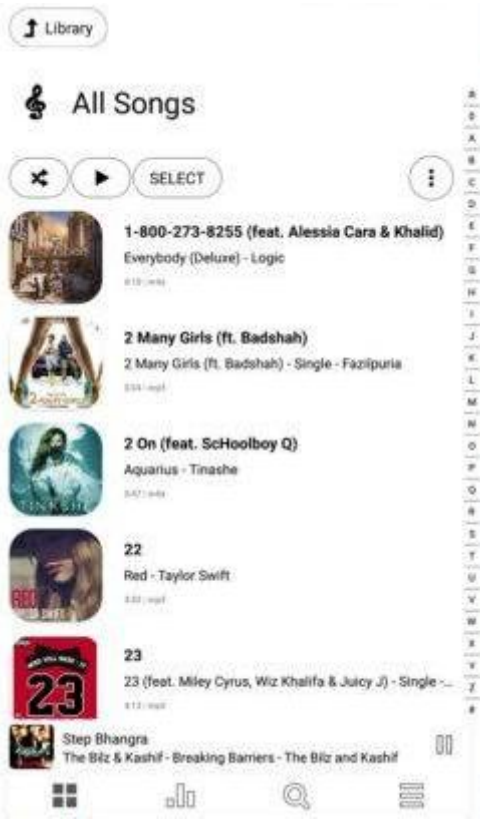


Рисунок 1.2 – Інтерфейс еквалайзера додатку Poweramp

AIMP – це легкий, швидкий аудіоплеєр, який користується популярністю завдяки простоті, низькому споживанню ресурсів та якісному відтворенню [9]. Додаток підтримує відтворення з папок, створення плейлистів, а також має зручний таймер сну. Інтерфейс (див. рисунок 1.3) доволі простий, але не надто сучасний. AIMP ідеально підходить для користувачів, яким потрібен базовий і надійний плеєр без зайвих функцій.


Рисунок 1.3 – Інтерфейс пошуку файлів додатку AIMP

JetAudio HD – це ще один функціональний плеєр, який підтримує багато аудіоформатів та має вбудований еквалайзер, звукові ефекти та широкий набір налаштувань звуку [10]. Є підтримка віджетів, обкладинок (див. рисунок 1.4), текстів пісень. Основним мінусом є наявність платної версії, у якій відкриваються більшість розширених можливостей.

Spotify є одним з найбільших стримінгових сервісів у світі, який також виконує функції аудіоплеєра [11]. Користувачі можуть слухати як онлайн, так і офлайн треки, створювати плейлисти, ділитися ними з іншими, слідкувати за артистами. Інтерфейс сучасний (див. рисунок 1.5) і постійно оновлюється. Недоліком є необхідність підписки для доступу до більшості функцій, наприклад: офлайн-режим, відсутність реклами, якість звуку, а також залежність від підключення до Інтернету.

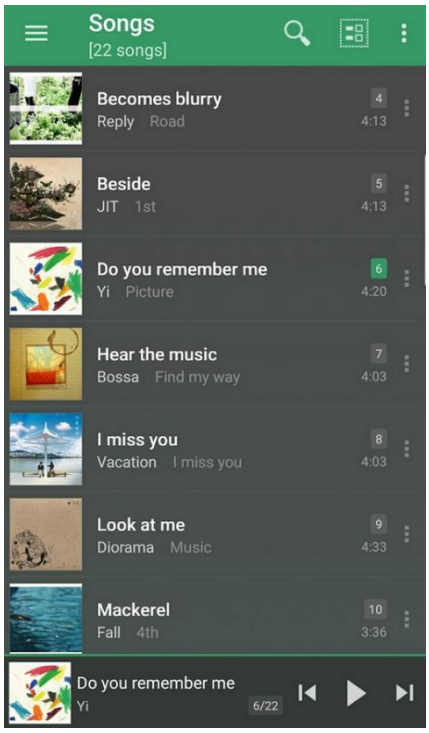


Рисунок 1.4 – Інтерфейс навігації файлів додатку JetAudio HD


Рисунок 1.5 – Інтерфейс додатку Spotify

YouTube Music (див. рисунок 1.6) – офіційний музичний сервіс Google [12], який об’єднує традиційне прослуховування музики з відеоконтентом. Перевага – доступ до величезної бібліотеки пісень і відеокліпів. Додаток має рекомендаційні алгоритми, можливість завантаження контенту, синхронізацію з акаунтом Google. Проте без підписки користувач обмежений у фоновому відтворенні та прослуховуванні без реклами.

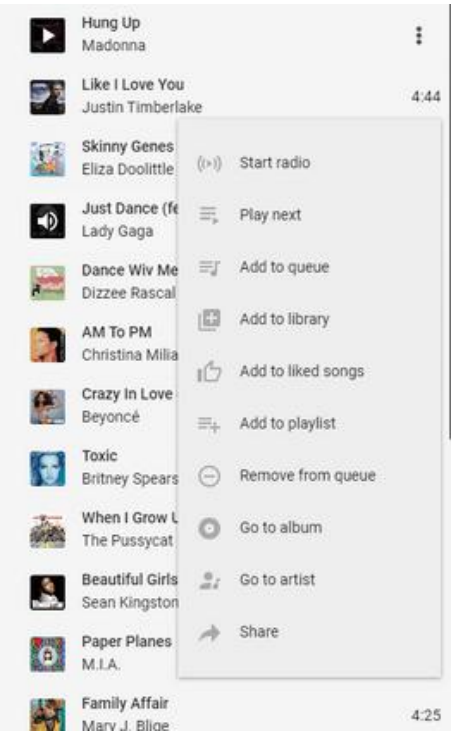


Рисунок 1.6 – Інтерфейс додатку YouTube Music

Аналізуючи сучасні аудіоплеєри, можна зробити висновок, що більшість з них або орієнтовані на досвідчених користувачів і мають складний функціонал, або навпаки – надто спрощені.

Також помітна тенденція до переходу в хмарні стримінгові сервіси, однак це обмежує автономність роботи додатку та вимагає створення алгоритмів для підбирання музики користувачу, значних витрат на ліцензії на музику та створення серверів для надання послуг.

Для більш детального розгляду та порівняння аналогів з власною розробкою було створено таблицю (таблиця 1.1) для порівняння їх функціоналу.

Таблиця 1.1 – Порівняльні характеристика мобільних застосунків

| | VLC | Poweramp | AIMP | JetAudio HD | Spotify | YouTube Music | Власна розробка |
|---|---|---|---|---|---|---|---|
| Незалежність від мережі Інтернет | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| Зручний інтерфейс | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| Доступ до великої бібліотеки файлів | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| Екалайзер для налаштування звучання | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| Таймер сну | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| Налаштованість вигляду інтерфейсу | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| Сумарний коефіцієнт | 3 | 4 | 2 | 4 | 2 | 2 | 5 |

Відповідно до наведеної таблиці порівняльного аналізу, розробка власної системи аудіоплеєра для смартфонів на платформі Андроїд обіцяє значні переваги порівняно з існуючими альтернативами на ринку, такими як VLC, AIMP, JetAudio HD та інші.

На відміну від аналогів, власна розробка забезпечить користувачам оптимальний плеєр для використання без доступу мережі Інтернет, зручний у використанні інтерфейс, вигляд якого можна змінювати, потужний еквалайзер для налаштування звучання музики та таймер сну, який вимикатиме програвання після вказаного користувачем часу.

Загальний коефіцієнт для власної розробки складає 5, що є найвищим показником в порівнянні з іншими аналізованими системами, що підкреслює її

переваги та інноваційний потенціал. Такий підхід не лише забезпечить високий рівень задоволеності користувачів завдяки зручності, але й сприятиме покращенню взаємодії користувачів з власними пристроями. Проаналізувавши дані з таблиці 1, відзначимо, що власна розробка показує кращі результати за розглянутими критеріями в порівнянні з аналогами Poweramp, JetAudio HD та на 20% (100% - 4/5 = 20%), VLC на 40% (100% -3/5 = 40%), AIMP, YouTube Music та Spotify – на 60% (100% - 2/5 = 60%).

Таким чином, є попит на легкий, зручний, автономний аудіоплеєр з сучасним інтерфейсом, базовим функціоналом для відтворення музики, керування плейлистами, роботи з системою сповіщень та відсутністю нав'язливої реклами чи залежності від інтернету. Саме такий застосунок є метою розробки в межах даного курсового проєкту.

### 1.3 Аналіз методів розв'язання задачі

Створення системи аудіоплеєра для смартфонів на платформі Андроїд вимагає визначитись з середовищем та мовою розробки продукту, але передусім для цього необхідно визначитись з типом розробки мобільного додатку.

Вибір типу розробки мобільного додатку є одним з найважливіших етапів у створенні успішного продукту, оскільки він визначає не лише технічні характеристики додатку, а й впливає на його функціональність, швидкість роботи, вартість розробки, а також досвід користувача. Існують різні підходи до розробки мобільних додатків, кожен з яких має свої переваги і недоліки: нативна, кроссплатформна та гібридна розробка. Кожен з цих методів має свої особливості, що зумовлюють їх вибір залежно від конкретних вимог проєкту. Саме тому важливо правильно оцінити, який підхід буде найбільш ефективним для досягнення поставлених цілей, мінімізації витрат та забезпечення високої якості кінцевого продукту.

Далі розглянемо типи розробки мобільного додатку: нативний, кросплатформенний та гібридний.

У використанні нативного типу розробки мобільних додатків, необхідно окремо розробляти додатки для кожної платформи окремо. Вони можуть використовувати всі переваги функцій пристрою, такі як вібрація, камера, GPS, тощо. Вони можуть використовуватись без необхідності доступу до мобільної мережі, оскільки вони повністю завантажені на пристрій, а вже розроблені настанови дозволять полегшити створення інтерфейсу застосунку, та дозволить користувачам ознайомитись з ним швидше завдяки загальним характеристикам. Нативні застосунки також мають кращий контроль над орієнтацією, розміром та розширенням зображення та доступ до особливостей компонування, що дозволяють пришвидшити розробку [13].

Цей тип також обмежує гнучкість розробки, оскільки вони розробляється для специфічної платформи, вони вимагають окремого досвіду для кожної платформи, більшої витрати ресурсів на розробку та обслуговування, вартість якої доволі висока та становить 15-20% з коштів на розробку. Також такі застосунки вимагають більшої затрати ресурсів пристрою, оскільки їх можливість працювати без доступу до мережі Інтернет часто вимагає більшого розміру застосунку.

Кросплатформний тип розробки застосунків дозволяє створювати один мобільний застосунок, що має здатність плавно працювати на декількох операційних системах. Такий підхід дозволяє використовувати частину, або ж увесь розроблений код та розгорнути додаток на декількох платформах без необхідності кодування “з нуля” для кожної платформи, що дозволяє зекономити час та витрати на розробку, оновлення та підтримку, а існування додатку на декількох платформах дозволяє досягти ширшої аудиторії [14].

Такий тип розробки теж має свої обмеження, часто пов’язані з продуктивністю. Процеси оптимізації розробки кросплатформенних застосунків можуть привести до негативного досвіду користувачів через меншу продуктивність в порівнянні з нативними застосунками, особливо для ресурсоємних функцій, а обмежений доступ до вбудованих можливостей пристрою часто робить реалізацію необхідних функцій складнішою та вимагати специфічних API для їх роботи. Можливі обмеження в UI/UX також вимагатимуть

компромісів під час розробки додатків для різних платформ, та робить перехід з використання додатку на різних платформах менш гладким.

Гібридний тип розробки мобільних застосунків теж мусить бути встановленим на пристрій для його роботи як і будь-який інший застосунок. Такі додатки відрізняються тим, що вони мають елементи нативних застосунків, розроблених для певної платформи, з елементами веб застосунків та веб сайтів, що працюють немов вони не були встановлені, а просо були запущені через браузер за допомоги мережі Інтернет. Такі додатки розгортаються за допомоги нативного контейнера, який використовує мобільний об’єкт WebView [15]. Це дозволяє відображати контент створений з використанням веб технологій під час роботи застосунку незалежно від платформи та скоротити час та витрати на розробку та підтримку, оскільки основна частина коду пишеться лише раз.

Обмеження гібридного типу розробки сильно обмежує можливості розробки інтерфейсу, та не дозволяє йому отримати нативного відчуття, а використання нативного контейнера не дозволяє додатку користуватися повним функціоналом пристрою. Крім того, комбінація обмежень кожної платформи вимагає від розробників використання необхідних плагінів, що ще більше ускладнює розробку проєктів. Також гібридні додатки завжди залежатимуть від доступу до мережі Інтернет та будуть обмежені його швидкістю.

Детально вивчивши типи розробки додатків, можна визначити, що обмеження гібридного типу розробки не дозволяє нам створити незалежний від мережі Інтернет додаток, тому є не припустимим для використання, а обмеження кросплатформеного типу сильно ускладнить розробку інтерфейсу додаткую.

Таким чином було визначено що додаток буде створено використовуючи нативний тип розробки, його обмеження щодо необхідності розробки додатків для кожної платформи окремо не впливатиме на сам процес, оскільки додаток розробляється на платформу Андроїд, а легший доступ до функцій пристрою полегшить розробку інтерфейсу та функцій застосунку.

Цей метод дозволяє використовувати декілька мов для реалізації застосунку. Java досі є популярною у використанні для розробки чи підтримки

нативних додатків на платформі Android, а завдяки своєму довгому існуванні має велику екосистему фреймворків, бібліотек та інструментів розробки [16], але, попри свої переваги, поступово програє конкуренцію Kotlin.

Kotlin є сучасною, статично типізованою мовою програмування, офіційно підтримуваною Google для розробки Android-застосунків [17]. Вона забезпечує високу безпеку коду завдяки усуненню помилок, пов’язаних із null-посиланнями, скорочує обсяг коду завдяки лаконічному синтаксису, підтримує функціональне програмування та є повністю сумісною з Java, що спрощує інтеграцію з існуючими бібліотеками. Kotlin також покращує читаність і підтримуваність коду, що робить розробку ефективнішою та зручнішою для роботи.

Завдяки його характеристикам було обрано Kotlin для використання під час розробки програмного продукту, та залишається ще один крок – визначення оптимального середовища розробки.

Android Studio є офіційним середовищем розробки для створення Android-застосунків і має повну інтеграцію з Kotlin [18]. Воно підтримує автодоповнення, рефакторинг, налагодження, профілювання продуктивності та інші засоби розробки. Android Studio автоматично налаштовує Kotlin у нових проєктах, забезпечуючи зручний старт для розробників. Крім того, воно містить інструменти для інтеграції Kotlin-коду з Java, що важливо для використання сторонніх бібліотек.

IntelliJ IDEA – це середовище розробки від компанії JetBrains, яка і створила мову Kotlin [19]. IDE має найкращу підтримку для цієї мови, включаючи повну інтеграцію з Gradle, Maven, тестуванням, налагодженням та сучасними засобами аналізу коду. IntelliJ IDEA підходить як для Android-, так і для серверної або десктопної розробки на Kotlin. Особливо корисною є можливість створення повноцінних Kotlin-проєктів для JVM, JS або мультиплатформенних застосунків.

Eclipse підтримує Kotlin через встановлення відповідного плагіна [20]. Хоча інтеграція не така глибока і стабільна, як у Android Studio або IntelliJ IDEA, вона дозволяє працювати з Kotlin-проєктами у середовищі, до якого звикли деякі розробники Java. Використання Eclipse з Kotlin доцільне, якщо вже є проєкти, що

розробляються в цьому середовищі, але для нових проєктів перевага зазвичай віддається JetBrains IDE.

Visual Studio Code підтримує Kotlin через розширення [21]. Це легка IDE з мінімалістичним інтерфейсом, яка підходить для редагування Kotlin-файлів, компіляції, а також базового запуску програм. Однак VS Code не має такої глибокої інтеграції з Android SDK або Gradle, як Android Studio, тому використовується переважно для консольних або серверних Kotlin-додатків.

Зважаючи на всі характеристики, Eclipse та Visual Studio Code не підходять для розробки, оскільки вони не має глибокої інтеграції з Android SDK, а відсутність необхідності взаємодії з серверною та десктопною розробкою робить IntelliJ IDEA середовищем розробки яке має чудовий, але не потрібний для даної розробки функціонал.

Отже для розробки системи аудіоплеєра для смартфонів на платформі Андроїд буде використовуватись Android Studio.

### 1.4 Постановка задачі дослідження

Проаналізувавши існуючі системи, було визначено головні завдання для розробки аудіоплеєра:

- програма повинна реалізовувати функції відтворення аудіофайлів;
- користувач має доступ до навігації по плейлистах;
- користувач може керувати відтворенням (пауза, перемотування, повтор, перемішування, тощо);
- застосунок має зрозумілий інтерфейс;
- необхідно створити підтримку фонової роботи та керування програванням через панель сповіщень.

Для реалізації програмного продукту буде використано Android Studio як основне середовище розробки, а мова програмування Kotlin забезпечить сучасний, безпечний та лаконічний синтаксис. У процесі розробки буде дотримано принципів модульності, повторного використання коду та забезпечено мінімальне споживання системних ресурсів, що важливо для мобільних пристроїв.

## 1.5 Висновки

У першому розділі було досліджено загальні вимоги та сучасний стан розробки мобільних аудіоплеєрів на платформі Android. Було визначено, що попит на локальні музичні плеєри залишається стабільно високим, незважаючи на домінування стримінгових сервісів. Це пов’язано з тим, що багато користувачів потребують швидкого доступу до особистої музичної бібліотеки без підключення до мережі, зайвої реклами або додаткових підписок.

Аналіз популярних мобільних застосунків, таких як YouTube Music, VLC, Poweramp, AIMP, дозволив виявити їхні сильні та слабкі сторони. Стрімінгові сервіси мають потужну рекомендаційну систему, але втрачають актуальність у відсутності інтернету. Натомість плеєри для локального використання відзначаються стабільністю, широким форматом підтримки та додатковими можливостями, але мають певні недоліки.

У результаті аналізу було сформульовано вимоги до власного програмного продукту, який вимагатиме надійне відтворення аудіофайлів, простий інтерфейс, підтримка фонової роботи, керування через панель сповіщень та мінімалізм у дизайні. Ці характеристики дозволять створити зручний у роботі інструмент для повсякденного користування на Android-пристроях.

Окрему увагу було приділено мові програмування Kotlin, яка забезпечує високу продуктивність, безпечне управління пам’яттю та зручний синтаксис. Android Studio, як офіційне середовище розробки, дозволяє реалізувати проєкт.

Таким чином, задача розробки програмного продукту полягає в створенні стабільного, легкого у використанні Android-застосунку, що відповідає базовим потребам користувачів у локальному аудіовідтворенні, не перевантажений непотрібними можливостями та не залежить від доступу до Інтернету.

# 2 АНАЛІЗ ВХДНИХ ДАНИХ, РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ ФУНКЦІОНАЛУ СИСТЕМИ

## 2.1 Аналіз вхідних даних системи

У розробці мобільного аудіоплеєра важливу роль відіграє правильне опрацювання та інтерпретація вхідних даних, які забезпечують коректну роботу програми. Вхідні дані системи поділяються на кілька основних груп залежно від джерела, формату та ролі у функціонуванні застосунку.

Основним джерелом даних для програми є аудіофайли, що зберігаються у внутрішній або зовнішній пам’яті пристрою. Зазвичай це файли у форматах MP3, WAV, FLAC, OGG та інші, підтримувані Android API [22]. Кожен файл має технічні характеристики: назва, шлях до розташування, розмір, тривалість, частота дискретизації, а також вбудовані метадані. Ці дані зчитуються з аудіофайлів та використовуються для відображення інформації користувачеві. До них належать:

- назва треку (обов’язкова);
- ім’я виконавця (якщо воно вказане);
- назва альбому (якщо вона вказана);
- обкладинка (якщо вона є);
- довжина аудіофайлу;
- шлях аудіофайлу.

Ці дані зчитуються за допомогою бібліотеки MediaMetadataRetriever та інших вбудованих бібліотек Android SDK.

Дії користувача також є вхідними даними. До них належать:

- вибір файлу для відтворення зі списку;
- керування програванням;
- регулювання гучності;
- вибір режиму (повтор, перемішування);
- створення та редагування плейлистів.

Такі дії обробляються через UI-компоненти: кнопки, слайдери, списки, тощо, і вимагають налаштування слухачів подій.

Система може надсилати певні події, які впливають на поведінку плеєра:

- підключення або відключення навушників;
- блокування екрану;
- зміна файлу, що відтворюється;
- зміна гучності;
- зміна налаштувань еквалайзера;
- завершення роботи системи.

Обробка таких подій є критично важливою для коректної та безперервної роботи застосунку. Для цього застосовуються механізми, такі як “BroadcastReceiver”, “AudioManager” та “Service”.

Налаштування користувача є додатковими параметрами, що зберігаються у локальному сховищі та зчитуються під час запуску програми. Ці параметри зберігають інформацію про останній програний трек, обрану тему інтерфейсу та режим прослуховування.

Отже, розробка програмних модулів для системи аудіоплеєра працює з різними видами вхідних даних, серед яких є аудіофайли, метадані, події користувача та сигнали від операційної системи. Швидка обробка цих даних є основою для створення зручного, стабільного та адаптивного застосунку.

## 2.2 Розробка моделі системи

Розробка моделі системи є важливим етапом у процесі створення програмного забезпечення, оскільки він дозволяє чітко визначити основні компоненти, їх взаємодію, а також логіку функціонування системи ще до написання коду. Необхідно детально розглянути архітектуру застосунку, визначити основні модулі, їх функції та взаємозв’язки, а також представити діаграми, що описують роботу системи.

Відкривши застосунок, користувач матиме доступ до необхідних функцій, що дозволять відтворити аудіофайл, знайти аудіофайл за назвою, створити

плейлист, перейти до налаштувань, створити випадковий порядок відтворення та переглянути метадані аудіофайлу. Відтворивши аудіофайл, аудіоплеєр надасть користувачу доступ до розширеного інтерфейсу відтворення. В ньому користувач зможе взаємодіяти з процесом відтворення вибираючи інші аудіофайли, змінювати режим відтворення, гучність, змінити налаштування еквалайзера, додати файл до списку улюблених та використати таймер. Повернувшись до головного екрану користувач матиме такі ж самі можливості, як і до початку програвання, а також змогу зупиняти та продовжувати відтворення.

Взаємодія користувача із системою представлена на рис. 2.1 у вигляді моделі діяльності.

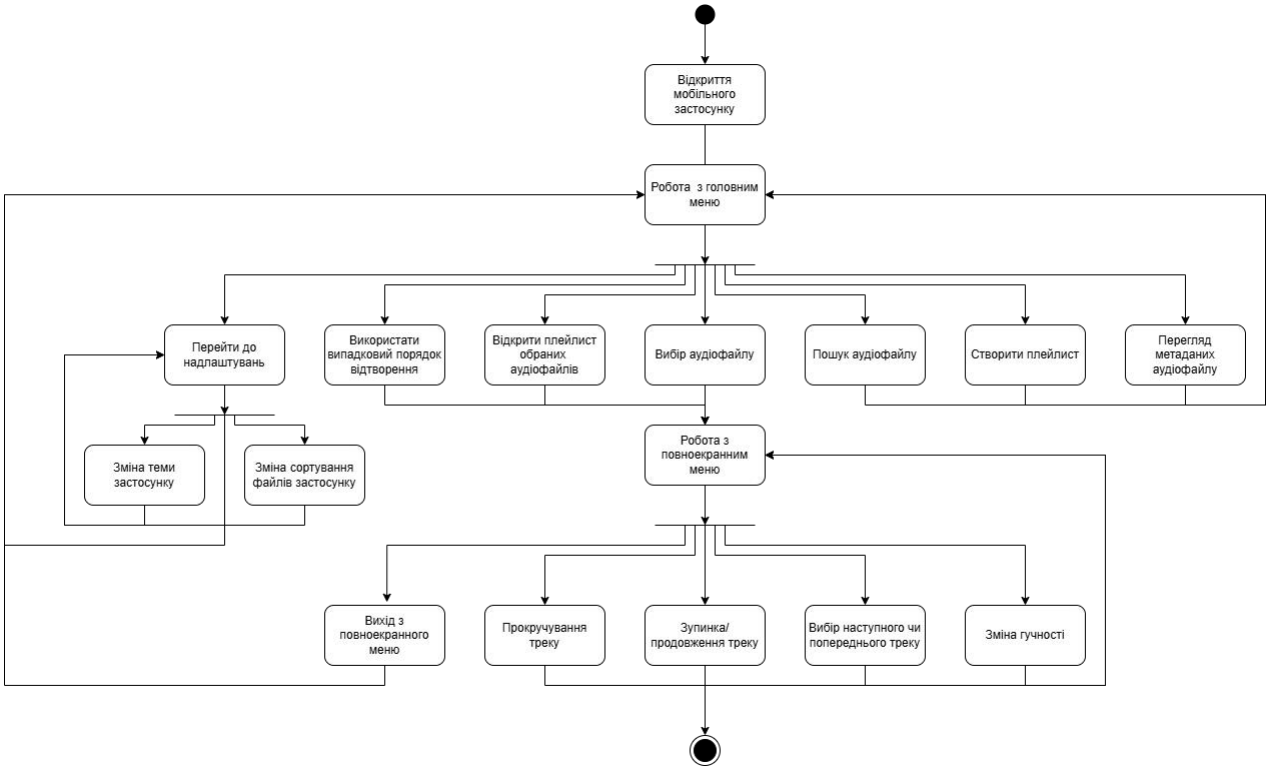


Рисунок 2.1 – Модель роботи системи у вигляді діаграми діяльності

Для реалізації аудіоплеєра обрано архітектурну модель MVVM (Model–View–ViewModel), яка дозволяє чітко відокремити логіку даних, бізнес-логіку та

відображення. Це покращує підтримуваність і тестованість коду, а також спрощує розширення функціоналу в майбутньому.

У цій моделі Model відповідає за доступ до даних, включає структури треків, зчитування метаданих, роботу з файлами, View реалізує графічний інтерфейс користувача, а ViewModel виступає посередником, що обробляє логіку відтворення, реагує на дії користувача та передає оновлення до інтерфейсу.

Модуль аудіовідтворення відповідає за запуск, паузу, перемотування та зупинку відтворення. Реалізований з використанням класу “MediaPlayer” . Модуль також обробляє зміну аудіофокусу, фонову роботу та переходи між треками.

Модуль управління плейлистами дозволяє створювати, зберігати, редагувати та видаляти списки відтворення. Працює з файловою системою пристрою для збереження структури плейлистів.

Модуль відображення даних забезпечує виведення інформації про поточний трек, список композицій, плейлисти та обкладинки альбомів. Інтерфейс адаптований до різних розмірів екранів і підтримує темну та світлу теми.

Сервіс фонового відтворення є окремим Android-сервісом, який забезпечує безперервне відтворення при згорнутому застосунку. Дозволяє керувати музикою з панелі сповіщень.

Модуль обробки подій реагує на взаємодію користувача з додатком (натискання кнопок, прокручування, вибір треків), а також на системні події (підключення навушників, зміна аудіофокусу, тощо).

Також для моделювання аудіотреків використовується спеціальний клас “Music”, який містить такі поля:

- id: String;
- title: String;
- artist: String;
- album: String;
- duration: Long;
- path: String;
- artUri: String.

Плейлист моделюється як список таких треків, збережених у локальній пам’яті.

- Music – модель даних, що містить метадані аудіофайлу.

- AudioPlayerManager – клас-контролер, який працює з MediaPlayer, управляє станами відтворення.

- TrackListAdapter – адаптер для виводу списку треків у RecyclerView.

- MainViewModel – ViewModel, який координує дії між AudioPlayerManager та UI.

- MainActivity / PlayerFragment – компоненти інтерфейсу, які взаємодіють з ViewModel та виводять дані.

У підсумку, модель системи охоплює логічну, архітектурну та структурну частини мобільного застосунку. Такий підхід забезпечує надійність, масштабованість та зручність розробки.

### 2.3 Розробка алгоритмів роботи системи

Розробка алгоритмів – це один з найважливіших етапів проєктування функціональної частини програмного забезпечення, який визначає порядок виконання операцій, логіку обробки подій та взаємодію модулів. У цьому розділі розглядаються основні алгоритми, реалізовані в аудіоплеєрі, з описом їх функцій, вхідних та вихідних даних, а також особливостей реалізації у середовищі Android з використанням мови Kotlin.

Алгоритм ініціалізації застосунку зчитатує аудіофайли з пристрою, щоб проаналізувати метадані та відобразити список треків.

1. Запит дозволу на читання зовнішнього сховища для версій Android 10 і нижче.

2. Сканування пам’яті пристрою за допомогою ContentResolver.

3. Формування списку треків у вигляді об’єктів Music.

4. Зчитування метаданих (назва, виконавець, тривалість, обкладинка).

5. Передача списку до ViewModel.

6. Відображення треків у RecyclerView.

Алгоритм обробки вибору треку зображений на рис. 2.2, та має на меті запустити відтворення вибраного треку, після чого оновити інтерфейс.

1. Користувач натискає на елемент у списку.

2. View викликає метод у ViewModel, передаючи ID треку.

3. ViewModel ініціалізує AudioPlayerManager з обраним треком.

4. AudioPlayerManager створює або за необхідності перезапускає MediaPlayer.

5. Починається відтворення.

6. ViewModel оновлює LiveData з інформацією про активний трек.

7. UI автоматично відображає назву, виконавця та обкладинку.

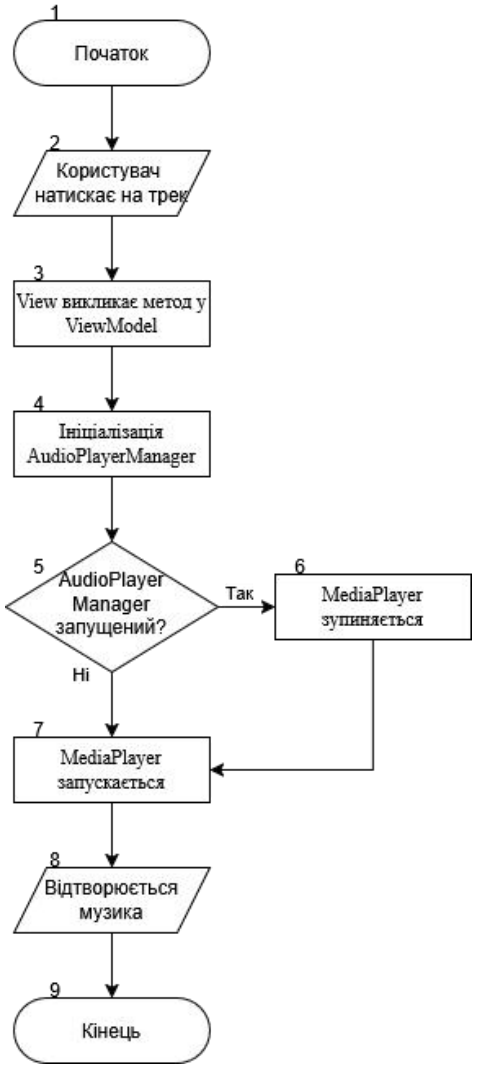


Рисунок 2.2 – Блок-схема алгоритму для вибору треку

Алгоритм фонової роботи застосунку дозволяє забезпечити безперервне відтворення у фоновому режимі.

1. При запуску відтворення створюється ForegroundService.

2. У сервісі створюється повідомлення з кнопками керування.

3. Обробники кнопок  дозволяють призупинити/відновити трек та перейти до наступного/попереднього треку.

4. Сервіс підтримується активним до завершення відтворення або зупинки користувачем.

5. Після зупинки – сервіс припиняє свою роботу.

Алгоритм переходу між треками реалізує функцію перемикання треків, з урахуванням порядку та режимів.

1. Користувач натискає кнопку «Next» або «Previous».

2. ViewModel визначає поточну позицію треку у списку.

3. Враховується режим (звичайний, повтор, випадковий).

4. Обирається новий трек відповідно до логіки.

5. ViewModel оновлює AudioPlayerManager і запускає новий трек.

6. Оновлюється UI відповідно до нового об’єкта Music.

Алгоритм оновлення прогресу треку дозволяє відобразити поточну позицію програвання у реальному часі.

1. У AudioPlayerManager запускається таймер .

2. Кожні N мілісекунд оновлюється значення позиції.

3. ViewModel оновлює LiveData з поточним прогресом.

4. UI-елемент SeekBar отримує нове значення та візуально оновлюється.

5. За потреби користувач може перемотати вручну, що також передається у MediaPlayer.

Алгоритм завершення роботи надає можливість коректно завершити відтворення, зупинити сервіси та звільнити ресурси.

1. Користувач натискає кнопку «Stop» або виходить із застосунку.

2. ViewModel надсилає команду на зупинку у AudioPlayerManager.

3. MediaPlayer зупиняється і звільняє ресурси.

4. Зупиняється ForegroundService.

5. Очищуються LiveData та посилання на об’єкти, щоб уникнути витік пам’яті.

Таким чином розроблено поведінку самої системи, а на рисунку 2.5 зображено поведінку користувача із самою системою.

Розроблені алгоритми забезпечують логічну та послідовну взаємодію між користувачем, інтерфейсом та механізмами відтворення. Вони гарантують стабільну роботу програми, ефективне використання системних ресурсів і зручне керування музикою у фоновому режимі. Правильно структуровані алгоритми дозволяють легко масштабувати та доповнювати функціонал у майбутньому.

### 2.4 Розробка методу пошуку аудіофайлів на пристрої

Для роботи системи відтворення аудіофайлів, застосунок вимагає доступу до усіх аудіофайлів пристрою. Але створення списку існуючих файлів потребує сканування файлової системи, що, починаючи з Android 10 частково забороняється самою системою, що результує у не повному скануванні файлів або ж сканування не відбудеться зовсім.

Таким чином, у випадку створення власного сканера файлової системи, використовуючи пошук у глибину, швидкодія буде завжди O(|V|+(|F|+|A|)), де |F| – це кількість інших файлів що не потрібні застосунку, |M| – це кількість медіафайлів, серед яких є необхідні для роботи аудіофайли, а |E| – кількість директорій, в яких знаходяться файли. Це означає, що якщо певний пристрій має велику кількість файлів, що не відповідають критеріям пошуку, для прикладу 10 тисяч файлів, то на обробку такої кількості може піти від 2 до 30 секунд, що не є прийнятним.

Шляхом використання контент-провайдера в якості посередника між застосунком та файловою системою пристрою, можна значно скоротити швидкодію до O(|V|+|A|), оскільки в такому випадку застосунок не муситиме перевіряти кожен файл директорій, а матиме прямий доступ до медіафайлів пристрою.

Таким чином, використання контент-провайдера вбудованого в пристрій дозволить скоротити час пошуку файлів від декількох секунд, до 100-300 мілісекунд.

### 2.5 Висновки

Отже, у цьому розділі було спроєктовано модель взаємодії користувача з системою та найважливіші алгоритми, що визначають поведінку аудіоплеєра на платформі Android. Всі алгоритми розроблені з урахуванням принципів надійності та відповідності особливостям мобільної платформи, особливо для пристроїв з обмеженими ресурсами,  та необхідністю забезпечення безперервного користувацького досвіду.

Розроблені алгоритми охоплюють основні сценарії взаємодії користувача з додатком від ініціалізації та зчитування аудіофайлів до керування відтворенням у фоновому режимі. Важливою частиною проєкту є також алгоритми керування переходами між треками, оновленням прогресу та обробки подій. Вони дозволяють реалізувати сучасний, зручний для користувача інтерфейс, що реагує на дії користувача використовуючи мінімум ресурсів пристрою та часу, а розроблений метод сканування файлів на пристрої дозволить значно збільшити швидкодію застосунку та зменшити затримки під час кожного сканування.

Таким чином, розроблена модель системи повністю відповідає поставленим задачам і дозволяє перейти до безпосередньої реалізації візуальної та програмної частини аудіоплеєра.

# 3. РОЗРОБКА ІНТЕРФЕЙСУ ТА ПРОГРАМНИХ МОДУЛІВ СИСТЕМИ АУДІОПЛЕЄРА ДЛЯ СМАРТФОНІВ НА ПЛАТФОРМІ АНДРОЇД

## 3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Ефективність розробки програмного забезпечення значною мірою залежить від правильно обраних інструментів, мови програмування та середовища розробки. У цьому розділі проведено порівняльний аналіз можливих варіантів реалізації мобільного застосунку, після чого обґрунтовано вибір конкретного технологічного стеку для розробки аудіоплеєра.

Для реалізації мобільного аудіоплеєра було використано підхід нативної розробка для Android.

Нативна розробка дає повний доступ до всіх можливостей платформи, забезпечує кращу продуктивність і точний контроль над системними компонентами. Для аудіоплеєра, який працює з потоковим відтворенням, сервісами та взаємодіє з медіа функціями пристрою, саме нативний підхід є оптимальним.

Для створення Android-застосунку використовувалося середовище Android Studio.

Android Studio (див. рисунок 3.1) є офіційним середовищем розробки від Google, спеціально орієнтованим на Android-додатки. Воно має вбудовані засоби проєктування інтерфейсу, відлагодження, інструменти профілювання продуктивності, інтеграцію з Gradle, підтримку Jetpack Compose та повну інтеграцію з Kotlin. Через ці якості Android Studio було обрано середовищем розробки аудіоплеєра.

Важливою функцією Android Studio є можливість емулювати середовище роботи різних пристроїв для тестування роботи програмної системи без застосування сторонніх пристроїв.

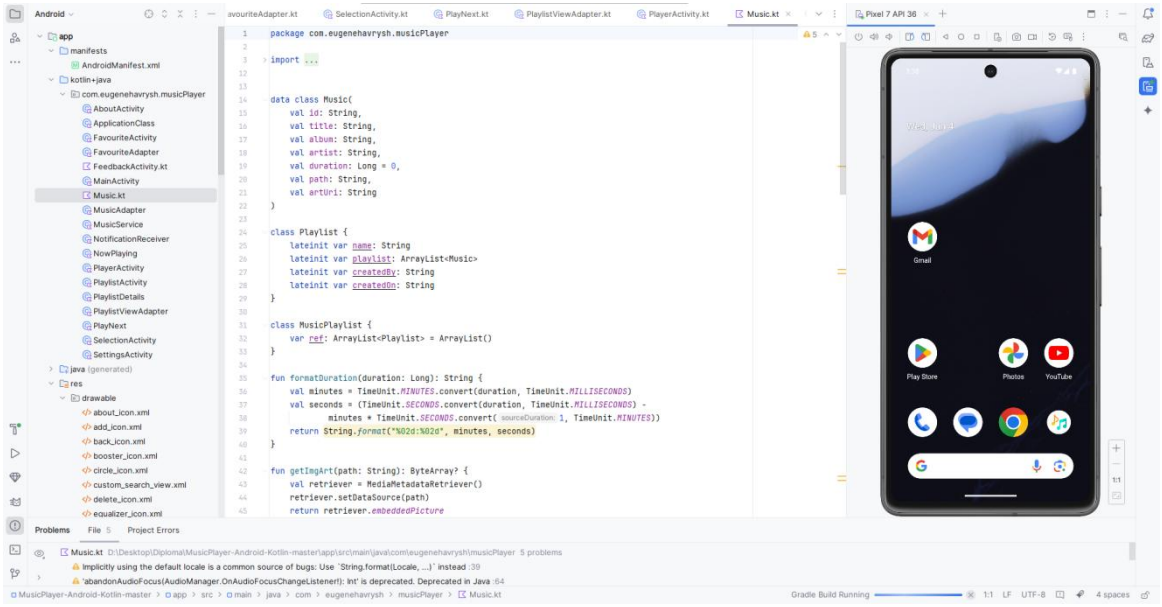

Рисунок 3.1 – Зображення інтерфейсу Android Studio

Android Studio підтримує розробку застосунків на мовах програмування Java та Kotlin.

Java – одна з найпопулярніших мов програмування загального призначення, яка підтримує розробку додатків для Android та має високу сумісність з багатьма бібліотеками [16].

Kotlin – це сучасна мова, яка офіційно підтримується Google для Android з 2017 року. Вона дозволяє писати компактний та читабельний код, також має підтримку асинхронності, безпечну роботу з null-значеннями, а велика кількість бібліотек створених для Java можуть використовуватись і нею [17].

Через це Kotlin було обрано мовою реалізації застосунку, а у процесі розробки застосунку планується використання таких інструментів:

- MediaPlayer для відтворення аудіо;
- Room для локального зберігання плейлистів;
- LiveData та ViewModel для реалізації моделі MVVM;
- Data Binding / View Binding для зручного зв’язку між UI та логікою;
- ForegroundService для фонового відтворення;
- Notification API для створення елементів керування в панелі повідомлень.

Вибір цих технологій обумовлений:

- високим рівнем інтеграції інструментів;
- підтримкою сучасних архітектур;
- стабільною роботою з мультимедійними API;
- високою продуктивністю та масштабованістю розробки.

Такі технології дозволяють реалізувати якісний, функціональний та надійний аудіоплеєр із підтримкою всіх необхідних функцій для мобільного програвача аудіофайлів. Такий підхід забезпечує хорошу підтримку коду, можливість подальшого розширення та зручність для розробника.

### 3.2 Розробка інтерфейсу програмного додатку

У процесі розробки системи аудіоплеєра на платформі Андроїд, створенню зручного та інтуїтивно зрозумілого користувацького інтерфейсу необхідно виділити особливу кількість уваги.

Інтерфейс користувача є посередником користувача та пристрою та охоплює все, що користувач бачить та може використовувати на екрані пристрою. Саме тому зручний інтерфейс може сильно спростити ознайомлення нового користувача з новою системою, що дозволить користувачам працювати з ним без зайхив зусиль, що особливо важливо під час розробки мобільних застосунків.

Для аудіоплеєра на платформі Android необхідно зробити декілька екранів: головний екран, що бачить користувач запустивши застосунок з боковою панеллю, що дозволяє перейти до налаштувань, панель налаштувань застосунку для зміни його роботи та графічного інтерфейсу, а також екран з усіма функціями, необхідних для аудіоплеєра.

Перший екран, з яким зустрічається користувач, представляє собою головний екран додатку, де користувач одразу може вибрати файл та прослуховувати музику (див. рисунок 3.2). Також користувач може використати пошук, за натиском кнопки створити плейлист створений із випадково обраних треків, перейти до обраних файлів, створених плейлистів чи переглянути порядок

програвання, який користувач може змінювати вручну, затиснувши назву треку та натиснувши відповідну кнопку.

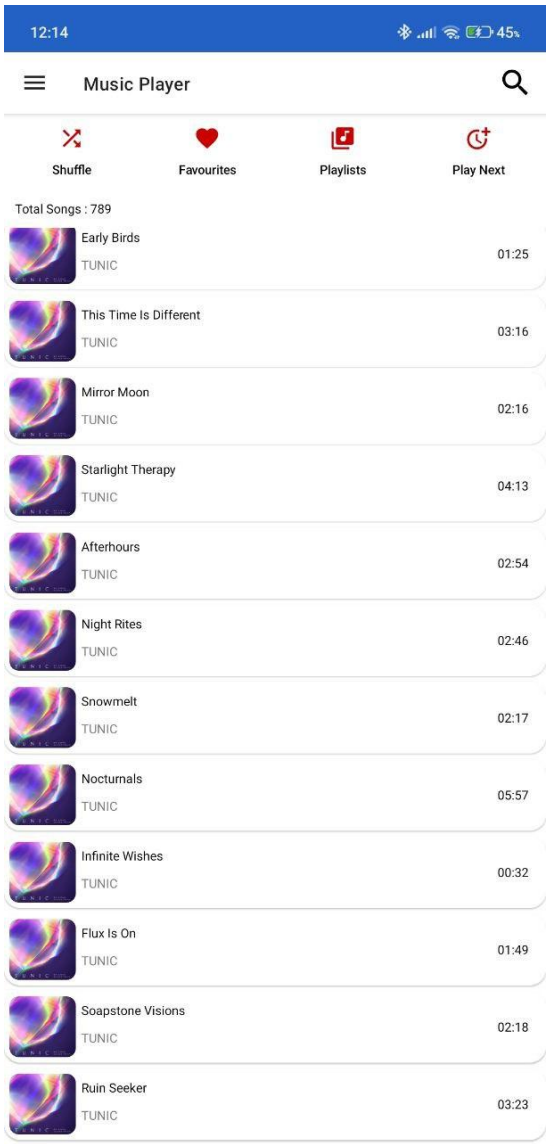


Рисунок 3.2 – Зображення головного екрану застосунку

Як можна побачити на рисунку 3.3, кожен елемент списку складається із таких елементів:

- зображення;
- назви треку;
- альбому;
- довжини мелодії.


Рисунок 3.3 – Зображення елементу списку на головному екрані застосунку

Обравши мелодію, розкривається меню прослуховування, зображене на рис. 3.4, де користувач має можливість змінити мелодію на попередню чи наступну, зупинити та продовжити програвання, змінити сегмент файлу, що програється, змінити режим повторення файлу, перейти до налаштувань еквалайзера, налаштувати таймер, та змінити гучність аудіоплеєра.

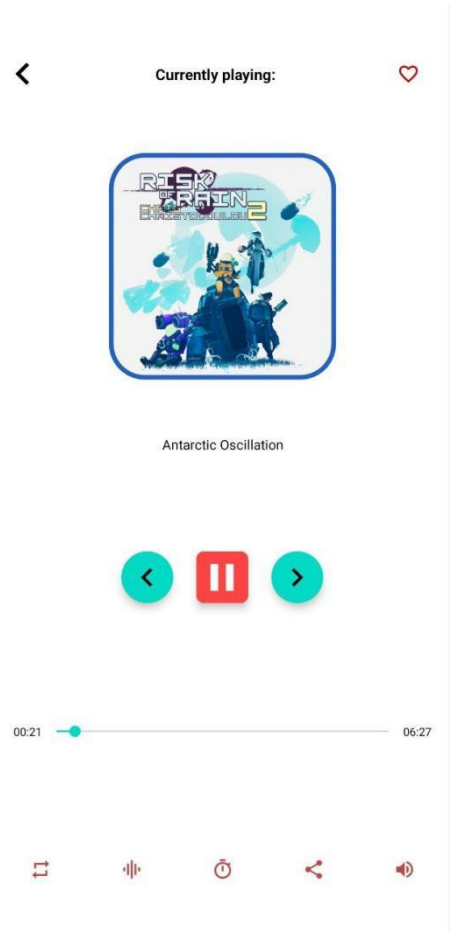


Рисунок 3.4 – Зображення розширеного меню керування аудіофайлами

Натиснувши на еквалайзер, користувач відкриває меню еквалайзера, представлене на рис. 3.5, що дозволяє змінити певні частоти аудіофалу, змінивши їх звучання. Також, користувач може вибрати інші профілі, або ж налаштувати власний.

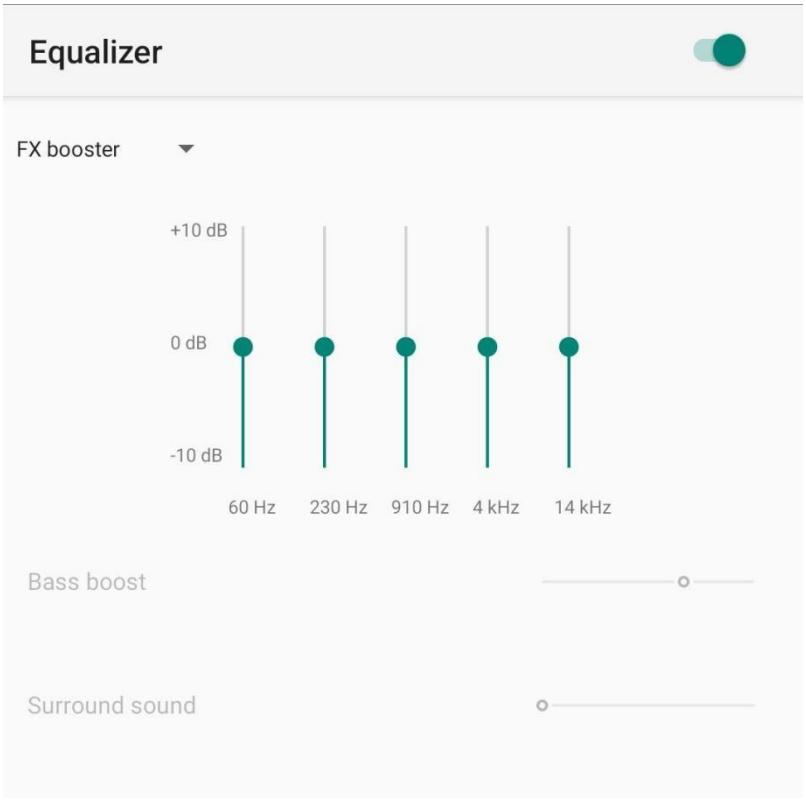


Рисунок 3.5 – Зображення інтерфейсу еквалайзера

Натиснувши на таймер, користувач отримує декілька варіантів для зупинки програвання у випадаючому меню, як показано на рис. 3.6.


Рисунок 3.6 – Зображення інтерфайсу таймера

Обравши мелодію та повернувшись в головне меню, знизу на головному екрані (див. рисунок 3.7) з’являється меню для швидкого керування, що необхідно для користувача та дозводяє переглянути назву файлу, що програється, зупинити та продовжити відтворення, перейти до наступного треку, в також відкрити переходу до розширеного меню.

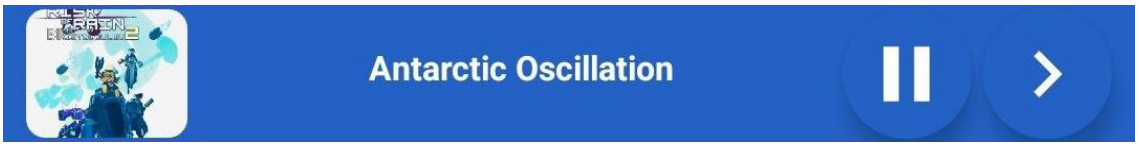


Рисунок 3.7 – Зображення елементу керування треком на головному екрані застосунку

У боковому меню користувач має можливість перейти до налаштувань застосунку, зображениому на рис. 3.8, де користувач може змінити тему застосунку та порядок сортування аудіофайлів на головному екрані.

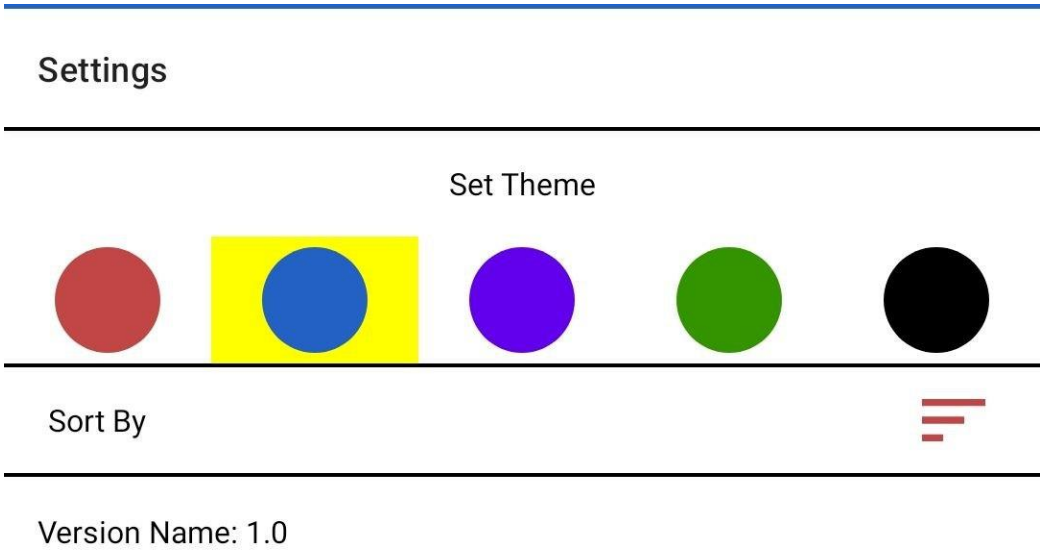


Рисунок 3.8 – Зображення налаштувань застосунку

Для розробки інтерфейсу користувача мобільного аудіоплеєра на платформі Android було використано сучасні технології та засоби, що відповідають актуальним стандартам Android-розробки.

Базова структура екранів застосунку була розроблена за допомогою XML-файлів (див. рисунок 3.9). У них описується розташування елементів інтерфейсу, їх розміри, відступи, кольори, шрифти тощо, та взаємне позиціонування. XML забезпечує чіткий поділ логіки і презентації, що спрощує розробку та підтримку.

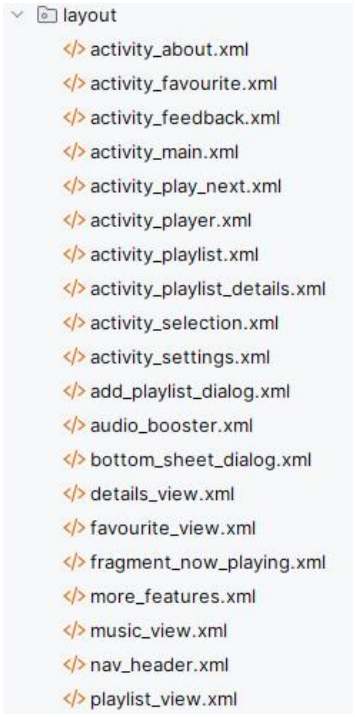


Рисунок 3.9– Зображення структури XML-файлів

У якості основного контейнера для побудови макетів було обрано ConstraintLayout, що дозволяє створювати адаптивні, складні інтерфейси з мінімальним рівнем вкладеності. Він забезпечує високу продуктивність та зручність налаштування відносних позицій елементів, що особливо важливо для різних розмірів екранів. Інтерфейс реалізовано відповідно до принципів Material Design, що забезпечує сучасний вигляд та зручну взаємодію. Було використано

бібліотеку “com.google.android.material” для основних дій користувача, у разі реалізації кількох розділів та для повідомлень та підтверджень дій.

Інтерфейс налаштовано на підтримку системних тем: темної та світлої. Це реалізовано через використання ресурсів тем (themes.xml), кольорових палітр (colors.xml) і стилів, що автоматично змінюються залежно від налаштувань пристрою.

Вся логіка взаємодії з інтерфейсом реалізована мовою Kotlin. Події натискання кнопок, прокручування, вибору треків тощо обробляються через слухачі setOnClickListener, OnSeekBarChangeListener та RecyclerView.Adapter у відповідних активностях або фрагментах.

Таким чином, комбінація XML-розмітки, стандартних і Material-компонентів Android SDK, ConstraintLayout та Kotlin дозволила створити адаптивний, функціональний і візуально привабливий інтерфейс користувача, що відповідає сучасним вимогам до мобільних застосунків.

### 3.3 Розробка модулів застосунку

Першим функціональним модулем застосунку є MainActivity (див. рисунок 3.10), який відповідає за ініціалізацію головного інтерфейсу, відображення списку аудіофайлів та взаємодію з елементами інтерфейсу на пристрої у застосунку. Клас реалізує інтерфейс activity_main та currently_playing, що дозволяє переглядати відтворення поточного аудіофайлу.

Методи цього класу ініціалізують інтерфейс з урахуванням налаштувань, роблять перевірку необхідних дозволів на доступ до аудіофайлів, отримують список усіх медіафайлів на пристрої, відображають створений список, що сортується відповідно до налаштувань, відображають блок активного програвання та роботу з елементами інтерфейсу.

```
class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding
    private lateinit var toggle: ActionBarDrawerToggle
    private lateinit var musicAdapter: MusicAdapter

    companion object{...}


    override fun onCreate(savedInstanceState: Bundle?) {...}
    //For requesting permission
    private fun requestRuntimePermission() :Boolean{...}


    override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<out String>, grantResults: IntArray) {...}

    override fun onOptionsItemSelected(item: MenuItem): Boolean {...}

    @SuppressLint("SetTextI18n")
    private fun initializeLayout(){...}

    @SuppressLint("Recycle", "Range")
    private fun getAllAudio(): ArrayList<Music> {...}

    override fun onDestroy() {...}

    override fun onResume() {...}

    override fun onCreateOptionsMenu(menu: Menu?): Boolean {...}
}
```

Рисунок 3.10 – Зображення класу MainActivity

Наступний клас, представлений на рис. 3.11, MusicAdapter, обробляє список аудіофайлів та формує графічний інтерфейс для користувача в залежності від того, чи переглядаються всі файли чи лише аудіофайли з певного плейлисту.

```
class MusicAdapter(private val context: Context, private var musicList: ArrayList<Music>, private val playlistDetails: Boolean = false,
private val selectionActivity: Boolean = false)
    : RecyclerView.Adapter<MyHolder>() {

    class MyHolder(binding: MusicViewBinding) : RecyclerView.ViewHolder(binding.root) {...}

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyHolder {...}

    override fun onBindViewHolder(holder: MyHolder, position: Int) {...}

    override fun getItemCount(): Int {...}

    fun updateMusicList(searchList : ArrayList<Music>){...}

    private fun sendIntent(ref: String, pos: Int){...}

    private fun addSong(song: Music): Boolean{...}

    fun refreshPlaylist(){...}
}
```

Рисунок 3.11 – Зображення класу MusicAdapter

Модуль MusicService відповідає за відтворення аудіофайлів та обробку елементів керування, що використовуються для зупинки, продовження та переходів між аудіофайлами. NotificationReceiver має подібний функціонал, але обробляє дії користувача при взаємодії користувача з плеєром в панелі сповіщень пристрою.

Модулі FavouriteAdapter та FavouriteActivity обробляють роботу з обраними користувачем аудіофайлами, відображають їх та надають можливість швидкого та зручного доступу.

Велику частину функціоналу також обробляє модуль PlayerActivity, що відповідає за відображання інформації у розширеному меню та роботу з ним, а також роботу з таймером та еквалайзером.

Отже, завдяки такій структурі застосунок обробляє всі дії користувача, та надає доступ до всього необхідного функціоналу.

### 3.4 Висновки

У третьому розділі було безпосередньо реалізовано основні функціональні компоненти аудіоплеєра для смартфонів на платформі Android.

Розроблений інтерфейс є зручним та не навантажує користувача великою кількості інформації, а вкладка налаштувань дозволяє швидко змінити зовнішній вигляд та критерій сортування аудіофайлів.

Окрема увага була приділена розробці модулів відтворення та обробки аудіофайлів, що реалізують логіку керування відтворенням (запуск, пауза, відновлення, перехід після завершення треку). Такий підхід дає змогу керувати медіаплеєром і адаптувати поведінку програми залежно від стану пісні, а також дозволяє користуватися інтерфейсом програми за допомоги панелі сповіщень, для фонової роботи.

Таким чином, усі елементи програмного забезпечення були реалізовані у відповідності до вимог функціональності та з урахуванням принципів ефективного, гнучкого та масштабованого програмування для мобільної платформи Android.

# 4 ТЕСТУВАННЯ ПРОГРАМИ

## 4.1 Тестування системи

Тестування розробленого аудіоплеєра для Android є важливим етапом перевірки його працездатності, відповідності функціональним вимогам і стабільності під час використання. Цей процес охоплює функціональне, інтеграційне та інтерфейсне тестування з метою виявлення можливих помилок у роботі окремих модулів та системи в цілому.

Мета тестування переконатися, що застосунок правильно відтворює аудіофайли та коректно реагує на дії користувача, правильно обробляє дані медіатеки пристрою, не допускає помилок при взаємодії з інтерфейсом та стабільно працює на різних версіях Android.

Для тестування було використано ручне тестування на емуляторах з версіями Android 11 та 16.

На рис. 4.1 представлено успішне запущення аудіоплеєра на емульованому пристрої з версією Android 16, а на рис. 4.2 – аудіоплеєра на емульованому пристрої з версією Android 11.

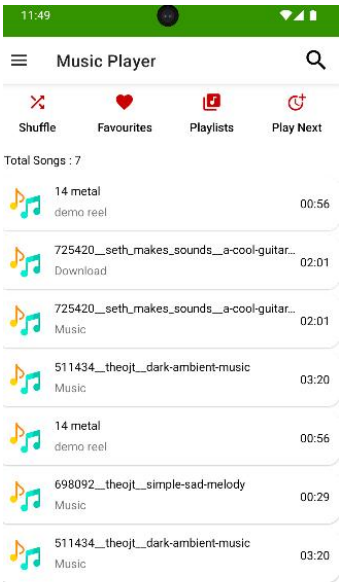


Рисунок 4.1 – Перевірка роботи аудіоплеєра на Android 16

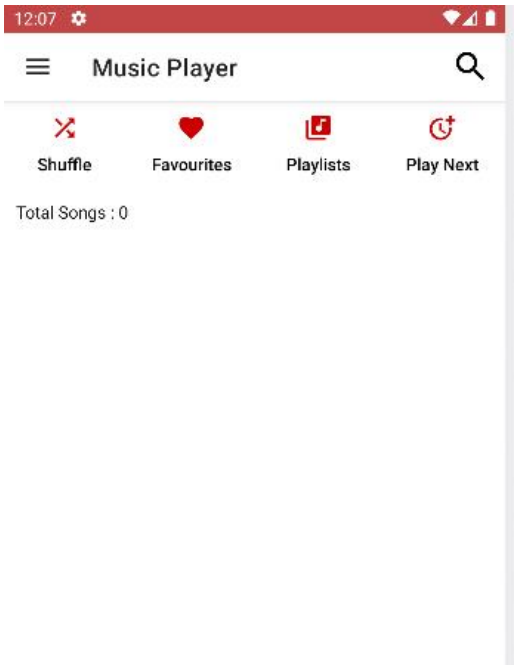


Рисунок 4.2 – Перевірка роботи аудіоплеєра на Android 16

Також, було сформовано низку тестів для перевірки основних функцій системи:

Тестування відображення списку пісень.

Після запуску застосунку очікується, що користувач бачить список пісень з назвою, виконавцем, обкладинкою та тривалістю.

Результат успішний (див. рис. 4.3), дані завантажуються через ViewModel і RecyclerView.

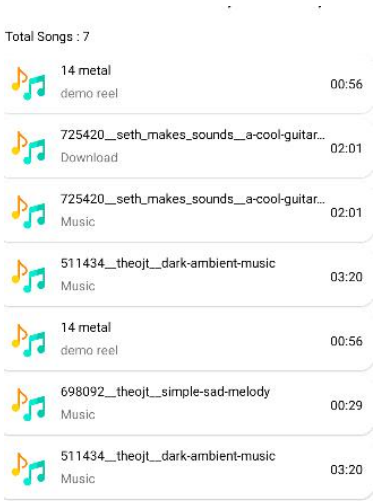


Рисунок 4.3 – Перевірка роботи відображення списку пісень

Тестування відтворення пісні.

Очікування: після натискання на пісню вона починає відтворюватися, розкривається розширене меню для користувача.

В результаті застосунок успішно починає програвання аудіофайлу та розкривається розширене меню (див. рис. 4.4) для подальшої роботи користувача з ним.

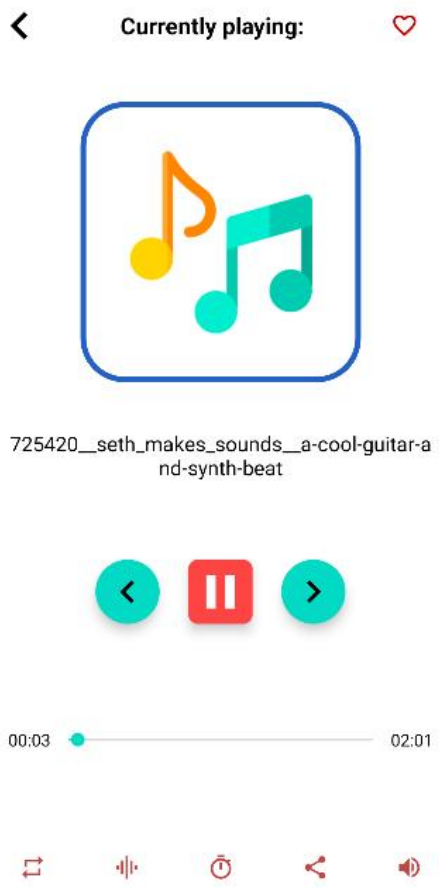


Рисунок 4.4 – Перевірка відтворення пісні

Тестування призупинення та продовження відтворення.

Очікується, що при натисканні кнопки «пауза» відтворення припиняється, а при повторному натисканні – відновлюється.

Результат успішний (див. рис. 4.5), зміна стану медіаплеєра обробляється правильно.

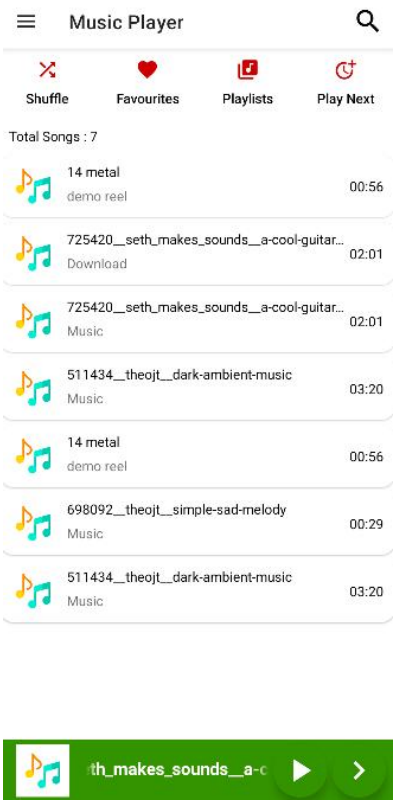


Рисунок 4.5 – Перевірка призупинення та продовження відтворення

Тестування відображення обкладинки альбому.

Очікується, що під час відтворення пісні зображення альбому оновлюється.

Результат у більшості випадків – успішно (див. рис. 4.6), також, при відсутності зображення використовується зображення за замовчуванням.

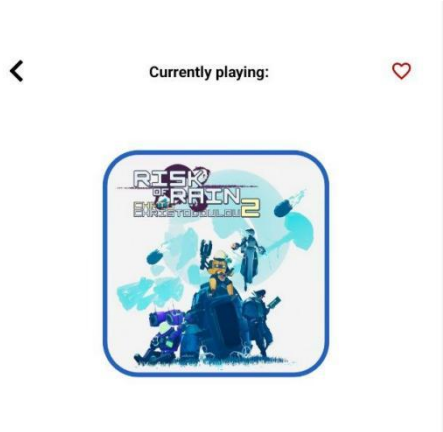


Рисунок 4.6 – Перевірка відображення обкладинки альбому

Тестування еквалайзера.

Очікується, що відкривши еквалайзер користувач може змінити профіль звуку аудіоплеєра, використовуючи існуючі профілі, або ж створивши власний.

Результат: успішний, відкривши еквалайзер користувач може обрати один з вбудованих профілів у випадаючому списку або ж створити свій. Будь-які модифікації вбудованого профілю зберігають усі зміни в профіль користувача, що показано на рис. 4.7.

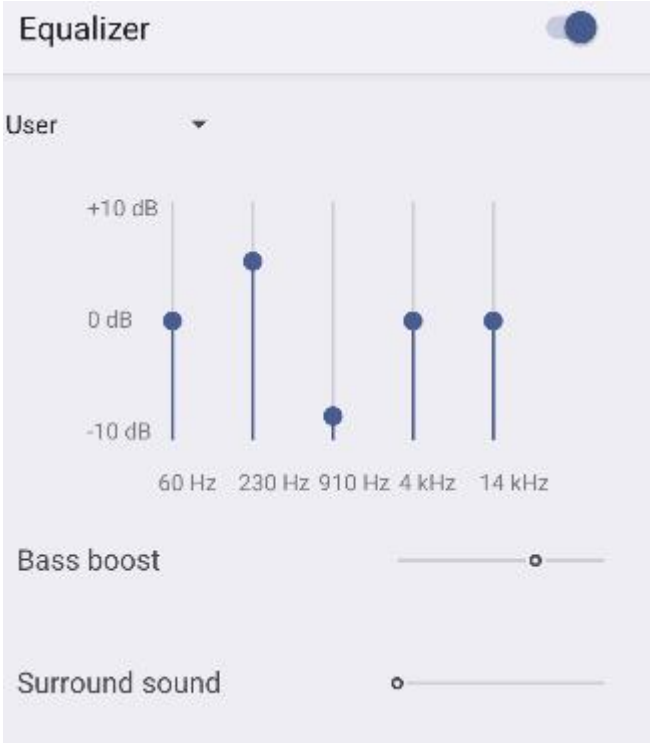


Рисунок 4.7 – Перевірка роботи еквалайзера

Тестування роботи аудіоплеєра у панелі сповіщень.

Очікування: під час використання застосунку у режимі фонової роботи, користувач може використовувати панель сповіщень для доступу до певних функцій.

Результат: успішний, як показано на рис. 4.8, користувач може використовувати панель сповіщень для зміни активного аудіофайлу, його прокручування та зупинки й продовження відтворення. Панель сповіщень працює

лише якщо користувач мав відтворюваний аудіофайл на момент переходу застосунку у фоновий режим. В інакшому випадку панель не створюється для зменшення використання ресурсів пристрою. У випадку зупинки мелодії, функція працює допоки користувач або пристрій повністю не завершить роботу програми.

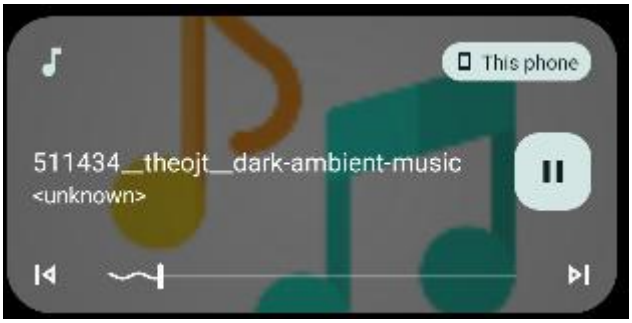


Рисунок 4.8 – Перевірка роботи аудіоплеєра у панелі сповіщень

Тестування роботи таймера сну.

Очікується, що при натисканні на таймер користувач може обрати час, через який застосунок самостійно зупиняє відтворення аудіофайлів.

В результаті застосунок надає три опції (див. рис. 4.9), які вказують через який час програма зупиняє відтворення.

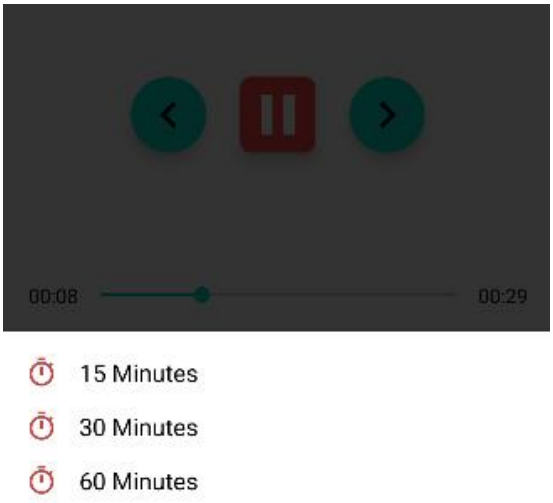


Рисунок 4.9 – Перевірка таймера сну

Тестуання налаштувань застосунку.

Очікується, що в боковому меню користувач має доступ до налаштувань застосунку, де він може змінити вигляд та порядок сортування файлів на головному екрані.

Результатом перевірки є успішна зміна інтерфейсу та порядку аудіофайлів у списку. Користувач має доступ до чотирьох світлих тем інтерфейсу, та однієї темної, як показано на рис. 4.10, а також може змінити порядок аудіофайлів відповідно до їх розміру, назви чи дати завантаження.


Рисунок 4.10 – Перевірка налаштувань застосунку

Тестування створення та видалення плейлистів.

Очікується, що у верхній панелі користувач має доступ до створення окремих списків відтворення, де він може створити їх вручну, а також почати їх відтворення.

Результатом перевірки є екран (див. рис. 4.11), де користувач може створювати та видаляти списки відтворювання. Для створення плейлиста необхідно натиснути на відповідну кнопку та ввести назву та автора. Для видалення плейлиста необхідно натиснути на кнопку видалення, яка є в кожного плейлиста окремо, після чого користувач повинен підтвердити або ж відхилити дію.

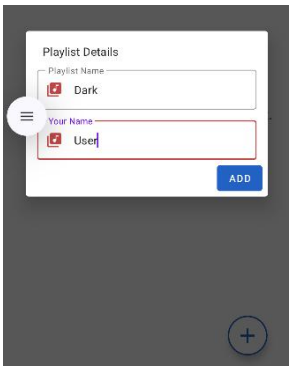


Рисунок 4.11 – Перевірка створення та видалення плейлистів

Тестування роботи з плейлистами.

Очікується, що обравши плейлист користувач може додати та відтворити аудіофайли в ньому, а також внести зміни, додавши або ж видаливши певні аудіофайли.

У результаті застосунок дозволяє додати та видалити аудіофайли в окремому меню, після чого користувач має можливість прослуховувати їх у власноруч створеному порядку, або ж використати функцію перемішування, що показано на рис. 4.12. Також є опція видалення усього списку аудіофайлів плейлиста, обравши яку користувач матиме можливість підтвердити чи змінити свій вибір.

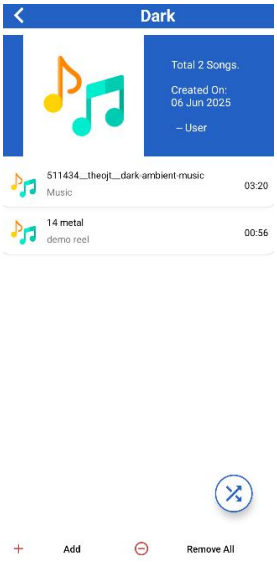


Рисунок 4.12 – Перевірка роботи з плейлистами

Тестування роботи з обраними аудіофайлами.

Очікування: після того, як користувач натисне в розширеному меню опцію додати аудіофайл до обраних, в меню обраних буде додано цей файл та інші попередньо додані файли в порядку додавання.

У результаті перевірки було додано декілька файлів зі списку, після чого вони успішно були додані до екрану обраних, що показано на рис. 4.13. Користувач може натиснути на файл та розпочати відтворення, або ж вибрати опцію перемішування, яка почне їх відтворювати у випадковому порядку.

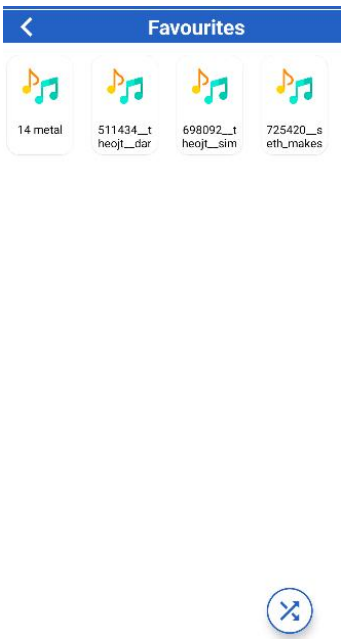


Рисунок 4.13 – Перевірка роботи з обраними аудіофайлами

У процесі тестування було виявлено декілька незначних недоліків:

- на старших версіях Android нижче 11 можливі труднощі з доступом до деяких аудіофайлів через зміну політик безпеки;

- при дуже великій кількості пісень список може прокручуватись із затримками на пристроях з обмеженими ресурсами;

- зручність користування аудіоплеєром у фоновому режимі сильно залежить від пристрою користувача;

- відсутність опції створення свого часу для таймера;

- при запуску аудіофайлу певні пристрої отримують звуковий сигнал сповіщення через створення такого для роботи у фоновому режимі.

Тестування підтвердило, що система в цілому працює стабільно та виконує всі поставлені функціональні вимоги. Основні модулі взаємодіють між собою коректно, а інтерфейс реагує на дії користувача швидко та без збоїв. Програма готова до подальшого розширення та оптимізації з урахуванням виявлених недоліків.

### 4.2 Розробка інструкції користувача

Інструкція користувача потребує визначення технічних вимог для запуску програмного продукту. Деталі щодо мінімальної та рекомендованої конфігурації пристрою для запуску застосунку можна знайти в таблицях 4.1 та 4.2.

Таблиця 4.1 – Мінімальна конфігурація:

| Тип процесора | Qualcomm Snapdragon 680 |
|---|---|
| Об’єм оперативної пам’яті | 2 ГБ для 64-разрядної системи |
| Місце на жорсткому диску | 1 ГБ |
| Операційна система | Android 11 |

Таблиця 4.2 – Рекомендована конфігурація:

| Тип процесора | 4× 1.8 GHz Cortex-A55 |
|---|---|
| Об’єм оперативної пам’яті | 4 ГБ для 64-разрядної системи |
| Місце на жорсткому диску | 4 ГБ |
| Операційна система | Android 14, Android 15, Android 16 |

Цей розділ містить покрокову інструкцію щодо встановлення, запуску та використання мобільного аудіоплеєра, створеного в рамках бакалаврської

кваліфікаційної роботи. Інтерфейс застосунку побудований таким чином, щоб бути інтуїтивно зрозумілим навіть для користувачів без технічної підготовки.

Інструкція встановлення застосунку через APK-файл, створеного для бакалаврської кваліфікаційної роботи:

1. Завантажте зібраний APK-файл на пристрій.
2. Увімкніть дозвіл на встановлення програм з невідомих джерел.
3. Встановіть застосунок вручну, відкривши файл.
4. Після завершення – відкрийте застосунок із головного екрана.

При першому запуску програма запитує дозвіл на доступ до мультимедійних файлів пристрою. Після надання дозволу автоматично виконується сканування аудіофайлів. На головному екрані з’явиться список доступних треків.

Основні функції:

1. Перегляд списку пісень: виводиться перелік треків із назвою, виконавцем, тривалістю та обкладинкою альбому (якщо така є), сортування за назвою виконується автоматично.

2. Відтворення музики відбувається при натисканні на будь-яку пісню зі списку, назва треку відображається у нижній частині екрана та супроводжується анімацією прокрутки.

3. Для паузи/продовження необхідно натиснути кнопку паузи, щоб зупинити трек. При повторному натисканні відтворення відновиться з того ж місця.

4. Після завершення одного треку автоматично починається відтворення наступного (при наявності логіки автопрогравання).

5. Динамічне оновлення інтерфейсу: при зміні списку аудіофайлів (наприклад, після додавання нових пісень) – список оновлюється без перезапуску застосунку.

Для коректної роботи переконайтесь, що на пристрої збережені аудіофайли у форматі .mp3, .wav, .ogg або інші, підтримувані MediaPlayer. Якщо зображення альбому не відображається, це означає, що воно відсутнє в медіатеці – у такому

випадку використовується стандартна обкладинка. Застосунок підтримує більшість пристроїв із Android 11 та вище.

Для виходу з програми скористайтесь кнопкою Назад або просто закрийте її з диспетчера задач. Відтворення музики автоматично зупиняється при закритті застосунку.

Інструкція є достатньо простою для користувача та охоплює всі основні аспекти роботи з застосунком. У разі майбутнього зміни або ж розширення функціональності ця інструкція може бути редагована чи доповнена відповідними пунктами.

### 4.3 Висновки

У цьому розділі було створено детальну інструкцію користувача, що описує процес встановлення, запуску та експлуатації мобільного застосунку – аудіоплеєра на платформі Android. Вона охоплює всі ключові дії, які користувач має виконати для початку роботи із застосунком, а також пояснює основні функціональні можливості інтерфейсу.

Завдяки простоті взаємодії та логічній структурі елементів управління, застосунок може бути зручно використаний широким колом користувачів, включно з тими, хто не має технічної підготовки. Ретельно продумане відображення інформації про трек (назва, виконавець, обкладинка, тривалість), а також інтуїтивно зрозуміле керування відтворенням роблять застосунок привабливим та зручним.

Також були враховані особливості взаємодії з файловою системою Android, що дозволило забезпечити стабільне відображення й обробку локальних медіафайлів. Анімація назви треку та обробка відсутніх обкладинок додають завершеності візуальному сприйняттю інтерфейсу.

Таким чином, інструкція користувача виконує важливу роль у забезпеченні доступності та комфортного користування розробленою системою, а також підвищує її загальну зручність та якість взаємодії з кінцевим користувачем.

# ВИСНОВКИ

Під час виконання бакалаврської кваліфікаційної роботи було розроблено програмне забезпечення аудіоплеєра для смартфонів на платформі Андроїд, яка дозволяє користувачам зручно та швидко відтворювати аудіофайли різних форматів.

Проведений аналіз стану сучасних тенденцій у сфері аудіоплеєрів, дозволив визначити вимоги користувачів до таких систем, а аналіз існуючих систем виявив їх переваги та недоліки, за допомоги яких визначено вимоги та набір функцій власної розробки, що необхідний для користування програмною системою.

Також проаналізовано типові методи розробки програмного забезпечення на платформу Андроїд та обрано нативний тип розробки для роботи, після чого визначено задачі дослідження у бакалаврській кваліфікаційній роботі.

Далі досліджено вхідні дані системи, що є необхідними для роботи користувача, опісля створено модель роботи користувача з системою у вигляді діаграми взаємодії, яка містить логіку дій користувача під час користування аудіоплеєром.

Розроблено алгоритми роботи окремих функцій системи, що необхідні для її роботи та покращено метод сканування системи для пошуку аудіофайлів, яка вимагає найбільшої оптимізації.

Представлено створення візуального інтерфейсу системи, з яким користувач може взаємодіяти, а програма може реагувати на його дії та виконувати відповідні функції.

Розроблено модулі та функції програми, які дозволяють прослуховувати та працювати з аудіофайлами як в застосунку, так і за його межами у фоновому режимі, а також створено функціонал, що дозволяє налаштувати роботу та вигляд інтерфейсу відповідно до бажань користувача.

Підтверджено роботу системи аудіоплеєра, шляхом перевірки роботи застосунку на різних версіях платформи Андроїд з використанням емулятора,

вбудованого в середовище розробки. Визначено, що в результаті перевірки система працювала стабільно, функції працювали без помилок, а система відповідала прийнятно швидко.

Створено інструкцію користувача, що допомагає використовувати систему, а також визначено мінімальні та оптимальні вимоги до пристрою, на якому може використовуватись система аудіоплеєра.

Таким чином, розроблено систему відтворення аудіофайлів, що не вимагає мережевого підключення, має весь необхідний функціонал для вибору аудіофайлів та об’єднання їх в окремі списки відтворення, може працювати та керуватися у фоновому режимі, має налаштування гучності та звучання, вигляду та роботи інтерфейсу. Даний функціонал відповідає сучасним потребам користувачів, а зручний інтерфейс дозволяє швидко та легко використовувати застосунок.

Отже, під час роботи над бакалаврською кваліфікаційною роботою успішно досягнуто виконання поставлених цілей та завдань. Розроблена система аудіоплеєра для смартфонів на платформі Андроїд готова для використання у реальних умовах, надаючи зручний та плавний у використанні застосунок для відтворення аудіофайлів.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ліщинська Л. Б., Гавриш Є. О. Типи мобільних застосунків, їх особливості та інструменти їх реалізації // Матеріали LIV всеукраїнської науково-технічної конференції підрозділів Вінницького національного технічного університету (2025). – Вінниця: ВНТУ, 2025. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/24282/20105

2. Екскурс в історію – еволюція аудіоплеєрів від 70х до наших днів. URL: https://www.moyo.ua/ua/news/evolyuciya_audiopleerov_s_1979_goda_i_do_segodnya.html (дата звернення: 19.04.2025).

3. What is the importance of mobile phone nowadays? URL: https://www.imei.info/uk/news/importance-mobile-phone-nowadays/ (last accessed: 19.04.2025).

4. Tech Talk: Do Smartphones Make Good Digital Audio Players? | StereoNET International. URL: https://stereonet.com/features/tech-talk-do-smartphones-make-good-digital-audio-players (last accessed: 19.04.2025).

5. Rise of Music Streaming | EBSCO Research Starters. URL: https://www.ebsco.com/research-starters/music/rise-music-streaming (last accessed: 22.04.2025).

6. Streaming Music Industry Challenges. URL: https://www.miquido.com/blog/streaming-music-industry-challenges/ (last accessed: 22.04.2025).

7. Official download of VLC media player, the best Open Source player - VideoLAN. URL: https://www.videolan.org/vlc/ (last accessed: 22.04.2025).

8. Poweramp – Music Player for Android. URL: https://powerampapp.com/ (last accessed: 22.04.2025).

9. AIMP – Безкоштовне завантаження та інсталяція у Windows | Microsoft Store. URL: https://apps.microsoft.com/detail/9pclllh15smt?hl=uk-UA&gl=US (дата звернення: 22.04.2025).

10. JetAudio. URL: https://www.jetaudio.com/download/ (last accessed: 22.04.2025).

11. Spotify - Web Player: Music for everyone/ URL: https://open.spotify.com/ (last accessed: 22.04.2025).

12.What is YouTube Music? - YouTube Music Help. URL: https://support.google.com/youtubemusic/answer/6313529?hl=en (last accessed: 22.04.2025).

13. Native app | Definition. URL: https://uxcam.com/glossary/native-app (last accessed: 23.03.2025).

14.What is cross-platform mobile development? URL: https://www.jetbrains.com/help/kotlin-multiplatform-dev/cross-platform-mobile-development.html (last accessed: 23.03.2025).

15. Hybrid Apps: The benefits, limitations & consequences for your Testing Phases. URL: https://www2.stardust-testing.com/en/blog-en/hybrid-apps (last accessed: 23.03.2025).

16. Java | Oracle. URL: https://www.java.com/en/ (last accessed: 27.04.2025).

17. Kotlin Programming Language. URL: https://kotlinlang.org/ (last accessed: 27.04.2025).

18. Download Android Studio & App Tools - Android Developers. URL: https://developer.android.com/studio (last accessed: 27.04.2025).

19. IntelliJ IDEA – the IDE for Pro Java and Kotlin Development. URL: https://www.jetbrains.com/idea/ (last accessed: 27.04.2025).

20. Kotlin plugin for Eclipse. URL: https://marketplace.eclipse.org/content/kotlin-plugin-eclipse (last accessed: 27.04.2025).

21. Setup VSCode For Kotlin Development. URL: https://in-kotlin.com/ide/vscode/setup-vscode-for-kotlin-development/ (last accessed: 27.04.2025).

22. Android Supported Media Formats. URL: https://www.dre.vanderbilt.edu/~schmidt/android/android-4.0/out/target/common/docs/doc-comment-check/guide/appendix/media-formats.html (last accessed: 14.05.2025).

# ДОДАТКИ

## Додаток А – Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., проф.
О. Н. Романюк _____
"25" 03 2025 р.

**Технічне завдання**

**на бакалаврську кваліфікаційну роботу «Розробка програмного забезпечення аудіоплеєра для смартфонів на платформі Андроїд»**

**за спеціальністю 121 – Інженерія програмного забезпечення**

Керівник бакалаврської кваліфікаційної роботи:

_____ д.т.н., проф. Л.Б. Ліщинська

"24" 03 2025 р.

Виконав:

_____ студент гр.4ПІ-21б Є.О. Гавриш

"24" 03 2025 р.

Вінниця – 2025 року

Вінниця – 2025 року

**1. Найменування та галузь застосування**

Бакалаврська кваліфікаційна робота: «Розробка програмного забезпечення аудіоплеєра для смартфонів на платформі Андроїд».

Галузь застосування – медіа і розваг.

**2. Підстава для розробки.**

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від 20 березня 2025 р. ректора ВНТУ про закріплення тем БКР.

**3. Мета та призначення розробки.**

Метою дослідження є розробка та оптимізація рекомендаційної системи для онлайн кінотеатрів з метою підвищення ефективності та точності рекомендаційного процесу.

Призначення роботи – розробка та реалізація програмного забезпечення рекомендаційної системи для онлайн кінотеатрів.

**4. Вихідні дані для проведення НДР**

Перелік основних літературних джерел, на основі яких буде виконуватись БКР:

1. Kotlin Programming Language. URL: https://kotlinlang.org

2. Download Android Studio & App Tools - Android Developers. URL: https://developer.android.com/studio

**5. Технічні вимоги**

Тип програми – Андроїд-застосунок; вхідні дані: аудіофайли, попередні дії користувача; вихідні дані: відтворений аудіфайл; середовище розробки – Android Studio; мова програмування – Kotlin.

**6. Конструктивні вимоги**

Графічна та текстова документація повинна відповідати діючим стандартам України.

**7. Перелік технічної документації, що пред'являється по закінченню робіт:**

1. Пояснювальна записка до БКР.
2. Технічне завдання.
3. Лістинги програми.

**8. Вимоги до рівня уніфікації та стандартизації**

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

**9. Стадії та етапи розробки:**

| № з\п | Назви етапів бакалаврської кваліфікаційної роботи | Строк виконання етапів роботи |
|---|---|---|
| 1 | Аналіз розвитку систем аудіоплеєра для смартфонів на платформі андроїд | 25.03.25 – 11.04.25 |
| 2 | Розробка інтерфейсу програмного забезпечення | 11.04.25 – 21.04.25 |
| 3 | Розробка алгоритмів роботи програмного забезпечення | 21.04.25 – 28.04.25 |
| 4 | Розробка програмних модулів та методів реалізації системи аудіоплеєра | 28.04.25 – 19.05.25 |
| 5 | Тестування програмного забезпечення | 19.05.25 –26.05.25 |
| 6 | Оформлення матеріалів до захисту БКР | 26.05.25 – 30.05.25 |

**10. Порядок контролю та прийняття**

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

## Додаток Б – Протокол перевірки на плагіат

### ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: _Розробка програмного забезпечення аудіоплеєра для смартфонів на платформі Андроїд_____

Тип роботи: _бакалаврська кваліфікаційна робота_____

(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ _кафедра програмного забезпечення, ФІТКІ_____

(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism __9.5__%

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____

(прізвище, ініціали, посада) (підпис)

_____ _____

(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ _____д.т.н., проф. Л.Б. Ліщинська_____

(підпис) (прізвище, ініціали, посада)

Здобувач _____ _____Гавриш Є.О_____

(підпис) (прізвище, ініціали)

# Додаток В – Лістинг програми

## Music.kt

```
package com.eugenehavrysh.musicPlayer

import android.content.Context
import android.graphics.Bitmap
import android.graphics.Color
import android.media.MediaMetadataRetriever
import androidx.appcompat.app.AlertDialog
import com.google.android.material.color.MaterialColors
import java.io.File
import java.util.concurrent.TimeUnit
import kotlin.system.exitProcess


data class Music(
    val id: String,
    val title: String,
    val album: String,
    val artist: String,
    val duration: Long = 0,
    val path: String,
    val artUri: String
)

class Playlist {
    lateinit var name: String
    lateinit var playlist: ArrayList<Music>
    lateinit var createdBy: String
    lateinit var createdOn: String
}

class MusicPlaylist {
    var ref: ArrayList<Playlist> = ArrayList()
}

fun formatDuration(duration: Long): String {
    val minutes = TimeUnit.MINUTES.convert(duration, TimeUnit.MILLISECONDS)
    val seconds = (TimeUnit.SECONDS.convert(duration, TimeUnit.MILLISECONDS) -
```

```
        minutes * TimeUnit.SECONDS.convert(1, TimeUnit.MINUTES))
    return String.format("%02d:%02d", minutes, seconds)
}

fun getImgArt(path: String): ByteArray? {
    val retriever = MediaMetadataRetriever()
    retriever.setDataSource(path)
    return retriever.embeddedPicture
}

fun setSongPosition(increment: Boolean) {
    if (!PlayerActivity.repeat) {
        if (increment) {
            if (PlayerActivity.musicListPA.size - 1 == PlayerActivity.songPosition)
                PlayerActivity.songPosition = 0
            else ++PlayerActivity.songPosition
        } else {
            if (0 == PlayerActivity.songPosition)
                PlayerActivity.songPosition = PlayerActivity.musicListPA.size - 1
            else --PlayerActivity.songPosition
        }
    }
}

fun exitApplication() {
    if (PlayerActivity.musicService != null) {
        PlayerActivity.musicService!!.audioManager.abandonAudioFocus(PlayerActivity.musicService)
        PlayerActivity.musicService!!.stopForeground(true)
        PlayerActivity.musicService!!.mediaPlayer!!.release()
        PlayerActivity.musicService = null
    }
    exitProcess(1)
}

fun favouriteChecker(id: String): Int {
    PlayerActivity.isFavourite = false
    FavouriteActivity.favouriteSongs.forEachIndexed { index, music ->
        if (id == music.id) {
            PlayerActivity.isFavourite = true
            return index
        }
```

```
    }
    return -1
}

fun checkPlaylist(playlist: ArrayList<Music>): ArrayList<Music> {
    val indicesToRemove = mutableListOf<Int>()

    playlist.forEachIndexed { index, music ->
        if (!File(music.path).exists()) indicesToRemove.add(index)
    }

    indicesToRemove.sortDescending()
    indicesToRemove.forEach { index -> playlist.removeAt(index) }
    return playlist
}

fun setDialogBtnBackground(context: Context, dialog: AlertDialog) {
    //setting button text
    dialog.getButton(android.app.AlertDialog.BUTTON_POSITIVE)?.setTextColor(
        MaterialColors.getColor(context, R.attr.dialogTextColor, Color.WHITE)
    )
    dialog.getButton(android.app.AlertDialog.BUTTON_NEGATIVE)?.setTextColor(
        MaterialColors.getColor(context, R.attr.dialogTextColor, Color.WHITE)
    )

    //setting button background
    dialog.getButton(android.app.AlertDialog.BUTTON_POSITIVE)?.setBackgroundColor(
        MaterialColors.getColor(context, R.attr.dialogBtnBackground, Color.RED)
    )
    dialog.getButton(android.app.AlertDialog.BUTTON_NEGATIVE)?.setBackgroundColor(
        MaterialColors.getColor(context, R.attr.dialogBtnBackground, Color.RED)
    )
}

fun getMainColor(img: Bitmap): Int {
    val newImg = Bitmap.createScaledBitmap(img, 1, 1, true)
    val color = newImg.getPixel(0, 0)
    newImg.recycle()
    return color
}
```

## MainActivity.kt

```
package com.eugenehavrysh.musicPlayer

import android.annotation.SuppressLint
import android.content.Intent
import android.content.pm.PackageManager
import android.content.res.Configuration
import android.net.Uri
import android.os.Build
import android.os.Bundle
import android.provider.MediaStore
import android.view.Menu
import android.view.MenuItem
import android.view.View
import android.widget.LinearLayout
import android.widget.Toast
import androidx.appcompat.app.ActionBarDrawerToggle
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.widget.SearchView
import androidx.core.app.ActivityCompat
import androidx.recyclerview.widget.LinearLayoutManager
import com.google.android.material.dialog.MaterialAlertDialogBuilder
import com.google.gson.GsonBuilder
import com.google.gson.reflect.TypeToken
import com.eugenehavrysh.musicPlayer.databinding.ActivityMainBinding
import java.io.File

class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding
    private lateinit var toggle: ActionBarDrawerToggle
    private lateinit var musicAdapter: MusicAdapter

    companion object{
        lateinit var MusicListMA : ArrayList<Music>
        lateinit var musicListSearch : ArrayList<Music>
        var search: Boolean = false
        var themeIndex: Int = 0
        val currentTheme = arrayOf(R.style.coolPink, R.style.coolBlue, R.style.coolPurple, R.style.coolGreen,
R.style.coolBlack)
```

```
        val currentThemeNav = arrayOf(R.style.coolPinkNav, R.style.coolBlueNav, R.style.coolPurpleNav,
R.style.coolGreenNav,
            R.style.coolBlackNav)
        val currentGradient = arrayOf(R.drawable.gradient_pink, R.drawable.gradient_blue,
R.drawable.gradient_purple, R.drawable.gradient_green,
        R.drawable.gradient_black)
        var sortOrder: Int = 0
        val sortingList = arrayOf(MediaStore.Audio.Media.DATE_ADDED + " DESC",
MediaStore.Audio.Media.TITLE,
        MediaStore.Audio.Media.SIZE + " DESC")
    }
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val themeEditor = getSharedPreferences("THEMES", MODE_PRIVATE)
        themeIndex = themeEditor.getInt("themeIndex", 0)
        setTheme(currentThemeNav[themeIndex])
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        //for nav drawer
        toggle = ActionBarDrawerToggle(this, binding.root,R.string.open, R.string.close)
        binding.root.addDrawerListener(toggle)
        toggle.syncState()
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
        //checking for dark theme
        if(themeIndex == 4 &&  resources.configuration.uiMode and Configuration.UI_MODE_NIGHT_MASK
== Configuration.UI_MODE_NIGHT_NO)
            Toast.makeText(this, "Black Theme Works Best in Dark Mode!!", Toast.LENGTH_LONG).show()

        if(requestRuntimePermission()){
            initializeLayout()
            //for retrieving favourites data using shared preferences
            FavouriteActivity.favouriteSongs = ArrayList()
            val editor = getSharedPreferences("FAVOURITES", MODE_PRIVATE)
            val jsonString = editor.getString("FavouriteSongs", null)
            val typeToken = object : TypeToken<ArrayList<Music>>(){}.type
            if(jsonString != null){
                val data: ArrayList<Music> = GsonBuilder().create().fromJson(jsonString, typeToken)
                FavouriteActivity.favouriteSongs.addAll(data)
            }
            PlaylistActivity.musicPlaylist = MusicPlaylist()
            val jsonStringPlaylist = editor.getString("MusicPlaylist", null)
```

```
            if(jsonStringPlaylist != null){
                val dataPlaylist: MusicPlaylist = GsonBuilder().create().fromJson(jsonStringPlaylist,
MusicPlaylist::class.java)
                PlaylistActivity.musicPlaylist = dataPlaylist
            }
        }

        binding.shuffleBtn.setOnClickListener {
            val intent = Intent(this@MainActivity, PlayerActivity::class.java)
            intent.putExtra("index", 0)
            intent.putExtra("class", "MainActivity")
            startActivity(intent)
        }
        binding.favouriteBtn.setOnClickListener {
            startActivity(Intent(this@MainActivity, FavouriteActivity::class.java))
        }
        binding.playlistBtn.setOnClickListener {
            startActivity(Intent(this@MainActivity, PlaylistActivity::class.java))
        }
        binding.playNextBtn.setOnClickListener {
            startActivity(Intent(this@MainActivity, PlayNext::class.java))
        }
        binding.navView.setNavigationItemSelectedListener{
            when(it.itemId)
            {
//              R.id.navFeedback -> startActivity(Intent(this@MainActivity, FeedbackActivity::class.java))
                R.id.navSettings -> startActivity(Intent(this@MainActivity, SettingsActivity::class.java))
                R.id.navAbout -> startActivity(Intent(this@MainActivity, AboutActivity::class.java))
                R.id.navExit -> {
                    val builder = MaterialAlertDialogBuilder(this)
                    builder.setTitle("Exit")
                        .setMessage("Do you want to close app?")
                        .setPositiveButton("Yes"){ _, _ ->
                            exitApplication()
                        }
                        .setNegativeButton("No"){dialog, _ ->
                            dialog.dismiss()
                        }
                    val customDialog = builder.create()
                    customDialog.show()
```

```
                    setDialogBtnBackground(this, customDialog)
                }
            }
            true
        }
    }
    //For requesting permission
    private fun requestRuntimePermission() :Boolean{
        if(Build.VERSION.SDK_INT < Build.VERSION_CODES.TIRAMISU){
            if(ActivityCompat.checkSelfPermission(this,
android.Manifest.permission.READ_EXTERNAL_STORAGE)
                != PackageManager.PERMISSION_GRANTED) {
                ActivityCompat.requestPermissions(this,
arrayOf(android.Manifest.permission.READ_EXTERNAL_STORAGE), 13)
                return false
            }
        }else{
            //android 13 or Higher permission request
            if(ActivityCompat.checkSelfPermission(this, android.Manifest.permission.READ_MEDIA_AUDIO)
                != PackageManager.PERMISSION_GRANTED) {
                ActivityCompat.requestPermissions(this,
arrayOf(android.Manifest.permission.READ_MEDIA_AUDIO), 13)
                return false
            }
        }
        return true
    }

    override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<out String>, grantResults:
IntArray) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults)
        if(requestCode == 13){
            if(grantResults.isNotEmpty() && grantResults[0] == PackageManager.PERMISSION_GRANTED){
                Toast.makeText(this, "Permission Granted",Toast.LENGTH_SHORT).show()
                initializeLayout()
            }
    //                                                    else    ActivityCompat.requestPermissions(this,
arrayOf(android.Manifest.permission.WRITE_EXTERNAL_STORAGE), 13)
        }
    }
```

```
    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        if(toggle.onOptionsItemSelected(item))
            return true
        return super.onOptionsItemSelected(item)
    }

    @SuppressLint("SetTextI18n")
    private fun initializeLayout(){
        search = false
        val sortEditor = getSharedPreferences("SORTING", MODE_PRIVATE)
        sortOrder = sortEditor.getInt("sortOrder", 0)
        MusicListMA = getAllAudio()
        binding.musicRV.setHasFixedSize(true)
        binding.musicRV.setItemViewCacheSize(13)
        binding.musicRV.layoutManager = LinearLayoutManager(this@MainActivity)
        musicAdapter = MusicAdapter(this@MainActivity, MusicListMA)
        binding.musicRV.adapter = musicAdapter
        binding.totalSongs.text  = "Total Songs : "+musicAdapter.itemCount

        //for refreshing layout on swipe from top
        binding.refreshLayout.setOnRefreshListener {
            MusicListMA = getAllAudio()
            musicAdapter.updateMusicList(MusicListMA)

            binding.refreshLayout.isRefreshing = false
        }
    }

    @SuppressLint("Recycle", "Range")
    private fun getAllAudio(): ArrayList<Music> {
        val tempList = ArrayList<Music>()

        // Filter Only Music or Audio Files
        val selection = MediaStore.Audio.Media.IS_MUSIC + " != 0 AND " +
MediaStore.Audio.Media.MIME_TYPE + " LIKE 'audio/%'"
        val projection = arrayOf(
            MediaStore.Audio.Media._ID,
            MediaStore.Audio.Media.TITLE,
            MediaStore.Audio.Media.ALBUM,
            MediaStore.Audio.Media.ARTIST,
            MediaStore.Audio.Media.DURATION,
```

```
                MediaStore.Audio.Media.DATE_ADDED,
                MediaStore.Audio.Media.DATA,
                MediaStore.Audio.Media.ALBUM_ID
            )
            val cursor = this.contentResolver.query(
                MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, projection, selection, null,
                sortingList[sortOrder], null
            )
            if (cursor != null) {
                if (cursor.moveToFirst()) {
                    do {
                        val titleC = cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.TITLE)) ?:
"Unknown"
                        val idC = cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media._ID)) ?: "Unknown"
                        val albumC = cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.ALBUM)) ?:
"Unknown"
                        val artistC = cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.ARTIST)) ?:
"Unknown"
                        val pathC = cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.DATA))
                        val durationC = cursor.getLong(cursor.getColumnIndex(MediaStore.Audio.Media.DURATION))
                        val albumIdC =
cursor.getLong(cursor.getColumnIndex(MediaStore.Audio.Media.ALBUM_ID)).toString()
                        val uri = Uri.parse("content://media/external/audio/albumart")
                        val artUriC = Uri.withAppendedPath(uri, albumIdC).toString()

                        // Only add the music file if the duration is greater than 0
                        if (durationC > 0) {
                            val music = Music(
                                id = idC,
                                title = titleC,
                                album = albumC,
                                artist = artistC,
                                path = pathC,
                                duration = durationC,
                                artUri = artUriC
                            )

                            if (File(music.path).exists()) tempList.add(music)
                        }
                    } while (cursor.moveToNext())
                }
```

```
            cursor.close()
        }
        return tempList
    }

    override fun onDestroy() {
        super.onDestroy()
        if(!PlayerActivity.isPlaying && PlayerActivity.musicService != null){
            exitApplication()
        }
    }

    override fun onResume() {
        super.onResume()

        //for storing favourites data using shared preferences
        val editor = getSharedPreferences("FAVOURITES", MODE_PRIVATE).edit()
        val jsonString = GsonBuilder().create().toJson(FavouriteActivity.favouriteSongs)
        editor.putString("FavouriteSongs", jsonString)
        val jsonStringPlaylist = GsonBuilder().create().toJson(PlaylistActivity.musicPlaylist)
        editor.putString("MusicPlaylist", jsonStringPlaylist)
        editor.apply()

        //for sorting
        val sortEditor = getSharedPreferences("SORTING", MODE_PRIVATE)
        val sortValue = sortEditor.getInt("sortOrder", 0)
        if(sortOrder != sortValue){
            sortOrder = sortValue
            MusicListMA = getAllAudio()
            musicAdapter.updateMusicList(MusicListMA)
        }
        if(PlayerActivity.musicService != null) binding.nowPlaying.visibility = View.VISIBLE
    }

    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        menuInflater.inflate(R.menu.search_view_menu, menu)
        //for setting gradient

findViewById<LinearLayout>(R.id.linearLayoutNav)?.setBackgroundResource(currentGradient[themeIndex])

        val searchView = menu?.findItem(R.id.searchView)?.actionView as SearchView
```

```
        searchView.setOnQueryTextListener(object : SearchView.OnQueryTextListener{
            override fun onQueryTextSubmit(query: String?): Boolean = true
            override fun onQueryTextChange(newText: String?): Boolean {
                musicListSearch = ArrayList()
                if(newText != null){
                    val userInput = newText.lowercase()
                    for (song in MusicListMA)
                        if(song.title.lowercase().contains(userInput))
                            musicListSearch.add(song)
                    search = true
                    musicAdapter.updateMusicList(searchList = musicListSearch)
                }
                return true
            }
        })
        return super.onCreateOptionsMenu(menu)
    }
}
```

## MusicAdapter.kt

```
package com.eugenehavrysh.musicPlayer

import android.content.Context
import android.content.Intent
import android.graphics.Color
import android.graphics.drawable.ColorDrawable
import android.text.SpannableStringBuilder
import android.text.format.DateUtils
import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.appcompat.app.AlertDialog
import androidx.core.content.ContextCompat
import androidx.core.text.bold
import androidx.recyclerview.widget.RecyclerView
import com.bumptech.glide.Glide
import com.bumptech.glide.request.RequestOptions
import com.google.android.material.dialog.MaterialAlertDialogBuilder
import com.google.android.material.snackbar.Snackbar
import com.eugenehavrysh.musicPlayer.MusicAdapter.MyHolder
import com.eugenehavrysh.musicPlayer.databinding.DetailsViewBinding
import com.eugenehavrysh.musicPlayer.databinding.MoreFeaturesBinding
```

```
import com.eugenehavrysh.musicPlayer.databinding.MusicViewBinding

class MusicAdapter(private val context: Context, private var musicList: ArrayList<Music>, private val playlistDetails: Boolean = false,
private val selectionActivity: Boolean = false)
  : RecyclerView.Adapter<MyHolder>() {

  class MyHolder(binding: MusicViewBinding) : RecyclerView.ViewHolder(binding.root) {
    val title = binding.songNameMV
    val album = binding.songAlbumMV
    val image = binding.imageMV
    val duration = binding.songDuration
    val root = binding.root
  }

  override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyHolder {
    return MyHolder(MusicViewBinding.inflate(LayoutInflater.from(context), parent, false))
  }

  override fun onBindViewHolder(holder: MyHolder, position: Int) {
    holder.title.text = musicList[position].title
    holder.album.text = musicList[position].album
    holder.duration.text = formatDuration(musicList[position].duration)
    Glide.with(context)
      .load(musicList[position].artUri)
      .apply(RequestOptions().placeholder(R.drawable.music_player_icon_slash_screen).centerCrop())
      .into(holder.image)

    //for play next feature
    if(!selectionActivity)
      holder.root.setOnLongClickListener {
        val customDialog = LayoutInflater.from(context).inflate(R.layout.more_features, holder.root, false)
        val bindingMF = MoreFeaturesBinding.bind(customDialog)
        val dialog = MaterialAlertDialogBuilder(context).setView(customDialog)
          .create()
        dialog.show()
        dialog.window?.setBackgroundDrawable(ColorDrawable(0x99000000.toInt()))

        bindingMF.AddToPNBtn.setOnClickListener {
          try {
            if(PlayNext.playNextList.isEmpty()){
```

```
                        PlayNext.playNextList.add(PlayerActivity.musicListPA[PlayerActivity.songPosition])
                        PlayerActivity.songPosition = 0
                    }

                    PlayNext.playNextList.add(musicList[position])
                    PlayerActivity.musicListPA = ArrayList()
                    PlayerActivity.musicListPA.addAll(PlayNext.playNextList)
                }catch (e: Exception){
                    Snackbar.make(context, holder.root,"Play A Song First!!", 3000).show()
                }
                dialog.dismiss()
            }

            bindingMF.infoBtn.setOnClickListener {
                dialog.dismiss()
                val detailsDialog = LayoutInflater.from(context).inflate(R.layout.details_view, bindingMF.root,
false)
                val binder = DetailsViewBinding.bind(detailsDialog)
                binder.detailsTV.setTextColor(Color.WHITE)
                binder.root.setBackgroundColor(Color.TRANSPARENT)
                val dDialog = MaterialAlertDialogBuilder(context)
        //            .setBackground(ColorDrawable(0x99000000.toInt()))
                    .setView(detailsDialog)
                    .setPositiveButton("OK"){self, _ -> self.dismiss()}
                    .setCancelable(false)
                    .create()
                dDialog.show()
                dDialog.getButton(AlertDialog.BUTTON_POSITIVE).setTextColor(Color.RED)
                setDialogBtnBackground(context, dDialog)
                dDialog.window?.setBackgroundDrawable(ColorDrawable(0x99000000.toInt()))
                val str = SpannableStringBuilder().bold { append("DETAILS\n\nName: ") }
                    .append(musicList[position].title)
                    .bold { append("\n\nDuration:
") }.append(DateUtils.formatElapsedTime(musicList[position].duration/1000))
                    .bold { append("\n\nLocation: ") }.append(musicList[position].path)
                binder.detailsTV.text = str
            }

            return@setOnLongClickListener true
        }
```

```
    when{
        playlistDetails ->{
            holder.root.setOnClickListener {
                sendIntent(ref = "PlaylistDetailsAdapter", pos = position)
            }
        }
        selectionActivity ->{
            holder.root.setOnClickListener {
                if(addSong(musicList[position]))
                    holder.root.setBackgroundColor(ContextCompat.getColor(context, R.color.cool_pink))
                else
                    holder.root.setBackgroundColor(ContextCompat.getColor(context, R.color.white))

            }
        }
        else ->{
            holder.root.setOnClickListener {
            when{
                MainActivity.search -> sendIntent(ref = "MusicAdapterSearch", pos = position)
                musicList[position].id == PlayerActivity.nowPlayingId ->
                    sendIntent(ref = "NowPlaying", pos = PlayerActivity.songPosition)
                else->sendIntent(ref="MusicAdapter", pos = position) } }
    }

     }
}

override fun getItemCount(): Int {
    return musicList.size
}

fun updateMusicList(searchList : ArrayList<Music>){
    musicList = ArrayList()
    musicList.addAll(searchList)
    notifyDataSetChanged()
}

private fun sendIntent(ref: String, pos: Int){
    val intent = Intent(context, PlayerActivity::class.java)
    intent.putExtra("index", pos)
    intent.putExtra("class", ref)
```

```
            ContextCompat.startActivity(context, intent, null)
        }

        private fun addSong(song: Music): Boolean{
            PlaylistActivity.musicPlaylist.ref[PlaylistDetails.currentPlaylistPos].playlist.forEachIndexed { index, music
->
                if(song.id == music.id){
                    PlaylistActivity.musicPlaylist.ref[PlaylistDetails.currentPlaylistPos].playlist.removeAt(index)
                    return false
                }
            }
            PlaylistActivity.musicPlaylist.ref[PlaylistDetails.currentPlaylistPos].playlist.add(song)
            return true
        }

        fun refreshPlaylist(){
            musicList = ArrayList()
            musicList = PlaylistActivity.musicPlaylist.ref[PlaylistDetails.currentPlaylistPos].playlist
            notifyDataSetChanged()
        }
    }
```

## PlayerActivity.kt

```
package com.eugenehavrysh.musicPlayer

import android.annotation.SuppressLint
import android.content.ComponentName
import android.content.Intent
import android.content.ServiceConnection
import android.database.Cursor
import android.graphics.BitmapFactory
import android.graphics.drawable.ColorDrawable
import android.graphics.drawable.GradientDrawable
import android.media.AudioManager
import android.media.MediaPlayer
import android.media.audiofx.AudioEffect
import android.media.audiofx.LoudnessEnhancer
import android.net.Uri
import android.os.Bundle
import android.os.IBinder
import android.provider.MediaStore
```

```
import android.view.LayoutInflater
import android.widget.LinearLayout
import android.widget.SeekBar
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.core.content.ContextCompat
import com.bumptech.glide.Glide
import com.bumptech.glide.request.RequestOptions
import com.google.android.material.bottomsheet.BottomSheetDialog
import com.google.android.material.dialog.MaterialAlertDialogBuilder
import com.eugenehavrysh.musicPlayer.databinding.ActivityPlayerBinding
import com.eugenehavrysh.musicPlayer.databinding.AudioBoosterBinding

class PlayerActivity : AppCompatActivity(), ServiceConnection, MediaPlayer.OnCompletionListener {


    companion object {
        lateinit var musicListPA : ArrayList<Music>
        var songPosition: Int = 0
        var isPlaying:Boolean = false
        var musicService: MusicService? = null
        @SuppressLint("StaticFieldLeak")
        lateinit var binding: ActivityPlayerBinding
        var repeat: Boolean = false
        var min15: Boolean = false
        var min30: Boolean = false
        var min60: Boolean = false
        var nowPlayingId: String = ""
        var isFavourite: Boolean = false
        var fIndex: Int = -1
        lateinit var loudnessEnhancer: LoudnessEnhancer
    }

    @SuppressLint("SetTextI18n")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setTheme(MainActivity.currentTheme[MainActivity.themeIndex])
        binding = ActivityPlayerBinding.inflate(layoutInflater)
        setContentView(binding.root)
        if(intent.data?.scheme.contentEquals("content")){
            songPosition = 0
```

```
    val intentService = Intent(this, MusicService::class.java)
    bindService(intentService, this, BIND_AUTO_CREATE)
    startService(intentService)
    musicListPA = ArrayList()
    musicListPA.add(getMusicDetails(intent.data!!))
    Glide.with(this)
        .load(getImgArt(musicListPA[songPosition].path))
        .apply(RequestOptions().placeholder(R.drawable.music_player_icon_slash_screen).centerCrop())
        .into(binding.songImgPA)
    binding.songNamePA.text = musicListPA[songPosition].title
}
else initializeLayout()

//audio booster feature
binding.boosterBtnPA.setOnClickListener {
    val customDialogB = LayoutInflater.from(this).inflate(R.layout.audio_booster, binding.root, false)
    val bindingB = AudioBoosterBinding.bind(customDialogB)
    val dialogB = MaterialAlertDialogBuilder(this).setView(customDialogB)
        .setOnCancelListener { playMusic() }
        .setPositiveButton("OK"){self, _ ->
            loudnessEnhancer.setTargetGain(bindingB.verticalBar.progress * 100)
            playMusic()
            self.dismiss()
        }
        .setBackground(ColorDrawable(0x803700B3.toInt()))
        .create()
    dialogB.show()

    bindingB.verticalBar.progress = loudnessEnhancer.targetGain.toInt()/100
    bindingB.progressText.text = "Audio Boost\n\n${loudnessEnhancer.targetGain.toInt()/10} %"
    bindingB.verticalBar.setOnProgressChangeListener {
        bindingB.progressText.text = "Audio Boost\n\n${it*10} %"
    }
    setDialogBtnBackground(this, dialogB)
}

binding.backBtnPA.setOnClickListener { finish() }
binding.playPauseBtnPA.setOnClickListener{ if(isPlaying) pauseMusic() else playMusic() }
binding.previousBtnPA.setOnClickListener { prevNextSong(increment = false) }
binding.nextBtnPA.setOnClickListener { prevNextSong(increment = true) }
binding.seekBarPA.setOnSeekBarChangeListener(object : SeekBar.OnSeekBarChangeListener{
```

```
                override fun onProgressChanged(seekBar: SeekBar?, progress: Int, fromUser: Boolean) {
                    if(fromUser) {
                        musicService!!.mediaPlayer!!.seekTo(progress)
                        musicService!!.showNotification(if(isPlaying) R.drawable.pause_icon else R.drawable.play_icon)
                    }
                }
                override fun onStartTrackingTouch(seekBar: SeekBar?) = Unit
                override fun onStopTrackingTouch(seekBar: SeekBar?) = Unit
            })
            binding.repeatBtnPA.setOnClickListener {
                if(!repeat){
                    repeat = true
                    binding.repeatBtnPA.setColorFilter(ContextCompat.getColor(this, R.color.purple_500))
                }else{
                    repeat = false
                    binding.repeatBtnPA.setColorFilter(ContextCompat.getColor(this, R.color.cool_pink))
                }
            }
            binding.equalizerBtnPA.setOnClickListener {
            try {
                val eqIntent = Intent(AudioEffect.ACTION_DISPLAY_AUDIO_EFFECT_CONTROL_PANEL)
                eqIntent.putExtra(AudioEffect.EXTRA_AUDIO_SESSION,
musicService!!.mediaPlayer!!.audioSessionId)
                eqIntent.putExtra(AudioEffect.EXTRA_PACKAGE_NAME, baseContext.packageName)
                eqIntent.putExtra(AudioEffect.EXTRA_CONTENT_TYPE, AudioEffect.CONTENT_TYPE_MUSIC)
                startActivityForResult(eqIntent, 13)
            }catch (e: Exception){Toast.makeText(this, "Equalizer is not supported",
Toast.LENGTH_SHORT).show()}
            }
            binding.timerBtnPA.setOnClickListener {
                val timer = min15 || min30 || min60
                if(!timer) showBottomSheetDialog()
                else {
                    val builder = MaterialAlertDialogBuilder(this)
                    builder.setTitle("Stop Timer")
                        .setMessage("Stop Timer?")
                        .setPositiveButton("Yes"){ _, _ ->
                            min15 = false
                            min30 = false
                            min60 = false
                            binding.timerBtnPA.setColorFilter(ContextCompat.getColor(this, R.color.cool_pink))
```

```
                }
                .setNegativeButton("No"){dialog, _ ->
                    dialog.dismiss()
                }
                val customDialog = builder.create()
                customDialog.show()
                setDialogBtnBackground(this, customDialog)
            }
        }
        binding.shareBtnPA.setOnClickListener {
            val shareIntent = Intent()
            shareIntent.action = Intent.ACTION_SEND
            shareIntent.type = "audio/*"
            shareIntent.putExtra(Intent.EXTRA_STREAM, Uri.parse(musicListPA[songPosition].path))
            startActivity(Intent.createChooser(shareIntent, "Sharing Music File!!"))

        }
        binding.favouriteBtnPA.setOnClickListener {
            fIndex = favouriteChecker(musicListPA[songPosition].id)
            if(isFavourite){
                isFavourite = false
                binding.favouriteBtnPA.setImageResource(R.drawable.favourite_empty_icon)
                FavouriteActivity.favouriteSongs.removeAt(fIndex)
            } else{
                isFavourite = true
                binding.favouriteBtnPA.setImageResource(R.drawable.favourite_icon)
                FavouriteActivity.favouriteSongs.add(musicListPA[songPosition])
            }
            FavouriteActivity.favouritesChanged = true
        }
    }
//Important Function
    private fun initializeLayout(){
        songPosition = intent.getIntExtra("index", 0)
        when(intent.getStringExtra("class")){
            "NowPlaying"->{
                setLayout()
                binding.tvSeekBarStart.text = formatDuration(musicService!!.mediaPlayer!!.currentPosition.toLong())
                binding.tvSeekBarEnd.text = formatDuration(musicService!!.mediaPlayer!!.duration.toLong())
                binding.seekBarPA.progress = musicService!!.mediaPlayer!!.currentPosition
                binding.seekBarPA.max = musicService!!.mediaPlayer!!.duration
```

```
                    if(isPlaying) binding.playPauseBtnPA.setIconResource(R.drawable.pause_icon)
                    else binding.playPauseBtnPA.setIconResource(R.drawable.play_icon)
                }
                "MusicAdapterSearch"-> initServiceAndPlaylist(MainActivity.musicListSearch, shuffle = false)
                "MusicAdapter" -> initServiceAndPlaylist(MainActivity.MusicListMA, shuffle = false)
                "FavouriteAdapter"-> initServiceAndPlaylist(FavouriteActivity.favouriteSongs, shuffle = false)
                "MainActivity"-> initServiceAndPlaylist(MainActivity.MusicListMA, shuffle = true)
                "FavouriteShuffle"-> initServiceAndPlaylist(FavouriteActivity.favouriteSongs, shuffle = true)
                "PlaylistDetailsAdapter"->
                    initServiceAndPlaylist(PlaylistActivity.musicPlaylist.ref[PlaylistDetails.currentPlaylistPos].playlist,
shuffle = false)
                "PlaylistDetailsShuffle"->
                    initServiceAndPlaylist(PlaylistActivity.musicPlaylist.ref[PlaylistDetails.currentPlaylistPos].playlist,
shuffle = true)
                "PlayNext"->initServiceAndPlaylist(PlayNext.playNextList, shuffle = false, playNext = true)
            }
        if (musicService!= null && !isPlaying) playMusic()
        }

        private fun setLayout(){
            fIndex = favouriteChecker(musicListPA[songPosition].id)
            Glide.with(applicationContext)
                .load(musicListPA[songPosition].artUri)
                .apply(RequestOptions().placeholder(R.drawable.music_player_icon_slash_screen).centerCrop())
                .into(binding.songImgPA)
            binding.songNamePA.text = musicListPA[songPosition].title
            if(repeat)                  binding.repeatBtnPA.setColorFilter(ContextCompat.getColor(applicationContext,
R.color.purple_500))
            if(min15 || min30 || min60) binding.timerBtnPA.setColorFilter(ContextCompat.getColor(applicationContext,
R.color.purple_500))
            if(isFavourite) binding.favouriteBtnPA.setImageResource(R.drawable.favourite_icon)
            else binding.favouriteBtnPA.setImageResource(R.drawable.favourite_empty_icon)

            val img = getImgArt(musicListPA[songPosition].path)
            val image = if (img != null) {
                BitmapFactory.decodeByteArray(img, 0, img.size)
            } else {
                BitmapFactory.decodeResource(
                    resources,
                    R.drawable.music_player_icon_slash_screen
                )
```

```
            }
            val bgColor = getMainColor(image)
            val gradient = GradientDrawable(GradientDrawable.Orientation.BOTTOM_TOP, intArrayOf(0xFFFFFF,
bgColor))
            binding.root.background = gradient
            window?.statusBarColor = bgColor
        }

        private fun createMediaPlayer(){
            try {
                if (musicService!!.mediaPlayer == null) musicService!!.mediaPlayer = MediaPlayer()
                musicService!!.mediaPlayer!!.reset()
                musicService!!.mediaPlayer!!.setDataSource(musicListPA[songPosition].path)
                musicService!!.mediaPlayer!!.prepare()
                binding.tvSeekBarStart.text = formatDuration(musicService!!.mediaPlayer!!.currentPosition.toLong())
                binding.tvSeekBarEnd.text = formatDuration(musicService!!.mediaPlayer!!.duration.toLong())
                binding.seekBarPA.progress = 0
                binding.seekBarPA.max = musicService!!.mediaPlayer!!.duration
                musicService!!.mediaPlayer!!.setOnCompletionListener(this)
                nowPlayingId = musicListPA[songPosition].id
                playMusic()
                loudnessEnhancer = LoudnessEnhancer(musicService!!.mediaPlayer!!.audioSessionId)
                loudnessEnhancer.enabled = true
            }catch (e: Exception){Toast.makeText(this, e.toString(), Toast.LENGTH_LONG).show()}
        }

        private fun playMusic(){
            isPlaying = true
            musicService!!.mediaPlayer!!.start()
            binding.playPauseBtnPA.setIconResource(R.drawable.pause_icon)
            musicService!!.showNotification(R.drawable.pause_icon)
        }

        private fun pauseMusic(){
            isPlaying = false
            musicService!!.mediaPlayer!!.pause()
            binding.playPauseBtnPA.setIconResource(R.drawable.play_icon)
            musicService!!.showNotification(R.drawable.play_icon)

        }
        private fun prevNextSong(increment: Boolean){
```

```
            if(increment)
            {
                setSongPosition(increment = true)
                setLayout()
                createMediaPlayer()
            }
            else{
                setSongPosition(increment = false)
                setLayout()
                createMediaPlayer()
            }
        }
        override fun onServiceConnected(name: ComponentName?, service: IBinder?) {
            if(musicService == null){
                val binder = service as MusicService.MyBinder
                musicService = binder.currentService()
                musicService!!.audioManager = getSystemService(AUDIO_SERVICE) as AudioManager
                musicService!!.audioManager.requestAudioFocus(musicService,        AudioManager.STREAM_MUSIC,
AudioManager.AUDIOFOCUS_GAIN)
                }
            createMediaPlayer()
            musicService!!.seekBarSetup()

        }

        override fun onServiceDisconnected(name: ComponentName?) {
            musicService = null
        }

        override fun onCompletion(mp: MediaPlayer?) {
            setSongPosition(increment = true)
            createMediaPlayer()
            setLayout()

            //for refreshing now playing image & text on song completion
            NowPlaying.binding.songNameNP.isSelected = true
            Glide.with(applicationContext)
                .load(musicListPA[songPosition].artUri)
                .apply(RequestOptions().placeholder(R.drawable.music_player_icon_slash_screen).centerCrop())
                .into(NowPlaying.binding.songImgNP)
            NowPlaying.binding.songNameNP.text = musicListPA[songPosition].title
```

```
}

@Deprecated("Deprecated in Java")
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if(requestCode == 13 || resultCode == RESULT_OK)
        return
}

private fun showBottomSheetDialog(){
    val dialog = BottomSheetDialog(this@PlayerActivity)
    dialog.setContentView(R.layout.bottom_sheet_dialog)
    dialog.show()
    dialog.findViewById<LinearLayout>(R.id.min_15)?.setOnClickListener {
        Toast.makeText(baseContext,  "Music will stop after 15 minutes", Toast.LENGTH_SHORT).show()
        binding.timerBtnPA.setColorFilter(ContextCompat.getColor(this, R.color.purple_500))
        min15 = true
        Thread{Thread.sleep((15 * 60000).toLong())
        if(min15) exitApplication()}.start()
        dialog.dismiss()
    }
    dialog.findViewById<LinearLayout>(R.id.min_30)?.setOnClickListener {
        Toast.makeText(baseContext,  "Music will stop after 30 minutes", Toast.LENGTH_SHORT).show()
        binding.timerBtnPA.setColorFilter(ContextCompat.getColor(this, R.color.purple_500))
        min30 = true
        Thread{Thread.sleep((30 * 60000).toLong())
            if(min30) exitApplication()}.start()
        dialog.dismiss()
    }
    dialog.findViewById<LinearLayout>(R.id.min_60)?.setOnClickListener {
        Toast.makeText(baseContext,  "Music will stop after 60 minutes", Toast.LENGTH_SHORT).show()
        binding.timerBtnPA.setColorFilter(ContextCompat.getColor(this, R.color.purple_500))
        min60 = true
        Thread{Thread.sleep((60 * 60000).toLong())
            if(min60) exitApplication()}.start()
        dialog.dismiss()
    }
}

private fun getMusicDetails(contentUri: Uri): Music{
    var cursor: Cursor? = null
```

```
            try {
                val projection = arrayOf(MediaStore.Audio.Media.DATA, MediaStore.Audio.Media.DURATION)
                cursor = this.contentResolver.query(contentUri, projection, null, null, null)
                val dataColumn = cursor?.getColumnIndexOrThrow(MediaStore.Audio.Media.DATA)
                val durationColumn = cursor?.getColumnIndexOrThrow(MediaStore.Audio.Media.DURATION)
                cursor!!.moveToFirst()
                val path = dataColumn?.let { cursor.getString(it) }
                val duration = durationColumn?.let { cursor.getLong(it) }!!
                return Music(id = "Unknown", title = path.toString(), album = "Unknown", artist = "Unknown", duration
= duration,
                artUri = "Unknown", path = path.toString())
            }finally {
                cursor?.close()
            }
        }

        override fun onDestroy() {
            super.onDestroy()
            if(musicListPA[songPosition].id == "Unknown" && !isPlaying) exitApplication()
        }
        private fun initServiceAndPlaylist(playlist: ArrayList<Music>, shuffle: Boolean, playNext: Boolean = false){
            val intent = Intent(this, MusicService::class.java)
            bindService(intent, this, BIND_AUTO_CREATE)
            startService(intent)
            musicListPA = ArrayList()
            musicListPA.addAll(playlist)
            if(shuffle) musicListPA.shuffle()
            setLayout()
            if(!playNext) PlayNext.playNextList = ArrayList()
        }
    }
```

## Settings.kt

```
package com.eugenehavrysh.musicPlayer

import android.graphics.Color
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import com.google.android.material.dialog.MaterialAlertDialogBuilder
import com.eugenehavrysh.musicPlayer.databinding.ActivitySettingsBinding
```

```
class SettingsActivity : AppCompatActivity() {

    lateinit var binding: ActivitySettingsBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setTheme(MainActivity.currentThemeNav[MainActivity.themeIndex])
        binding = ActivitySettingsBinding.inflate(layoutInflater)
        setContentView(binding.root)
        supportActionBar?.title = "Settings"
        when(MainActivity.themeIndex){
            0 -> binding.coolPinkTheme.setBackgroundColor(Color.YELLOW)
            1 -> binding.coolBlueTheme.setBackgroundColor(Color.YELLOW)
            2 -> binding.coolPurpleTheme.setBackgroundColor(Color.YELLOW)
            3 -> binding.coolGreenTheme.setBackgroundColor(Color.YELLOW)
            4 -> binding.coolBlackTheme.setBackgroundColor(Color.YELLOW)
        }
        binding.coolPinkTheme.setOnClickListener { saveTheme(0) }
        binding.coolBlueTheme.setOnClickListener { saveTheme(1) }
        binding.coolPurpleTheme.setOnClickListener { saveTheme(2) }
        binding.coolGreenTheme.setOnClickListener { saveTheme(3) }
        binding.coolBlackTheme.setOnClickListener { saveTheme(4) }
        binding.versionName.text = setVersionDetails()
        binding.sortBtn.setOnClickListener {
            val menuList = arrayOf("Recently Added", "Song Title", "File Size")
            var currentSort = MainActivity.sortOrder
            val builder = MaterialAlertDialogBuilder(this)
            builder.setTitle("Sorting")
                .setPositiveButton("OK"){ _, _ ->
                    val editor = getSharedPreferences("SORTING", MODE_PRIVATE).edit()
                    editor.putInt("sortOrder", currentSort)
                    editor.apply()
                }
                .setSingleChoiceItems(menuList, currentSort){ _,which->
                    currentSort = which
                }
            val customDialog = builder.create()
            customDialog.show()

            setDialogBtnBackground(this, customDialog)
        }
```

```
    }

    private fun saveTheme(index: Int){
        if(MainActivity.themeIndex != index){
            val editor = getSharedPreferences("THEMES", MODE_PRIVATE).edit()
            editor.putInt("themeIndex", index)
            editor.apply()
            val builder = MaterialAlertDialogBuilder(this)
            builder.setTitle("Apply Theme")
                .setMessage("Apply chosen theme?")
                .setPositiveButton("Yes"){ _, _ ->
                    exitApplication()
                }
                .setNegativeButton("No"){dialog, _ ->
                    dialog.dismiss()
                }
            val customDialog = builder.create()
            customDialog.show()

            setDialogBtnBackground(this, customDialog)
        }
    }
    private fun setVersionDetails():String{
        return "Version Name: 1.0"
    }
}
```

# Додаток Г – Ілюстративна частина

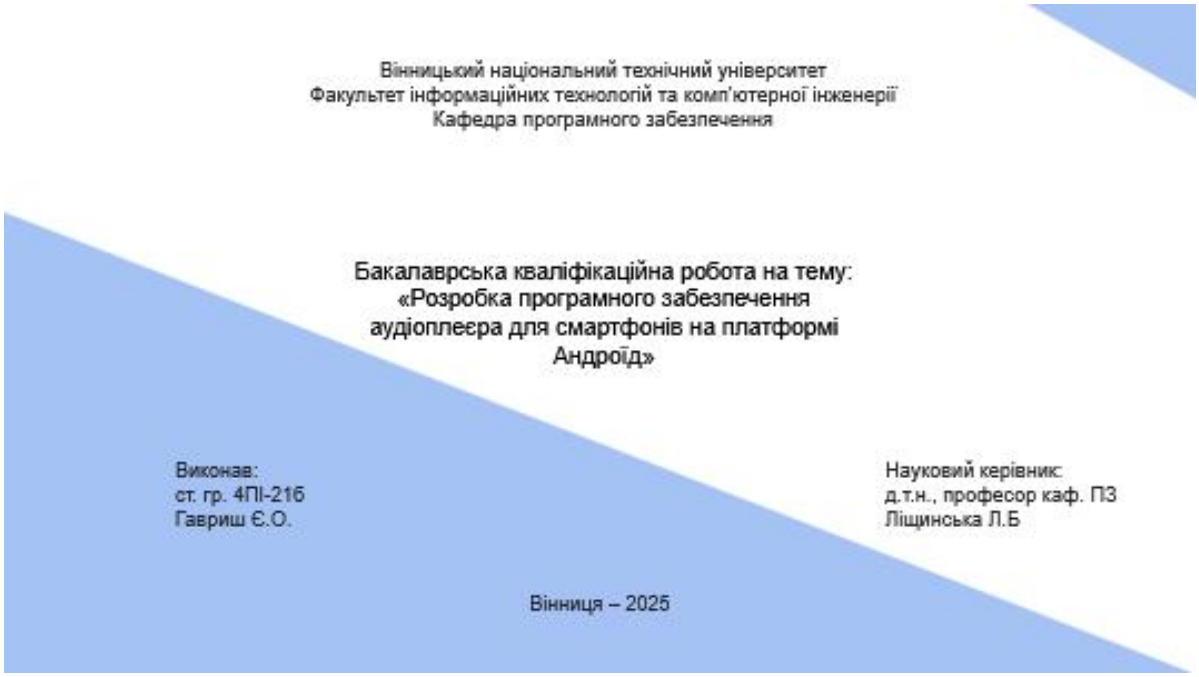


Рисунок Г.1 – Титульний слайд


Рисунок Г.2 – Актуальність дослідження

## Мета, об’єкт та предмет дослідження

Мета дослідження – спрощення управління користувачами прослуховування аудіофайлів.

Об'єкт дослідження – процеси управління аудіофайлами та робота з їх метаданими.

Предмет дослідження – методи та засоби процесів управління аудіофайлами та роботи з їх метаданими.

Рисунок Г.3 – Мета, об’єкт та предмет дослідження

## Цілі дослідження

– Проаналізувати предметну галузь;

– Проаналізувати аналоги;

– Розробити алгоритм взаємодії користувача із застосунком;

– Визначити необхідний для роботи користувачам із застосунку функціонал;

– Розробити інтерфейс на основі даних, отриманих після аналізу існуючих застосунків;

– Розробити функціонал програмного продукту, необхідного для роботи;

– Протестувати розроблений програмний продукт.

Рисунок Г.4 – Цілі дослідження

## Аналоги

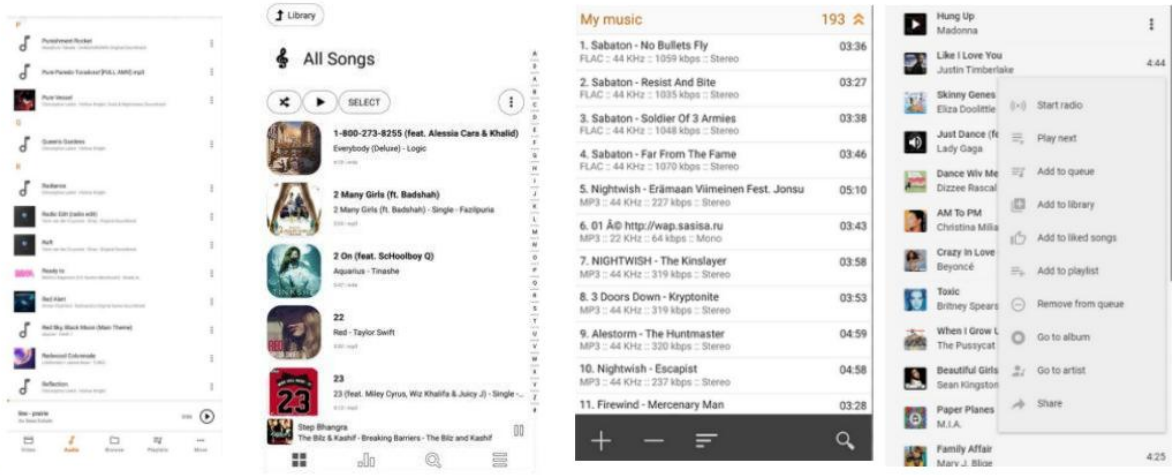


Рисунок Г.5 – Аналоги

## Порівняльний аналіз аналогів

| Аналоги | VLC | Poweramp | AIMP | JetAudio HD | Spotify | YouTube Music | Власна розробка |
|---|---|---|---|---|---|---|---|
| Незалежність від мережі Інтернет | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| Зручний інтерфейс | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| Доступ до великої бібліотеки файлів | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| Еквалайзер для налаштування звучання | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| Таймер сну | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| Налаштованість вигляду інтерфейсу | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| Сумарний коефіцієнт | 3 | 4 | 2 | 4 | 2 | 2 | 5 |

Рисунок Г.6 – Порівняльний аналіз аналогів

## Діаграма взаємодії

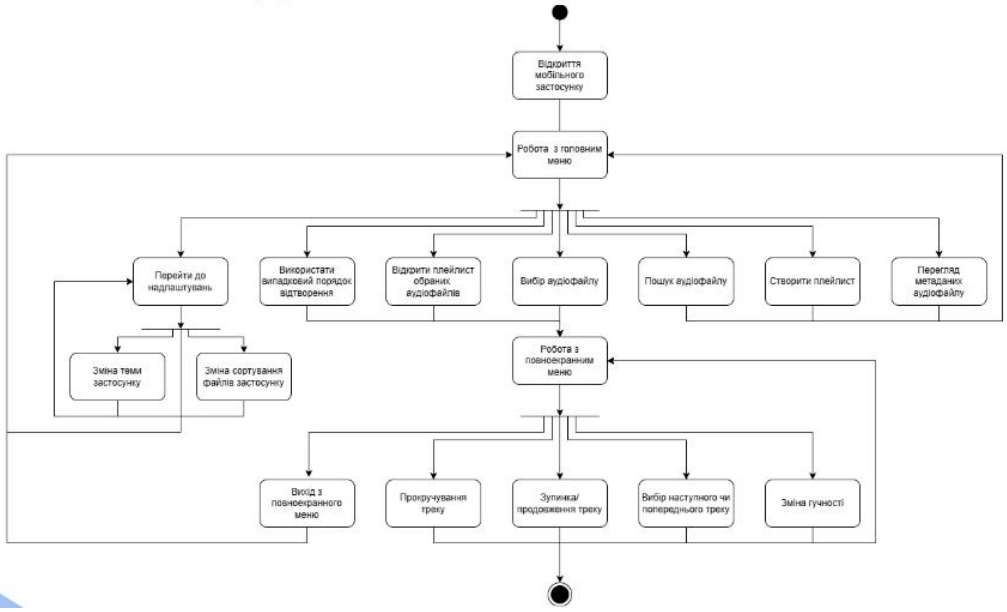


Рисунок Г.7 – Діаграма взаємодії користувача

## Вибір середовища розробки


Рисунок Г.8 – Вибір середовища розробки

## Інтерфейс застосунку

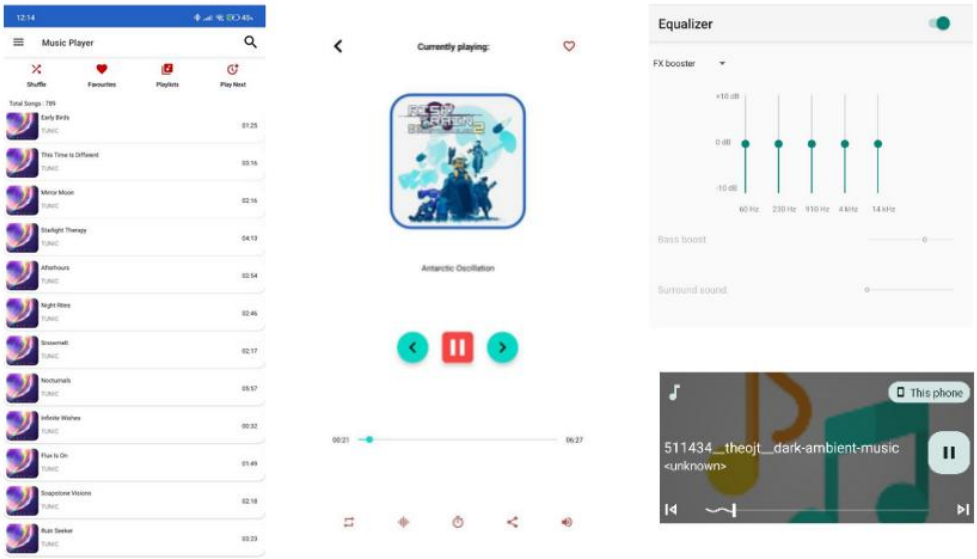


Рисунок Г.9 – Інтерфейс застосунку

## Вимоги до пристрою

Мінімальна конфігурація:

| Тип процесора | Qualcomm Snapdragon 680 |
|---|---|
| Об'єм оперативної пам'яті | 2 ГБ для 64-разрядної системи |
| Місце на жорсткому диску | 1 ГБ |
| Операційна система | Android 11 |

Рекомендована конфігурація:

| Тип процесора | 4× 1.8 GHz Cortex-A55 |
|---|---|
| Об'єм оперативної пам'яті | 4 ГБ для 64-разрядної системи |
| Місце на жорсткому диску | 4 ГБ |
| Операційна система | Android 14, Android 15, Android 16 |

Рисунок Г.10 – Вимоги до пристрою