МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

СТВОРЕННЯ MP3-ПЛЕЄРА ДЛЯ МОБІЛЬНИХ ПРИСТРОЇВ НА БАЗІ ANDROID

Спеціальність 122 «Комп'ютерні науки»

122 – КРБ – 401.22010317

Виконав студент 4-го курсу, групи 401
_____ А. В. Жулай
«20» червня 2024 р.

Керівник: канд. фіз.-мат. наук, доцент
_____І. В. Кулаковська
«20» червня 2024 р.

Миколаїв – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Рівень вищої освіти **бакалавр**
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2024 р.

З А В Д А Н Н Я
на виконання кваліфікаційної роботи

Видано студенту групи 402 факультету комп'ютерних наук Жулаю Анатолію Вікторовичу

1. Тема кваліфікаційної роботи «Створення MP3-плеєра для мобільних пристроїв на базі Android».

Керівник роботи Кулаковська Інесса Василівна, кандидат фізико-математичних наук, доцент кафедри інтелектуальних інформаційних систем факультету комп'ютерних наук.

Затв. наказом Ректора ЧНУ ім. Петра Могили від «28» грудня 2023 р. № 271

2. Строк представлення кваліфікаційної роботи студентом «20» червня 2024 р.

3. Вхідні (початкові) дані до роботи: приклади існуючих застосунків, підходи та методи до створення застосунків для мобільних пристроїв на базі операційної системи Android.

Очікуваний результат: розроблений застосунок для прослуховування музики для мобільних пристроях на базі Android, шляхом використання середовища розробки Android Studio.

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):

- розкрити теоретичні засади створення мобільних застосунків;
- огляд існуючих аналогів;
- обґрунтувати вибір інструментальних засобів розробки застосунку;
- розробити та здійснити програмну реалізацію MP3-плеєру.

5. Перелік графічного матеріалу: –

6. Завдання до спеціальної частини: –

7. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	Алексєєва А. О., доцент кафедри екології	

Керівник роботи кандидат фіз.-мат. наук, доцент. Кулаковська І. В.

(наук. ступінь, вчене звання, прізвище та ініціали)

_____

(підпис)

Завдання прийнято до виконання Жулай А. В.

(прізвище та ініціали)

_____

(підпис)

Дата видачі завдання « 14 » січня 2024

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

Тема: Створення MP3-плеєра для мобільних пристроїв на базі Android

№	Найменування роботи	Початок	Закінчення	Примітки
1	Подання заяви на затвердження теми та керівників КРБ	10.11.2023	15.11.2023	Виконано
2	Отримання завдання на виконання КРБ	10.01.2024	15.01.2024	Виконано
3	Складання календарного плану роботи на весь період виконання КРБ	16.01.2024	30.01.2024	Виконано
4	Отримання завдання на переддипломну практику	15.04.2024	29.04.2024	Виконано
5	Проходження переддипломної практики, збір та аналіз матеріалів до КРБ	29.04.2024	11.05.2024	Виконано
6	Розробка звіту з переддипломної практики	12.05.2024	15.05.2024	Виконано
7	Виконання КРБ: створення MP3-плеєра для мобільних пристроїв на базі Android	13.05.2024	22.06.2024	Виконано
8	Перший попередній захист КРБ на засіданні комісії кафедри	27.05.2024	27.05.2024	Виконано
9	Доробка та остаточне оформлення КРБ	28.05.2024	09.06.2024	Виконано
10	Другий попередній захист КРБ на засіданні комісії кафедри	10.06.2024	10.06.2024	Виконано
11	Подання КРБ рецензенту	13.06.2024	13.06.2024	Виконано
11	Подання КРБ, її електронної копії та інших документів (відгуку, рецензії) до захисту	17.06.2024	21.06.2024	Виконано
12	Захист БКР перед екзаменаційною комісією (ЕК)	24.06.2024	28.06.2024	

Розробив студент Жулай А. В. _____

(прізвище, ім'я, по батькові студента) *(підпис)*

Керівник роботи канд. фіз.-мат. наук, доцент. Кулаковська І. В. _____

(*посада, прізвище, ім'я, по батькові)* *(підпис)*

« 29 » січня 2024

АНОТАЦІЯ

кваліфікаційної роботи студента 401 групи ЧНУ ім. Петра Могили

Жулая Анатолія Вікторовича

Тема: «Створення mp3-плеєра для мобільних пристроїв на базі Android»

Створення якісного MP3-плеєра для мобільних пристроїв на базі Android є актуальним завданням через високу популярність цієї операційної системи та постійний попит на функціональні та зручні застосунки. Крім того, розвиток технологій та покращення апаратного забезпечення мобільних пристроїв відкривають нові можливості для розробки застосунків з розширеними можливостями і покращеною продуктивністю.

Об'єктом роботи є процеси розробки мобільних мультимедійних застосунків.

Предметом роботи є методи та технології розробки MP3-плеєра для мобільних пристроїв на базі Android, а також функціональні та користувацькі аспекти такого програмного забезпечення.

Метою кваліфікаційної роботи є розробка та реалізація MP3-плеєра для мобільних пристроїв на базі операційної системи Android, який буде мати широкий набір функцій, зручний інтерфейс та високу продуктивність.

Робота складається з фахового розділу і спеціальної частини з охорони праці. Пояснювальна записка складається зі вступу, трьох розділів та висновків.

Кваліфікаційна робота магістра містить 72 сторінки, 18 рисунки, 1 таблицю, 25 літературних джерел.

Ключові слова: Android, MP3-плеєр, Kotlin, Android Studio, мобільний застосунок.

ABSTRACT

to the bachelor's qualification work

by the student of the group 401 of Petro Mohyla Black Sea National University

Zhulai Anatoly

«Creation of an mp3 player for mobile devices based on Android»

Creating a high-quality MP3 player for mobile devices based on Android is a relevant task due to the high popularity of this operating system and the constant demand for functional and user-friendly applications. Additionally, the development of technologies and the improvement of mobile device hardware open new opportunities for the development of applications with extended capabilities and improved performance.

The object of this work is the processes of developing mobile multimedia applications.

The subject of this work is the methods and technologies for developing an MP3 player for mobile devices based on Android, as well as the functional and user aspects of such software.

The aim of the qualification work is to develop and implement an MP3 player for mobile devices based on the Android operating system, which will have a wide range of features, a convenient interface, and high performance.

The work consists of a professional section and a special part on occupational safety. The explanatory note consists of an introduction, three chapters, and conclusions.

The Bachelor's qualification work contains 72 pages, 18 figures, 1 table, and 25 literary sources.

Keywords: Android, MP3 player, Kotlin, Android Studio, mobile application.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ОС – операційна система

ПЗ – програмне забезпечення

API – Application Programming Interface

APK – Android Package

CSS – Cascading Style Sheets

HTML – HyperText Markup Language

IDE – Integrated Development Environment

JVM – Java Virtual Machine

MP3 – MPEG-1 Audio Layer III

MVP – Minimum Viable Product

QA – Quality Assurance

SDK – Software Development Kit

SDLC – Software Development Life Cycle

UI – User Interface

URI – Uniform Resource Identifier

UX – User Experience

XML – eXtensible Markup Language

ВСТУП

Розробка MP3-плеєра для мобільних пристроїв на базі Android є важливим кроком у створенні зручних та функціональних медіа-застосунків. Сучасний світ, з його швидким розвитком технологій, вимагає високоякісних рішень для відтворення мультимедійного контенту. Мобільні застосунки для відтворення музики стали невід’ємною частиною повсякденного життя, надаючи користувачам можливість слухати улюблену музику будь-де і будь-коли. Зважаючи на це, створення надійного та ефективного MP3-плеєра є актуальною задачею.

Об'єктом роботи є процеси розробки мобільних мультимедійних застосунків. Предметом роботи є архітектури програмного забезпечення та інструменти, що використовуються для створення MP3-плеєрів на платформі Android. Метою роботи є розробка функціонального MP3-плеєра з використанням сучасних технологій та інструментів.

Для досягнення цієї мети було виконано наступні завдання: проведено аналіз сучасних підходів до створення медіа-застосунків, обрано оптимальні архітектури та інструменти для розробки, розроблено та впроваджено основні функції MP3-плеєра, включаючи відтворення музики та інтеграцію системи сповіщень.

У першому розділі роботи розглядається сучасний стан задачі створення мобільних мультимедійних застосунків, аналізуються наявні рішення та публікації, визначаються задачі, які необхідно виконати для успішної реалізації проекту. Другий розділ містить огляд існуючих технологій та алгоритмів, проведено аналіз архітектур програмного забезпечення для поставленої задачі. У третьому розділі описується процес розробки MP3-плеєра, включаючи вибір технологій, архітектури застосунку, реалізацію основних функцій та тестування.

1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Опис предметної сфери

Мобільні пристрої, зокрема смартфони та планшети на базі операційної системи Android, є невід'ємною частиною сучасного життя. Вони використовуються для виконання широкого спектру задач, включаючи комунікацію, навігацію, доступ до інформації, розваги та багато іншого. Для більшого розуміння теми кваліфікаційної роботи варто навести деякі відомості про дану ОС.

Android – це операційна система, яка розроблена компанією Google. Вона використовується на різних пристроях, таких як смартфони, планшети, телевізори і навіть ноутбуки.

Головним призначенням Android є забезпечення функціональності, підтримки застосунків та взаємодії з користувачем на мобільних пристроях. Він надає доступ до різноманітних застосунків з Google Play Store, де користувачі можуть завантажувати і встановлювати різні програми та ігри.

Операційна система Android також забезпечує можливості для налаштування пристрою, зручність використання інтерфейсу, можливість синхронізації з обліковим записом Google та багато іншого.

Відкритий характер Android дозволяє розробникам створювати власні застосунки та модифікації, що сприяє широкому розповсюдженню цієї операційної системи і розвитку мобільної індустрії загалом (рис. 1.1).

Рисунок 1.1 – T-Mobile G1 – перший мобільний пристрій на базі Android

Перший прототип операційної системи Android був представлений у 2007 році. У 2008 році, компанія Google анонсувала випуск першого смартфона на базі Android - HTC Dream (T-Mobile G1). Це був перший пристрій, що працював на операційній системі Android та відкривав нову еру для мобільних телефонів.

З того часу Android швидко набув популярності, завоювавши значну частку ринку мобільних пристроїв. Зараз Android є найпопулярнішою операційною системою для мобільних пристроїв у світі.

Варто зазначити плюси та мінуси ОС Андроїд, серед плюсів:

– великий вибір пристроїв – на ринку представлено багато різних моделей смартфонів і планшетів;

– доступні ціни – багато пристроїв Android мають конкурентоспроможну ціну, що робить їх доступними для широкого кола споживачів;

– можливість налаштування – Android дозволяє користувачам налаштовувати свій девайс за власними потребами, змінювати розміщення

елементів, встановлювати різні застосунки тощо;

– велика кількість застосунків – Google Play магазин надає доступ до величезної кількості застосунків для всіх можливих потреб користувача;

– замінний рівень функціоналу – можливість встановлювати альтернативні запущені екрани, клавіатури, месенджери та інше.

Мінуси Android:

– різні версії інтерфейсу – через велику кількість виробників і моделей, деякі користувачі можуть мати різну версію операційної системи, що може призводити до несумісності та проблем з оновленнями;

– вразливість до вірусів – більша відкритість системи може створювати більший ризик для безпеки даних та можливості отримання вірусних програм;

– перевантажений інтерфейс – деякі виробники накладають свої шари інтерфейсу, що може змінювати вигляд та функціонал пристрою;

– оновлення – деякі виробники можуть бути повільними в оновленні операційної системи, що може призводити до недоступності нових функцій та безпекових оновлень для деяких моделей;

– залежність від Google – багато сервісів Android пов'язані з Google, що може бути обмеженням для користувачів, які бажають використовувати інші екосистеми.

Зважаючи на наведені положення та власний досвід використання мобільних пристроїв, саме Android було обрано в якості цільової ОС застосунка, який буде створено.

Також варто описати функціонал створюваного застосунка, а саме відтворення музики.

Кожен погодиться, що потяг до прослуховування музики завжди був невід’ємним супутником людини. Відповідно, чимало зусиль було вкладено у реалізацію все більш доступних засобів для відтворення музичних композицій. Одними з перших винаходів, які цьому сприяли були фонограф та грамофон, винайдені ще в 19 сторіччі силами американських науковців.

З тих пір минуло чимало часу, та місце серед найпоширеніших засобів прослуховування музики зараз почали займати застосунки для відтворення музики на мобільних пристроях. З ними доводилось зтикатись чи не кожному користувачу смартфону, але така популярність, нажаль, все ще не гарантує те, що користувач отримає задовільний сервіс. Це в основному трапляється через декілька аспектів:

– деякі пристрої мають незручний, погано функціонуючий вбудований MP3 плеєр, або ж взагалі його не мають;

– застосунки, які наявні у Play Market з великою вірогідністю будуть мати настирливу рекламу, або платний контент;

– крім того, застосунки з Play Market нерідко можуть потребувати чималих ресурсів системи та працювати із несправностями на слабких в плані обчислювальної потужності пристроях.

Зазначені фактори ведуть до очевидного висновку: ідея створення простого та зручного в використанні, безкоштовного застосунку для прослуховування музики без реклами та з підтримкою на відносно слабких пристроях може мати досить немалий успіх в плані допомоги тисячам користувачів смартфонів та здобуття цінного досвіду та проекту у власне портфоліо. В наступному розділі проведено огляд та аналіз наявних аналогів та публікацій для формування орієнтирів для власного застосунка.

1.2 Огляд та аналіз наявних аналогів

Першим для аналізу було обрано застосунок «Music Player – Музичний плеєр» («Audify»). Застосунок має середній відгук користувачів 4.7 зірок та більше 50 мільйонів завантажень в Play Market.

При відкритті застосунку на час завантаження демонструється сплешскрін (екран з логотипом), після цього відкривається головна активність (рис. 1.2). На ній присутні: логотип застосунку; іконка пошуку та меню; вкладки «Пісні», «Списки відтворення», «Папки», «Альбоми», «Художники», «Жанри», «Заспокійливі звуки» в верхній частині екрану; фрагмент активної вкладки в основній частині екрану; панель відтворюваної композиції та меню навігації в нижній частині екрану.

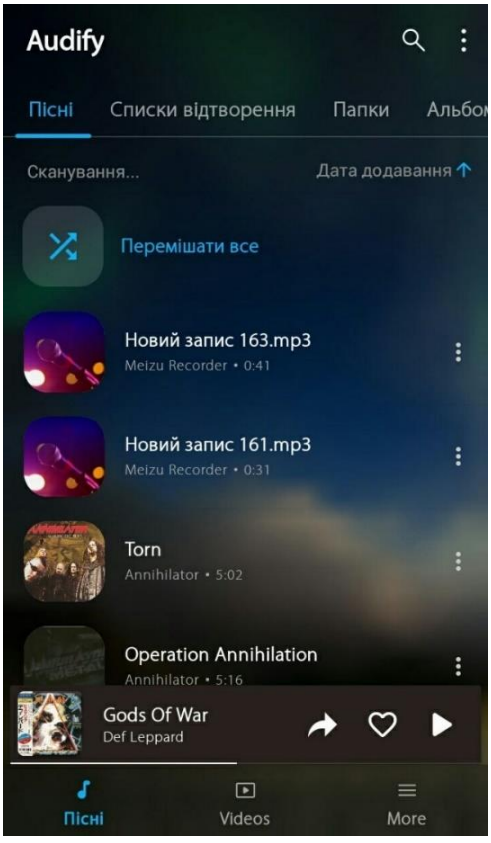

Рисунок 1.2 – Головна активність застосунку «Audify»

Варто описати вміст вкладок. «Пісні» містить список всіх виявлених в результаті сканування пісень, який відображає окрему композицію та містить її

назву, виконавця, тривалість, зображення альбому та іконку опцій; кнопку вибору порядку сортування елементів; кнопку початку відтворення композицій у випадковому порядку.

Вкладка «Списки відтворення» містить перелік плейлистів, серед яких вже наявні окремі списки для улюблених пісень, пісень з текстом, останні відтворені, найчастіше відтворювані та нещодавно доданих. Користувач має можливість створювати власні списки відтворення та упорядковувати їх.

Вкладка «Папки» містить провідник для знаходження музики як в локальних директоріях, так і хмарному середовищі.

Вкладка «Альбоми» містить список альбомів виявлених при скануванні, який містить елементи з зображенням альбому, його назвою та кількістю наявних композицій.

Вкладка «Художники» має вміст подібний до попередньо описаної, але відображає наявних виконавців.

В двох останніх вкладках відображені жанри та списки із заспокійливими звуками, які можна завантажити з мережі. Загалом застосунок справив гарне враження, він містить багато функціоналу та відносно не багато реклами. Єдиним помітним недоліком є проблеми з локалізацією (перекладом).

Даний застосунок також пропонує засіб перегляду відео файлів наявних в локальному сховищі пристрою. Користувач має можливість продивитись список всіх виявлених файлів та переглядати кожен з них. Для цього застосунок надає зручний інтерфейс, який дозволяє не тільки дивитись відео, а й прослуховувати відокремлену аудіо доріжку з нього. Також є можливість додавати відео в список улюблених, які можна знайти окремим списком.

Також застосунок надає такі функції як еквалайзер, таймер сну, ідентифікація музики зафіксованої мікрофоном пристрою, обмін файлами з іншими пристроями.

Наступним застосунком для аналізу став «Music Player & MP3 - DDMusic». Він має середній відгук користувачів 4.8 зірок та, як і попередній, більше 50 мільйонів завантажень.

Як і в попередньому плеєрі, при відкритті «DDMusic» користувач бачить сплешскрін, на цей раз він містить анімований індикатор прогресу завантаження, після завершення якого відкривається основна активність (рис 1.3).

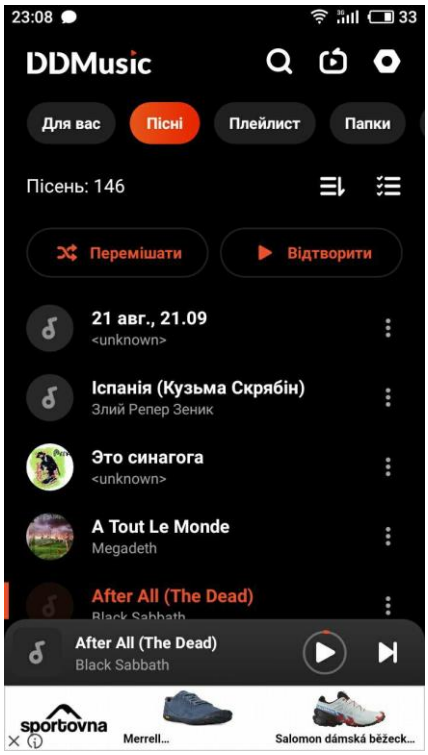

Рисунок 1.3 – Головна активність застосунку «DDMusic»

Головна активність містить інтерфейс схожий за функціоналом з тим, що мав застосунок «Audify». Даний аналог відрізняється більш привабливим дизайном, але містить більше реклами: вона відображається не лише окремим вікнами, які відкриваються в процесі користування, а й вбудованими фрагментами, один з таких можна побачити на рисунку 1.3 знизу.

Даний застосунок, як і попередній здатен відтворювати не лише аудіо, а й відео файли. Якщо в «Audify» для навігації до модуля з відтворення відео необхідно було натиснути відповідний пункт на нижньому меню, то в «DDMusic» для цього

необхідно натиснути іконку в верхній частині екрану. Це дозволило звільнити місце в нижній частині екрану та додати фрагмент з рекламою.

Користуючись застосунком «DDMusic» можна додавати віджети на головний екран пристрою. Користувачу наданий вибір серед різних варіантів панелі керування програванням музики, в якій реалізовано базовий функціонал, як-от кнопки припинення та продовження відтворення композиції, переходу до попереднього та наступного треків.

Аналог під назвою «Poweramp Music Player (Trial)» відрізняється тим, що не містить рекламу, але є безкоштовним лише не невеликий термін. Незважаючи на це застосунок за невідомих причин продовжує функціонувати навіть після завершення періоду. Він має середню оцінку 4.5 з 5 та більше 50 мільйонів завантажень з Play Market.

Основним елементом навігації в застосунку є панель, яка розташована в нижній частині екрану та має чотири розділи. Перший розділ містить вертикальний список з розділів плеєру, серед них: «Всі пісні», «Теки», «Ієрархія тек», «Альбоми», «Виконавці», «Виконавці альбому», «Жанри», «Роки», «Композитори», «Списки відтворення», «Потоки», «Черга», «Часто відтворювані», «Високо оцінені», «Низько оцінені», «Нещодавно додані» , «Довгий». Другий розділ містить багатофункціональний інтерфейс для керування налаштуваннями звуку, серед них: десяти канальний еквалайзер, опції регулювання низьких та високих частот, темпу, балансу, гучності. Третій розділ містить інтерфейс для пошуку треків, альбомів, виконавців за назвою. Четвертий демонструє меню з пунктами «Налаштування», «Список змін», «Підказки».

Серед переваг даного аналогу:

– дуже привабливий інтерфейс;

– потужні можливості регулювання налаштувань звуку;

– наявність великої кількості розділів, які не зустрічались в інших аналогах, на приклад розділ «Роки» зі списком композицій розподілених за датою релізу;

– відсутність реклами (рис 1.4).

Помітних недоліків знайдено не було.

Рисунок 1.4 – Інтерфейс застосунку «Poweramp»

Ще одним прикладом реалізації музичного плеєру вартим уваги є застосунок «Spotify». Spotify – це шведський стрімінговий музичний сервіс, який запустився ще у 2008 році і з часом суттєво змінив правила гри в музичній індустрії.

Розробники «Spotify» дали користувачам таку очевидну сьогодні, але дуже неочевидну 12 років тому можливість: за відносно недорогу підписку легально слухати музику онлайн. Завантажувати файли на свій пристрій стало не потрібно - так само, як купувати окремий альбом чи пісню.

Коли Spotify тільки виходив на ринок, засновник Apple Стів Джобс пророкував сервісу та закладеній у ньому бізнес-моделі провал – кому, мовляв, потрібна музика в оренду. Як і чимало інших критиків ідеї, Джобс сильно помилився.

Сьогодні аналогічних сервісів – чимало. У 2015 році Apple запустила власний – Apple Music – і тепер намагається надолужити втрачений час. [1]

Spotify — це цифровий сервіс музики, подкастів і відео, що відкриває доступ мільйонів пісень та іншого контенту від авторів з усього світу.

Основні функції, як-от прослуховування музики, абсолютно безкоштовні, але користувач також може перейти на Spotify Premium.

Серед можливостей, доступних користувачам як безкоштовної версії підписки, так і Premium отримання рекомендацій на основі смаків, створення колекції музики та подкастів та багато чого іншого.

Сервіс Spotify доступний на низці пристроїв, зокрема на комп'ютерах, телефонах, планшетах, колонках, телевізорах і автомобілях. Користувач може з легкістю переключатися між пристроями за допомогою функції «Spotify Connect». [2]

Мобільний застосунок «Spotify» має середній відгук 4.1 з 5 та більше одного мільярда завантажень. Перший раз відкриваючи «Spotify» користувач має увійти або зареєструватись, для цього користувачу надається можливість авторизуватись ввівши адрес електронної пошти та пароль, або ввійти за допомогою Google чи Facebook. Авторизувавшись користувач потрапляє до основної активності, яка містить розділи «Головна», «Пошук», «Ваша бібліотека», «Premium».

Головною відмінністю застосунка «Spotify» від попередньо розглянутих аналогів, є те, що користуючись ним можна прослуховувати композиції з хмарного середовища без необхідності завантажувати їх на пристрій. Користуючись пошуком можна швидко отримувати доступ до мільярдів композицій та подкастів. В розділі «Для вас» можна побачити популярних виконавців, радіо та альбоми, запропоновані алгоритмами застосунка добірки музики.

Користуючись «Spotify» можна зручно поширювати композиції в соцмережах, це є надзвичайно привабливим для користувачів смартфонів, оскільки дозволяє поєднати прослуховування музики з соціальною активністю. Завдяки цьому мільйони людей мають змогу доступно ділитись своїми музичними

вподобаннями, брати участь у різноманітних заходах, до прикладу, має популярність звичай ділитись кожен день піснею, яка відповідає певній темі впродовж одного місяця року (рис. 1.5).

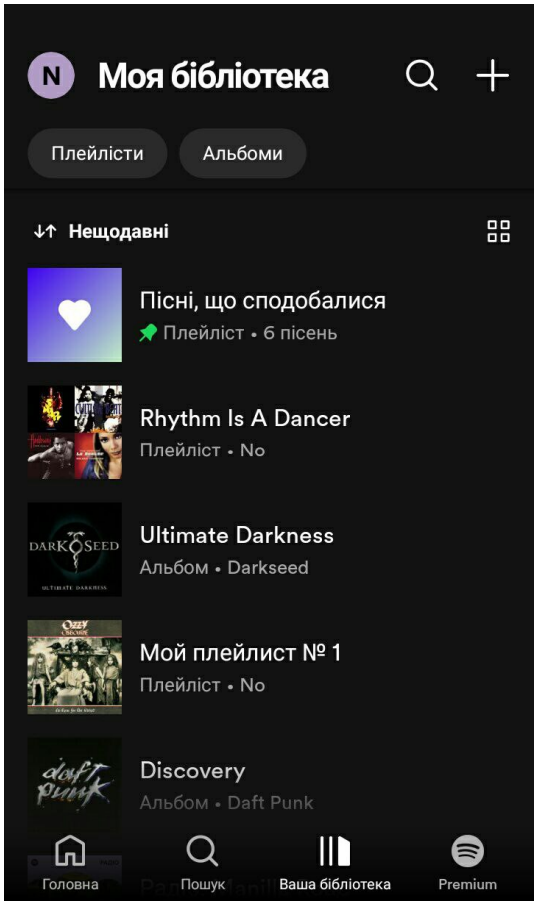

Рисунок 1.5 – Інтерфейс застосунку «Spotify»

До недоліків мобільного застосунку «Spotify» можна віднести наявність аудіо реклами та неможливість прослуховувати треки з локального сховища.

Розглянуті аналоги є гарними зразками для втілення у власному проекті.

1.3 Постановка задачі

Метою даної роботи є поглиблення практичних навичок у створенні програмного забезпечнення для мобільних пристоїв на базі ОС Android. А тому варто поглибитись в тему життєвого циклу ПЗ та методології його розробки.

Розробка якісного продукту починається з визначення його життєвого циклу. Це чіткий план дій, що дозволяє зрозуміти, що має вийти в розробників, як досягти результату та які методи для цього використати. Методологія розробки програмного забезпечення — це перевірені способи та практики, що дозволяють створити діджитал-продукт правильно та якісно.

Кожен проєкт має власний життєвий цикл. Це етапи, якими проходить вся розробка. Починається він зі створення ідеї та прийняття рішень. Студент вирішив зробити сайт або застоснок — у цей момент життєвий цикл уже розпочато.

Далі виконується підготовка та аналіз, пошук концепції та шляху створення продукту. Коли стає зрозумілим, який резльтат бажано отримати, потрібно вирішити, як цього досягти. На етапі аналітики ідея перетворюється на план дій, підбирається стек технологій, обираються зокрема й методології розробки програмного забезпечення.

Це підхід, за яким команда розроблятиме продукт. І від вибору цього підходу залежить в тому числі й якість кінцевого продукту. Адже вибір моделі розробки ПЗ дозволяє визначити порядок виконання та реалізації завдань, розробити систему контролю та оцінки розробки, сформувати терміни створення продукту, визначити вартість. Вибір методики дозволяє досягти стабільності під час розробки, а це одне з основних завдань.

Сам процес розробки програмного забезпечення складається з приблизно однакових етапів. [3]

Життєвий цикл розробки ПЗ за Agile-методологією (Agile SDLC) – це процес розробки ПЗ, який враховує зміну вимог, ринкові умови та потреби клієнтів. Він ґрунтується на співпраці, командній роботі та постійному вдосконаленні. У

випадку даної роботи, а саме створення застосунку для відтворення музики, будь яка командна робота відсутня, але базові концепції даної методології можна взяти за приклад організації процесу виконання робіт.

Процес розробки за методологією Agile включає кілька ключових етапів: планування, проєктування, розробку, тестування, розгортання та обслуговування (рис. 1.6).

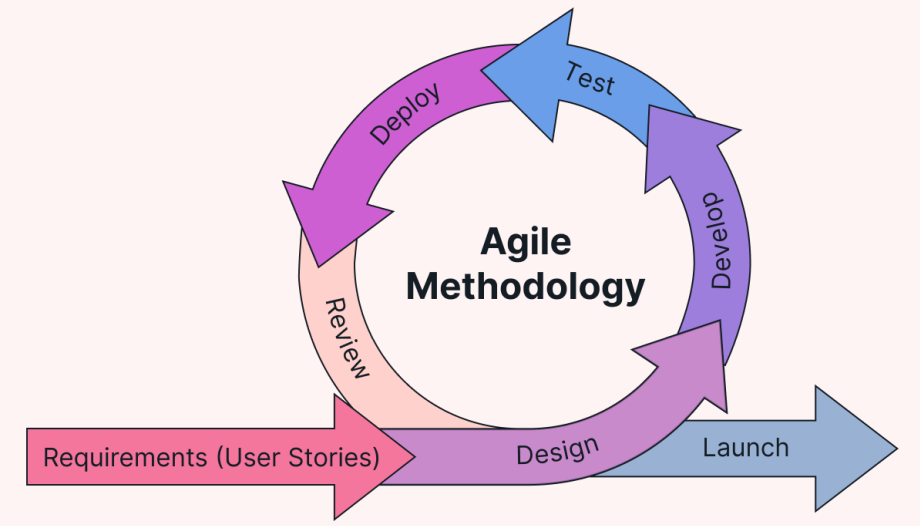

Рисунок 1.6 – Графічне представлення циклу розробки ПЗ
за методологією Agile

Варто розглянути докладніше кожний етап та сформувати план дій для кожного етапу.

1.3.1 Етап планування

Планування є початковою стадією життєвого циклу розробки ПЗ. Проєктна команда працює зі стейкхолдерами, щоб з’ясувати чи продукт є життєздатним і цінним та чи можливо його реалізувати.

Стейкхолдер проекту (англ. stakeholder) - це зацікавлена в проекті особа або сторона, яка має до нього певне відношення. [5]

Найкращим результатом буде вироблена стратегія щодо продукту, яка

визначає бачення, цілі, обсяг та мінімально життєздатний продукт (MVP – minimum viable product), а також містить задокументовані вимоги та беклог продукту (перелік функцій, які той має містити). Ключовими людьми тут є менеджери та власники продукту та бізнес-аналітики. Відтак команда вирішує, які технології та ресурси використовувати для створення продукту. За це відповідають деліверi-менеджер і архітектор рішень.

У випадку реалізації даного проекту всі ролі уособлює здобувач освітнього рівня бакалавра, весь етап планування втілюється у аналізі існуючих аналогів та доступної інформації в мережі інтернет.

1.3.2 Етап проєктування

На цьому етапі, проєктна команда починає працювати над архітектурою та UI/UX продукту. Дизайнери створюють перші каркаси, макети та прототипи, щоб допомогти стейкхолдерам візуалізувати кінцевий продукт і перевірити користувацький досвід. Вкрай важливо запропонувати щось матеріальне якомога швидше, щоб прискорити зворотний зв’язок від стейкхолдерів, а в ідеальному світі ще й кінцевих користувачів.

Для даного проекту основними вимогами до продукту можна назвати наявність активності зі списком доступних на пристрої композицій, можливість відтворювати кожну з них та контролювати цей процес користуючись інтерфейсом застосунка.

1.3.3 Етап розробки

Проєктування завершено. Розпочинається стадія розробки. Команда починає будувати продукт, застосовуючи ітеративний та інкрементний підхід, або інакше – спринти. Вони поділяють беклог продукту на менші, здійсненні завдання, які називаються історіями користувачів, а потім визначають їхню пріоритетність, оцінюють і працюють над ними у порядку, встановленому беклогом продукту. Зазвичай, команда складається з власників продукту, розробників, бізнес-

аналітиків та QA-інженерів, а в даному випадку, з однієї особи.

1.3.4 Етап тестування

Після етапу розробки треба переконатися, що все працює належним чином і відповідає очікуванням стейкхолдерів і кінцевих користувачів. Ключовими людьми на цій стадії є QA-інженери, які використовують різні методології тестування, як-от модульне, інтеграційне, системне та приймальне тестування, щоб переконатися, що продукт є функціональним, надійним і зручним для користувача.

Для тестування проекту кваліфікаційної роботи буде використано мануальне тестування функціоналу та верстки.

1.3.5 Етап розгортання

Наступним етапом у цьому процесі є розгортання. Щоб пересвідчитися, що продукт працює належним чином, команда розгортає його у виробничому середовищі та проводить останній раунд тестування та перевірки. DevOps-інженер налаштовує середовища та конвеєри та керує процесом розгортання.

1.3.5 Етап обслуговування

Завершальним етапом є обслуговування, коли команда розробки постійно підтримує продукт. Вони відстежують його продуктивність, усувають будь-які проблеми чи помилки, що виникають, а також надають оновлення та покращення для поліпшення функціональності продукту та взаємодії з користувачем. [4]

Отже, дотримуючись цієї методології, завдяки її гнучкості та швидкому зворотному зв’язку, проєктні команди можуть ефективніше розробляти високоякісне, зручне програмне забезпечення, такий підхід стане корисним для організації процесу розробки мобільного застосунку.

У розробці мобільного застосунку, задачею можна назвати будь-яке конкретне завдання або набір завдань, які необхідно виконати для досягнення

певної мети на кожному етапі процесу. Ось приклади задач для кожного з етапів:

– провести аналіз цільової аудиторії для визначення основних потреб і вимог;

– скласти детальний план проєкту, включаючи часові рамки та необхідні ресурси;

– визначити архітектуру застосунку та обрати технології, які будуть використовуватись;

– виконати тестування на різних пристроях для забезпечення сумісності;

– виправляти помилки, що виникають у користувачів після випуску;

– додавати нові функції та оновлення для підтримки актуальності застосунку.

Кожна з цих задач спрямована на досягнення конкретного результату та є частиною загального процесу розробки мобільного застосунку.

Висновки до розділу 1

У ході написання першого розділу було проведено дослідження теоретичних аспектів створення MP3 плеєра для мобільних пристроїв на базі операційної системи Android.

В результаті проведеного дослідження існуючих аналогів було сформульовано чітке уявлення про функціональні можливості та архітектуру MP3 плеєра для Android. Для аналізу було взято декілька з найпопулярніших застосунків для прослуховування музики з магазину застосунків Play Market.

Узагальнюючи, проведений аналіз та постановка задачі створили всебічний огляд предметної сфери, конкурентного середовища та конкретних завдань для створення конкурентоспроможного мобільного застосунку. Це забезпечує чітке розуміння вимог та завдань, що необхідно виконати для успішної реалізації проєкту та створення високоякісного кінцевого продукту.

2 МОДЕЛІ, МЕТОДИ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ДЛЯ ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

2.1 Методи для вирішення задачі

Існує 2 методи розробки мобільних застосунків:

– нативна розробка;

– кросплатформенна, або гібридна розробка (ReactNative, Flutter, Xamarin).

Нативна розробка має на увазі створення програми для мобільного пристрою на конкретній мові під конкретну платформу. Нативні застосунки досить продуктивні і не мають обмежень в розробці (Java і Kotlin – для Android, а Swift – для iOS). До плюсів такої розробки можна віднести досить швидку реакцію на дії користувача, можливість мати прямий доступ до апаратної частини і розробити найбільш звичний для користувача конкретної платформи інтерфейс. До недоліків можна віднести досить високу вартість розробки і підтримки, і тривалий час, необхідний на розробку.

Розробляти один і той самий додаток під різні платформи (iOS / Android) довго і дорого, до того ж їх тестування також займе досить тривалий час (наприклад, компіляція оновлень для Андроїд займає близько 30 секунд), з цього якщо потрібно створити простий додаток відразу для двох платформ, вдаються до гібридного способу розробки. Кросплатформенна розробка проводиться за допомогою web-технологій – HTML, CSS і JavaScript, які дозволяють розробити додаток відразу на кілька платформ. Але для того, щоб додаток працював у відповідності зі своєю платформою, його потрібно “перевести” на зрозумілу платформі мову, або додати проміжну ланку-перекладач. До переваг можна віднести низьку вартість розробки, адже для цього іноді достатньо буде задіяти одного фахівця. А ось до недоліків можна віднести можливі труднощі в роботі всіх функцій і затримки в реакції на дію користувача (додаток може бути повільним), також інтерфейс буде досить простим і потрібно буде його додатково

допрацьовувати.

Для створення застосунку було обрано нативну розробку, а саме середовище Android Studio.

Android Studio — це офіційне інтегроване середовище розробки (IDE) для розробки застосунків Android. Він заснований на IntelliJ IDEA, інтегрованому Java-середовищі розробки програмного забезпечення, і містить його інструменти для редагування коду та розробника.

Щоб підтримувати розробку застосунків в операційній системі Android, Android Studio використовує систему збірки на основі Gradle, емулятор Android, шаблони коду та інтеграцію GitHub. Кожен проект в Android Studio має одну або кілька модальностей із вихідним кодом і файлами ресурсів. Ці модальності включають модулі програми Android, модулі бібліотеки та модулі Google App Engine.

Android Studio використовує функцію «Застосувати зміни», щоб надсилати зміни коду та ресурсів до запущеної програми. Редактор коду допомагає розробнику писати код і пропонує доповнення, заломлення та аналіз коду. Програми, створені в Android Studio, потім компілюються у формат APK для надсилання в магазин Google Play.

Програмне забезпечення було вперше анонсовано на Google I/O у травні 2013 року, а перша стабільна збірка була випущена в грудні 2014 року. Android Studio доступний для настільних платформ macOS, Windows і Linux. Він замінив Eclipse Android Development Tools (ADT) як основну IDE для розробки застосунків Android. Android Studio та пакет розробки програмного забезпечення можна завантажити безпосередньо з Google.

Android Studio містить інструменти для розробки рішень для смартфонів і планшетів, а також нові технологічні рішення для Android TV, Android Wear, Android Auto, Glass і додаткові контекстуальні модулі (рис. 2.1).

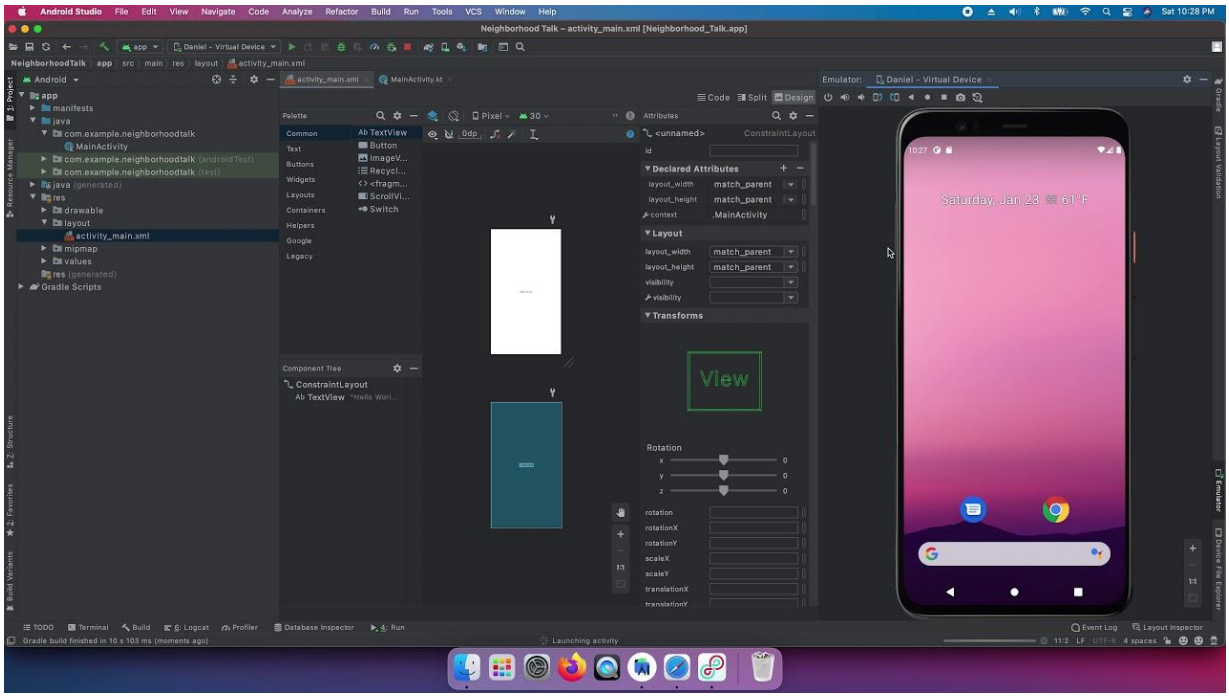

Рисунок 2.1 – Інтерфейс Android Studio

Середовище Android Studio призначене як для невеликих команд розробників мобільних застосунків (навіть однієї людини), так і для великих міжнародних організацій з GIT або іншими подібними системами управління версіями. Досвідчені розробники зможуть вибрати інструменти, які найбільше підходять для масштабних проектів. Рішення для Android розробляються в Android Studio за допомогою Java або C++.

В основі робочого процесу Android Studio закладено концепт безперервної інтеграції, що дозволяє відразу виявляти наявні проблеми.

Тривала перевірка коду забезпечує можливість ефективного зворотного зв'язку з розробниками. Така опція дозволяє швидше опублікувати версію мобільного застосунка в Google Play App Store. Для цього є також підтримка інструментів LINT, Pro-Guard і App Signing.

За допомогою засобів оцінки продуктивності визначається стан файлу з пакетом прикладних програм. Візуалізація графіки дає змогу дізнатися, чи програма відповідає орієнтиру Google в 16 мілісекунд (рис. 2.2).

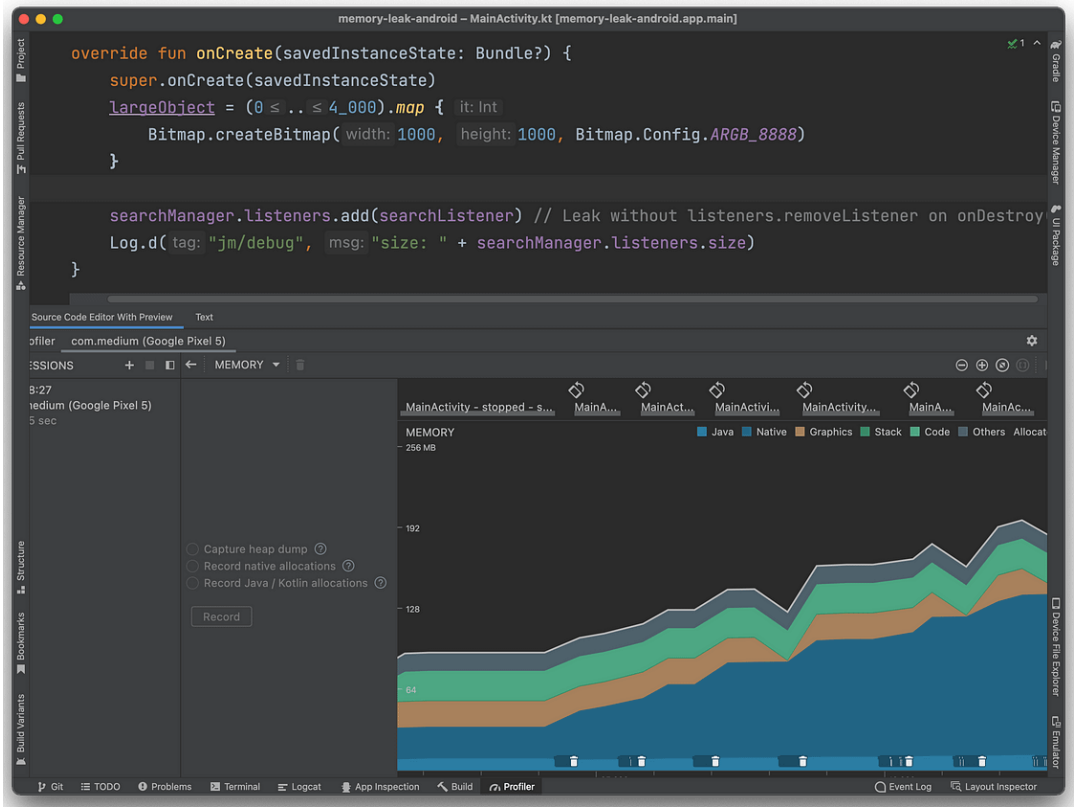

Рисунок 2.2 – Інструмент для візуалізації пам'яті в Android Studio

За допомогою інструмента для візуалізації пам'яті розробник дізнається, коли його додаток буде використовувати занадто багато оперативної пам'яті і коли відбудеться збір сміття.

Інструменти для аналізу батареї показують, яке навантаження лягає на пристрій.

Android Studio сумісна з платформою Google App Engine для швидкої інтеграції у хмари нових API та функцій. У розробці є різні API, такі як Google Play, Android Pay і Health.

Є підтримка всіх платформ Android, починаючи з Android 1.6.

Для початку роботи з середовищем Android Studio необхідно ознайомитись з базовими принципами роботи в ньому.

2.1.1 Встановлення Android Studio

Для початку розробки застосунків необхідно встановити Android Studio, для цього треба виконати наступні дії:

1) завантажити Android Studio з офіційного сайту, вибравши версію для конкретної ОС (Windows, macOS або Linux);

2) встановити Android Studio, дотримуючись інструкцій майстра встановлення, вказавши компоненти і створивши ярлик для зручності доступу;

3) встановити Android SDK – набір інструментів і бібліотек для розробки Android-застосунків;

4) налаштуйте Android Virtual Device (AVD), створивши емулятор Android для тестування застосунків.

2.1.2 Вибір мови програмування

Основні дві мови, які підтримуються для Android-розробки, – це Java і Kotlin, їх характеристики наведені в таблиці 2.1.

Таблиця 2.1 – Характеристики мов програмування Java та Kotlin

Характеристика	Java	Kotlin
Синтаксис	Старий, об’ємний	Сучасний, лаконічний, null safety
Інтероперабельність	Хороша, можна поступово переходити	100% інтероперабельність, легкий імпорт
Безпека	Не підтримує null safety	Підтримує null safety
Швидкість розробки	Повільніша	Швидка і продуктивна
Суспільство та ресурси	Велика спільнота, безліч ресурсів	Активна спільнота, багато ресурсів
Рівень абстракції	Більш низький	Більш високий, зручні функції
Складність	Складніше для вивчення і використання	Простіший в освоєнні та використанні

Для поточного проекту було обрано мову програмування Kotlin.

2.1.3 Створення нового проекту

Створення нового проекту в Android Studio – це перший крок до розробки мобільного застосунку. Варто розглянути цей процес детальніше, починаючи зі створення нового проекту і вибору відповідного шаблону. Для цього необхідно відтворити наступні кроки:

1) запуск Android Studio: переконатися, що Android Studio встановлена та налаштована на комп'ютері та запустити програму;

2) створення нового проекту: вибрати "Start a new Android Studio project" з головного меню або на стартовому екрані;

3) шаблон проекту: Android Studio пропонує різні шаблони проектів, такі як "Empty Activity" і "Basic Activity", призначені для різних типів застосунків. Необхідно вибрати відповідний шаблон, який визначить структуру проекту;

4) налаштування основних параметрів: необхідно вказати назву проекту, унікальне ім'я пакета ("com.example.myapp"), місце збереження та вибрати мову програмування (Java або Kotlin). Натиснути "Finish" або "Next", залежно від версії Android Studio.

Тепер проект створено, і можна почати розробку Android-застосунку. Створення нового проекту – лише початок шляху Android-розробки, і наступним кроком буде створення користувацького інтерфейсу та написання логіки програми. Ці аспекти розробки будуть описані в третьому розділі звіту.

2.2 Технології розробки системи

2.2.1 Опис мови програмування Kotlin

Kotlin — це сучасний, статично типізований і один із найбільш швидко прогресуючих мов програмування, створений і розвивається компанією JetBrains. Kotlin можна використовувати для створення самих різних програм. Це і

застосунки для мобільних пристроїв - Android, iOS. При чому Kotlin дозволяє писати кросплатформенний код, який буде застосовуватися на всіх платформах. Це і веб-застосунки, причому як серверні застосунки, які обробляють на стороні сервера – бекенда, так і браузерні клієнтські застосунки – фронтенд. Kotlin також можна застосовувати для створення настільних програм, для Data Science і так далі.

Таким чином, цілий ряд платформ, для яких можна створювати застосунки на Kotlin, надзвичайно широкий - Windows, Linux, MacOS, iOS, Android.

Найпопулярнішим напрямком, де застосовується Kotlin, є перш за все розробка під ОС Android. При чому настільки популярний, що компанія Google на конференції Google I/O 2017 оголосила Kotlin однією з офіційних мов для розробки під Android (наряду з Java і C++), а інструменти для роботи з даними мовою були за замовчуванням включені у функціональну середу розробки Android Studio починаючи з версією 3.0.

Перша версія мови вийшла 15 лютого 2016 року. Хоча сама розробка мови була з 2010 року. Поточною версією мови на даний момент є версія 1.9, яка вийшла в липні 2023 року.

Свою назву мови Kotlin отримав від назви острова, який розташований в Балтійському морі.

Kotlin випробував вплив багатьох мов: Java, Scala, Groovy, C#, JavaScript, Swift і дозволяє писати програми як в об'єктно-орієнтованому, так і у функціональному стилі. Він має ясний і зрозумілий синтаксис і досить легкий для навчання. Файли написані цією мовою мають розширення .kt.

Але Kotlin - це не просто чергова мова програмування. На сьогоднішній день це ціла екосистема (рис. 2.3).

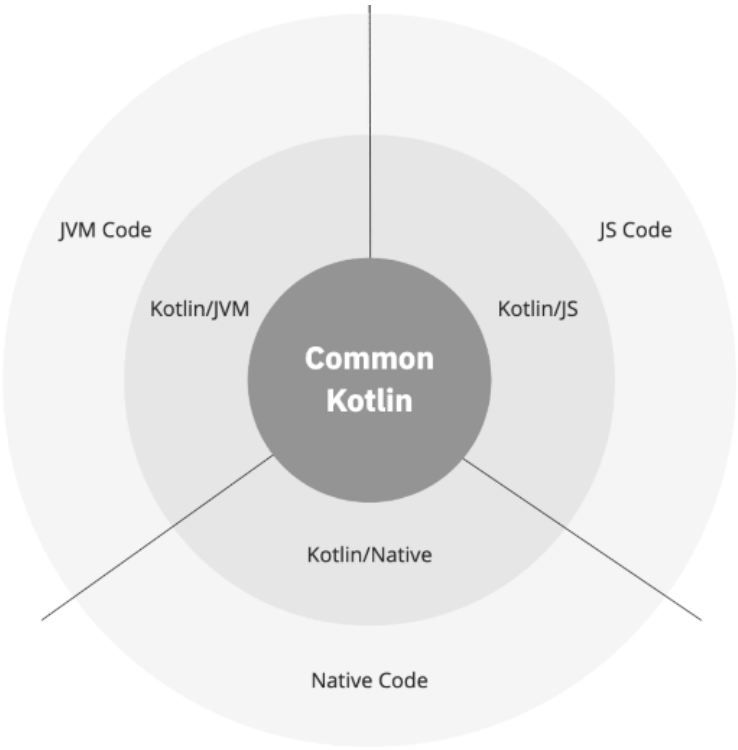

Рисунок 2.3 – Візуалізація екосистеми Kotlin

Ядро цієї екосистеми - Common Kotlin, який включає в себе власну мову, основні бібліотеки та базові інструменти для побудови програми.

Для взаємодії з конкретною платформою призначені для цієї платформи версії Kotlin: Kotlin/JVM, Kotlin/JS і Kotlin/Native. Ці специфічні версії представляють розширення для мови Kotlin, а також специфічні для конкретної платформи бібліотеки та інструменти розробки.

Перед початком роботи з Kotlin слід відзначити, що потрібен JDK (Java Development Kit). Однак знати саму мову програмування Java необов'язково. Достатньо лише знати, що в загальному випадку Kotlin працює поверх віртуальної машини Java (JVM). Завдяки цьому Kotlin може використовувати бібліотеки, написані на Java. Аналогічно з програми на Java можна запускати код Kotlin. Однак Kotlin/Native дозволяє писати застосунки, які компілюються в нативний код і які виконуються без JVM. [11]

2.2.2 Огляд інструменту Jetpack Compose

Jetpack Compose — це набір інструментів для побудови сучасних UI (користувацьких інтерфейсів) в Android-застосунках. Компанія Google анонсувала Jetpack Compose у 2019 році, а вже в березні 2021 року випустила бета-версію фреймворка.

Стабільний Jetpack Compose 1.0, який користувачі чекали майже 2 роки, вийшов 28 липня 2021 року. До цього моменту ми в GO Digital вже були підковані в бібліотеках знань і активно тестували розробку інтерфейсу користувача за допомогою фреймворка (без XML-файлів) у поточних проектах.

З впровадженням Jetpack Compose стало легше працювати над інтерфейсом користувача для Android. Прості API допомагають створювати приємні інтерфейси. Кількість коду зменшується в середньому на 30%. Швидкість розробки також скорочується, відбувається оптимізація інтерфейсу користувача завдяки потужним фічам: попередній перегляд, дії редактора, інспектор макета, ітеративна розробка коду, анімація.

Jetpack Compose побудований на базі мови програмування Kotlin від компанії JetBrains. Kotlin особливо популярний в мобільній розробці і дає великі переваги для Compose. Наприклад, користувач пише клас даних для використання в системах, дефолт-значення – у методах, функції розширення – для розширення функціональності.

В основі технології Jetpack Compose – декларативний підхід. Вже було зазначено, що розробка відбувається без xml-макетів (макетів). Цей факт і характеризує декларативний UI.

Варто перерахувати переваги та недоліки даної технології:

– декларативний інтерфейс користувача: Jetpack Compose використовує декларативний синтаксис, який легко читати та приймати. Це напряму впливає на швидкість і якість розробки;

– нескладний в опануванні: фреймворк має низький поріг входження, тому що прискорює розробку користувацьких інтерфейсів і допомагає зменшити обсяг коду;

– тестування інтерфейсу користувача: Jetpack Compose спрощує опис тестів інтерфейсу користувача на Kotlin;

– інтерактивний інтерфейс користувача: Jetpack Compose забезпечує оновлення в режимі реального часу при взаємодії користувача з інтерфейсом. Таким чином, додаток виглядає більш привабливо в очах юзера, а його досвід користувача покращується;

– повторне використання: за допомогою Jetpack Compose можна створювати багатократно використовувані компоненти інтерфейсу користувача, а потім використовувати їх на кількох екранах і навіть у різних застосунках. Це особливо важливо при створенні складних інтерфейсів користувача;

– типобезпечність: Jetpack Compose побудовано на базі мови Kotlin, яка забезпечує точну типізацію та допомагає виявити помилки на етапі компіляції, а не під час виконання програми. Це свідчить про стабільність і надійність застосунків;

– просте відлагодження: Jetpack Compose поставляється з набором інструментів відлагодження, які можуть допомогти розробникам швидко виявити та виправити проблеми у своєму коді інтерфейсу користувача;

– гнучкість: Jetpack Compose пропонує безліч можливостей для проектування інтерфейсу користувача. Програмісти можуть легко створювати макети, додавати анімації та кастомізувати зовнішній вигляд застосунків, не користуючись написанням великої кількості коду;

– симпатії спільноти: Jetpack Compose – це відносно новий *toolkit*, але він уже отримав багато уваги серед розробників Android. Фахівці вносять свій вклад у розвиток фреймворка, суспільство росте, а значить, інструмент може розраховувати на підтримку та ресурси.

Недоліки:

– підтримує тільки Kotlin. Це може бути недоліком для розробників, які вже впевнено володіють іншими мовами програмування, наприклад, Java, C++ або Swift. Це також може бути проблемою для команди розробників з різними мовними перевагами або рівнями навиків. Крім того, не всі сторонні бібліотеки чи інструменти, які розробники бажають використовувати у своїх проектах, можуть бути написані на Kotlin. Тому виникають проблеми сумісності або додаткова робота, необхідна для їх інтеграції з Jetpack Compose;

– проблеми з вимальовкою UI-компонентів: під час рендеринга можуть виникнути лаги та затримки. Це пов’язано з тим, що Compose призначений для представлення динамічного рендерингу, на відміну від традиційного підходу з використанням попередньо нарисованих XML-макетів. Це означає, що процес рендеринга може зайняти більше часу, особливо при роботі з анімацією. Однак проблеми з продуктивністю часто можна пом’якшити, оптимізувавши розташування UI-компонентів і мінімізувавши кількість рекомпозицій, що відбуваються в процесі рендеринга;

– міграція: якщо є готовий додаток на XML-макетах, для переходу на Jetpack Compose можуть знадобитися більші зміни коду. А це займає час. [10]

Висновки до розділу 2

Розглянуті методи та технології забезпечують ефективну реалізацію MP3-плеєра для Android. Android Studio надає всі необхідні інструменти для розробки та тестування застосунка, Jetpack Compose спрощує створення інтерфейсу користувача, а ExoPlayer забезпечує надійне та якісне відтворення аудіофайлів. Використання цих інструментів дозволить створити функціональний, зручний і продуктивний MP3-плеєр, що відповідає вимогам сучасних користувачів.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Опис програмної реалізації

Створення програм із функціями відтворення мультимедіа пов'язана з низкою труднощів, які ускладнюють розробку. Однак 2022 року з'явилася можливість використати Jetpack Media3 — рішення, яке повністю змінює процес взаємодії з мультимедіа.

Щоб відтворити мультимедіа файл, необхідні дві складові: сам медіаплеєр, який програє аудіо та відео, а також UI, що відображає деякі метадані (наприклад, назву, тривалість аудіотреку або відео та поточний стан відтворення).

Крім того, інтерфейс користувача дозволяє взаємодіяти з плеєром через кнопки — програма отримує команди «Відтворення», «Пауза», «Навігація за часом» і виводить повідомлення при зміні стану.

Проблема полягає в тому, що в цій архітектурі Android та інші програми не отримують інформації про відтворення мультимедіа. Як наслідок, користувачі позбавляються зручних способів перегляду та керування плеєром поза нашою програмою.

Щоб вирішити цю проблему, необхідно підключити плеєр до сеансу мультимедіа - тоді система отримає дані про те, що зараз відтворюється файл, і надасть можливість керування. При цьому заходити в сам додаток не потрібно.

Завдяки цьому стають доступними багато корисних інтеграцій. Наприклад, можна керувати відтворенням файлу через розумний годинник або скористатися кнопками на навушниках. Надалі планується реалізація управління з інших застосунків, а також голосового помічника.

Таким чином, подібна архітектура дозволяє повніше використовувати можливості як самого плеєра, так і всієї системи в цілому.

Головне питання, з якими стикаються розробники в цій сфері – це вибір підходящої бібліотеки для реалізації функції відтворення та управління мультимедіа (рис. 3.1).

Вирішення цього питання – Jetpack Media3. Але щоб краще зрозуміти її переваги, спочатку зробимо огляд деяких бібліотек, які використовувалися раніше.

Androidx.media2. У ній реалізовані модулі, які відповідають за відтворення, компоненти інтерфейсу, контроль сеансу мультимедіа, а також загальні структури даних та функції.

ExoPlayer. Це бібліотека для відтворення мультимедіа. Її використовують у сотнях тисяч програм. У цих API є модулі, які практично дублюють один одного та виконують загалом однакові функції.

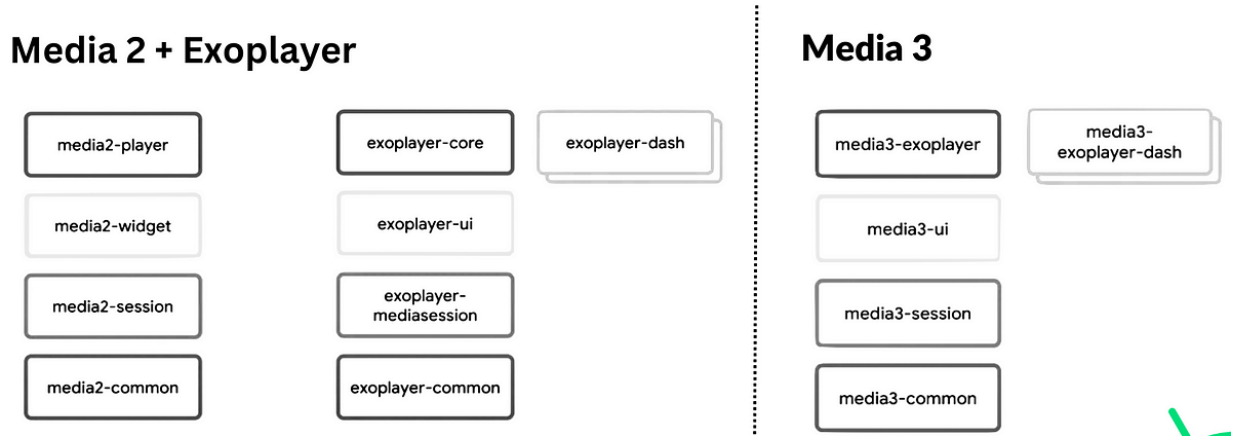

Рисунок 3.1 – Основні бібліотеки Android для роботи з мультимедіа

Також існують інші бібліотеки, включаючи оригінальну compat-бібліотеку androidx.media.

Безліч подібних бібліотек, їхні можливості та недоліки, різні умови сумісності - все це ускладнює вибір бібліотеки для використання.

Для зручності розробників у Jetpack Media3 вирішили поєднати ці бібліотеки.

– було створено один загальний модуль;

– ExoPlayer був стандартизований, при цьому зберігся його великий набір допоміжних модулів;

– був реалізований один модуль з компонентами інтерфейсу користувача та виду плеєра і ще один модуль з API для роботи з медіасесіями системи.

Результатом став androidx.media3, який надає пов'язаний набір бібліотек різних варіантів використання мультимедіа.

Варто описати головні юзкейси та виділити переваги Jetpack Media3 у порівнянні з минулими версіями.

У варіанті відтворення контенту, при якому програма розташована на передньому плані можна використовувати архітектуру, коли UI, плеєр та медіасеанс знаходяться в одній активності. У цьому випадку медіасеанс дозволяє підтримувати події мультимедійних клавіш та елементи керування відображенням "картинка в картинці" (рис. 3.2).

Рисунок 3.2 – Архітектура роботи в режимі «foreground service» застарілих API

Раніше в цьому підході виникали складнощі, тому що з попередніми API сеанс мультимедіа не міг безпосередньо взаємодіяти з плеєром. Потрібний об'єкт-конектор (рис 3.3), який би переводив команди та зворотні виклики між цими компонентами.

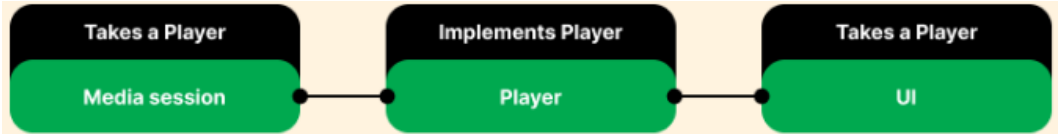

Рисунок 3.3 – Архітектура роботи в режимі «foreground service» Media3

Щоб реалізувати весь набір команд, які може отримувати сеанс мультимедіа, і обробки різних станів плеєра потрібно писати багато коду. Через це з'являлися

додаткові складності і підвищувався ризик виникнення помилок.

Для вирішення цього питання в Media3 відмовилися від конектора. Media3 надає ExoPlayer як свій стандартний плеєр — і саме в ньому реалізується інтерфейс Player. Тому в Google оновили MediaSession і віджети UI, щоб вони сприймали той же Player-інтерфейс, зв'язавши їх безпосередньо один з одним.

У разі фонового відтворення дещо складніше. Архітектура розділена між сервісом, що містить програвач, і активністю для інтерфейсу користувача (рис 3.4). Сервіс запускає сеанс мультимедіа, який використовується для оголошення відтворення та передачі команд плеєру. І всередині активності створюється медіаконтролер, який служить для зв'язку з сеансом мультимедіа.

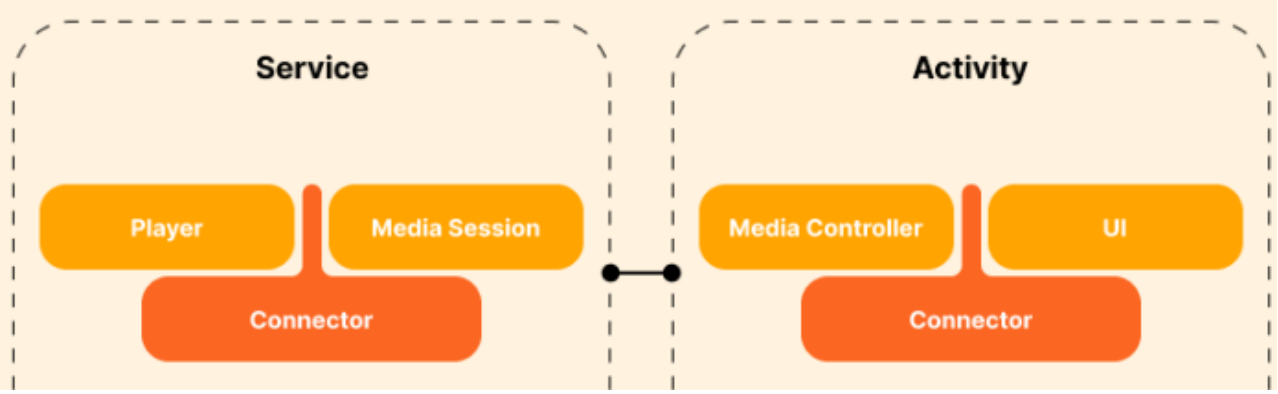

Рисунок 3.4 – Архітектура роботи в режимі «background service» застарілих API

Як вже говорилося раніше, плеєр не може взаємодіяти безпосередньо з сеансом - потрібен конектор. Також він необхідний для медіаконтролера та UI. В результаті виникає та ж проблема, що і з програванням на передньому плані – з'єднувачі ускладнюють код, що призводить до збоїв (рис. 3.5).

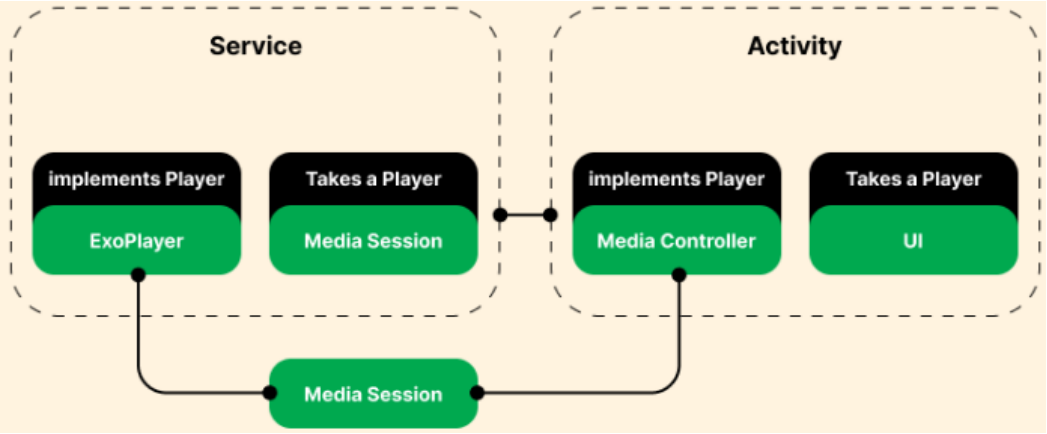

Рисунок 3.5 – Архітектура роботи в режимі «background service» Media3

Щоб вирішити це питання, було реалізовано спільний інтерфейс Player. Тепер ExoPlayer безпосередньо сумісний із сеансом мультимедіа, а медіаконтролер безпосередньо сумісний із UI. В результаті таких змін виходить код, який легше підтримувати і який менше схильний до помилок.

Важливим э питання, як застосунок може працювати з іншими програмами в системі Android.

Існують два основні варіанти. У першому випадку відкривається доступ до медіасеансу, щоб інші програми та пристрої могли керувати нашим медіаплеєром та відтворенням. Для користувача такий варіант може бути найзручнішим.

У другому випадку передається контент відтворення - інші програми можуть використовувати свій юзер-інтерфейс. Це важливо, наприклад, для Android Auto, який надає власний, зручний для водія UI для нашого контенту. [9]

3.1.1 Опис базової структури проєкту

Проект Android може складатися із різних модулів. За замовчуванням, коли створюється проект, створюється один модуль – app. Модуль має три підпапки:

– *manifests*: зберігає файл маніфесту *AndroidManifest.xml*, який описує конфігурацію програми та визначає кожен із компонентів цієї програми;

– *java*: зберігає файли коду мовою Java, які структуровані окремими пакетами. Так, у папці com.example.myapp (назва якого було вказано на етапі створення проекту) є за замовчуванням файл *MainActivity.java* з кодом на мові Java, який представляє клас *MainActivity*, який запускається за замовчуванням при старті програми;

– *res*: містить ресурси, що використовуються в застосунку. Усі ресурси розбиті на підпапки.

Директорія res може містити наступні підпапки та файли:

1) папка *drawable* призначена для зберігання зображень, що використовуються в програмі;

2) папка *layout* призначена для зберігання файлів, що визначають графічний інтерфейс. За промовчанням тут є файл *activity_main.xml*, який визначає інтерфейс для класу *MainActivity* у вигляді xml;

3) папки *mipmap* містять файли зображень, які призначені для створення іконки програми за різних роздільних здатностей екрана;

4) папка *values* зберігає різні XML-файли, що містять колекції ресурсів – різних даних, які застосовуються у застосунку. За замовчуванням тут є два файли та одна папка:

– файл *colors.xml* зберігає опис кольорів, що використовуються у програмі;

– файл *strings.xml* містить рядкові ресурси, які використовуються у застосунку;

– папка *themes* зберігає дві теми програми – світлу (денна) і темну (нічна) (рис. 3.6).

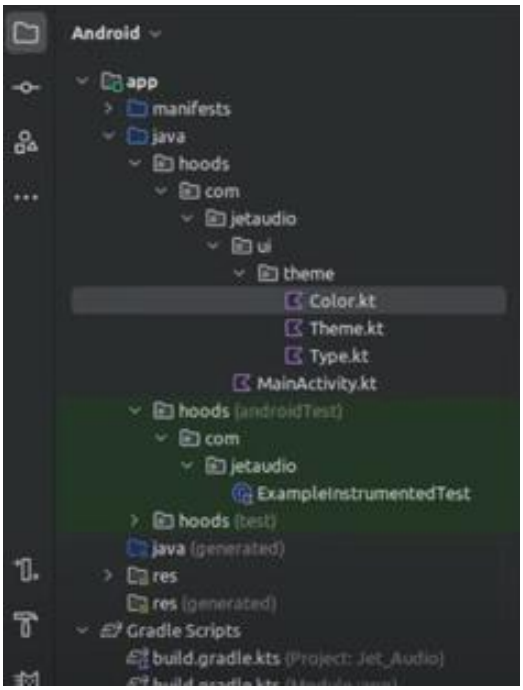

Рисунок 3.6 – Базова структура проєкту

Варто зазначити, що при використанні інструменту Jetpack Compose зникає необхідність у використанні більшості файлів з директорії res, до прикладу, в даному проекті вони взагалі не зазнавали змін.

Окремий елемент *Gradle Scripts* містить низку скриптів, які використовуються при побудові програми.

У всій цій структурі слід виділити файл *MainActivity.kt*, який містить логіку програми та власне з нього починається виконання програми.

3.1.2 Реалізація рівню доступу до даних

Перше, що необхідно для функціонування створюваного застосунку, це доступ до даних, а саме, до медіафайлів. Для цього спершу необхідно було створити директорію *data* в папці jetaudio (цю папку можна побачити на рисунку 3.6), та в ній директорію *local*. Останню було названо так через місце розташування даних – в локальному сховищі. Для реалізації доступу до даних із мережі можна було б розташувати код в окремій папці *remote*.

Для функціонування програми необхідно створити клас-модель даних, ним

стане клас *Audio* в підпапці *model*. Клас буде містити наступний код:

```
data class Audio(
    val uri: Uri,
    val displayName: String,
    val id: Long,
    val artist: String,
    val data: String,
    val duration: Int,
    val title: String,
)
```

URI (Uniform Resource Identifier) – уніфікований ідентифікатор ресурсу.

URI — це символьний рядок, що дозволяє ідентифікувати будь-який ресурс: документ, зображення, файл, службу, скриньку електронної пошти. Перш за все йдеться, звичайно, про ресурси мережі Інтернет та Всесвітньої павутини. URI надає простий та розширюваний спосіб ідентифікації ресурсів. Розширюваність URI означає, що вже існує кілька схем ідентифікації всередині URI, і ще більше буде створено в майбутньому. [6] Крім об’єктів в мережі інтернет URI дозволяє звертатись до файлів локального сховища.

Значення інших полів класу Audio відповідають їх назві.

Наступним кроком є реалізація безпосередньо отримання даних, код, який виконує цю функцію знаходиться в файлі *ContentResolverHelper.kt* директорії *local,* його вміст можна побачити в додатку А.

Цей код призначений для отримання даних про аудіофайли з пристрою користувача, використовуючи *ContentResolver*. Він містить клас *ContentResolverHelper*, який допомагає взаємодіяти з медіа-контентом на Android-пристрої.

Варто описати компоненти класу *ContentResolverHelper*, серед них є анотації та залежності, поля та методи.

Анотації та залежності:

– *@Inject*: Анотація для ін'єкції залежностей з Dagger Hilt.

– *@ApplicationContext*: Анотація для отримання контексту застосунка.

Поля класу:

– *context*: Контекст застосунка, ін'єктований через конструктор.

– *mCursor*: Об'єкт Cursor, що використовується для зчитування даних з бази даних медіа.

– *projection*: Масив рядків, що визначає стовпці, які будуть витягнуті з бази даних.

– *selectionClause*: Умова відбору рядків (*SQL WHERE clause*).

– *selectionArg*: Аргументи для умови відбору.

– *sortOrder*: Порядок сортування результатів.

Методи класу:

– *getAudioData*(): Публічний метод, що викликає приватний метод getCursorData() для отримання списку об'єктів *Audio*.

– *getCursorData*(): Приватний метод, що виконує запит до бази даних медіа і зчитує дані, повертаючи їх у вигляді списку об'єктів *Audio*.

Логіка роботи цього коду полягає в формуванні запиту до бази даних MediaStore та обробці отриманих результатів.

Запит до бази даних передбачає наступні дії:

– використовується *context.contentResolver.query()* для запиту до зовнішнього медіаконтенту *(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI)*;

– у запиті використовується *projection*, щоб визначити, які стовпці витягувати;

– використовується *selectionClause* і *selectionArg*, щоб відфільтрувати лише музику, виключаючи певні MIME-типи;

– *sortOrder* визначає порядок сортування результатів за назвою файлу.

Обробка результатів в свою чергу передбачає наступні дії:

– якщо *mCursor* не дорівнює *null*, курсор використовується для зчитування даних.

– визначаються індекси стовпців через *cursor.getColumnIndexOrThrow*();

– у циклі *while (cursor.moveToNext())* дані зчитуються з кожного рядка курсору;

– створюються об'єкти *Audio* для кожного аудіофайлу і додаються до списку *audioList*;

– якщо курсор порожній *(count == 0)*, в лог виводиться повідомлення про помилку.

Цей код забезпечує ефективний спосіб отримання та обробки аудіофайлів на пристрої Android, використовуючи потужний API ContentResolver та інструменти з Dagger Hilt для управління залежностями.

Для виклику методів попереднього класу асинхронно та формування результуючої множини даних було реалізовано клас *AudioRepository* в директорії repository.

Клас містить єдиний метод *getAudioData*, який не приймає параметрів та повертає результат типу *List<Audio>*.

```
suspend fun getAudioData(): List<Audio> = withContext(Dispatchers.IO
{ contentResolver.getAudioData() }
```

Dispatchers.IO є одним з диспетчерів потоків (диспетчерів контекстів) у бібліотеці корутин Kotlin, який оптимізований для виконання операцій введення-виведення (I/O), таких як робота з файлами, мережеві запити або взаємодія з базами даних.

Функція *getAudioData* визначена як *suspend* (призупиняюча), що означає, що вона може бути призупинена і виконана асинхронно в корутині.

Саме робота вищеописаних класів реалізує доступ до даних створюваного застосунку.

3.1.3 Реалізація функціоналу плеєру

Код, відповідальний за даний функціонал буде розміщено в директорії *player*. В ній буде міститись дві піддерикторії: *notification* та *service* в яких буде розміщуватись код керування процесами відтворення аудіо. Папка «notification» міститиме файли *JetAudioNotificationAdapter.kt* та *JetAudioNotificationManager.kt*; папка *service* – файли *JetAudioService.kt* та *JetAudioServiceHandler.kt*.

Клас *JetAudioNotificationAdapter* відповідає за налаштування сповіщення для музичного плеєра з використанням бібліотеки *androidx.media3* (Media3) та бібліотеки Glide для завантаження зображень. Цей клас реалізує інтерфейс *PlayerNotificationManager.MediaDescriptionAdapter*, що дозволяє налаштовувати вигляд сповіщення для музичного плеєра. Код класу розміщений в додатку В.

Конструктор класу приймає наступні параметри:

– *context*: Контекст застосунка, який використовується для доступу до ресурсів;

– *pendingIntent*: *PendingIntent*, що викликається при натисканні на сповіщення.

Клас містить методи *getCurrentContentTitle*, *createCurrentContentIntent*, *getCurrentContentText*, *getCurrentLargeIcon*.

Метод *getCurrentContentTitle* повертає заголовок сповіщення, що відображає назву альбому з метаданих поточного медіа. Якщо назва альбому відсутня, повертає *Unknown*.

Метод *createCurrentContentIntent* повертає *PendingIntent*, що викликається при натисканні на сповіщення.

Метод *getCurrentContentText* повертає текст сповіщення, що відображає заголовок треку з метаданих поточного медіа. Якщо заголовок відсутній, повертає *Unknown*.

Метод getCurrentLargeIcon виконує наступні функції:

– використовує бібліотеку Glide для завантаження великого значка (обкладинки альбому) для сповіщення;

– завантажує зображення з URI, вказаного в метаданих поточного медіа (*player.mediaMetadata.artworkUri*);

– використовує *CustomTarget<Bitmap>* для асинхронного завантаження зображення;

– коли зображення завантажено, викликає *callback.onBitmap(resource)*, щоб встановити зображення в сповіщенні;

– повертає *null*, тому що зображення завантажується асинхронно.

JetAudioNotificationAdapter налаштовує сповіщення для музичного плеєра, використовуючи метадані поточного медіафайлу для відображення інформації про трек та альбом. Завантаження зображень здійснюється асинхронно з використанням Glide, що забезпечує плавну та ефективну роботу застосунка.

Клас *JetAudioNotificationManager* відповідає за створення та управління сповіщеннями для музичного плеєра на основі ExoPlayer та *MediaSession* (додаток Г). Він містить константи *NOTIFICATION_ID* – ідентифікатор для сповіщення; *NOTIFICATION_CHANNEL_NAME* – назва каналу сповіщень; *NOTIFICATION_CHANNEL_ID* – ідентифікатор каналу сповіщення.

Конструктор класу виконує наступні дії:

– інжекція залежностей: використовує Dagger Hilt для ін'єкції *Context* і *ExoPlayer*;

– *notificationManager*: ініціалізація *NotificationManagerCompat* для управління сповіщеннями;

– init блок: перевіряє версію ОС і створює канал сповіщень, якщо ОС Android 8.0 (Oreo) або вище (рис. 3.7).

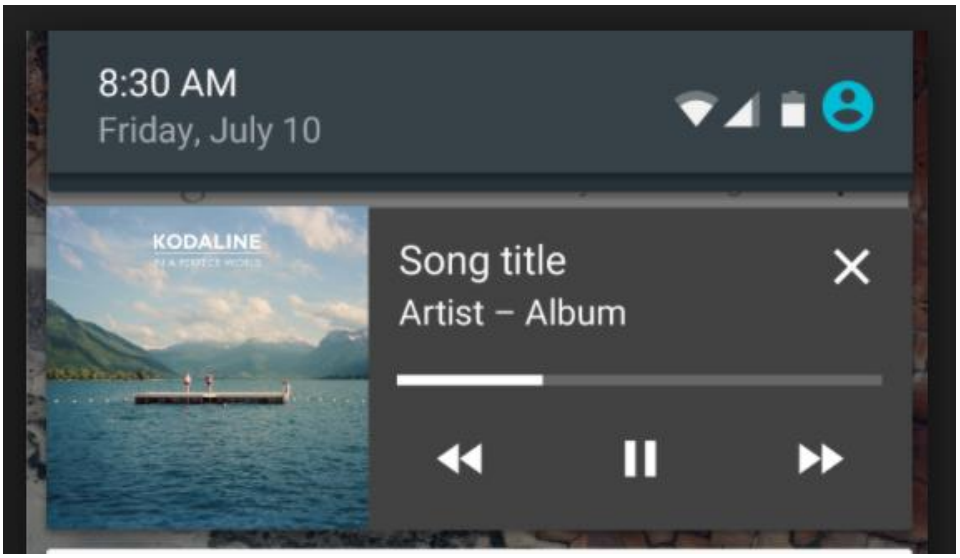

Рисунок 3.7 – Приклад сповіщення про відтворення музики

Клас містить методи *startNotificationService*, *startForeGroundNotificationService*, *buildNotification*, *createNotificationChannel*.

Метод startNotificationService приймає вхідні параметри: mediaSessionService і mediaSession та викликає *buildNotification* для створення сповіщення та *startForeGroundNotificationService* для запуску сервісу у передньому плані.

Метод *startForeGroundNotificationService* створює сповіщення за допомогою *Notification.Builder* та запускає сервіс у передньому плані з використанням startForeground.

Метод buildNotification будує сповіщення з використанням *PlayerNotificationManager.Builder*; налаштовує *JetAudioNotificationAdapter* для адаптації опису медіа та встановлює маленький значок, токен сесії, дії в компактному вигляді, пріоритет і плеєр.

Метод *createNotificationChannel* створює канал сповіщень з низьким пріоритетом та реєструє: Канал через *notificationManager*.

JetAudioNotificationManager забезпечує створення та управління сповіщеннями для аудіоплеєра, використовуючи *ExoPlayer*, *MediaSession*, *NotificationManagerCompat* і *PlayerNotificationManager*. Він створює канал сповіщень для Android 8.0 і вище, будує сповіщення з медіа-метаданими, запускає

сервіс у передньому плані для забезпечення тривалого відтворення музики, навіть коли додаток працює у фоновому режимі.

3.1.4 Модуль впровадження залежностей

Hilt це нова бібліотека для впровадження залежностей, побудована на основі Dagger. Вона дозволяє використовувати можливості Dagger у застосунках для Android спрощеним способом. У цьому розділі описано основні функціональні можливості бібліотеки та їх використання в проєкті.

Після встановлення всіх необхідних елементів і модулів, що підключаються, щоб використовувати Hilt, необхідно задати класу Application анотацію *@HiltAndroidApp*. Більше нічого не потрібно робити, а також не потрібно викликати Hilt безпосередньо.

При написанні коду, в якому використовується впровадження залежностей, є два основні компоненти, які слід враховувати:

– класи, що мають залежність, які ви збираєтеся впровадити;

– класи, які можуть бути впроваджені як залежність.

Вони є взаємовиключними: у часто клас одночасно є впроваджуваним і має залежності.

Щоб в Hilt зробити об'єкт, що впроваджується, необхідно вказати для Hilt спосіб створення екземпляра цього об'єкта. Такі інструкції називаються прив'язками (рис. 3.8).

Є три способи визначення прив'язки Hilt:

– додати до конструктора анотацію *@Inject*;

– використовувати *@Binds* у модулі;

– використовувати *@Provides* у модулі.

Будь-який клас може мати конструктора з анотацією *@Inject*, що дозволяє використовувати його як залежність у будь-якому місці проекту.

Два інших способи перетворення об'єктів у впроваджувані Hilt пов'язані з використанням модулів.

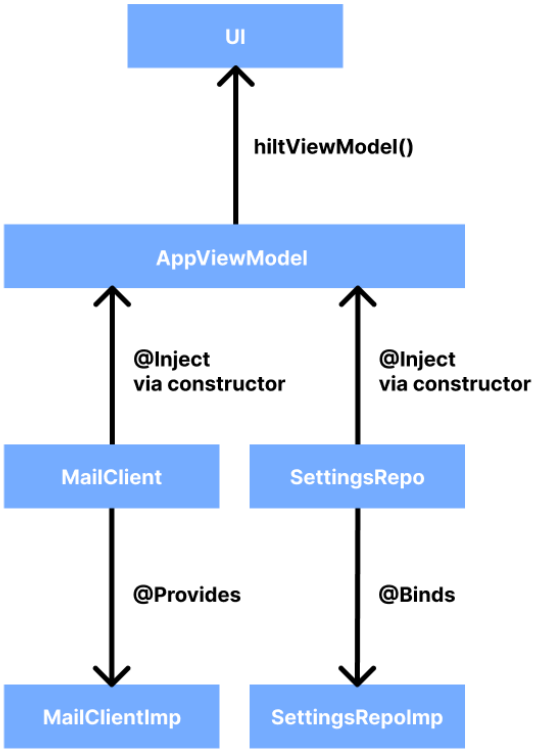

Рисунок 3.8 – Діаграма взаємодії залежностей організованих з Hilt

Модуль Hilt можна вважати набором «рецептів», які вказують Hilt, як створити екземпляр того чи іншого об'єкта, який не має конструктора – наприклад, інтерфейсу або системної служби.

Крім того, у тестах будь-який модуль можна замінити іншим модулем. Наприклад, це дозволяє легко замінювати реалізацію інтерфейсів мок-об'єктами. [7]

Модулі встановлюються компонент Hilt, який вказується за допомогою анотації *@InstallIn*. Одним з інструментів впровадження залежностей в даному проєкті стави модуль *MediaModule* (додаток И).

Клас MediaModule у налаштовує залежності, пов'язані з медіаплеєром, за допомогою Dagger Hilt. Він постачає компоненти, такі як *ExoPlayer, MediaSession, JetAudioNotificationManager* та інші, через відповідні методи, позначені анотаціями *@Provides* та *@Singleton*.

Метод *provideAudioAttributes* створює та надає об'єкт *AudioAttributes* з типом вмісту *C.AUDIO_CONTENT_TYPE_MOVIE* і використанням *C.USAGE_MEDIA*.

Метод *provideExoPlayer* створює та налаштовує *ExoPlayer*, встановлюючи аудіоатрибути, обробку аудіо, що переривається, та трек-селектор. Метод *provideMediaSession* створює *MediaSession*, використовуючи контекст та *ExoPlayer*. Метод *provideNotificationManager* створює *JetAudioNotificationManager*, використовуючи контекст і *ExoPlayer*. Метод *provideServiceHandler* створює *JetAudioServiceHandler*, використовуючи *ExoPlayer*.

Клас *MediaModule* забезпечує впровадження залежностей для компонентів медіаплеєра в застосунку, використовуючи Dagger Hilt. Він створює та надає необхідні об'єкти, такі як *ExoPlayer, MediaSession, JetAudioNotificationManager* та інші, забезпечуючи їхню доступність протягом усього життєвого циклу застосунку. Це спрощує управління залежностями та сприяє підтримуваності й масштабованості коду.

3.1.5 Реалізація інтерфейсу користувача

Для реалізації UI було створено два класи: *AudioViewModel.kt* та *Home.kt*.

Клас *AudioViewModel* є частиною архітектури MVVM (Model-View-ViewModel) для застосунка, який використовує Jetpack Compose і Hilt для управління станом та залежностями. Цей *ViewModel* відповідає за взаємодію з аудіо-сервісом та управління станом інтерфейсу користувача.

AudioViewModel використовує *JetAudioServiceHandler* для управління аудіо-плеєром і *AudioRepository* для отримання даних про аудіо. Залежності впроваджуються через конструктор, який позначений анотацією *@Inject*, що дозволяє Hilt автоматично забезпечити необхідні об'єкти.

Зберігання стану здійснюється за допомогою *SavedStateHandle*, що дозволяє зберігати стан *ViewModel* навіть при зміні конфігурацій. Змінні стану, такі як *duration, progress, progressString, isPlaying, currentSelectedAudio* та *audioList,* ініціалізуються за допомогою *mutableStateOf* та *saveable*.

При створенні *ViewModel* викликається метод *loadAudioData*, який запускає корутину для отримання даних аудіо з репозиторію і налаштовує медіа-елементи

для аудіо-плеєра.

ViewModel підписується на зміни стану аудіо-сервісу через *audioServiceHandler.audioState.collectLatest*, і відповідним чином оновлює стан інтерфейсу користувача *(_uiState).* Це включає обробку різних станів, таких як *Initial, Buffering, Playing, Progress, CurrentPlaying* та *Ready*.

Метод *onUiEvents* обробляє події, що надходять з інтерфейсу користувача, такі як *PlayPause, SeekToNext, Backward, Forward, SeekTo, SelectedAudioChange* та *UpdateProgress*. Відповідні події надсилаються до аудіо-сервісу через audioServiceHandler.

Метод *calculateProgressValue* обчислює поточний прогрес відтворення та оновлює відповідні змінні стану. Метод *formatDuration* форматує тривалість у вигляді «хвилини:секунди».

Метод *onCleared* забезпечує зупинку відтворення аудіо при знищенні *ViewModel*, викликаючи подію *PlayerEvent.Stop* у аудіо-сервісі.

AudioViewModel забезпечує ефективне управління станом аудіо-плеєра та інтеграцію з аудіо-сервісом у рамках архітектури MVVM, використовуючи сучасні інструменти Android, такі як Hilt, Jetpack Compose та Kotlin корутини. Код класу наведений в додатку Е.

Відповідно, компонент *HomeScreen* з файлу *Home.kt* (додаток Ж) відповідає за відображення списку аудіофайлів та управління програвачем. Використовуючи Jetpack Compose, він включає елементи інтерфейсу, такі як список аудіофайлів (*LazyColumn*), панель програвача (*BottomBarPlayer*), та елементи управління (*MediaPlayerController*, *PlayerIconItem*). Впровадження залежностей, таких як репозиторій аудіофайлів та обробник сервісу, відбувається через Hilt, що забезпечує модульність і розширюваність коду. Компоненти та функції організовані для забезпечення інтерактивного та зручного інтерфейсу користувача для прослуховування музики.

3.2 Тестування застосунку

Тестування мобільних застосунків – це процес, за допомогою якого прикладне програмне забезпечення, розроблене для портативних мобільних пристроїв, перевіряється на його функціональність, зручність використання та сумісність. Тестування може бути мануальним чи автоматизованим. Проєкт даної роботи було протестовано мануально на пристрої Meizu M5.

Для поглиблення теоретичних навичок варто перерахувати деякі з видів тестування мобільних пристроїв.

Функціональне тестування є базовим тестом для будь-якої програми, для перевірки відповідності вимогам. Подібно до інших програм, заснованих на інтерфейсі користувача, мобільні застосунки вимагають ряду взаємодій людини в сценаріях користувача.

Тестування сумісності має найвищу важливість, коли доходить до тестування мобільних застосунків. Мета тесту на сумісність мобільного застосунка, як правило, полягає в тому, щоб ключові функції програми працювали належним чином на конкретному пристрої. Сама сумісність повинна займати лише кілька хвилин і може бути спланована заздалегідь. Вирішити, які тести на сумісність мобільних пристроїв слід виконати не легке завдання (оскільки тестування з усіма наявними пристроями просто неможливе). Тому необхідно підготувати тестову матрицю з кожною можливою комбінацією та розставити пріоритети для клієнта (рис. 3.9)..

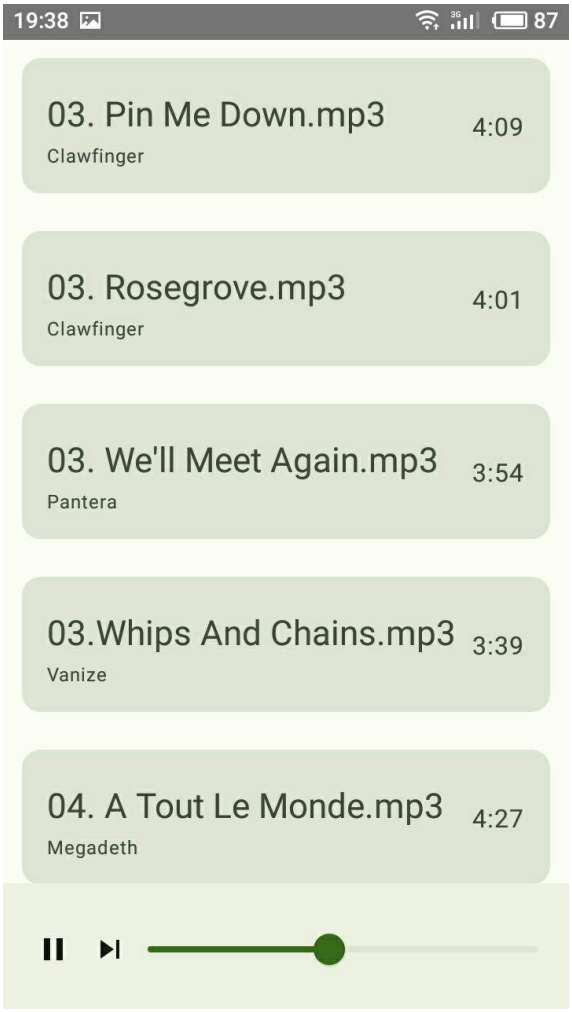

Рисунок 3.9 – Інтерфейс створеного застосунку

Localization Testing. В даний час більшість програм призначені для глобального використання, і дуже важливо піклуватися про регіональні особливості, такі як мови, часові пояси і т.д. Важливо перевірити функціональність програми, коли хтось змінює часовий пояс. Необхідно враховувати, що іноді західні дизайни можуть працювати з аудиторією зі східних країн чи навпаки.

Aboratory testing, які зазвичай проводять мережевими операторами, виконуються шляхом моделювання всієї бездротової мережі. Цей тест виконується для виявлення будь-яких збоїв, коли мобільний додаток використовує передачу голосу та/або даних для виконання деяких функцій.

Performance Testing охоплює продуктивність клієнтських програм, сервера та

мережі. Завдяки *Performance Testing* можна ідентифікувати існуючі мережі, сервери та вузькі місця серверних програм, враховуючи зумовлене навантаження та поєднання транзакцій.

Stress Testing є обов'язковим тестуванням на шляху виявлення винятків, зависань і взаємоблокувань, що може залишитися непоміченими під час тестування функціональності та інтерфейсу користувача.

Security Testing допомагає виявити всі можливі вразливості щодо політик злому, автентифікації та авторизації, безпеки даних, управління сеансами та інших стандартів безпеки. Програми повинні шифрувати ім'я користувача та паролі під час аутентифікації користувача по мережі.

Usability Testing оцінює додаток на основі таких трьох критеріїв для цільової аудиторії: ефективність; точність та повнота; задоволеність. Дуже важливо провести юзабіліті-тестування з раннього етапу розробки програми. Цей вид тестування вимагає активної участі користувачів, і результати можуть вплинути на дизайн програми, що дуже важко змінити на пізніших етапах проекту. [8]

В результаті функціонального тестування створеного застосунку було зроблено висновок, що програма процює справно: будь-яка композиція доступна для відтворення, прогрес бар виконує свою функцію та дозволяє переходити до вибраного моменту пісні, але наявні деякі проблеми з відтворенням у фоновому режимі, що спричинено застарілою версією Android на пристрої.

Висновки до розділу 3

Розробка мобільного застосунку для ОС Android включала створення функціонального інтерфейсу користувача за допомогою Jetpack Compose, що забезпечує декларативний підхід до UI. Для впровадження залежностей було використано Hilt, що спрощує управління життєвим циклом залежностей і робить код більш модульним і тестованим. Для відтворення медіа-контенту застосунок використовує Media3, що надає потужні інструменти для роботи з аудіо та відео, а для завантаження і кешування зображень використовується бібліотека Glide.

Тестування застосунку включало перевірку коректності роботи основних функцій, таких як відтворення, пауза, перемотування та відображення сповіщень. Тестування UI було проведено для забезпечення зручності і зрозумілості інтерфейсу для користувачів.

Підсумовуючи даний розділ можна сказати, що застосунок для прослуховування музики було успішно реалізовано, хоча він і має значно примітивніший функціонал ніж розглянуті в першому розділі аналоги.

ВИСНОВКИ

При виконанні кваліфікаційної роботи було досліджено та проаналізовано існуючі аналоги та технології необхідні для створення застосунку для відтворення аудіофайлів та реалізовано його програмно. Розробка мобільного застосунку MP3-плеєра для ОС Android включала аналіз сучасних підходів та технологій, що використовуються в мобільних мультимедійних застосунках. Було досліджено архітектури та компоненти, необхідні для створення функціонального і зручного інтерфейсу користувача, а також технології для забезпечення високої якості відтворення аудіо.

Під час розробки застосунку були використані сучасні бібліотеки та інструменти, такі як Jetpack Compose для декларативного UI, Hilt для управління залежностями, та Media3 для роботи з медіа-контентом. Також було застосовано бібліотеку Glide для завантаження та кешування зображень.

Тестування застосунку включало перевірку коректності роботи основних функцій, таких як завантаження аудіофайлів, відтворення, пауза, перемотування та відображення сповіщень. Завдяки використанню архітектури MVVM та впровадженню залежностей через Hilt, тестування було полегшене, оскільки це дозволяє ізолювати окремі компоненти і тестувати їх незалежно.

Загалом, завдання кваліфікаційної роботи було виконано повною мірою: проаналізовано сучасні рішення для створення мультимедійних застосунків; розроблено та реалізовано функціональний MP3-плеєр для Android, що відповідає сучасним вимогам до зручності та продуктивності; проведено тестування застосунку для забезпечення його стабільної роботи в реальних умовах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Що таке Spotify? *Spotify:* вебсайт. URL: https://support.spotify.com/ua-uk/article/what-is-spotify/ (станом на 10.06.2024);

2. Spotify в Україні. Чим він особливий і чому всі про нього говорять. *BBC News Україна:* вебсайт. URL: https://www.bbc.com/ukrainian/features-53421179 (станом на 10.06.2024);

3. Ключові методології розробки програмного забезпечення: робота команди зсередини. *WEZOM:* вебсайт. URL: https://wezom.com.ua/ua/blog/metodologija-razrabotki-programmnogo-obespechenija (станом на 10.06.2024);

4. Життєвий цикл розробки програмного забезпечення за Agile-методологією. Що це таке і які фази охоплює? *EPAM Campus:* вебсайт. URL: https://training.epam.ua/ua/blog/581 (станом на 10.06.2024);

5. Хто такі стейкхолдери та як з ними дружити. *Brain Rain:* вебсайт. URL: https://brainrain.com.ua/kto-takie-steykholderi/ (станом на 10.06.2024);

6. Що таке URI з прикладами. *АйТі Бубен:* вебсайт. URL: https://wiki.dieg.info/uri (станом на 10.06.2024);

7. Практичний посібник з використання Hilt з Kotlin. URL: https://habr.com/ru/companies/otus/articles/532066/ (станом на 10.06.2024);

8. Тестування мобільних застосунків. *QALight:* вебсайт. URL: https://qalight.ua/ru/baza-znaniy/testirovanie-mobilnyh-prilozhenij/ (станом на 10.06.2024);

9. Нові можливості AndroidX Media та ExoPlayer. *Хабр:* вебсайт. URL: https://habr.com/ru/companies/cleverpumpkin/articles/689022/ (станом на 10.06.2024);

10. Jetpack Compose – сила Android-розробки. *Хабр:* вебсайт. URL: https://habr.com/ru/articles/722040/ (станом на 10.06.2024);

11. Введення в мову Kotlin. *METANIT:* вебсайт. URL: https://metanit.com/kotlin/tutorial/1.1.php (станом на 10.06.2024);

12. Санітарно–гігієнічні норми і правила роботи за комп'ютером. *Сайт дистанційної освіти АЕФК ПДАА:* вебсайт. URL: https://agrokoledg–online.pl.ua/wp– content/uploads/2020/11/sanitarno–gigiyenichni–normy–i–pravyla–roboty–za–kompyuterom.pdf (станом на 10.06.2024);

13. Робота в офісі: основні санітарно–гігієнічні вимоги. *Управління інспекційної діяльності у Тернопільській області Південно-Західного міжрегіонального управління Державної служби з питань праці:* вебсайт. URL: https://te.dsp.gov.ua/robota–v–ofisi–osnovni–sanitarno–gigiyenichni–vymogy/ (станом на 10.06.2024);

14. Вимоги до електробезпеки у офісних приміщеннях з комп'ютерною технікою. *Навчально-науковий Центр перепідготовки та заочного навчання:* вебсайт. URL: https://cpo.stu.cn.ua/Oksana/posibnik/1140.html (станом на 10.06.2024);

15. Вимоги до організації робочого місця з обслуговування, ремонту та налагодження ЕОМ. *Studfiles:* вебсайт. URL: https://studfile.net/preview/5228962/page:8/ (станом на 10.06.2024);

16. Вимоги безпеки під час обслуговування, ремонту та налагодження ЕОМ. *Vuzlit:* вебсайт. URL: https://vuzlit.com/135901/vimogi_bezpeki_obslugovuvannya_remontu_nalagodzhennya (станом на 10.06.2024);

17. Вимоги до режимів праці і відпочинку при роботі ВДТ. *Бібліотека економіста:* вебсайт. URL: https://library.if.ua/book/9/975.html (станом на 10.06.2024);

18. Дейтел Х. М., Дейтел П. Дж., Сантрі С. И., Технології програмування на Java 2: Книга 2. Розподілені застосунки. Пер. з англ. – М.: ООО «Біном-Пресс», 2003. 464 с.

19. Java: основи програмування. Пер. з англ. – К.: Видав. Група ВНИ. 1997. 320 с.

20. Основи інформатики. Текстовий редактор Word 2002: навчально-методичний посібник – Миколаїв: Вид-во МГДУ ім. Петра Могили. 2005. 132 с.

21. Сучасні комп'ютері технології і засоби масової комунікації: Аспекти застосування – К.: ІЗМН. 1996. 180 с.

22. Охорона праці користувачів комп'ютерів – Львів: Афіша. 2000. 176 с.

23. Доктрина інформаціологічного розвитку людства у XXI столітті / За ред. Ю. М. Воронцова, І. Й. Юзвілина – 2. Вид. – Луганськ. 2001. 51 с.

24. Верлань А. Ф., Широчин В. П. Інформатика і ЄВМ – К.: Техніка. 1987. 344 с.

25. Логіка і комп'ютер / Г. Л. Бузук – М.: 1995. 208 с.

ДОДАТОК А

Код з файлу ContentResolverHelper.kt

```
package hoods.com.jetaudio.data.local

import android.content.ContentUris
import android.content.Context
import android.database.Cursor
import android.provider.MediaStore
import android.util.Log
import androidx.annotation.WorkerThread
import dagger.hilt.android.qualifiers.ApplicationContext
import hoods.com.jetaudio.data.local.model.Audio
import javax.inject.Inject

class ContentResolverHelper @Inject
constructor(@ApplicationContext val context: Context) {
    private var mCursor: Cursor? = null

    private val projection: Array<String> = arrayOf(
        MediaStore.Audio.AudioColumns.DISPLAY_NAME,
        MediaStore.Audio.AudioColumns._ID,
        MediaStore.Audio.AudioColumns.ARTIST,
        MediaStore.Audio.AudioColumns.DATA,
        MediaStore.Audio.AudioColumns.DURATION,
        MediaStore.Audio.AudioColumns.TITLE,
    )

    private var selectionClause: String? =
        "${MediaStore.Audio.AudioColumns.IS_MUSIC} = ? AND
${MediaStore.Audio.Media.MIME_TYPE} NOT IN (?, ?, ?)"
    private var selectionArg = arrayOf("1", "audio/amr", "audio/3gpp", "audio/aac")

    private val sortOrder = "${MediaStore.Audio.AudioColumns.DISPLAY_NAME} ASC"


    @WorkerThread
    fun getAudioData(): List<Audio> {
        return getCursorData()
    }


    private fun getCursorData(): MutableList<Audio> {
        val audioList = mutableListOf<Audio>()

        mCursor = context.contentResolver.query(
            MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,
            projection,
            selectionClause,
            selectionArg,
            sortOrder
        )

        mCursor?.use { cursor ->
            val idColumn =
                cursor.getColumnIndexOrThrow(MediaStore.Audio.AudioColumns._ID)
            val displayNameColumn =
```

```
cursor.getColumnIndexOrThrow(MediaStore.Audio.AudioColumns.DISPLAY_NAME)
                val artistColumn =
                    cursor.getColumnIndexOrThrow(MediaStore.Audio.AudioColumns.ARTIST)
                val dataColumn =
                    cursor.getColumnIndexOrThrow(MediaStore.Audio.AudioColumns.DATA)
                val durationColumn =
                    cursor.getColumnIndexOrThrow(MediaStore.Audio.AudioColumns.DURATION)
                val titleColumn =
                    cursor.getColumnIndexOrThrow(MediaStore.Audio.AudioColumns.TITLE)

                cursor.apply {
                    if (count == 0) {
                        Log.e("Cursor", "getCursorData: Cursor is Empty")
                    } else {
                        while (cursor.moveToNext()) {
                            val displayName = getString(displayNameColumn)
                            val id = getLong(idColumn)
                            val artist = getString(artistColumn)
                            val data = getString(dataColumn)
                            val duration = getInt(durationColumn)
                            val title = getString(titleColumn)
                            val uri = ContentUris.withAppendedId(
                                MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,
                                id
                            )

                            audioList += Audio(
                                uri, displayName, id, artist, data, duration, title
                            )
                        }
                    }
                }
            }
            return audioList
        }
    }
```

ДОДАТОК Б

Код з файлу MediaModule.kt

```
package hoods.com.jetaudio.di
import android.content.Context
import androidx.media3.common.AudioAttributes
import androidx.media3.common.C
import androidx.media3.common.util.UnstableApi
import androidx.media3.exoplayer.ExoPlayer
import androidx.media3.exoplayer.trackselection.DefaultTrackSelector
import androidx.media3.session.MediaSession
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.android.qualifiers.ApplicationContext
import dagger.hilt.components.SingletonComponent
import hoods.com.jetaudio.player.notification.JetAudioNotificationManager
import hoods.com.jetaudio.player.service.JetAudioServiceHandler
import javax.inject.Singleton
@Module
@InstallIn(SingletonComponent::class)
object MediaModule {
    @Provides
    @Singleton
    fun provideAudioAttributes(): AudioAttributes = AudioAttributes.Builder()
        .setContentType(C.AUDIO_CONTENT_TYPE_MOVIE)
        .setUsage(C.USAGE_MEDIA)
        .build()
    @Provides
    @Singleton
    @UnstableApi
    fun provideExoPlayer(
        @ApplicationContext context: Context,
        audioAttributes: AudioAttributes,
    ): ExoPlayer = ExoPlayer.Builder(context)
        .setAudioAttributes(audioAttributes, true)
        .setHandleAudioBecomingNoisy(true)
        .setTrackSelector(DefaultTrackSelector(context))
        .build()
    @Provides
    @Singleton
    fun provideMediaSession(
        @ApplicationContext context: Context,
        player: ExoPlayer,
    ): MediaSession = MediaSession.Builder(context, player).build()
    @Provides
    @Singleton
    fun provideNotificationManager(
        @ApplicationContext context: Context,
        player: ExoPlayer,
    ): JetAudioNotificationManager = JetAudioNotificationManager(
        context = context,
        exoPlayer = player
    )
    @Provides
    @Singleton
    fun provideServiceHandler(exoPlayer: ExoPlayer): JetAudioServiceHandler =
        JetAudioServiceHandler(exoPlayer)
}
```

ДОДАТОК В

Код з файлу JetAudioNotificationAdapter.kt

```
package hoods.com.jetaudio.player.notification

import android.app.PendingIntent
import android.content.Context
import android.graphics.Bitmap
import android.graphics.drawable.Drawable
import androidx.media3.common.Player
import androidx.media3.common.util.UnstableApi
import androidx.media3.ui.PlayerNotificationManager
import com.bumptech.glide.Glide
import com.bumptech.glide.load.engine.DiskCacheStrategy
import com.bumptech.glide.request.target.CustomTarget
import com.bumptech.glide.request.transition.Transition

@UnstableApi
class JetAudioNotificationAdapter(
    private val context: Context,
    private val pendingIntent: PendingIntent?,
) : PlayerNotificationManager.MediaDescriptionAdapter {
    override fun getCurrentContentTitle(player: Player): CharSequence =
        player.mediaMetadata.albumTitle ?: "Unknown"

    override fun createCurrentContentIntent(player: Player): PendingIntent? =
pendingIntent

    override fun getCurrentContentText(player: Player): CharSequence =
        player.mediaMetadata.displayTitle ?: "Unknown"

    override fun getCurrentLargeIcon(
        player: Player,
        callback: PlayerNotificationManager.BitmapCallback,
    ): Bitmap? {
        Glide.with(context)
            .asBitmap()
            .load(player.mediaMetadata.artworkUri)
            .diskCacheStrategy(DiskCacheStrategy.ALL)
            .into(object : CustomTarget<Bitmap>() {
                override fun onResourceReady(resource: Bitmap, transition: Transition<in
Bitmap>?) {
                    callback.onBitmap(resource)
                }

                override fun onLoadCleared(placeholder: Drawable?) = Unit
            })
        return null
    }
}
```

ДОДАТОК Г

Код з файлу JetAudioNotificationManager.kt

```
package hoods.com.jetaudio.player.notification

import android.app.Notification
import android.app.NotificationChannel
import android.app.NotificationManager
import android.content.Context
import android.os.Build
import androidx.annotation.RequiresApi
import androidx.core.app.NotificationCompat
import androidx.core.app.NotificationManagerCompat
import androidx.media3.common.util.UnstableApi
import androidx.media3.exoplayer.ExoPlayer
import androidx.media3.session.MediaSession
import androidx.media3.session.MediaSessionService
import androidx.media3.ui.PlayerNotificationManager
import dagger.hilt.android.qualifiers.ApplicationContext
import hoods.com.jetaudio.R
import javax.inject.Inject

private const val NOTIFICATION_ID = 101
private const val NOTIFICATION_CHANNEL_NAME = "notification channel 1"
private const val NOTIFICATION_CHANNEL_ID = "notification channel id 1"

class JetAudioNotificationManager @Inject constructor(
    @ApplicationContext private val context: Context,
    private val exoPlayer: ExoPlayer,
) {
    private val notificationManager: NotificationManagerCompat =
        NotificationManagerCompat.from(context)

    init {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            createNotificationChannel()
        }
    }

    @RequiresApi(Build.VERSION_CODES.O)
    @UnstableApi
    fun startNotificationService(
        mediaSessionService: MediaSessionService,
        mediaSession: MediaSession,
    ) {
        buildNotification(mediaSession)
        startForeGroundNotificationService(mediaSessionService)
    }

    @RequiresApi(Build.VERSION_CODES.O)
    private          fun          startForeGroundNotificationService(mediaSessionService:
MediaSessionService) {
        val notification = Notification.Builder(context, NOTIFICATION_CHANNEL_ID)
            .setCategory(Notification.CATEGORY_SERVICE)
            .build()
        mediaSessionService.startForeground(NOTIFICATION_ID, notification)
    }
```

```
    @UnstableApi
    private fun buildNotification(mediaSession: MediaSession) {
        PlayerNotificationManager.Builder(
            context,
            NOTIFICATION_ID,
            NOTIFICATION_CHANNEL_ID
        )
            .setMediaDescriptionAdapter(
                JetAudioNotificationAdapter(
                    context = context,
                    pendingIntent = mediaSession.sessionActivity
                )
            )
            .setSmallIconResourceId(R.drawable.ic_microphone)
            .build()
            .also {
                it.setMediaSessionToken(mediaSession.sessionCompatToken)
                it.setUseFastForwardActionInCompactView(true)
                it.setUseRewindActionInCompactView(true)
                it.setUseNextActionInCompactView(true)
                it.setPriority(NotificationCompat.PRIORITY_LOW)
                it.setPlayer(exoPlayer)
            }
    }

    @RequiresApi(Build.VERSION_CODES.O)
    private fun createNotificationChannel() {
        val channel = NotificationChannel(
            NOTIFICATION_CHANNEL_ID,
            NOTIFICATION_CHANNEL_NAME,
            NotificationManager.IMPORTANCE_LOW
        )
        notificationManager.createNotificationChannel(channel)
    }
}
```

ДОДАТОК Д

Код з файлу JetAudioServiceHandler.kt

```
package hoods.com.jetaudio.player.service
import androidx.media3.common.MediaItem
import androidx.media3.common.Player
import androidx.media3.exoplayer.ExoPlayer
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.Job
import kotlinx.coroutines.delay
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.launch
import javax.inject.Inject
class JetAudioServiceHandler @Inject constructor(
    private val exoPlayer: ExoPlayer,
) : Player.Listener {
    private val _audioState: MutableStateFlow<JetAudioState> =
        MutableStateFlow(JetAudioState.Initial)
    val audioState: StateFlow<JetAudioState> = _audioState.asStateFlow()
    private var job: Job? = null
    init {
        exoPlayer.addListener(this)
    }
    fun addMediaItem(mediaItem: MediaItem) {
        exoPlayer.setMediaItem(mediaItem)
        exoPlayer.prepare()
    }
    fun setMediaItemList(mediaItems: List<MediaItem>) {
        exoPlayer.setMediaItems(mediaItems)
        exoPlayer.prepare()
    }
    suspend fun onPlayerEvents(
        playerEvent: PlayerEvent,
        selectedAudioIndex: Int = -1,
        seekPosition: Long = 0,
    ) {
        when (playerEvent) {
            PlayerEvent.Backward -> exoPlayer.seekBack()
            PlayerEvent.Forward -> exoPlayer.seekForward()
            PlayerEvent.SeekToNext -> exoPlayer.seekToNext()
            PlayerEvent.PlayPause -> playOrPause()
            PlayerEvent.SeekTo -> exoPlayer.seekTo(seekPosition)
            PlayerEvent.SelectedAudioChange -> {
                when (selectedAudioIndex) {
                    exoPlayer.currentMediaItemIndex -> {
                        playOrPause()
                    }
                    else -> {
                        exoPlayer.seekToDefaultPosition(selectedAudioIndex)
                        _audioState.value = JetAudioState.Playing(
                            isPlaying = true
                        )
                        exoPlayer.playWhenReady = true
                        startProgressUpdate()
```

```
                    }
                }
            }
            PlayerEvent.Stop -> stopProgressUpdate()
            is PlayerEvent.UpdateProgress -> {
                exoPlayer.seekTo(
                    (exoPlayer.duration * playerEvent.newProgress).toLong()
                )
            }
        }
    }
    override fun onPlaybackStateChanged(playbackState: Int) {
        when (playbackState) {
            ExoPlayer.STATE_BUFFERING -> _audioState.value =
                JetAudioState.Buffering(exoPlayer.currentPosition)

            ExoPlayer.STATE_READY -> _audioState.value =
                JetAudioState.Ready(exoPlayer.duration) }}
    override fun onIsPlayingChanged(isPlaying: Boolean) {
        _audioState.value = JetAudioState.Playing(isPlaying = isPlaying)
        _audioState.value = JetAudioState.CurrentPlaying(exoPlayer.currentMediaItemIndex)
        if (isPlaying) {
            GlobalScope.launch(Dispatchers.Main) {
                startProgressUpdate()
            }
        } else {
            stopProgressUpdate()}}
    private suspend fun playOrPause() {
        if (exoPlayer.isPlaying) {
            exoPlayer.pause()
            stopProgressUpdate()
        } else {
            exoPlayer.play()
            _audioState.value = JetAudioState.Playing(
                isPlaying = true)
            startProgressUpdate()}}
    private suspend fun startProgressUpdate() = job.run {
        while (true) {
            delay(500)
            _audioState.value = JetAudioState.Progress(exoPlayer.currentPosition)}}
    private fun stopProgressUpdate() {
        job?.cancel()
        _audioState.value = JetAudioState.Playing(isPlaying = false)}}
sealed class PlayerEvent {
    object PlayPause : PlayerEvent()
    object SelectedAudioChange : PlayerEvent()
    object Backward : PlayerEvent()
    object SeekToNext : PlayerEvent()
    object Forward : PlayerEvent()
    object SeekTo : PlayerEvent()
    object Stop : PlayerEvent()
    data class UpdateProgress(val newProgress: Float) : PlayerEvent()}
sealed class JetAudioState {
    object Initial : JetAudioState()
    data class Ready(val duration: Long) : JetAudioState()
    data class Progress(val progress: Long) : JetAudioState()
    data class Buffering(val progress: Long) : JetAudioState()
    data class Playing(val isPlaying: Boolean) : JetAudioState()
    data class CurrentPlaying(val mediaItemIndex: Int) : JetAudioState()}
```

ДОДАТОК Е

Код з файлу AudioViewModel.kt

```
package hoods.com.jetaudio.ui.audio
import androidx.compose.runtime.mutableStateOf
import androidx.core.net.toUri
import androidx.lifecycle.SavedStateHandle
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import androidx.lifecycle.viewmodel.compose.saveable
import androidx.media3.common.MediaItem
import androidx.media3.common.MediaMetadata
import dagger.hilt.android.lifecycle.HiltViewModel
import hoods.com.jetaudio.data.local.model.Audio
import hoods.com.jetaudio.data.repository.AudioRepository
import hoods.com.jetaudio.player.service.JetAudioServiceHandler
import hoods.com.jetaudio.player.service.JetAudioState
import hoods.com.jetaudio.player.service.PlayerEvent
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.flow.collectLatest
import kotlinx.coroutines.launch
import java.util.concurrent.TimeUnit
import javax.inject.Inject
private val audioDummy = Audio("".toUri(), "", 0L, "", "", 0, "")
@HiltViewModel
class AudioViewModel @Inject constructor(
    private val audioServiceHandler: JetAudioServiceHandler,
    private val repository: AudioRepository,
    savedStateHandle: SavedStateHandle,
) : ViewModel() {
    var duration by savedStateHandle.saveable { mutableStateOf(0L) }
    var progress by savedStateHandle.saveable { mutableStateOf(0f) }
    var progressString by savedStateHandle.saveable { mutableStateOf("00:00") }
    var isPlaying by savedStateHandle.saveable { mutableStateOf(false) }
    var currentSelectedAudio by savedStateHandle.saveable { mutableStateOf(audioDummy) }
    var audioList by savedStateHandle.saveable { mutableStateOf(listOf<Audio>()) }
    private val _uiState: MutableStateFlow<UIState> = MutableStateFlow(UIState.Initial)
    val uiState: StateFlow<UIState> = _uiState.asStateFlow()
    init {loadAudioData()}
    init {
        viewModelScope.launch {
            audioServiceHandler.audioState.collectLatest { mediaState ->
                when (mediaState) {
                    JetAudioState.Initial -> _uiState.value = UIState.Initial
                    is                      JetAudioState.Buffering                      ->
calculateProgressValue(mediaState.progress)
                    is JetAudioState.Playing -> isPlaying = mediaState.isPlaying
                    is                       JetAudioState.Progress                      ->
calculateProgressValue(mediaState.progress)
                    is JetAudioState.CurrentPlaying -> {
                        currentSelectedAudio = audioList[mediaState.mediaItemIndex]}
                    is JetAudioState.Ready -> {
                        duration = mediaState.duration
                        _uiState.value = UIState.Ready}}}}}
    private fun loadAudioData() {
```

```
        viewModelScope.launch {
            val audio = repository.getAudioData()
            audioList = audio
            setMediaItems()}}
    private fun setMediaItems() {
        audioList.map { audio ->
            MediaItem.Builder()
                .setUri(audio.uri)
                .setMediaMetadata(
                    MediaMetadata.Builder()
                        .setAlbumArtist(audio.artist)
                        .setDisplayTitle(audio.title)
                        .setSubtitle(audio.displayName)
                        .build()).build()
        }.also {audioServiceHandler.setMediaItemList(it) }}
    private fun calculateProgressValue(currentProgress: Long) {
        progress = if (currentProgress > 0) ((currentProgress.toFloat() /
duration.toFloat()) * 100f)
            else 0f
        progressString = formatDuration(currentProgress) }
    fun onUiEvents(uiEvents: UIEvents) = viewModelScope.launch {
        when (uiEvents) {
            UIEvents.Backward -> audioServiceHandler.onPlayerEvents(PlayerEvent.Backward)
            UIEvents.Forward -> audioServiceHandler.onPlayerEvents(PlayerEvent.Forward)
            UIEvents.SeekToNext                                                    ->
audioServiceHandler.onPlayerEvents(PlayerEvent.SeekToNext)
            is UIEvents.PlayPause -> {audioServiceHandler.onPlayerEvents(
                    PlayerEvent.PlayPause)}
            is UIEvents.SeekTo -> {
                audioServiceHandler.onPlayerEvents(
                    PlayerEvent.SeekTo,
                    seekPosition = ((duration * uiEvents.position) / 100f).toLong())}
            is UIEvents.SelectedAudioChange -> {
                audioServiceHandler.onPlayerEvents(
                    PlayerEvent.SelectedAudioChange,
                    selectedAudioIndex = uiEvents.index)}
            is UIEvents.UpdateProgress -> {
                audioServiceHandler.onPlayerEvents(
                    PlayerEvent.UpdateProgress(
                        uiEvents.newProgress)) progress = uiEvents.newProgress}}}
    fun formatDuration(duration: Long): String {
        val minute = TimeUnit.MINUTES.convert(duration, TimeUnit.MILLISECONDS)
        val seconds = (minute) - minute * TimeUnit.SECONDS.convert(1, TimeUnit.MINUTES)
        return String.format("%02d:%02d", minute, seconds) }
    override fun onCleared() {
        viewModelScope.launch {
            audioServiceHandler.onPlayerEvents(PlayerEvent.Stop)}
        super.onCleared()}}
sealed class UIEvents {
    object PlayPause : UIEvents()
    data class SelectedAudioChange(val index: Int) : UIEvents()
    data class SeekTo(val position: Float) : UIEvents()
    object SeekToNext : UIEvents()
    object Backward : UIEvents()
    object Forward : UIEvents()
    data class UpdateProgress(val newProgress: Float) : UIEvents()}
sealed class UIState {
    object Initial : UIState()
    object Ready : UIState()}
```

ДОДАТОК Ж

Код з файлу Home.kt

```
package hoods.com.jetaudio.ui.audio
import androidx.compose.foundation.BorderStroke
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.itemsIndexed
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.MusicNote
import androidx.compose.material.icons.filled.Pause
import androidx.compose.material.icons.filled.PlayArrow
import androidx.compose.material.icons.filled.SkipNext
import androidx.compose.material3.BottomAppBar
import androidx.compose.material3.Card
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Slider
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.core.net.toUri
import hoods.com.jetaudio.data.local.model.Audio
import hoods.com.jetaudio.ui.theme.JetAudioTheme
import kotlin.math.floor

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun HomeScreen(
    progress: Float,
    onProgress: (Float) -> Unit,
    isAudioPlaying: Boolean,
    currentPlayingAudio: Audio,
    audiList: List<Audio>,
    onStart: () -> Unit,
    onItemClick: (Int) -> Unit,
```

```
    onNext: () -> Unit,
) {
    Scaffold(
        bottomBar = {
            BottomBarPlayer(
                progress = progress,
                onProgress = onProgress,
                audio = currentPlayingAudio,
                onStart = onStart,
                onNext = onNext,
                isAudioPlaying = isAudioPlaying
            )
        }
    ) {
        LazyColumn(
            contentPadding = it
        ) {
            itemsIndexed(audiList) { index, audio ->
                AudioItem(
                    audio = audio,
                    onItemClick = { onItemClick(index) }
                ) }}}}

@Composable
fun AudioItem(
    audio: Audio,
    onItemClick: () -> Unit,
) {
    Card(
        modifier = Modifier
            .fillMaxWidth()
            .padding(12.dp)
            .clickable {
                onItemClick()
            },
    ) {
       Row(
           verticalAlignment = Alignment.CenterVertically,
           modifier = Modifier.padding(8.dp)
       ) {
           Column(
               modifier = Modifier
                   .weight(1f)
                   .padding(8.dp),
               verticalArrangement = Arrangement.Center
           ) {
               Spacer(modifier = Modifier.size(4.dp))
               Text(
                   text = audio.displayName,
                   style = MaterialTheme.typography.titleLarge,
                   overflow = TextOverflow.Clip,
                   maxLines = 1
               )
               Spacer(modifier = Modifier.size(4.dp))
               Text(
                   text = audio.artist,
                   style = MaterialTheme.typography.bodySmall,
                   maxLines = 1,
                   overflow = TextOverflow.Clip
```

```
                )

            }
            Text(
                text = timeStampToDuration(audio.duration.toLong())
            )
            Spacer(modifier = Modifier.size(8.dp))
        }

    }
}

private fun timeStampToDuration(position: Long): String {
    val totalSecond = floor(position / 1E3).toInt()
    val minutes = totalSecond / 60
    val remainingSeconds = totalSecond - (minutes * 60)
    return if (position < 0) "--:--"
    else "%d:%02d".format(minutes, remainingSeconds)
}

@Composable
fun BottomBarPlayer(
    progress: Float,
    onProgress: (Float) -> Unit,
    audio: Audio,
    isAudioPlaying: Boolean,
    onStart: () -> Unit,
    onNext: () -> Unit,
) {
    BottomAppBar(
        content = {
            Column(
                modifier = Modifier.padding(8.dp)
            ) {
                Row(
                    modifier = Modifier
                        .fillMaxWidth()
                        .height(56.dp),
                    horizontalArrangement = Arrangement.SpaceBetween,
                    verticalAlignment = Alignment.CenterVertically
                ) {
                    ArtistInfo(
                        audio = audio,
                        modifier = Modifier.weight(1f),
                    )
                    MediaPlayerController(
                        isAudioPlaying = isAudioPlaying,
                        onStart = onStart,
                        onNext = onNext
                    )
                    Slider(
                        value = progress,
                        onValueChange = { onProgress(it) },
                        valueRange = 0f..100f
                    )

                }
            }
        }
```

```
    )
}

@Composable
fun MediaPlayerController(
    isAudioPlaying: Boolean,
    onStart: () -> Unit,
    onNext: () -> Unit,
) {
    Row(
        verticalAlignment = Alignment.CenterVertically,
        modifier = Modifier
            .height(56.dp)
            .padding(4.dp)
    ) {
        PlayerIconItem(
            icon = if (isAudioPlaying) Icons.Default.Pause
            else Icons.Default.PlayArrow
        ) {
            onStart()
        }
        Spacer(modifier = Modifier.size(8.dp))
        Icon(
            imageVector = Icons.Default.SkipNext,
            modifier = Modifier.clickable {
                onNext()
            },
            contentDescription = null
        )
    }
}

@Composable
fun ArtistInfo(
    modifier: Modifier = Modifier,
    audio: Audio,
) {
    Row(
        modifier = modifier.padding(4.dp),
        verticalAlignment = Alignment.CenterVertically
    ) {
        PlayerIconItem(
            icon = Icons.Default.MusicNote,
            borderStroke = BorderStroke(
                width = 1.dp,
                color = MaterialTheme.colorScheme.onSurface
            )
        ) {}
        Spacer(modifier = Modifier.size(4.dp))
        Column {
            Text(
                text = audio.title,
                fontWeight = FontWeight.Bold,
                style = MaterialTheme.typography.titleLarge,
                overflow = TextOverflow.Clip,
                modifier = Modifier.weight(1f),
                maxLines = 1
            )
            Spacer(modifier = Modifier.size(4.dp))
```

```
            Text(
                text = audio.artist,
                fontWeight = FontWeight.Normal,
                style = MaterialTheme.typography.bodySmall,
                overflow = TextOverflow.Clip,
                maxLines = 1
            )
        }
    }
}

@Composable
fun PlayerIconItem(
    modifier: Modifier = Modifier,
    icon: ImageVector,
    borderStroke: BorderStroke? = null,
    backgroundColor: Color = MaterialTheme.colorScheme.surface,
    color: Color = MaterialTheme.colorScheme.onSurface,
    onClick: () -> Unit,
) {
    Surface(
        shape = CircleShape,
        border = borderStroke,
        modifier = Modifier
            .clip(CircleShape)
            .clickable {
                onClick()
            },
        contentColor = color,
        color = backgroundColor
    ) {
        Box(
            modifier = Modifier.padding(4.dp),
            contentAlignment = Alignment.Center,
        ) {
            Icon(
                imageVector = icon,
                contentDescription = null
            )
        }
    }
}

@Preview(showSystemUi = true)
@Composable
fun HomeScreenPrev() {
    JetAudioTheme {
        HomeScreen(
            progress = 50f,
            onProgress = {},
            isAudioPlaying = true,
            audiList = listOf(
                Audio("".toUri(), "Title One", 0L, "Said", "", 0, "Title One"),
                Audio("".toUri(), "Title Two", 0L, "Unknown", "", 0, "Title two"),
            ),
            currentPlayingAudio = Audio("".toUri(), "Title One", 0L, "Said", "", 0, ""),
            onStart = {},
            onItemClick = {},
            onNext = {})}}
```

ДОДАТОК И

Код з файлу MediaModule.kt

```
package hoods.com.jetaudio.di
import android.content.Context
import androidx.media3.common.AudioAttributes
import androidx.media3.common.C
import androidx.media3.common.util.UnstableApi
import androidx.media3.exoplayer.ExoPlayer
import androidx.media3.exoplayer.trackselection.DefaultTrackSelector
import androidx.media3.session.MediaSession
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.android.qualifiers.ApplicationContext
import dagger.hilt.components.SingletonComponent
import hoods.com.jetaudio.player.notification.JetAudioNotificationManager
import hoods.com.jetaudio.player.service.JetAudioServiceHandler
import javax.inject.Singleton
@Module
@InstallIn(SingletonComponent::class)
object MediaModule {
    @Provides
    @Singleton
    fun provideAudioAttributes(): AudioAttributes = AudioAttributes.Builder()
        .setContentType(C.AUDIO_CONTENT_TYPE_MOVIE)
        .setUsage(C.USAGE_MEDIA)
        .build()
    @Provides
    @Singleton
    @UnstableApi
    fun provideExoPlayer(
        @ApplicationContext context: Context,
        audioAttributes: AudioAttributes,
    ): ExoPlayer = ExoPlayer.Builder(context)
        .setAudioAttributes(audioAttributes, true)
        .setHandleAudioBecomingNoisy(true)
        .setTrackSelector(DefaultTrackSelector(context))
        .build()
    @Provides
    @Singleton
    fun provideMediaSession(
        @ApplicationContext context: Context,
        player: ExoPlayer,
    ): MediaSession = MediaSession.Builder(context, player).build()
    @Provides
    @Singleton
    fun provideNotificationManager(
        @ApplicationContext context: Context,
        player: ExoPlayer,
    ): JetAudioNotificationManager = JetAudioNotificationManager(
        context = context,
        exoPlayer = player)
    @Provides
    @Singleton
    fun provideServiceHandler(exoPlayer: ExoPlayer): JetAudioServiceHandler =
        JetAudioServiceHandler(exoPlayer)}
```