МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
01 червня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавр

зі спеціальності 122 – Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Персоналізований інтелектуальний музичний додаток»
здобувача групи ІН-03 Картавенка Едуарда Анатолійовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____Едуард КАРТАВЕНКО
(підпис)

Керівник,
старший викладач кафедри
комп'ютерних наук,
Кандидат технічних наук Артем КОРОБОВ _____
(підпис)

Суми – 2024

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«Затверджую»

В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-03 Картавенка Едуарда Анатолійовича

1. Тема роботи: «Персоналізований інтелектуальний музичний додаток» затверджую наказом по СумДУ від *«22» квітня 2024 р. № 0414-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 29 травня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми та актуальності розробки музичного додатку, конкурентоспроможності та цільової аудиторії, постановка й формування завдань дослідження. 2) Проектування системи, розробка дизайну, діаграми класів та структури бази даних 3) Огляд та вибір засобів програмної реалізації 4) Розробка інформаційної системи музичного додатку 5) Аналіз отриманих результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
6. Консультанти до проекту (роботи), із зазначенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «__» _____ 20__ р.

Завдання прийняв на виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, конкурентоспроможності, постановка й формування завдань дослідження*	06.05.2024 - 07.05.2024	
2	*Проектування системи, розробка дизайну, діаграми класів та структури бази даних*	07.05.2024 – 09.05.2024	
3	*Огляд засобів програмної реалізації*	09.05.2024 –	

		10.05.2024	
4	*Розробка інформаційної системи музичного додатку*	10.05.2024 - 17.05.2024	
5	*Аналіз отриманих результатів*	17.05.2024 – 18.05.2024	
6	*Оформлення пояснювальної записки до кваліфікаційної роботи*	18.05.2024 – 20.05.2024	

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 78 стор., 32 рис., 0 табл., 2 додатки, 12 використаних джерел.

Обґрунтування актуальності теми роботи – розробка персоналізованих музичних додатків є актуальною не лише з технологічної, але й з соціальної, культурної та економічної точок зору. Це підтверджується сучасними тенденціями на ринку та попитом серед користувачів, що робить цю тему вартою уваги та дослідження.

Об’єкт дослідження - персоналізований музичний додаток, який аналізує користувачів і надає індивідуально підібрані музичні рекомендації на основі їхніх вподобань.

Мета роботи – дослідження, аналіз та розробка персоналізованого музичного додатка, який надає користувачам індивідуально підібрані музичні рекомендації.

Методи дослідження - аналіз існуючих музичних додатків, інструменти проектування та розробки.

Результати - розроблено інформаційну систему, яка ефективно надає індивідуально підібрані музичні рекомендації, забезпечуючи зручний інтерфейс для користувачів та можливість прослуховування музики в інтерактивному форматі.

МУЗИЧНИЙ ДОДАТОК, АДАПТИВНИЙ ДИЗАЙН, ІНФОРМАЦІЙНА СИСТЕМА, KOTLIN, XML, SQLITE

ЗМІСТ

ВСТУП

Музика відіграє важливу роль у сучасному суспільстві і є не лише джерелом задоволення, а й засобом спілкування та самовираження. Музика здатна впливати на емоції, настрій і навіть фізіологічний стан людей. Музика – це універсальна мова, яка дає змогу людям із різних культур знаходити спільну мову. Вона також відіграє важливу роль у розвитку особистості, особливо в молодого покоління, впливаючи на їхній емоційний, естетичний і духовний розвиток.

Музичні додатки стали невід'ємною частиною повсякденного життя багатьох людей у сучасному світі. Дані з дослідження музичних вподобань українців у 2023 році показують що 85% опитаних слухають музику майже щодня, а у середньому, люди витрачають 105 хвилин на день на прослуховування музики[1]. Вони забезпечують зручний доступ до величезної кількості музики і дають змогу користувачам відкривати для себе нові жанри та виконавців, а також створювати персональні плейлисти. Ці додатки також сприяють музичній освіті, надаючи інструменти для навчання гри на інструментах, співу та теорії музики.

Завдяки музичним додаткам люди можуть слухати музику в будь-який час і в будь-якому місці, що робить її більш доступною та інтегрованою в повсякденне життя. Це також відкрило музикантам можливість ділитися своєю творчістю з аудиторією по всьому світу, незалежно від їхнього географічного положення.

Мета роботи

Робота присвячена розробці персоналізованого інтелектуального музичного додатку що надасть зручний доступ до прослуховування музичних треків, персоналізовані рекомендації та багато інших функцій, що покращують музичний досвід користувача.

Персоналізований інтелектуальний музичний додаток — це програма, яка дозволяє користувачам слухати музику, створювати власні плейлисти, використовувати рекомендації та взаємодіяти з інтуїтивно зрозумілим інтерфейсом.

У цьому звіті буде розглянуто основні етапи розробки додатку. Спочатку я проведу аналітичний огляд, щоб визначити вимоги до програми , згідно з якими можна визначити приблизні об’єми роботи та вміло розподілити час що буде витрачений на тей чи інший модуль роботи, та проаналізувати її конкурентоспроможність. Далі, я розгляну проектування системи, включаючи дизайн інтерфейсів для різних частин додатка та структури системи. У завершальному розділі я розгляну інформаційне та програмне забезпечення системи, що включає вибір технологій та опис програмної реалізації.

У процесі розробки персоналізованого інтелектуального музичного додатка ми будемо враховувати такі ключові аспекти:

- **Аналітичний огляд:** на цьому етапі я визначу основні вимоги до програми, враховуючи потреби користувачів та цільову аудиторію. Аналіз конкурентоспроможності допоможе визначити унікальні особливості, які виділяють наш додаток на тлі інших музичних програм.
- **Проектування системи:** проектування інтерфейсу є критично важливим для створення зручного користувацького досвіду. Я розроблю інтерфейс для вікна завантаження, основного вікна та решти вікон що можуть знадобитись користувачу при користуванні додатком. Також буде розроблено діаграму класів, яка відобразить структуру додатку.
- **Інформаційне та програмне забезпечення системи:** на цьому етапі я розповім про те які саме інструменти та технології для розробки додатків були обрані. Буде описано програмну реалізацію, з акцентом на важливі деталі, які забезпечують ефективність та стабільність додатку.

1. АНАЛІТИЧНИЙ ОГЛЯД

1.1 Основні етапи створення музичного додатку

Музичні додатки відіграють важливу роль у сучасному суспільстві, оскільки вони надають користувачам необмежений доступ до музики, дозволяючи відкривати для себе нові жанри та виконавців і створювати власні плейлисти. Вони роблять музику більш доступною та інтегрованою в повсякденне життя, дозволяючи людям слухати музику будь-де та будь-коли. І задля того щоб розробити такий додаток потрібно керуватися такими етапами:

- **Етап визначення ідеї та планування** – перше що потрібно зробити це звісно визначити тематику додатку, адже додатки можуть бути найрізноманітнішими, такі як - розважальні, бізнес додатки, освітні додатки та ін. Також одним з перших кроків в розробці є планування графіку, визначення дедлайнів, та визначення самого покрокового плану розробки.
- **Етап проектування** – це другий, але не менш важливий крок у створенні мобільного додатку. Саме на цьому етапі закладається фундамент, на якому буде будуватися весь проект. Тут визначається не лише те, як буде виглядати ваш додаток, але й те, як він буде працювати. В нього входить розробка дизайну додатку, визначення та розробка архітектури та технічних характеристик.
- **Етап розробки** – це третій, ключовий етап створення мобільного додатку, де задуми та ідеї втілюються в життя. Саме на цьому етапі пишеться код, створюються функції та модулі, які будуть забезпечувати роботу вашого додатку.
- **Етап тестування** – це один з найважливіших етапів створення мобільного додатку, адже саме тут визначається, наскільки він готовий до випуску та публікації. На цьому етапі виявляються та виправляються помилки, тестується продуктивність та загальна якість роботи додатку.

1.2 Аналіз вимог

Мобільні додатки стали невід'ємною частиною нашого повсякденного життя, пропонуючи рішення для майже будь-якої потреби або бажання. Від соціальних мереж до фінансових сервісів, від ігор до освітніх платформ.

Користувачів приваблює зручність та доступність, яку надають мобільні додатки. Вони цінують додатки, які пропонують контрастність елементів, очевидну навігацію, не навантажений елементами інтерфейс[2], швидкість, безпеку та персоналізацію.

Отже можна чітко визначити вимоги до музичного додатку, а саме:

Інтерфейс користувача - повинен мати інтуїтивно зрозумілий інтерфейс, який буде забезпечувати легку навігацію серед функціоналу музичного додатку, також правильне позиціювання, і контрастність елементів для зручної взаємодії між користувачем, основний текст буде відображений англійською мовою, адже вона є міжнародною і досить інтуїтивною.

Персоналізація – додаток має містити принаймні дві теми для персоналізації інтерфейсу, а саме світлу та темну. Також систему рекомендацій, яка аналізує вподобання користувачів та надає індивідуалізовані пропозиції.

Функції для прослуховування – додаток має містити справний функціонал для прослуховування музики, зокрема такі функції, як зчитування треків з файлової системи телефону, зупинка/відтворення, перемотування до наступного та попереднього треку, а також відтворення треків, коли телефон вимкнений або коли користувач перебуває поза межами додатку.

Функція створення плейлистів – мати можливість для користувачів створювати, зберігати та управляти власними плейлистами.

Адаптивність - додаток має бути сумісним з різними пристроями на базі android, також повинен бути оптимізований та мати чіткий вигляд на

смартфонах, які мають діагоналі екранів, що користуються найбільшою популярністю.

Безпека – додаток може мати доступ до файлів, що знаходяться в телефоні лише з дозволу користувача, що надається додатку при підтвердженні запиту на дозвіл.

Ці пункти допоможуть зрозуміти проблематику вимог які потрібно реалізувати задля того щоб додаток був не тільки функціональним, але й зручним, безпечним та привабливим для користувачів, покращуючи їх музичний досвід.

1.3 Аналіз конкурентоспроможності

Аналіз конкурентоспроможності є ключовим елементом для успіху будь-якого додатку. Цей аналіз дозволяє розробникам зрозуміти ринок, визначити сильні та слабкі сторони конкурентів, а також виявити потенційні можливості для свого продукту. Для аналізу конкурентоспроможності було обрано наступні популярні музичні додатки:

- Pulsar
- Lark Player
- BlackPlayer

Pulsar

Pulsar Music Player - є одним із найкращих музичних програвачів на Android. У цьому офлайн-аудіоплеєрі немає реклами, також Pulsar містить майже всі необхідні функції для задоволення ваших музичних потреб, включаючи: відтворення без інтервалів, відображення текстів пісень, перехресне згасання, регулювання швидкості відтворення, управління еквалайзером, та ін. Приклад вигляду додатку можна побачити на рис 1.3.1(фото були взяті зі сторінки додатку в google play, посилання: https://play.google.com/store/apps/details?id=com.rhmsoft.pulsar).

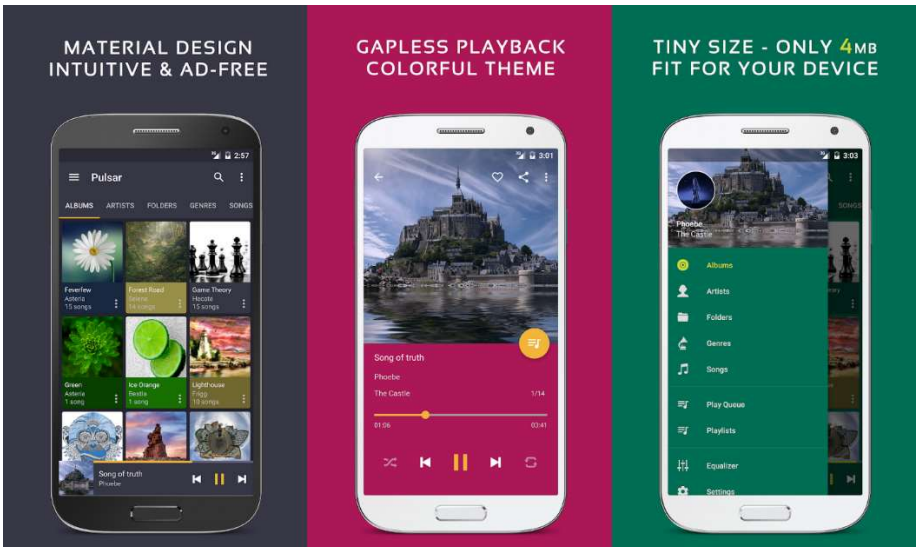


Рис 1.3.1 – Інтерфейс та функції додатку Pulsar

З основних переваг Pulsar має:

- **Дизайн та інтерфейс** - кожна окрема деталь його інтерфейсу відповідає принципам Material Design[3]. Загалом інтерфейс є інтуїтивно зрозумілим, не навантаженим різними елементами. Альбоми, виконавці, треки, папки систематизовано таким чином, щоб не було важко знайти ту, чи іншу композицію.
- **Еквалайзер** – згідно зі своїми вподобаннями можна налаштувати частоти звучання, еквалайзер має підсилання басу та тонкомпенсатор. Для налаштувань еквалайзеру можна зробити свій пресет, або вибрати з запропонованих. Проте важливо підмітити, що керування еквалайзером присутнє лише в платній версії додатку.
- **Персоналізація** – додаток містить декілька варіантів тем, а також надає змогу користувачу обирати те, які вкладки буде містити головна сторінка, що дає можливість користувачам налаштувати додаток під свої індивідуальні потреби та вподобання, роблячи його більш персональним.
- **Таймер сну** – це функція, яку можна налаштувати таким чином, щоб через вказаний користувачем час відтворення музики переривалось. Досить корисна функція, адже вона допомагає користувачам заснути, при цьому зупиняючи відтворення музики, що запобігає впливу шуму уві сні, та економить заряд смартфону.[4]

До недоліків можна віднести:

- **Немає підтримки Gapless playback -** Pulsar не підтримує безперервне відтворення, що може бути проблемою для деяких користувачів.
- **Відсутність підтримки синхронізації хмарного сховища -** Pulsar не підтримує синхронізацію музичної бібліотеки з хмарними службами зберігання.
- **Еквалайзер лише в платній версії додатку –** це важливий інструмент для меломанів, який дозволяє їм налаштувати звучання музики

відповідно до своїх уподобань. Проте, як я зазначав раніше, він доступний лише в платній версії додатку.

Отже Pulsar – це чудовий вибір для користувачів Android, який вирізняється своїм елегантним дизайном, потужними функціями та широкими можливостями налаштування.

Lark Player

Lark Player – це безкоштовний музичний програвач(рис 1.3.2) для Android, який пропонує користувачам зручний спосіб відтворювати музику та відеофайли на своїх пристроях (фото були взяті зі сторінки додатку в google play, посилання: https://play.google.com/store/apps/details?id=com.dywx.larkplayer&hl=en&gl=US).

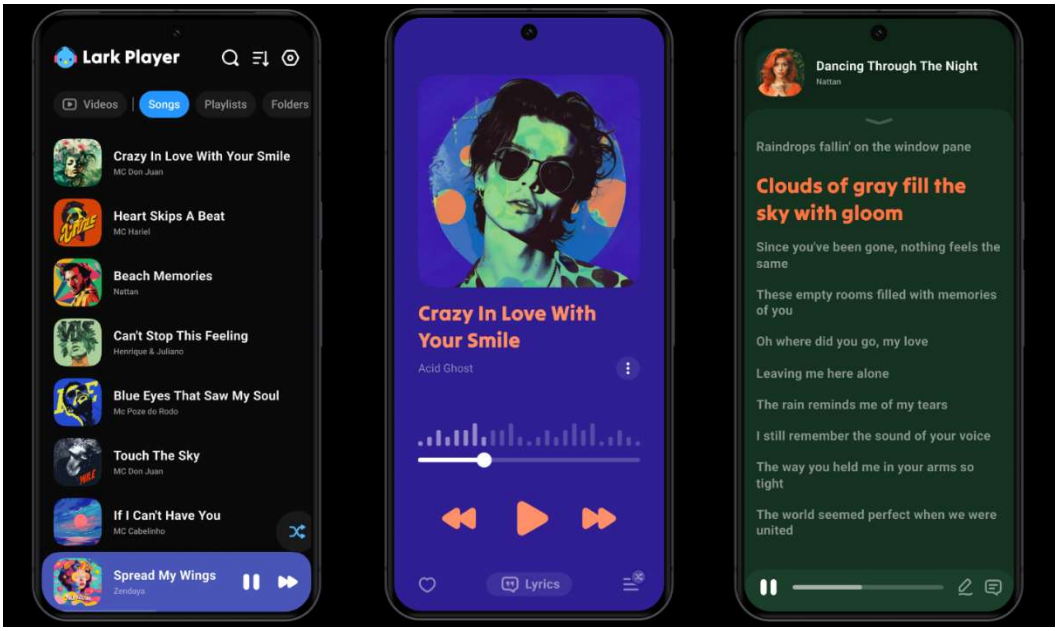


Рис 2.3.2 – Інтерфейс та функції додатку Lark Player

Ключовими особливостями Lark Player є:

- **Підтримка офлайн-відтворення** – одна з головних переваг Lark Player – можливість зберігати музичні файли на своєму пристрої та відтворювати їх без підключення до інтернету. Це корисно для ситуацій, коли у вас немає доступу до мобільного інтернету або Wi-Fi.

- **Підтримка багатьох форматів** – Lark Player підтримує відтворення різних форматів аудіофайлів, включаючи MP3, M4A, FLAC, WAV та інші. Він також може відтворювати відеофайли у форматах MP4, AVI, MKV тощо.
- **Еквалайзер** – Lark Player має вбудований еквалайзер, який дозволяє користувачам налаштувати звучання музики відповідно до своїх уподобань. Ця функція доступна безкоштовно.
- **Плейлисти** – користувачі можуть створювати та редагувати власні плейлисти, щоб легко отримувати доступ до улюбленої музики.
- **Тексти пісень** – деякі версії Lark Player можуть відображати тексти пісень під час відтворення музики (залежно від регіону та моделі телефону).
- **Плаваючий програвач** – Lark Player пропонує функцію плаваючого програвача, яка дозволяє користувачам зменшити програвач і продовжувати керувати відтворенням музики з будь-якого місця на екрані за рахунок плаваючого острівця.
- **Багатомовна підтримка** – Lark Player підтримує декілька мов, що робить його доступним для користувачів з різних країн.[4]

Недоліки Lark Player:

- **Реклама** – безкоштовна версія Lark Player містить рекламу, яка може дратувати деяких користувачів.
- **Проблеми з конфіденційністю** – деякі огляди Lark Player висловлюють занепокоєння щодо його політики конфіденційності та збору даних користувачів.

Загалом, Lark Player – це простий та безкоштовний музичний плеєр для Android, який пропонує базові функції для відтворення музики та відео в режимі офлайн. Однак користувачам слід звернути увагу на рекламу в безкоштовній версії та потенційні проблеми з конфіденційністю.

BlackPlayer

BlackPlayer – це безкоштовний музичний плеєр(рис 1.3.3) для Android, який вирізняється своїм елегантним дизайном та потужними функціями такі як 10-смуговий еквалайзер, Смарт-плейлисти, підтримка Chromecast, та ін(фото були взяті зі сторінки додатку в google play, посилання: https://play.google.com/store/apps/details?id=com.musicplayer.blackplayerfree&hl=en&gl=US).

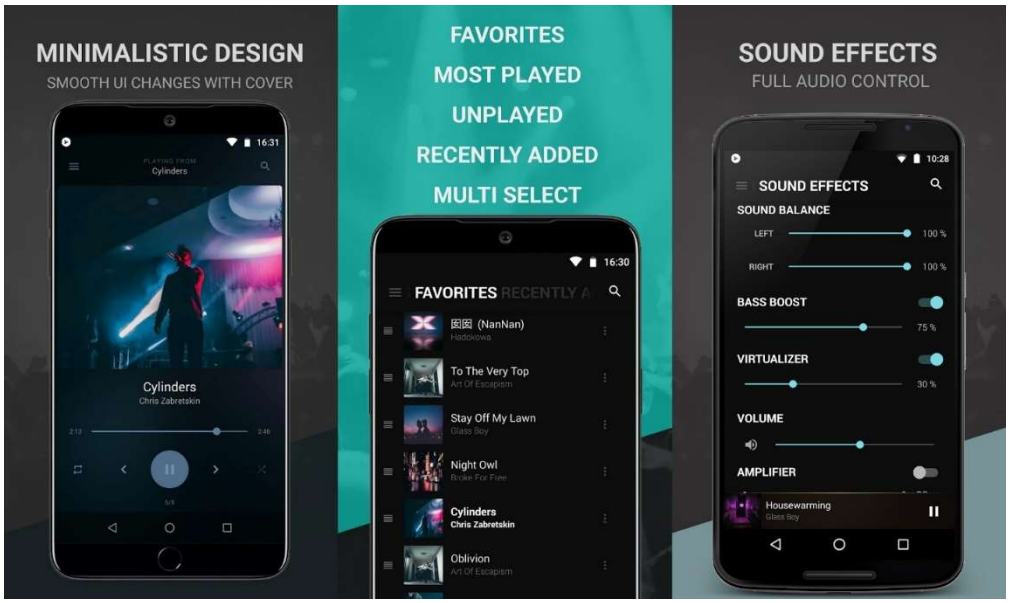


Рис 3.3.3 – Інтерфейс та функції додатку BlackPlayer

Ось деякі ключові особливості BlackPlayer:

- **10-смуговий еквалайзер** – BlackPlayer має вбудований 10-смуговий еквалайзер з підтримкою басового підсилювача та віртуального об'ємного звучання.
- **Gapless playback** – BlackPlayer забезпечує плавні переходи між треками, щоб ваша музика звучала безперервно.
- **Смарт-плейлисти** – BlackPlayer може автоматично створювати смарт-плейлисти на основі ваших уподобань, таких як нещодавно відтворені треки, найпопулярніші треки або треки з певного жанру.
- **Підтримка Chromecast та інших пристроїв** – можливість використовувати BlackPlayer для відтворення музики на Chromecast, розумних динаміках та інших підключених до Інтернету пристроях.

- **Віджети та сповіщення** – в BlackPlayer є різні віджети та сповіщення, які дозволяють вам керувати музикою з головного екрану або панелі сповіщення.
- **Широкі можливості налаштування** – BlackPlayer пропонує безліч налаштувань, що дозволяють персоналізувати інтерфейс та функції програми відповідно до ваших уподобань.
- **Таймер сну** – BlackPlayer має вбудований таймер сну, який дозволяє вам автоматично вимикати відтворення музики після закінчення встановленого часу.
- **Підтримка текстових пісень** – BlackPlayer може відображати тексти пісень під час відтворення музики (залежно від наявності).
- **Елегантний та інтуїтивно зрозумілий інтерфейс** – BlackPlayer має елегантний та інтуїтивно зрозумілий інтерфейс, який полегшує навігацію та використання.[4]

Із недоліків можу назвати:

- **Немає можливості синхронізації з хмарними сервісами** – BlackPlayer не підтримує синхронізацію вашої музичної бібліотеки з хмарними службами зберігання.
- **Реклама** – У безкоштовній версії BlackPlayer міститься реклама, яка може дратувати деяких користувачів.

Можна зазначити що BlackPlayer – гарний додаток, що має інтуїтивно зрозумілий та лаконічний інтерфейс, а також пропонує досить цікавий функціонал, відмінний від попередніх двох плеєрів.

Згідно з аналізом популярних плеєрів, що були зазначені вище, можна виокремити те, що задля того щоб заохотити користувачів, плеєр повинен мати гарний, зручний та інтуїтивний дизайн, можливість персоналізації додатку та стабільність роботи функцій, що пропонує додаток.

2. ВИБІР МЕТОДУ ВИРІШЕННЯ ЗАДАЧІ

2.1 Вибір засобів програмної реалізації

Вибір засобів програмної реалізації визначає основу проекту, забезпечуючи його технічну основу та визначаючи можливості і обмеження кінцевого продукту. У моєму випадку для розробки музичного додатку були обрані такі інструменти: Android Studio(версія - Iguana 2023.2.1), Kotlin, XML, Figma та SQLite. Кожен із них має свої переваги, які роблять їх ідеальними для реалізації цього проекту.

Середовище розробки

Android Studio[5] є офіційним інтегрованим середовищем розробки (IDE) для створення додатків на платформі Android. Вона пропонує широкий набір інструментів для розробників, включаючи вбудовані засоби для налагодження, тестування та аналізу коду. Завдяки своїй інтеграції з Gradle, Android Studio спрощує управління залежностями і процесом збірки проекту.

Рис 2.1.1 – Середовище розробки Android Studio

Основними перевагами Android Studio є:

- **Швидке програмування та швидка ітерація -** Android Studio базується на IntelliJ IDEA, що дозволяє швидше завершувати код та миттєво оцінювати робочий процес. Ви можете вносити зміни в код, не перезапускаючи додаток повністю, що забезпечує гнучкість при внесенні невеликих змін, коли додаток вже працює.
- **Швидкий та функціональний емулятор -** Android Studio має емулятор, який допомагає запускати додаток швидше, ніж на реальному пристрої. Емулятор може симулювати різні апаратні можливості, такі як GPS, багатократне натискання, датчики руху та прискорення тощо.
- **Редактор макетів -** Android Studio надає візуальний редактор для роботи з XML-файлами, що дозволяє швидко створювати та редагувати макети додатків.[6]
- **Розширені засоби налагодження та тестування -** Android Studio містить потужні засоби налагодження, включаючи інспектор макетів, аналізатор пам'яті та продуктивності, а також інтегровані емулятори для тестування додатків на різних версіях Android та різних пристроях. Це допомагає знаходити і виправляти помилки на ранніх етапах розробки.
- **Спільнота та підтримка -** Android Studio має велику та активну спільноту розробників, що забезпечує широкий доступ до ресурсів, прикладів коду, підручників та документації. Це робить процес навчання та вирішення проблем легшим і швидшим.

Мова програмування

Kotlin обрано як основну мову програмування для цього проекту. Kotlin є офіційно підтримуваною мовою для розробки Android-додатків і має ряд переваг над Java, включаючи більш лаконічний синтаксис, підвищену безпеку типів і кращу сумісність з сучасними мовними функціями. Це дозволяє розробляти більш надійний і зрозумілий код, що сприяє швидшому і безпомилковому розвитку додатку.

Рис 2.1.2 – Мова програмування Kotlin

Основними перевагами програмування на Kotlin є:

- **Лаконічність -** Kotlin є лаконічним, заощаджуючи час, який ви витратили б на написання шаблонного коду на Java.
- **Кросплатформеність -** Kotlin може працювати на будь-якій платформі, де використовується Java, що дозволяє створювати кросплатформенні додатки.
- **Обробка нульових значень -** нуль у Java може призвести до збою програми, якщо ви до цього не підготувалися, натомість у Kotlin ви можете додати простий оператор до змінних, які можуть бути нульовими, щоб запобігти цим збоям.[7]
- **Простота навчання -** Kotlin має простий і зрозумілий синтаксис, що полегшує його вивчення для розробників, які вже знайомі з Java або іншими мовами програмування. Це сприяє швидкому освоєнню мови і продуктивному переходу до її використання.

Розмітка Інтерфейсу (XML)

XML використовується для опису інтерфейсу користувача додатку. Використання XML дозволяє чітко структурувати компоненти UI[8] і забезпечує легкість їх модифікації та налаштування. Завдяки розділенню логіки додатку та його інтерфейсу, розробники можуть одночасно працювати над функціональністю та дизайном, що пришвидшує процес розробки.

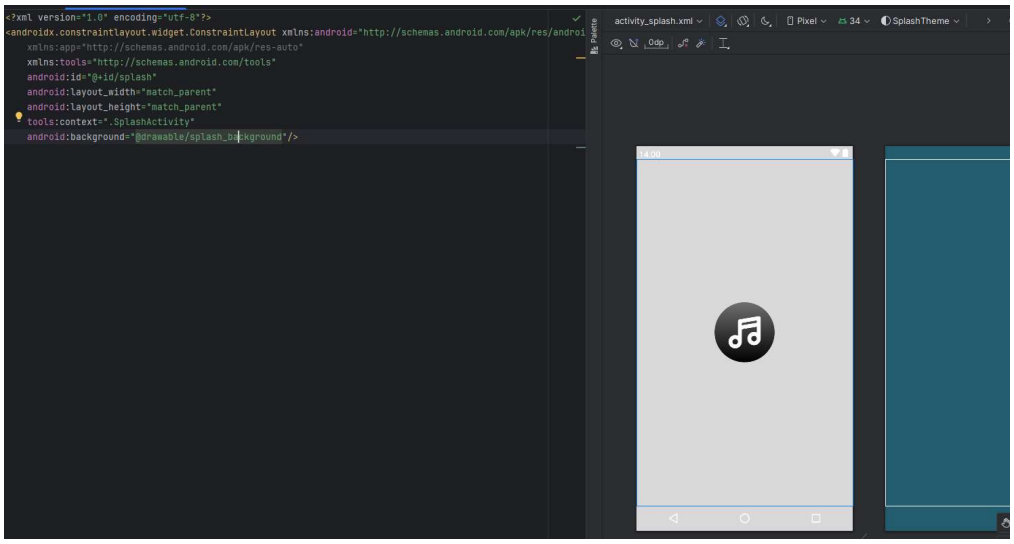


Рис 2.1.3 – XML, розмітка користувацького інтерфейсу

Переваги використання XML-розмітки:

- **Читабельність -** XML-файли легко читати та редагувати.
- **Розділення логіки та дизайну -** Розмітка дозволяє відокремити логіку додатка від його візуального представлення.
- **Підтримка різних пристроїв -** Завдяки розмітці можна створювати адаптивний інтерфейс для різних розмірів екранів.

SQLite

SQLite[9] вибрано як базу даних для цього додатку завдяки її легкості та інтегрованості в Android. SQLite є потужним, але легким рішенням для зберігання даних на пристрої, що дозволяє додатку працювати автономно без необхідності постійного з'єднання з інтернетом. Це особливо важливо для музичних додатків, де зберігання локальних даних, таких як списки відтворення та налаштування користувача, покращує користувацький досвід.

Рис 2.1.4 – СКБД SQLite

Переваги використання SQLite включають:

- **Простота використання** - це легка вбудовувана база даних, яка не вимагає налаштування сервера або складних конфігурацій. Її можна легко використовувати без значного досвіду в адмініструванні баз даних.
- **Низькі вимоги до ресурсів** - SQLite відома своєю ефективністю та низькими вимогами до ресурсів. Вона працює швидко навіть на пристроях з обмеженими ресурсами, таких як мобільні телефони та вбудовані системи.
- **Портативність** - SQLite підтримується на різних операційних системах, включаючи Windows, macOS, Linux та інші. Це робить її ідеальним вибором для розробки крос-платформових додатків.
- **Безпека та надійність** - SQLite має вбудовану підтримку для транзакцій, які забезпечують консистентність даних. Вона також має механізми для захисту вразливих місць безпеки.

Інструменти для Дизайну (Figma)

Для розробки дизайну інтерфейсу використовується Figma[10], онлайн-платформа для дизайну. Figma дозволяє створювати інтерактивні макети, прототипи та логотипи. А зберігати елементи можна в форматах jpg, svg, pdf, png.

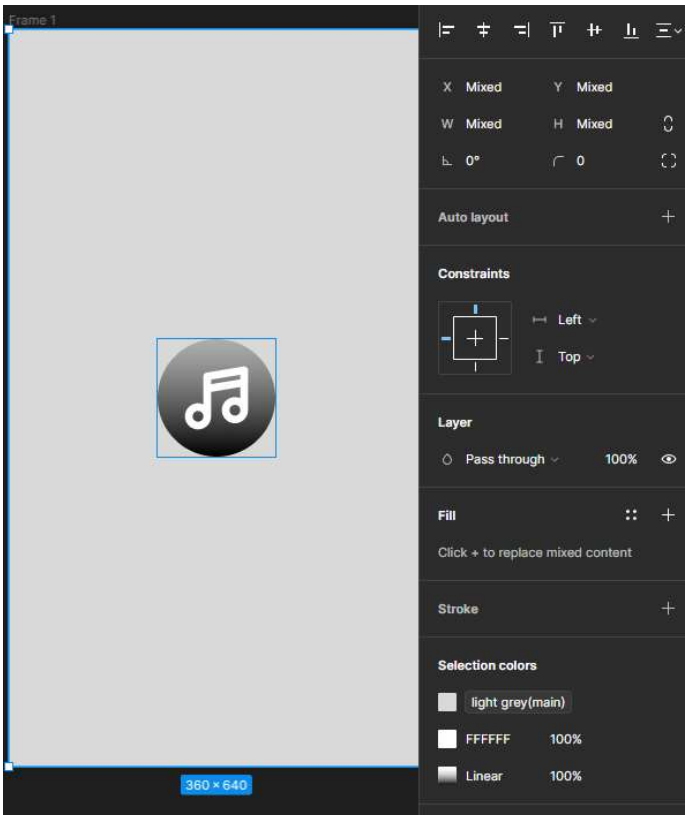


Рис 2.1.5 – Інтерфейс програми figma

Figma має такі переваги:

- **Векторна основа -** Figma працює з векторними зображеннями, що дозволяє зберігати чіткість та різкість дизайну на будь-якому роздільніку екрану.
- **Підтримка компонентів та стилів -** Figma дозволяє створювати та підтримувати однорідний дизайн інтерфейсу між проектами завдяки компонентам та стилям.
- **Плагіни та інтеграції -** Figma підтримує велику кількість плагінів, які розширюють її функціональні можливості. Це включає інструменти для автоматизації задач, імпорту та експорту даних, аналізу та інших функцій, що підвищують ефективність роботи.
- **Хмарне зберігання та доступність -** Figma працює в хмарі, що означає, що всі ваші проекти зберігаються онлайн. Це забезпечує легкий доступ до файлів з будь-якого пристрою та в будь-який час, без необхідності встановлення додаткового програмного забезпечення.

2.2 Проектування структур системи

Діаграма класів – це статичне представлення структури моделі в UML, вона відображає статичні (декларативні) елементи, такі як класи, типи даних, їх зміст та відношення.[11] На діаграмі класів показують класи, інтерфейси, об’єкти й кооперації, а також їхні відносини (зв’язки). Класи представленні у вигляді прямокутників, які містять:

- Назву класу.
- Атрибути - описують властивості класу.
- Операції - показують, які дії може виконувати клас.

Для побудови діаграми класів скористаємось застосунком draw.io.(Повна діаграма класів зазначена в Додатку 1)

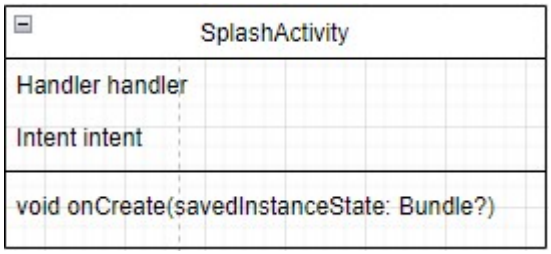


Рис 2.2.1 – Вікно SplashActivity

Клас SplashActivity реалізує класичний екран-заставку, який з'являється при запуску програми і через короткий проміжок часу передає управління основній активності (MainActivity). Він використовує корутини для асинхронної затримки і забезпечує плавний перехід між активностями, зберігаючи чуйність користувацького інтерфейсу.

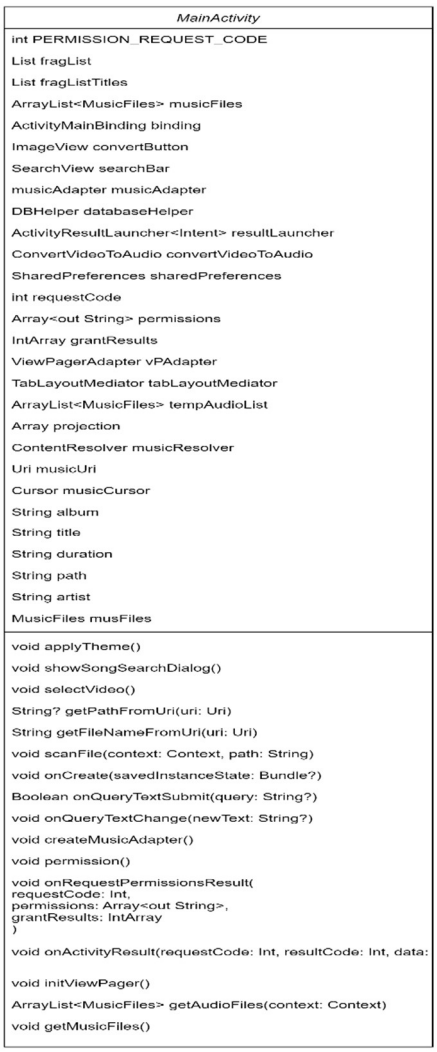


Рис 2.2.2 – Вікно MainActivity

Клас MainActivity - відповідає за основну логіку роботи додатку, включаючи управління правами доступу, відображення музичних файлів, їх пошук та конвертацію відео в аудіо. Він інтегрує роботу з фрагментами, базою даних та забезпечує зручну взаємодію користувача з додатком.

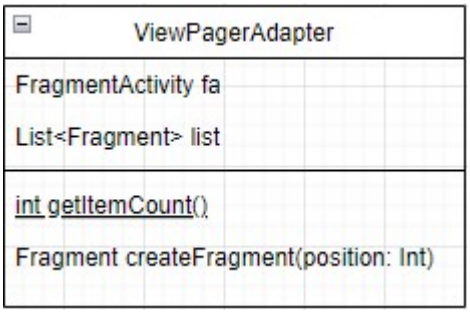


Рис 2.2.3 – Клас ViewPagerAdapter

Клас-адаптер ViewPagerAdapter - використовується для відображення фрагментів при переміщенні між вкладками або екранними областями у додатку. Адаптер ViewPagerAdapter розширює клас FragmentStateAdapter, який є частиною бібліотеки AndroidX, що забезпечує ефективне та гнучке використання фрагментів у ViewPager2.

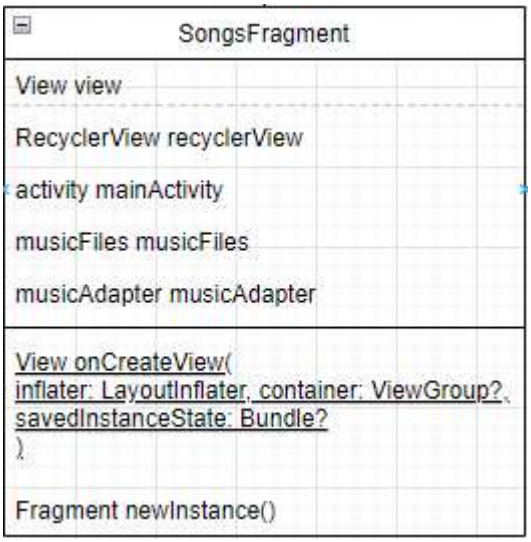


Рис 2.2.4 – Клас SongsFragment

Клас SongsFragment – є одним з фрагментів, котрий відображає клас ViewPagerAdapter. Цей клас відображає список пісень у RecyclerView. Він ініціалізує RecyclerView, отримує список пісень із MainActivity, і встановлює адаптер MusicAdapter. Це забезпечує структуру для відображення списку треків у музичному додатку.

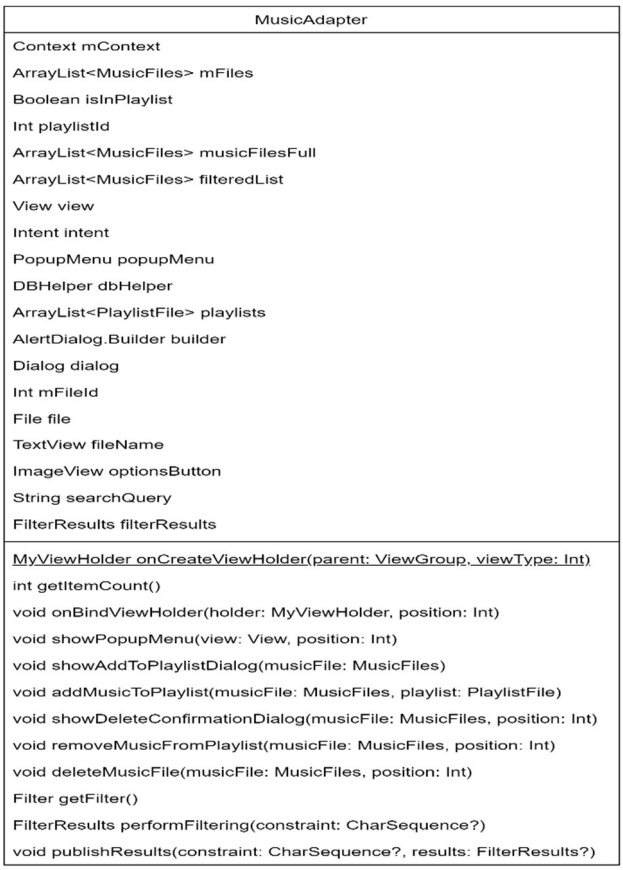


Рис 2.2.5 – Клас MusicAdapter

Клас MusicAdapter є адаптером для відображення списку музичних файлів у RecyclerView. Він забезпечує відображення музичних елементів і управління взаємодією з ними, включаючи можливості відтворення, видалення та додавання до плейлиста.

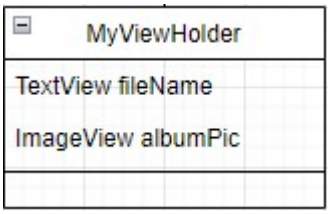


Рис 2.2.6 – Клас MyViewHolder

Клас MyViewHolder є частиною структури RecyclerView, який зберігає посилання на елементи інтерфейсу в одному елементі RecyclerView. Він використовується в адаптері для прив'язування даних та взаємодії з користувачем.

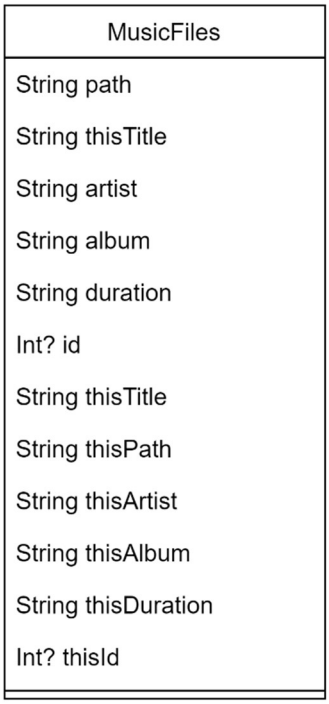


Рис 2.2.7 – Клас MusicFiles

Клас MusicFiles представляє дані про музичні файли або треки. Він містить основні атрибути, такі як назва, шлях, виконавець, альбом та тривалість. Використовуючи цей клас, музичний додаток може зберігати та обробляти інформацію про музичні файли, а також передавати ці дані між різними компонентами.

Рис 2.2.8 – Клас PlaylistFragment

Фрагмент PlaylistFragment є частиною додатку, котрий відображає клас ViewPagerAdapter. Цей клас дозволяє користувачам управляти своїми плейлистами: додавати нові, переглядати деталі існуючих та видаляти непотрібні плейлисти. Він інтегрує роботу з базою даних через DBHelper і відображає дані за допомогою RecyclerView та PlaylistAdapter.

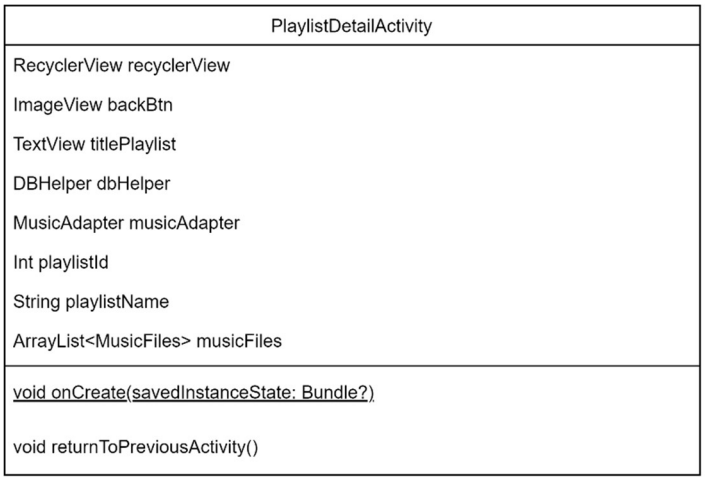


Рис 2.2.9 – Клас PlaylistDetailActivity

Активність PlaylistDetailActivity використовується для перегляду музичних файлів у конкретному плейлисті. Вона дозволяє користувачам бачити список музичних файлів, що входять до плейлиста, та взаємодіяти з ними. Користувач може повернутися до попередньої активності, натиснувши на кнопку повернення.

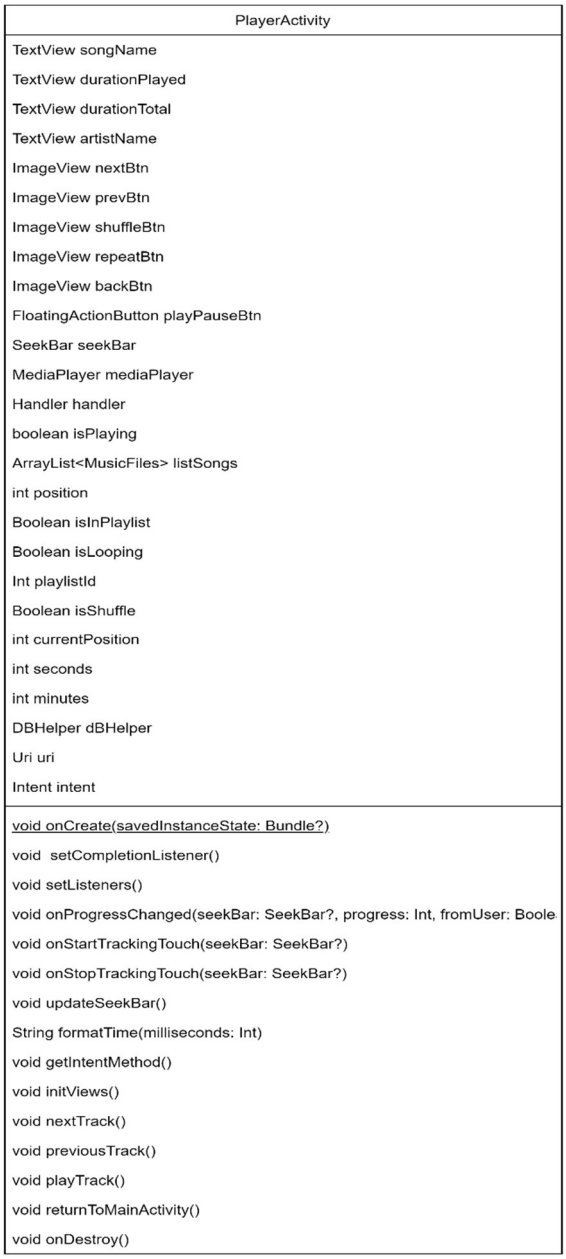


Рис 2.2.10 – Клас PlayerActivity

PlayerActivity — це активність музичного додатка, яка забезпечує відтворення музичних треків та взаємодію з користувачем. Вона має розширений функціонал музичного плеєра, включаючи перемикання треків, відтворення/паузу, повторення, шафл, і переміщення по треку. Клас також включає функції для ефективного оновлення SeekBar та запобігає витокам пам’яті при зупинці або знищенні активності.

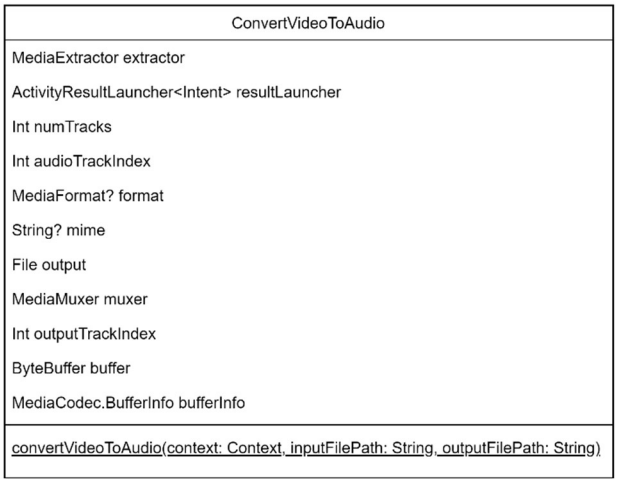


Рис 2.2.11 – Клас ConvertVideoToAudio

Клас ConvertVideoToAudio відповідає за конвертацію відеофайлу у аудіоформат. Використовуючи класи MediaExtractor і MediaMuxer, цей клас “витягує” аудіодоріжку з відео та створює новий аудіофайл.

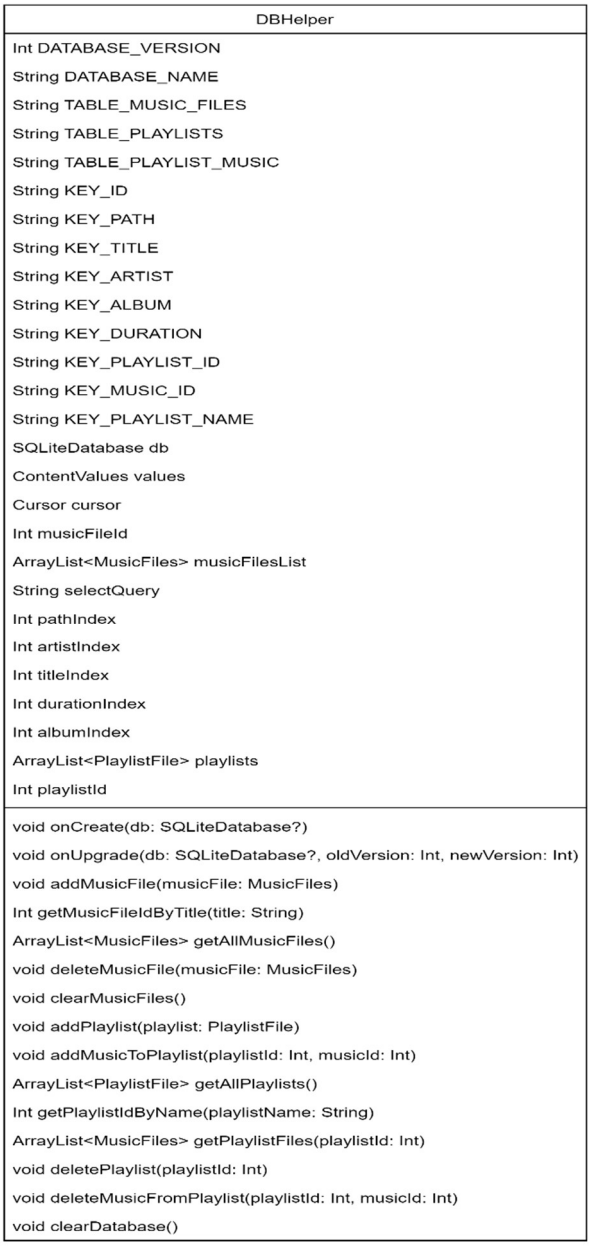


Рис 2.2.12 – Клас DBHelper

Клас DBHelper виконує функції допомоги для роботи з базою даних SQLite у додатку. Він надає методи для створення та оновлення таблиць, вставки, вибору та видалення даних з таблиць. Клас DBHelper використовується для зберігання та управління даними музичних файлів та плейлистів у додатку.

2.3 Структура бази даних

Схема «сутність-зв’язок» (ERD), також відома як ER-діаграма, є важливим інструментом для проектування та моделювання баз даних. Вона дозволяє візуально представити сутності, атрибути та взаємозв'язки між ними.[12]

Основними компонентами ER-діаграми є:

- **Сутність (Entity)** - представляє об'єкт або концепцію з реального світу, про яку зберігається інформація.
- **Атрибут (Attribute)** - це властивість або характеристика сутності.
- **Зв'язок (Relationship)** - Зв'язок показує, як сутності взаємодіють між собою.

Типи зв'язків:

- **Один-до-одного (One-to-One)** - кожна сутність А пов'язана з однією сутністю В, і навпаки.
- **Один-до-багатьох (One-to-Many)** - одна сутність А може бути пов'язана з багатьма сутностями В, але сутність В пов'язана лише з однією сутністю А.
- **Багато-до-багатьох (Many-to-Many)** - кожна сутність А може бути пов'язана з багатьма сутностями В, і кожна сутність В може бути пов'язана з багатьма сутностями А.

Отже для музичного додатку ER-діаграма виглядає таким чином:

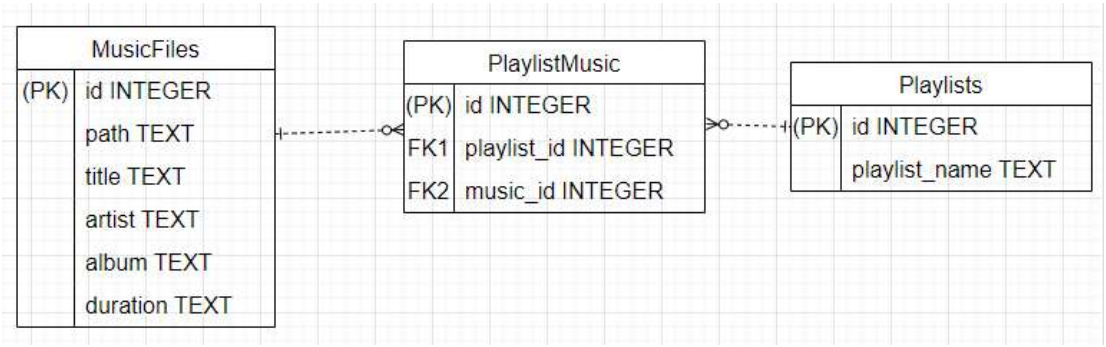


Рис 2.3.1 – ER діаграма бази даних додатку

Згідно з діаграмою можна зазначити що кожен плейлист може містити багато музичних файлів, і кожен музичний файл може бути в багатьох плейлистах.

Тому представлені сутності мають зв’язки: MusicFiles – PlaylistMusic (One-to-Many); Playlists – PlaylistMusic (One-to-Many).

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Розробка інтерфейсу

Створення інтуїтивно зрозумілого та привабливого інтерфейсу є ключовим фактором для успіху будь-якого мобільного додатку. Інтерфейс повинен бути таким, щоб користувачі могли легко його зрозуміти та використовувати, а також щоб він був візуально приємним.

Android Studio – це офіційне середовище розробки для Android, яке включає в себе інструменти для створення інтерфейсів користувачів за допомогою XML розмітки. XML розмітка – це текстовий формат, який використовується для опису структури та зовнішнього вигляду інтерфейсу.

Загальний опис інтерфейсу

Інтерфейс виконаний в мінімалістичному стилі, без навантаження великою кількістю елементів, що користувач міг легко орієнтуватися та взаємодіяти з елементами сторінок(рис 3.1.1).

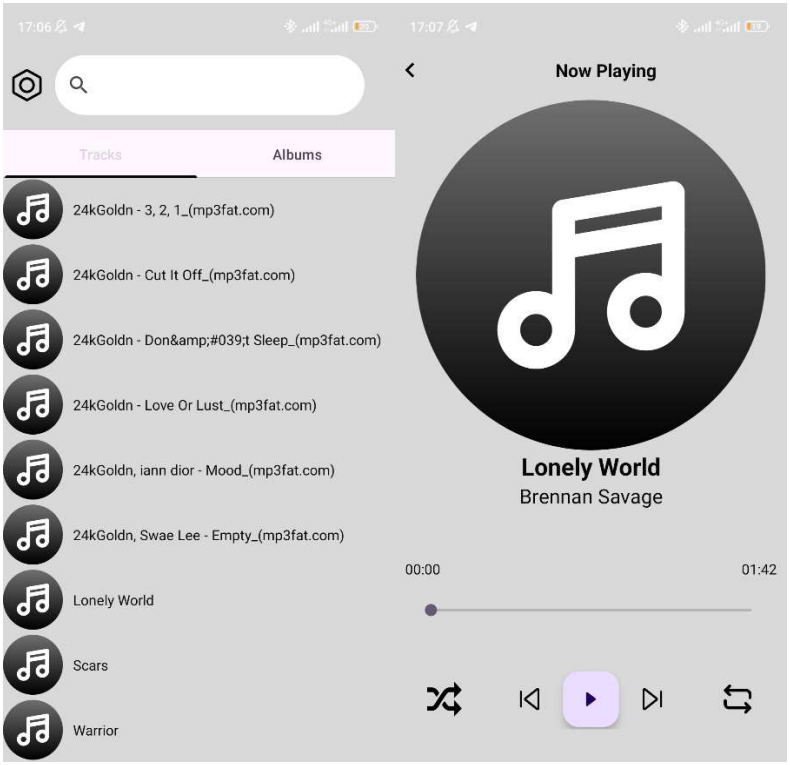


Рис 3.1.1 – Інтерфейс музичного додатку

Кольори теми по замовчуванню були ретельно підібрані – світлий сірий, що створює відчуття спокою та гармонії, підкреслюючи доброзичливість та доступність інтерфейсу. Цей нейтральний тон допомагає користувачеві відчувати зручність та легкість у використанні додатку.

Щоб забезпечити оптимальну читабельність та зручність взаємодії, іконки елементів та текст були виконані в темному стилі. Це додає контрастності та виділяє їх на світлому фоні, навіть при недостатньому освітленні, забезпечуючи миттєве сприйняття та зручність в користуванні.

Інтерфейс вікна завантаження програми

При відкритті додатку користувача зустрічає вікно завантаження програми(рис 3.1.2).

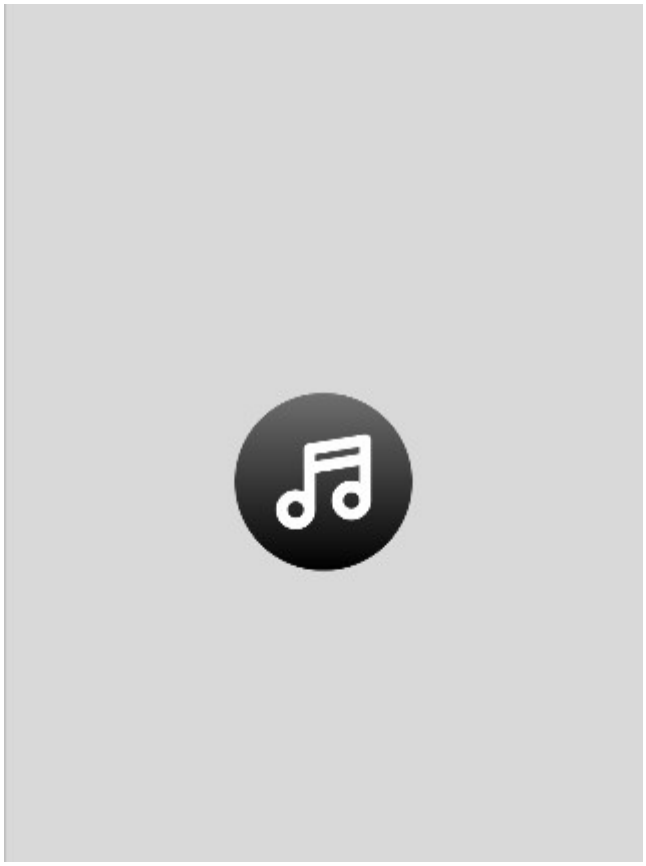


Рис 3.1.2 – Вікно завантаження програми

Цей екран є першою взаємодією користувача з додатком і виконує важливу функцію створення першого враження. Інтерфейс завантажувального вікна розроблений у відповідності до загального стилю додатку, що передбачає світлий фон та сучасний логотип.

Світлий фон створює відчуття легкості та простору, надаючи інтерфейсу естетично приємний вигляд, що сприяє позитивному користувацькому досвіду з першої миті взаємодії. Логотип додатку, розташований в центрі вікна, привертає увагу і є важливим елементом брендингу.

Логотип, розроблений за допомогою інструменту Figma[10], є втіленням елегантності та сучасності. Він поєднує в собі лінійний градієнт, що надає логотипу динамічності та візуальної привабливості, і символ ноти, що підкреслює музичну спрямованість додатку. Векторний формат логотипу забезпечує його чіткість і збереження якості на будь-якому екрані, незалежно від роздільної здатності чи масштабу.

Такий дизайн вікна завантаження створює гармонійний і професійний вигляд, який готує користувача до подальшого досвіду роботи з додатком, встановлюючи позитивний тон на початку використання.

Інтерфейс Основного вікна

Після вікна завантаження користувач перейде на основне вікно(рис 3.1.3).

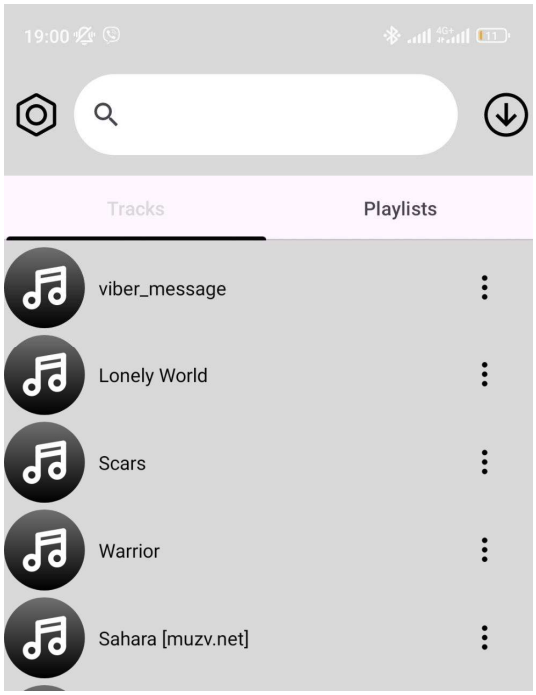


Рис 3.1.3 – Інтерфейс основного вікна

Основний екран містить такі елементи: список треків, вкладки з треками та плейлистами, поле пошуку, кнопку налаштувань та кнопку конвертації відео в аудіо.

Список треків представлений елементом ViewPager2, який містить фрагменти, що адаптуються під обрану вкладку. Кожен трек включає обкладинку (логотип додатку) та назву треку. Відступи між треками роблять список просторим і мінімалістичним. Вкладки з треками та плейлистами мають контрастні назви на світлому фоні. При виборі вкладки вона підкреслюється чорним кольором, а назва підсвічується, що дозволяє користувачеві інтуїтивно зрозуміти, на якій вкладці він знаходиться.

Поле пошуку містить поле для вводу тексту та кнопку пошуку. Поле має заокруглені кути, що додає дизайну доброзичливості та сучасності.

Список треків, виконаний у вигляді ViewPager2, забезпечує плавну навігацію між фрагментами треків. Обкладинка треку, виконана у вигляді логотипу додатку, додає впізнаваності, а назва треку є чіткою і контрастною, що полегшує читання. Відступи між треками створюють візуальний простір, роблячи інтерфейс менш захаращеним.

Вкладки з треками та плейлистами, що мають контрастні назви на світлому фоні, роблять користування додатком зручнішим. Підкреслення обраної вкладки чорним кольором і підсвічування назви вкладки полегшують навігацію.

Поле пошуку, яке містить поле для вводу тексту та кнопку пошуку, забезпечує швидкий і зручний пошук треків та плейлистів. Заокруглений дизайн поля пошуку робить інтерфейс доброзичливішим та сучаснішим.

Кнопка налаштувань, представлена у вигляді векторного зображення, забезпечує швидкий доступ до налаштувань додатку. Векторний формат забезпечує чіткість і збереження якості на будь-якому екрані.

Кнопка конвертації, також представлена у вигляді векторного зображення, дозволяє перейти до виконання функції конвертації відео.

Цей дизайн основного вікна забезпечує зручність використання, візуальну привабливість та інтуїтивну навігацію, що сприяє покращенню користувацького досвіду.

Інтерфейс вікна прослуховування треку

Коли користувач натискає на трек зі списку основного вікна, запускається нова активність(рис 2.4.1).

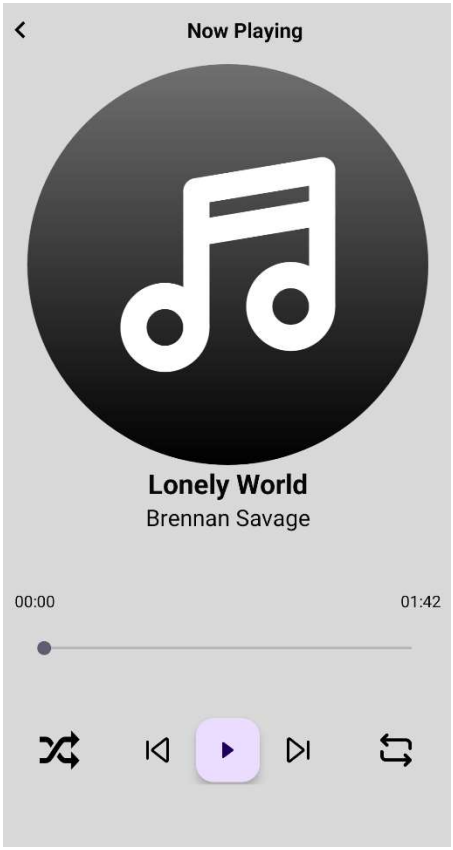


Рис 3.1.4 – Інтерфейс вікна прослуховування треку

Вікно прослуховування треку має такі елементи:

Кнопка повернення на домашню сторінку – векторне зображення, після натискання на яке користувач переходить на домашню сторінку додатку.

Обкладинка пісні(логотип)

Поля назва пісні та автор – текстові поля, що відображають назву пісні що грає та її автора.

Значення тривалості треку – це текстові поля, що відображають значення скільки трек грає, та скільки він буде грати. Значення програвання треку оновлюється динамічно кожні 100мс.

Шкала SeekBar – це шкала що показує скільки грає трек. При роботі повзунок проходить шлях зліва на право, а шкала змінює колір на темніший.

Кнопка “перемішати” програвання – натискання кнопки "Перемішати" випадковим чином змінює порядок треків у списку відтворення. При натисканні кнопка стає білою, сигналізуючи про її активність.

Кнопки “попередній трек” та “наступний трек” – при натисканні відповідної кнопки система переходить до наступного або попереднього треку.

Кнопка “пауза/відтворення” – це floatingActionButton котра зупиняє відтворення треку, тим самим зупиняючи відлік значення тривалості треку, та шкалу SeekBar. Після натискання на кнопку змінюється зображення всередині кнопки на “пауза” або “відтворення” відповідно.

Кнопка повторення треку – кнопка "Повторення" дозволяє зациклити відтворення поточного треку, створюючи ефект безперервного прослуховування. При її активації кнопка змінює колір на білий, сигналізуючи про те що вона натиснута.

Інтерфейс фрагменту з плейлистами

Якщо користувач натисне на вкладку плейлисти, він перейде до фрагменту, в якому можна створювати, видаляти та переглядати плейлисти.(рис 3.1.5)

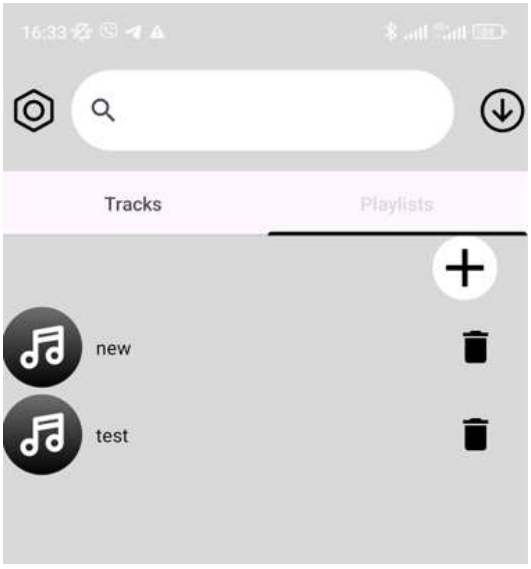


Рис 3.1.5 – Інтерфейс фрагменту з плейлистами

Інтерфейс налічує у собі: список плейлистів, вкладки з треками та плейлистами, поле пошуку, кнопку налаштувань та кнопку конвертації відео в аудіо.

Крім елементів що були зазначені вище, фрагмент налічує:

Кнопку для додавання плейлисту – векторне зображення, що має форму кола, з “плюсом”, що додає контрастності та інтуїтивності з боку користувача.

Список з плейлистів – елементи виконані майже у одному стилі як і треки, за вийнятком кнопки “видалення”. Кнопка видалення – це векторне зображення чорного кольору що має інтуїтивно вказувати користувачу на функцію видалення плейлисту.

Інтерфейс вмісту плейлисту

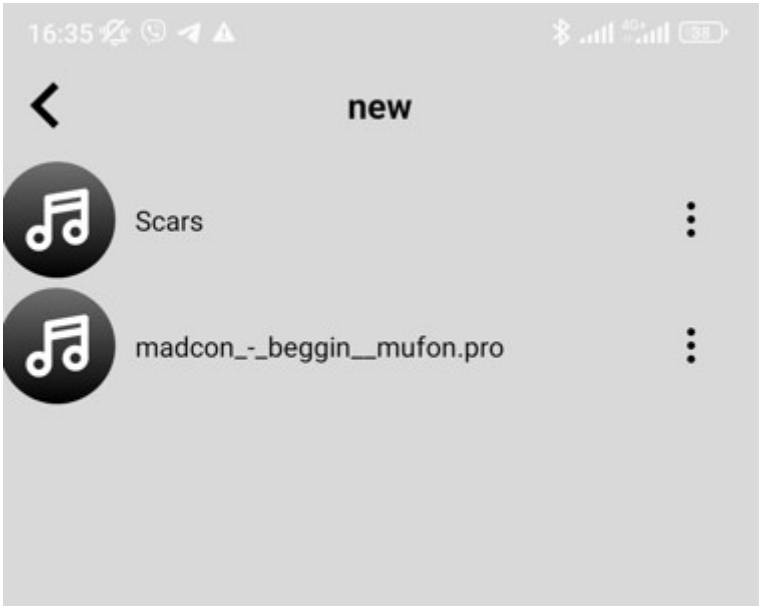


Рис 3.1.6 – Інтерфейс вмісту плейлисту

Інтерфейс виконаний в мінімалістичному стилі, та містить у собі елементи треків, їх властивості та кнопку повернення до попередньої активності.

3.2 Опис програмної реалізації

Основна структура додатка побудована на архітектурі з використанням Fragment для розділення функціональності між різними компонентами інтерфейсу. В основній активності, MainActivity, розміщується ViewPager2 з вкладками для переміщення між фрагментами, які представляють різні секції додатка, такі як - список пісень та плейлистів.

Опис роботи програми:

Користувач відкриває додаток, після чого його зустрічає екран-заставка SplashActivity і через короткий проміжок часу переходить до основної активності(MainActivity). В MainActivity, в першу чергу додаток перевіряє дозволи на читання та запис файлів на телефон, якщо дозволу немає то додаток робить запит у користувача на дозвіл. Коли користувач дасть дозвіл то додаток парсить файлову систему телефону та знаходить музичні файли, і заносить дані про назву музичного треку, виконавця, його шлях, альбом та тривалість, в базу даних за допомогою класу DBHelper. Після чого в основній активності, користувач може:

- **Виконати функцію пошуку треку** – натиснути на поле пошуку та ввести частину або ж повну назву треку;
- **Почати відтворення** - натиснувши на елемент списку котрий був сформований згідно зі знайденими файлами;
- **Перейти на вкладку “Плейлисти”** – після переходу на котру користувач може побачити кнопку для створення плейлисту, а після створення плейлисту користувач зможе додати трек у відповідний плейлист та прослуховувати музику що є в ньому;
- **Натиснути на кнопку налаштування** - для переходу до налаштувань додатку відповідно. А в налаштуваннях можна обрати тему додатку(світлу або темну).
- **Натиснути на кнопку для конвертації -** після того як користувач натисне на кнопку конвертації, програма покаже вікно в якому є кнопка

“Search”. Після натискання на кнопку “Search” користувач обирає відео для конвертації. В результаті відео конвертується в аудіофайл.

Після того як користувач натисне на трек та почне відтворення, почнеться активність – PlayerActivity. Вона містить в собі такі функції як:

- перемикання треків.
- відтворення/паузу.
- повторення, шафл.
- переміщення по треку.

Для виходу з PlayerActivity користувач може натиснути на відповідну кнопку для повернення на основну активність, в лівому верхньому куті інтерфейсу.

Розглянемо основні компоненти додатку, їх функції та опис:

SplashActivity:

Використання CoroutineScope для Затримки(delay(1500)).

Опис: Створюється корутина, яка виконує асинхронну затримку на 1,5 секунди. Це використовується для того, щоб екран-заставка відображався протягом певного часу.

MainActivity:

permission()

Опис: Функція перевіряє, чи додаток має дозвіл на читання зовнішньої пам'яті, якщо дозвіл не надано, функція запитує його через ActivityCompat.requestPermissions(). Якщо дозвіл є, викликаються методи initViewPager() і getAllAudioFiles(), щоб ініціалізувати інтерфейс і отримати список музичних файлів.

onRequestPermissionsResult()

Опис: Метод обробляє результати запиту на дозвіл. Він приймає requestCode, масив permissions, і масив grantResults.

initViewPager()

Опис: Метод ініціалізує ViewPager2 і встановлює адаптер ViewPagerAdapter з переліком фрагментів (fragList). Також використовується TabLayoutMediator для підключення вкладок TabLayout до ViewPager2.

getAllAudioFiles()

Опис: Метод виконує запит до MediaStore через ContentResolver, щоб отримати дані про аудіофайли. Він визначає проекцію (які дані потрібні), створює Cursor, і використовує цикл, щоб зчитувати дані з Cursor та передати їх до бази даних.

showSongSearchDialog()

Опис: Метод призначений для відображення діалогового вікна, яке дозволяє користувачеві шукати і вибирати відеофайл зі своєї галереї.

PlayerActivity:

setCompletionListener()

Опис: Встановлює слухача події завершення для MediaPlayer. Коли поточний трек закінчується, виконується перехід до наступного треку, якщо це можливо.

setListeners()

Опис: Встановлює обробники подій для різних елементів інтерфейсу, таких як кнопки, SeekBar, та інші.

updateSeekBar()

Опис: Використовує Handler для періодичного оновлення (postDelayed). Перевіряє, чи грає MediaPlayer, і оновлює прогрес SeekBar та час відтворення. Якщо MediaPlayer все ще грає, функція викликає сама себе для повторного оновлення.

formatTime()

Опис: Форматує час у формат мм:сс, а саме розраховує хвилини та секунди з даних у мілісекундах, і повертає форматований рядок.

getIntentMethod()

Опис: Отримує позицію з Intent. Відтворює музичний трек, встановлюючи необхідні значення в інтерфейсі (назва, виконавець, загальний

час). Оновлює SeekBar та викликає setCompletionListener() для обробки завершення відтворення.

initViews()

Опис: Отримує елементи з макету за допомогою findViewById і встановлює їх в якості полів класу (songName, durationPlayed, durationTotal, artistName, тощо).

PlaylistDetailActivity:

onCreate()

Опис: Метод створює екземпляр DBHelper для взаємодії з базою даних, отримує playlistId та playlistName задля відображення музичних файлів що були додані до відповідного плейлисту, а також задля встановлення назви плейлисту. Отримує список музичних файлів у плейлисті з бази даних та ініціалізує MusicAdapter з отриманими музичними файлами для відображення треків користувачу.

returnToPreviousActivity()

Опис: Метод закриває поточну активність і повертає користувача до попередньої активності.

SettingsActivity:

onCreate()

Опис: Метод ініціалізує SharedPreferences для зберігання налаштувань додатку, а також налаштовує обробники натискань для кнопок вибору теми та кнопки повернення.

setTheme()

Опис: Метод зберігає вибрану тему у SharedPreferences і перезапускає активність для застосування нової теми.

applyTheme()

Опис: Метод зчитує вибрану тему з SharedPreferences і застосовує її до активності.

3.3 Тестування

Для тестування додатку я використав метод мануального тестування, який дозволяє детально перевірити роботу всіх функцій і елементів інтерфейсу користувача. Нижче наведено опис процесу мануального тестування, який я виконав на своєму смартфоні.

Пристрій: Xiaomi Redmi note 9 pro(Android 12).

Підготовка

- **Встановлення додатку**: Завантажив та встановив APK-файл додатку на свій смартфон.
- **Налаштування середовища**: Переконався, що на пристрої встановлено необхідні версії Android та наявний стабільний доступ до Інтернету.

Тестування основних функцій:

1. Запит на доступ до медіафайлів та зчитування медіафайлів:

Мета: Перевірити функціонал запитів на доступ до медіафайлів, та зчитування з файлової системи телефону

Процедура: Зайти в додаток, натиснути дозволити доступ до медіафайлів, якщо такий запит наявний.

Результат: Відбулося відображення запиту додатку щодо дозволу доступу до медіафайлів. Всі медіафайли зчитались та були відображені у вигляді списку(рис 3.3.1).

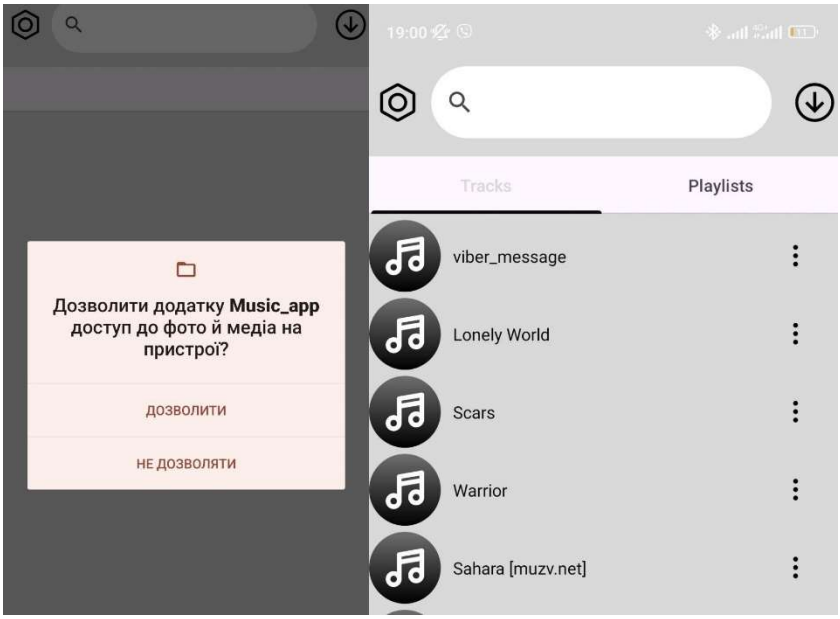


Рис 3.3.1 – Результати тестування №1

2. Навігація між екранами:

Мета: Перевірити коректність роботи навігації.

Процедура: Перейшов між різними екранами додатку (головний екран, екран плейлистів, екран налаштувань, екран програвання треків).

Результат: Навігація працює плавно, всі екрани завантажуються коректно(рис 3.3.2).

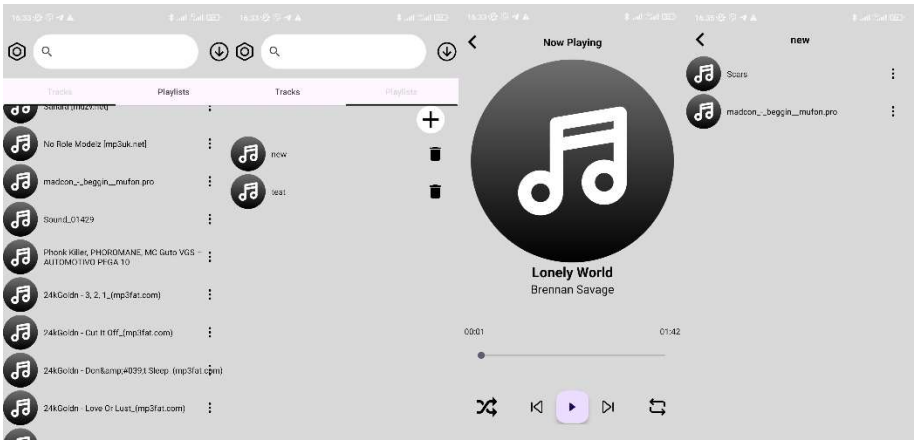


Рис 3.3.2 – Результати тестування навігації

3. Створення та редагування плейлистів:

Мета: Перевірити функціонал створення та редагування плейлистів.

Процедура: Створив новий плейлист, додав до нього музичні файли, видалив плейлист.

Результат: Плейлисти створюються та редагуються без проблем(рис3.3.3).

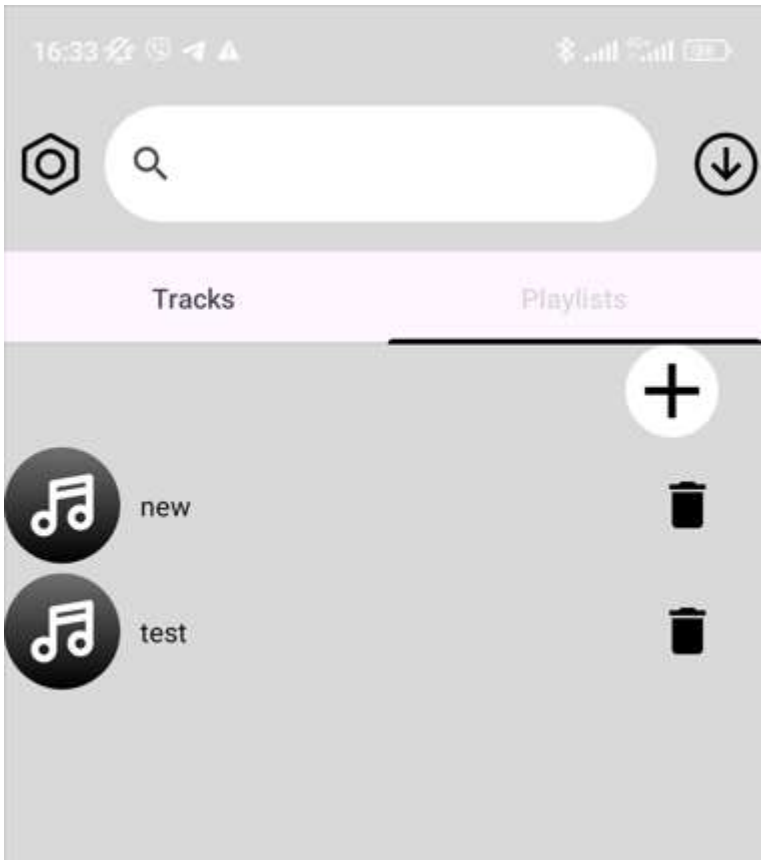


Рис 3.3.3 – Результати тестування плейлистів

4. Видалення музичних файлів:

Мета: Перевірити можливість видалення музичних файлів з бібліотеки.

Процедура: Натиснути на три крапки, що розміщені на одному рівні треку. У спливаючому вікні натиснути кнопку “видалити трек”. Підтвердити видалення треку.

Результат: Файли видаляються коректно, жодних помилок не виявлено.

5. Конвертація відео в аудіо:

Мета: Перевірити функцію конвертації відео в аудіо.

Процедура: Перейти на головну сторінку додатку. Натиснути на кнопку для конвертації(правий верхній куток). Натиснути кнопку “Search”. Обрати відео для конвертації.

Результат: Відео конвертовано коректно, жодних помилок не виявлено.(рис 3.3.4)

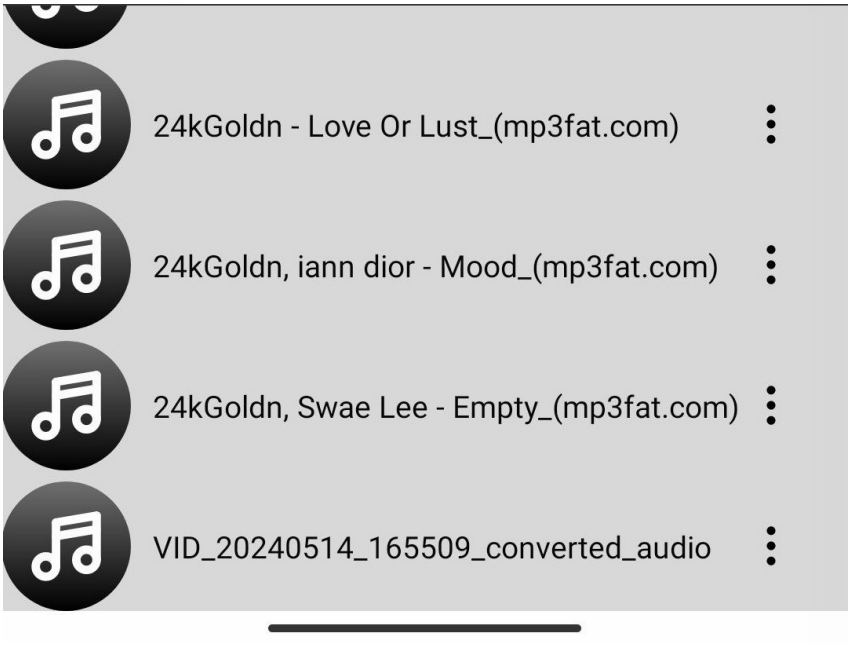


Рис 3.3.4 – Результати тестування функції конвертації відео в аудіо

ВИСНОВОК

У процесі розробки персоналізованого інтелектуального музичного додатка я проробив ряд важливих кроків, які привели до функціонального та інтуїтивно зрозумілого продукту. Вибір технологічного стеку, середовища розробки, мови програмування та інструментів для дизайну був здійснений з урахуванням сучасних вимог до продуктивності, зручності використання та гнучкості.

Як середовище розробки я використав Android Studio, яке містить у собі широкий набір інструментів для ефективної роботи над додатком. Kotlin був обраний як основна мова програмування завдяки його сучасності, а застосунок Figma — для досить легкого створення макетів, щоб надалі на базі цих макетів проектувати інтерфейс з допомогою XML-розмітки. Система керування базами даних SQLite була обрана як найбільш підходяща для потреб моєї програми через свою легкість використання, вбудовану підтримку у середовищі Android та ефективність у роботі з невеликими обсягами даних.

Реалізація програмного забезпечення була виконана з урахуванням вимог користувачів, що були виокремлені в аналізі конкурентоспроможності. Музичний додаток забезпечує взаємодію та необхідні функції для прослуховування музики, включаючи відтворення, паузу, перемикання треків, повтор, шафл та використання SeekBar для переміщення по треку.

Результати цієї роботи сприяли створенню музичного додатка, що відповідає сучасним вимогам і забезпечує приємний користувацький досвід. У процесі розробки я отримав практичні навички роботи з такими технологіями, як MediaStore для доступу до медіаконтенту, Handler для асинхронної обробки, із методами взаємодії з користувачем, SQLite як систему керування базами даних, та ін.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Суспільне Культура. (2023, липень 31). Що і як слухають українці: результати дослідження музичних вподобань у 2023 році. Суспільне Культура. URL: https://suspilne.media/culture/643078-so-i-ak-sluhaut-ukrainci-rezultati-doslidzenna-muzicnih-vpodoban-u-2023-roci/ (дата звернення: 19.05.2024).

2. QATestLab. (2022, серпень 15). Application requirements guideline. QATestLab Blog. URL: https://training.qatestlab.com/blog/technical-articles/application-requirements-guideline/ (дата звернення: 19.05.2024).

3. Salyga, P. (2016, серпня 2). Принципи матеріального дизайну і чим він загрожує дизайнерам. Web Design and Usability Analysis. Архів оригіналу за 1 вересня 2017. (дата звернення: 19.05.2024).

4. MOYO.UA. (2023). Кращі музичні плеєри для Android: ТОП-10. MOYO.UA. URL: https://www.moyo.ua/ua/news/luchshie_muzykalnye_pleery_dlya_android_top-10.html (дата звернення: 19.05.2024).

5. DEVELOPERS.ANDROID. Meet Android Studio. URL: https://developer.android.com/studio/intro (дата звернення: 19.05.2024).

6. IndianAppDevelopers. The Major Advantages of Android Studio App Development. URL: https://www.indianappdevelopers.com/blog/advantages-of-android-studio-app-development/ (дата звернення: 19.05.2024).

7. Codecademy. What Is Kotlin Used For? URL: https://www.codecademy.com/resources/blog/what-is-kotlin-used-for/ (дата звернення: 19.05.2024).

8. DEVELOPERS.ANDROID. Develop a UI with Views. URL: https://developer.android.com/studio/write/layout-editor (дата звернення: 19.05.2024).

9. SQLite. SQLite. URL: https://www.sqlite.org/ (дата звернення: 19.05.2024).

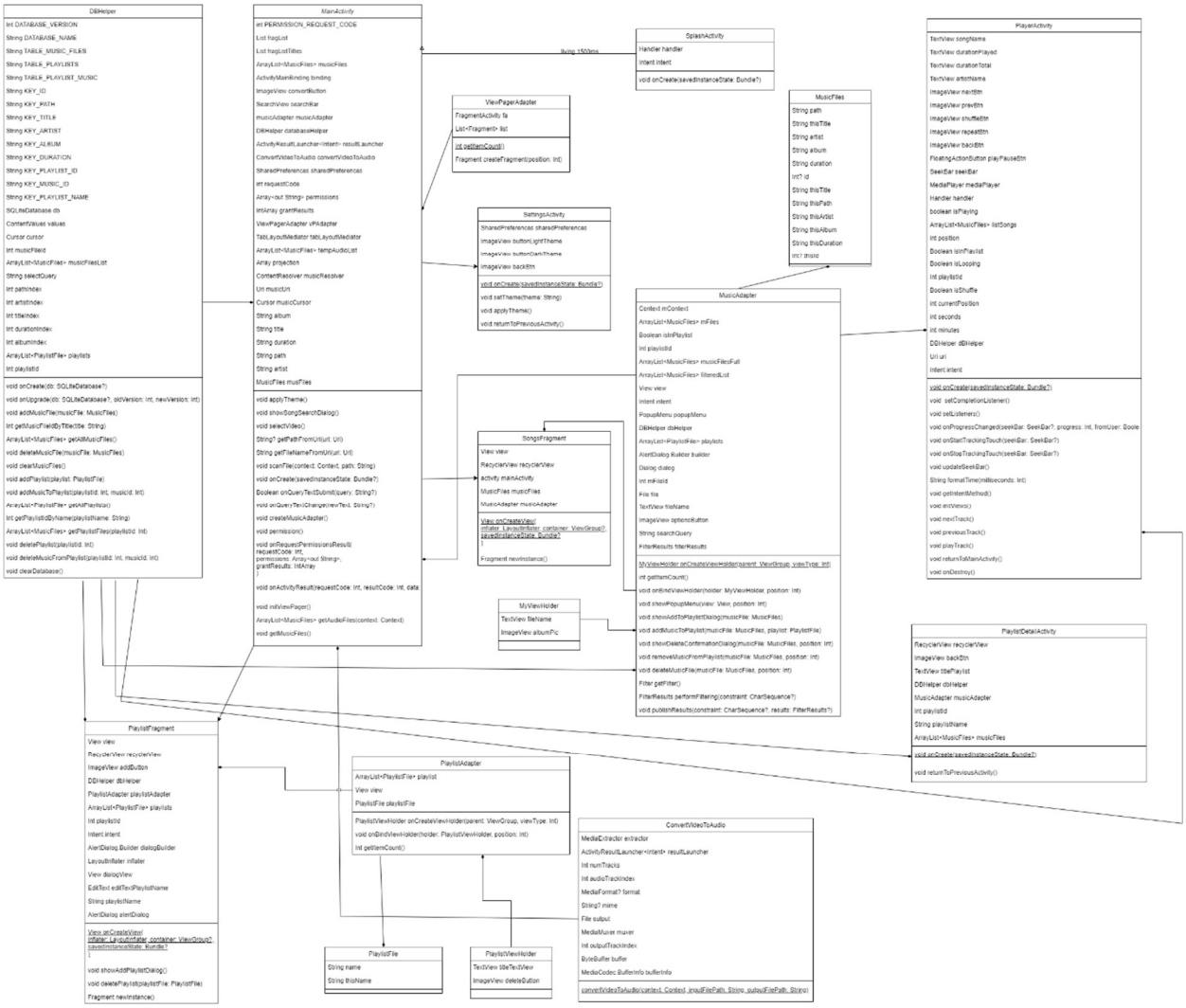
10. Figma. (2023). What is Figma? Figma Help Center. URL: https://help.figma.com/hc/en-us/articles/14563969806359-What-is-Figma#:~:text=Figma%20design%20is%20for%20people,and%20make%20better%20decisions%2C%20faster. (дата звернення: 19.05.2024).

11. Rumbaugh, J., Jacobson, I., & Booch, G. (1999). The unified modeling language reference manual. Addison Wesley Longman Inc. ISBN 0-201-30998-X. (дата звернення: 19.05.2024).

12. Lucidchart. What is an Entity Relationship Diagram (ERD)? URL: https://www.lucidchart.com/pages/er-diagrams (дата звернення: 19.05.2024).

Додатки

Додаток 1 – Діаграма класів



Додаток 2 – Код модулів додатку

SplashActivity.kt

```
package com.example.music_app

import android.annotation.SuppressLint
import android.content.Intent
import android.os.Bundle
import android.os.Handler
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.Job
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch


class SplashActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_splash)

        val handler = CoroutineScope(Dispatchers.Main).launch {
            delay(1500)
            val intent = Intent(this@SplashActivity,
MainActivity::class.java)
            startActivity(intent)
            finish()
        }
    }
}
```

MainActivity.kt

```
package com.example.music_app

import android.content.ContentResolver
import android.content.Context
import android.content.pm.PackageManager
import android.database.Cursor
import android.net.Uri
import android.os.Bundle
import android.Manifest
import android.app.Activity
import android.content.ActivityNotFoundException
import android.content.Intent
import android.content.SharedPreferences
import android.media.MediaScannerConnection
import android.os.Build
import android.os.Environment
import android.provider.MediaStore
import android.provider.Settings
import android.util.Log
import android.widget.Button
import android.widget.ImageView
import android.widget.SearchView
import android.widget.Toast
import androidx.activity.result.ActivityResultLauncher
```

```
import androidx.activity.result.contract.ActivityResultContracts
import androidx.appcompat.app.AlertDialog
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import com.example.music_app.adapters.ViewPagerAdapter
import com.example.music_app.adapters.MusicAdapter
import com.example.music_app.databinding.ActivityMainBinding
import com.example.music_app.fragments.PlaylistFragment
import com.example.music_app.fragments.SongsFragment
import com.example.music_app.helpers.DBHelper
import com.google.android.material.tabs.TabLayoutMediator
import java.io.File


class MainActivity : AppCompatActivity() {

    val PERMISSION_REQUEST_CODE = 1

    private val fragList = listOf(
        SongsFragment.newInstance(),
        PlaylistFragment.newInstance()
    )

    private val fragListTitles = listOf(
        "Tracks", "Playlists"
    )
    companion object {
        var musicFiles: ArrayList<MusicFiles> = ArrayList()
    }

    private lateinit var binding: ActivityMainBinding
    private lateinit var convertButton: ImageView
    private lateinit var searchBar: SearchView
    private lateinit var musicAdapter: MusicAdapter
    private lateinit var databaseHelper: DBHelper
    private lateinit var resultLauncher: ActivityResultLauncher<Intent>
    private val convertVideoToAudio = ConvertVideoToAudio()
    private lateinit var sharedPreferences: SharedPreferences

    override fun onCreate(savedInstanceState: Bundle?) {

        sharedPreferences = getSharedPreferences("AppSettings", MODE_PRIVATE)
        applyTheme()

        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        databaseHelper = DBHelper(this)
        permission()



        musicAdapter = createMusicAdapter()
        searchBar = findViewById(R.id.search_bar)
        val buttonSettings = findViewById<ImageView>(R.id.settings_button)

        buttonSettings.setOnClickListener {
            val intent = Intent(this, SettingsActivity::class.java)
            startActivity(intent)
        }

        searchBar.setOnQueryTextListener(object :
```

```
SearchView.OnQueryTextListener {
            override fun onQueryTextSubmit(query: String?): Boolean {
                return false
            }

            override fun onQueryTextChange(newText: String?): Boolean {
                musicAdapter.filter.filter(newText) // Передача запросу в
адаптер для фільтрації
                return true
            }
        })

        convertButton = findViewById(R.id.convert_button)
        convertButton.setOnClickListener{
            showSongSearchDialog()
        }

        resultLauncher =
registerForActivityResult(ActivityResultContracts.StartActivityForResult()) {
result ->
            if (result.resultCode == RESULT_OK) {
                val data: Intent? = result.data
                val selectedVideoUri = data?.data
                selectedVideoUri?.let { uri ->
                    val inputFilePath = getPathFromUri(uri)
                    inputFilePath?.let { path ->
                        val videoName = getFileNameFromUri(uri)
                        val outputDir =
File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWN
LOADS), "ConvertedAudio")
                        if (!outputDir.exists()) {
                            outputDir.mkdirs()
                        }
                        val outputFilePath =
"${outputDir.absolutePath}/${videoName}_converted_audio.mp3"
                        convertVideoToAudio.convertVideoToAudio(this,
inputFilePath, outputFilePath)
                        Toast.makeText(this, "The audio has been successfully
converted and saved to $outputFilePath", Toast.LENGTH_SHORT).show()
                        scanFile(this, outputFilePath)
                        getAudioFiles(this)
                    }
                }
            }
        }
    }

    private fun applyTheme() {
        when (sharedPreferences.getString("AppTheme", "light")) {
            "light" -> setTheme(R.style.LightTheme)
            "dark" -> setTheme(R.style.DarkTheme)
        }
    }

    private fun showSongSearchDialog() {
        val builder = AlertDialog.Builder(this)
        val view = layoutInflater.inflate(R.layout.dialog_download, null)

        val searchButton = view.findViewById<Button>(R.id.button_search)

        val dialog = builder.setView(view).create()

        searchButton.setOnClickListener {
```

```
            selectVideo()
            dialog.dismiss()
        }

        dialog.show()
    }

    private fun selectVideo() {
        val intent = Intent(Intent.ACTION_PICK,
MediaStore.Video.Media.EXTERNAL_CONTENT_URI)
        intent.type = "video/*"
        resultLauncher.launch(Intent.createChooser(intent, "Select video"))
    }

    private fun getPathFromUri(uri: Uri): String? {
        val projection = arrayOf(MediaStore.Video.Media.DATA)
        val cursor = contentResolver.query(uri, projection, null, null, null)
        cursor?.use {
            if (it.moveToFirst()) {
                val columnIndex =
it.getColumnIndexOrThrow(MediaStore.Video.Media.DATA)
                return it.getString(columnIndex)
            }
        }
        return null
    }

    private fun getFileNameFromUri(uri: Uri): String {
        var result: String? = null
        if (uri.scheme == "content") {
            val cursor = contentResolver.query(uri, null, null, null, null)
            cursor?.use {
                if (it.moveToFirst()) {
                    val columnIndex =
it.getColumnIndexOrThrow(MediaStore.Video.Media.DISPLAY_NAME)
                    result = it.getString(columnIndex)
                }
            }
        }
        if (result == null) {
            result = uri.path
            val cut = result?.lastIndexOf('/')
            if (cut != null && cut != -1) {
                result = result?.substring(cut + 1)
            }
        }
        return result?.substringBeforeLast(".") ?: "converted_audio"
    }

    private fun scanFile(context: Context, path: String) {
        MediaScannerConnection.scanFile(context, arrayOf(path), null) { _,
uri ->
            runOnUiThread {
                Toast.makeText(context, "File has been added into media base:
$uri", Toast.LENGTH_SHORT).show()
                getAudioFiles(context)
            }
        }
    }

    private fun createMusicAdapter(): MusicAdapter {
        return MusicAdapter(this, databaseHelper.getAllMusicFiles())
    }
```

```
    private fun permission() {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) {
            try {
                if (Environment.isExternalStorageManager()) {
                    // Permission has already been granted
                    initViewPager()
                    getAudioFiles(this)
                    musicFiles = databaseHelper.getAllMusicFiles()
                } else {
                    // Request the user to grant MANAGE_EXTERNAL_STORAGE
permission
                    val intent =
Intent(Settings.ACTION_MANAGE_ALL_FILES_ACCESS_PERMISSION)
                    intent.data = Uri.parse("package:$packageName")
                    startActivityForResult(intent, PERMISSION_REQUEST_CODE)
                }
            } catch (e: ActivityNotFoundException) {
                // Handling cases where intent is not supported
                val permissions = arrayOf(
                    Manifest.permission.READ_EXTERNAL_STORAGE,
                    Manifest.permission.WRITE_EXTERNAL_STORAGE
                )
                ActivityCompat.requestPermissions(this, permissions,
PERMISSION_REQUEST_CODE)
            }
        } else {
            val permissions = arrayOf(
                Manifest.permission.READ_EXTERNAL_STORAGE,
                Manifest.permission.WRITE_EXTERNAL_STORAGE
            )
            val permissionsToRequest = mutableListOf<String>()

            permissions.forEach {
                if (ContextCompat.checkSelfPermission(this, it) !=
PackageManager.PERMISSION_GRANTED) {
                    permissionsToRequest.add(it)
                }
            }

            if (permissionsToRequest.isNotEmpty()) {
                ActivityCompat.requestPermissions(this,
permissionsToRequest.toTypedArray(), PERMISSION_REQUEST_CODE)
            } else {
                initViewPager()
                getAudioFiles(this)
                musicFiles = databaseHelper.getAllMusicFiles()
            }
        }
    }

    override fun onRequestPermissionsResult(requestCode: Int, permissions:
Array<out String>, grantResults: IntArray) {
        super.onRequestPermissionsResult(requestCode, permissions,
grantResults)
        if (requestCode == PERMISSION_REQUEST_CODE) {
            if (grantResults.isNotEmpty() && grantResults.all { it ==
PackageManager.PERMISSION_GRANTED }) {
                initViewPager()
                getAudioFiles(this)
                musicFiles = databaseHelper.getAllMusicFiles()
            } else {
                Toast.makeText(this, "Permissions are required to continue
```

```
running the application.", Toast.LENGTH_SHORT).show()
            }
        }
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data:
Intent?) {
        super.onActivityResult(requestCode, resultCode, data)
        if (requestCode == PERMISSION_REQUEST_CODE) {
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) {
                if (Environment.isExternalStorageManager()) {
                    // Permission granted
                    initViewPager()
                    getAudioFiles(this)
                    musicFiles = databaseHelper.getAllMusicFiles()
                } else {
                    Toast.makeText(this, "Permissions are required to
continue running the application.", Toast.LENGTH_SHORT).show()
                }
            }
        }
    }

    private fun initViewPager(){

        val vPAdapter = ViewPagerAdapter(this,fragList)
        binding.viewPager.adapter = vPAdapter
        val tabLayoutMediator = TabLayoutMediator(binding.tabLayout,
binding.viewPager) { tab, position ->
            tab.text = fragListTitles[position]
        }
        tabLayoutMediator.attach()
    }


    private fun getAudioFiles(context: Context){
        databaseHelper.clearMusicFiles()
        val projection = arrayOf(
            MediaStore.Audio.Media.ALBUM,
            MediaStore.Audio.Media.TITLE,
            MediaStore.Audio.Media.DURATION,
            MediaStore.Audio.Media.DATA,
            MediaStore.Audio.Media.ARTIST
        )

        val musicResolver:ContentResolver = contentResolver
        val musicUri:Uri =
android.provider.MediaStore.Audio.Media.EXTERNAL_CONTENT_URI
        val musicCursor: Cursor? = musicResolver.query(musicUri, projection,
null, null, null)

        musicCursor?.use { cursor ->
            while (cursor.moveToNext()) {
                val album: String? = cursor.getString(0)
                val title: String? = cursor.getString(1)
                val duration: String? = cursor.getString(2)
                val path: String? = cursor.getString(3)
                val artist: String? = cursor.getString(4)

                if (album != null && title != null && duration != null &&
path != null && artist != null) {
                    val musicFile = MusicFiles(path, title, artist, album,
duration)
```

```
                    databaseHelper.addMusicFile(musicFile)
                }
            }
        }

    }

    fun getMusicFiles(): ArrayList<MusicFiles> {
        if (musicFiles.isEmpty()) {
            getAudioFiles(this)
            musicFiles = databaseHelper.getAllMusicFiles()
        }
        return musicFiles
    }
}
```

PlayerActivity.kt

```
package com.example.music_app

import android.content.Intent
import android.media.MediaPlayer
import android.net.Uri
import android.os.Bundle
import android.os.Handler
import android.widget.ImageView
import android.widget.SeekBar
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
import com.example.music_app.helpers.DBHelper
import com.google.android.material.floatingactionbutton.FloatingActionButton

class PlayerActivity : AppCompatActivity() {

    private lateinit var songName:TextView
    private lateinit var durationPlayed:TextView
    private lateinit var durationTotal:TextView
    private lateinit var artistName:TextView
    private lateinit var nextBtn:ImageView
    private lateinit var prevBtn:ImageView
    private lateinit var shuffleBtn:ImageView
    private lateinit var repeatBtn:ImageView
    private lateinit var backBtn:ImageView
    private lateinit var playPauseBtn:FloatingActionButton
    private lateinit var seekBar:SeekBar
    private lateinit var uri:Uri
    private var isLooping = false
    private var isShuffle = false

    private lateinit var mediaPlayer:MediaPlayer
    private val handler = Handler() // Для оновлення SeekBar
    private var isPlaying = true // Флаг для відстеження стану відтворення
музики

    private var listSongs:ArrayList<MusicFiles> = ArrayList()
    private var position = -1
    private var isInPlaylist = false
    private var playlistId = -1

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_player)
        initViews()
        getIntentMethod()
```

```
        setListeners()
        updateSeekBar()
        setCompletionListener()

    }

    private fun setCompletionListener() { // Функція переходу до наступного
треку після закінчення треку що грає
        if (position < listSongs.size - 1) { // Перевірка чи останній трек
грає
            mediaPlayer.setOnCompletionListener {
                nextTrack()
            }
        }
    }

    private fun setListeners() {
        playPauseBtn.setOnClickListener {// Прослуховувач для кнопки
програвання/паузи
            if (isPlaying) {
                mediaPlayer.pause()
                playPauseBtn.setImageResource(R.drawable.play)
            } else {
                mediaPlayer.start()
                playPauseBtn.setImageResource(R.drawable.pause)
                updateSeekBar()
            }
            isPlaying = !isPlaying
        }

        nextBtn.setOnClickListener {// Прослуховувач для кнопки переходу до
наступного треку
            nextTrack()
        }

        prevBtn.setOnClickListener {// Прослуховувач для кнопки переходу до
попереднього треку
            previousTrack()
        }

        repeatBtn.setOnClickListener {// Прослуховувач для кнопки повтору
треку
            isLooping = !isLooping
            mediaPlayer.isLooping = isLooping
            repeatBtn.setImageResource(if (isLooping) R.drawable.repeat_on
else R.drawable.repeat_off)
        }

        shuffleBtn.setOnClickListener {// Прослуховувач для кнопки шафлу
треків
            isShuffle = !isShuffle
            if (isShuffle) {
                listSongs.shuffle()
            } else {

            }
            shuffleBtn.setImageResource(if (isShuffle) R.drawable.shuffle_on
else R.drawable.shuffle_off)
        }

        backBtn.setOnClickListener {// Прослуховувач для кнопки переходу на
головне меню
            returnToPreviousActivity()
```

```
        }

        seekBar.setOnSeekBarChangeListener(object :
SeekBar.OnSeekBarChangeListener {// Прослуховувач для змін SeekBar
            override fun onProgressChanged(seekBar: SeekBar?, progress: Int,
fromUser: Boolean) {
                if (fromUser) { // Перевірка якщо зміна від користувача
                    mediaPlayer.seekTo(progress * 1000)
                }
            }

            override fun onStartTrackingTouch(seekBar: SeekBar?) {

            }

            override fun onStopTrackingTouch(seekBar: SeekBar?) {

            }
        })
    }


    private fun updateSeekBar() {// Функція для оновлення SeekBar
        handler.postDelayed({
            if (mediaPlayer.isPlaying) {
                val currentPosition = mediaPlayer.currentPosition / 1000
                seekBar.progress = currentPosition
                durationPlayed.text = formatTime(currentPosition)
                updateSeekBar()
            }
        }, 100)
    }


    fun formatTime(milliseconds: Int): String {// Функція для
переформатування часу
        val seconds = milliseconds % 60
        val minutes = (milliseconds / 60) % 60
        return String.format("%02d:%02d", minutes, seconds)
    }

    private fun getIntentMethod() {// Функція для ініціалізації відтворення
музичного треку
        position = intent.getIntExtra("position", -1)
        isInPlaylist = intent.getBooleanExtra("isInPlaylist", false)
        playlistId = intent.getIntExtra("playlistId", -1)
        val dBHelper = DBHelper(this)

        if(isInPlaylist && playlistId != -1){
            listSongs = dBHelper.getPlaylistFiles(playlistId)
        }
        else{
            listSongs = dBHelper.getAllMusicFiles()
        }

        if (position >= 0) {
            uri = Uri.parse(listSongs[position].thisPath)
            mediaPlayer = MediaPlayer.create(this, uri)
            mediaPlayer.start()
            seekBar.max = mediaPlayer.duration / 1000
            songName.text = listSongs[position].thisTitle
            artistName.text = listSongs[position].thisArtist
            durationTotal.text = formatTime(mediaPlayer.duration / 1000)
```

```
        }
    }

    private fun initViews(){// Функція для ініціалізації компонентів сторінки
        songName = findViewById(R.id.song_name)
        durationPlayed = findViewById(R.id.duration_played)
        durationTotal = findViewById(R.id.duration_total)
        artistName = findViewById(R.id.song_artist)
        nextBtn = findViewById(R.id.id_next)
        prevBtn = findViewById(R.id.id_prev)
        shuffleBtn = findViewById(R.id.id_shuffle)
        repeatBtn = findViewById(R.id.id_repeat)
        backBtn = findViewById(R.id.back_btn)
        playPauseBtn = findViewById(R.id.play_pause)
        seekBar = findViewById(R.id.seekBar)

    }

    private fun nextTrack() {// Функція для переходу до наступного треку
        if (position < listSongs.size - 1) {
            position++
            playTrack()
        }
    }

    private fun previousTrack() {// Функція для переходу до попереднього
треку
        if (position > 0) {
            position--
            playTrack()
        }
    }

    private fun playTrack() {
        if (mediaPlayer.isPlaying) {
            mediaPlayer.stop()
            mediaPlayer.release()
        }

        uri = Uri.parse(listSongs[position].thisPath)
        mediaPlayer = MediaPlayer.create(this, uri)
        mediaPlayer.isLooping = isLooping
        mediaPlayer.start()

        playPauseBtn.setImageResource(R.drawable.pause)
        seekBar.max = mediaPlayer.duration / 1000
        songName.text = listSongs[position].thisTitle
        artistName.text = listSongs[position].thisArtist
        durationTotal.text = formatTime(mediaPlayer.duration / 1000)

        updateSeekBar()
        setCompletionListener()
    }

    private fun returnToPreviousActivity() {
        finish()
    }

    override fun onDestroy() {// Функція для вивільнення ресурсів
        super.onDestroy()
        mediaPlayer.release()
        handler.removeCallbacksAndMessages(null)
```

```
    }
}
```

MusicFiles.kt

```
package com.example.music_app


class MusicFiles(private val path:String, private val title:String, private
val artist:String
                 , private val album:String, private val duration:String, val
id: Int? = null){

    val thisTitle:String
        get() = title

    val thisPath:String
        get() = path

    val thisArtist:String
        get() = artist

    val thisAlbum:String
        get() = album

    val thisDuration:String
        get() = duration

    val thisId:Int?
        get() = id
}
```

MusicAdapter.kt

```
package com.example.music_app.adapters

import android.content.Context
import android.content.Intent
import android.media.MediaScannerConnection
import android.util.Log
import android.view.LayoutInflater
import android.view.MenuItem
import android.view.View
import android.view.ViewGroup
import android.widget.Filter
import android.widget.Filterable
import android.widget.ImageView
import android.widget.PopupMenu
import android.widget.TextView
import android.widget.Toast
import androidx.appcompat.app.AlertDialog
import androidx.recyclerview.widget.RecyclerView
import com.example.music_app.PlayerActivity
import com.example.music_app.R
import com.example.music_app.MusicFiles
import com.example.music_app.PlaylistFile
import com.example.music_app.helpers.DBHelper
import java.io.File
import java.util.Locale

class MusicAdapter(private val mContext: Context?,
                   private var mFiles:ArrayList<MusicFiles>,
```

```
                    private val isInPlaylist: Boolean = false,
                    private val playlistId: Int = -1):
    RecyclerView.Adapter<MusicAdapter.MyViewHolder>(), Filterable{

    private var musicFilesFull: ArrayList<MusicFiles> = ArrayList(mFiles)
    private var filteredList = ArrayList<MusicFiles>()

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
MyViewHolder {
        val view:View =
LayoutInflater.from(mContext).inflate(R.layout.music_item, parent, false)
        return MyViewHolder(view)
    }

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

    override fun onBindViewHolder(holder: MyViewHolder, position: Int) {
        holder.fileName.text = mFiles[position].thisTitle
        holder.itemView.setOnClickListener{
            val intent = Intent(mContext, PlayerActivity::class.java)
            intent.putExtra("position", position)
            if (isInPlaylist){
                intent.putExtra("isInPlaylist", isInPlaylist)
                intent.putExtra("playlistId", playlistId)
            }
            mContext?.startActivity(intent)
        }
        holder.optionsButton.setOnClickListener {
            showPopupMenu(holder.optionsButton, position)
        }
    }

    private fun showPopupMenu(view: View, position: Int) {
        val popupMenu = PopupMenu(mContext, view)
        popupMenu.inflate(R.menu.music_item_menu)

        popupMenu.setOnMenuItemClickListener { item: MenuItem ->
            when (item.itemId) {
                R.id.menu_delete -> {
                    showDeleteConfirmationDialog(mFiles[position], position)
                    true
                }
                R.id.menu_add_to_playlist -> {
                    showAddToPlaylistDialog(mFiles[position])
                    true
                }
                else -> false
            }
        }
        popupMenu.show()
    }

    private fun showAddToPlaylistDialog(musicFile: MusicFiles) {
        val dbHelper = DBHelper(mContext!!)
        val playlists = dbHelper.getAllPlaylists()

        val playlistNames = playlists.map { it.thisName }.toTypedArray()

        val builder = AlertDialog.Builder(mContext)
        builder.setTitle("Choose a playlist")
        builder.setItems(playlistNames) { dialog, which ->
```

```
            val selectedPlaylist = playlists[which]
            addMusicToPlaylist(musicFile, selectedPlaylist)
            dialog.dismiss()
        }
        builder.setNegativeButton("Cancel") { dialog, _ ->
            dialog.dismiss()
        }
        val dialog = builder.create()
        dialog.show()
    }

    private fun addMusicToPlaylist(musicFile: MusicFiles, playlist:
PlaylistFile) {
        Log.d("addMusicToPlaylist", "Function called")
        val dbHelper = DBHelper(mContext!!)
        val playlistId = dbHelper.getPlaylistIdByName(playlist.thisName)
        val mFileId = dbHelper.getMusicFileIdByTitle(musicFile.thisTitle)

        Log.d("addMusicToPlaylist", "playlistId: $playlistId, mFileId:
$mFileId")

        if (playlistId != -1 && mFileId != -1) {
            dbHelper.addMusicToPlaylist(playlistId, mFileId)
            Toast.makeText(mContext, "Track added to playlist
${playlist.thisName}", Toast.LENGTH_SHORT).show()
        } else {
            Toast.makeText(mContext, "Error adding to playlist",
Toast.LENGTH_SHORT).show()
        }
    }


    private fun showDeleteConfirmationDialog(musicFile: MusicFiles, position:
Int) {
        val builder = AlertDialog.Builder(mContext!!)
        builder.setTitle("Delete Confirmation")
        builder.setMessage("Are you sure you want to delete
${musicFile.thisTitle}?")

        builder.setPositiveButton("Yes") { dialog, _ ->
            if (isInPlaylist) {
                removeMusicFromPlaylist(musicFile, position)
            } else {
                deleteMusicFile(musicFile, position)
            }
            dialog.dismiss()
        }

        builder.setNegativeButton("No") { dialog, _ ->
            dialog.dismiss()
        }

        val alertDialog = builder.create()
        alertDialog.show()
    }

    private fun removeMusicFromPlaylist(musicFile: MusicFiles, position: Int)
{
        val dbHelper = DBHelper(mContext!!)
        dbHelper.deleteMusicFromPlaylist(playlistId, musicFile.thisId ?: -1)

        mFiles.removeAt(position)
        notifyItemRemoved(position)
```

```
        notifyItemRangeChanged(position, mFiles.size)
    }

    private fun deleteMusicFile(musicFile: MusicFiles, position: Int) {
        // Deleting a file from disk
        val file = File(musicFile.thisPath)
        if (file.exists() && file.delete()) {
            // Removing a record from the database
            val dbHelper = DBHelper(mContext!!)
            dbHelper.deleteMusicFile(musicFile)

            // Media scanner notification
            MediaScannerConnection.scanFile(mContext,
arrayOf(musicFile.thisPath), null) { path, uri ->
                Log.d("MusicAdapter", "File deleted: $path, URI: $uri")
            }

            // Removing an item from a list and notifying the adapter
            mFiles.removeAt(position)
            notifyItemRemoved(position)
            notifyItemRangeChanged(position, mFiles.size)
        } else {
            Log.e("MusicAdapter", "Failed to delete file:
${musicFile.thisPath}")
        }
    }

    class MyViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val fileName:TextView = itemView.findViewById(R.id.music_file_name)
        val optionsButton: ImageView =
itemView.findViewById(R.id.button_options)
    }

    override fun getFilter(): Filter {
        return object : Filter() {
            override fun performFiltering(constraint: CharSequence?):
FilterResults {
                filteredList = ArrayList<MusicFiles>()
                val searchQuery =
constraint?.toString()?.lowercase(Locale.ROOT) ?: ""

                if (searchQuery.isEmpty()) {
                    filteredList.addAll(musicFilesFull)
                } else {
                    for (musicFile in musicFilesFull) {
                        if
(musicFile.thisTitle.lowercase(Locale.ROOT).contains(searchQuery)) {
                            filteredList.add(musicFile)
                        }
                    }
                }

                val filterResults = FilterResults()
                filterResults.values = filteredList
                return filterResults
            }

            override fun publishResults(constraint: CharSequence?, results:
FilterResults?) {
                mFiles.clear()
                if (results != null && results.values is ArrayList<*>) {
                    mFiles.addAll(results.values as ArrayList<MusicFiles>)
```

```
                    Log.d("musicAdapter", "Filtered List Size:
${filteredList.size}")
                    for (item in filteredList) {
                        Log.d("musicAdapter", "Filtered Item:
${item.thisTitle}")
                    }
                }
                notifyDataSetChanged()
            }
        }
    }


}
```

ViewPagerAdapter.kt

```
package com.example.music_app.adapters

import androidx.fragment.app.Fragment
import androidx.fragment.app.FragmentActivity
import androidx.fragment.app.FragmentManager
import androidx.fragment.app.FragmentPagerAdapter
import androidx.viewpager2.adapter.FragmentStateAdapter


class ViewPagerAdapter(fa:FragmentActivity, private val list: List<Fragment>)
: FragmentStateAdapter(fa) {
    override fun getItemCount(): Int {
        return list.size
    }

    override fun createFragment(position: Int): Fragment {
        return list[position]
    }


}
```

SongsFragment.kt

```
package com.example.music_app.fragments

import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.example.music_app.MainActivity
import com.example.music_app.R
import com.example.music_app.adapters.MusicAdapter


class SongsFragment() : Fragment() {
```

```
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        val view:View = inflater.inflate(R.layout.fragment_songs, container,
false)
        val recyclerView:RecyclerView = view.findViewById(R.id.recyclerView)
        recyclerView.setHasFixedSize(true)
        val mainActivity = activity as MainActivity
        val musicFiles = mainActivity.getMusicFiles()
        if (musicFiles.size >= 1) {
            val musicAdapter = MusicAdapter(context,musicFiles)
            recyclerView.adapter = musicAdapter
            recyclerView.layoutManager = LinearLayoutManager(context,
RecyclerView.VERTICAL, false)
        } else {

        }

        return view
    }

    companion object {

        @JvmStatic
        fun newInstance() = SongsFragment()
    }
}
```

PlaylistFragment.kt

```
package com.example.music_app.fragments

import android.content.DialogInterface
import android.content.Intent
import android.os.Bundle
import android.util.Log
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.EditText
import android.widget.ImageView
import androidx.appcompat.app.AlertDialog
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.example.music_app.PlaylistDetailActivity
import com.example.music_app.PlaylistFile
import com.example.music_app.R
import com.example.music_app.adapters.PlaylistAdapter
import com.example.music_app.helpers.DBHelper


class PlaylistFragment : Fragment() {

    private lateinit var recyclerView: RecyclerView
    private lateinit var addButton: ImageView
    private lateinit var dbHelper: DBHelper
    private lateinit var playlistAdapter: PlaylistAdapter
    private lateinit var playlists: ArrayList<PlaylistFile>
```

```
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val view:View = inflater.inflate(R.layout.fragment_playlist,
container, false)
        recyclerView= view.findViewById(R.id.recyclerPlaylist)
        addButton = view.findViewById(R.id.add_button)
        recyclerView.setHasFixedSize(true)
        recyclerView.layoutManager = LinearLayoutManager(context)

        dbHelper = DBHelper(requireContext())
        playlists = dbHelper.getAllPlaylists()

        playlistAdapter = PlaylistAdapter(playlists, { playlistFile ->
            val playlistId =
dbHelper.getPlaylistIdByName(playlistFile.thisName)
            val intent = Intent(requireContext(),
PlaylistDetailActivity::class.java)
            intent.putExtra("playlistId", playlistId)
            intent.putExtra("playlistName", playlistFile.thisName)
            startActivity(intent)
        }, { playlistFile ->// For deleting playlist
            deletePlaylist(playlistFile)
        })
        recyclerView.adapter = playlistAdapter

        addButton.setOnClickListener {
            // For adding new playlist
            showAddPlaylistDialog()
        }

        return view
    }

    private fun showAddPlaylistDialog() {
        val dialogBuilder = AlertDialog.Builder(requireContext())
        val inflater = requireActivity().layoutInflater
        val dialogView = inflater.inflate(R.layout.dialog_add_playlist, null)
        dialogBuilder.setView(dialogView)

        dialogBuilder.setTitle("Add Playlist")
        dialogBuilder.setMessage("Enter the name of the new playlist:")

        dialogBuilder.setPositiveButton("Add") { dialogInterface:
DialogInterface, i: Int ->
            val editTextPlaylistName =
dialogView.findViewById<EditText>(R.id.editTextPlaylist)
            val playlistName = editTextPlaylistName.text.toString()
            if (playlistName.isNotBlank()) {
                dbHelper.addPlaylist(PlaylistFile(playlistName))
                playlists.clear()
                playlists.addAll(dbHelper.getAllPlaylists())
                playlistAdapter.notifyDataSetChanged()
            }
        }

        dialogBuilder.setNegativeButton("Cancel") { dialogInterface:
DialogInterface, i: Int ->
            dialogInterface.dismiss()
        }
```

```
        val alertDialog = dialogBuilder.create()
        alertDialog.show()
    }

    private fun deletePlaylist(playlistFile: PlaylistFile) {
        val playlistId = dbHelper.getPlaylistIdByName(playlistFile.thisName)
        dbHelper.deletePlaylist(playlistId)
        playlists.clear()
        playlists.addAll(dbHelper.getAllPlaylists())
        playlistAdapter.notifyDataSetChanged()
    }

    companion object {

        @JvmStatic
        fun newInstance() = PlaylistFragment()


    }
}
```

PlaylistAdapter.kt

```
package com.example.music_app.adapters

import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ImageView
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.example.music_app.MusicFiles
import com.example.music_app.PlaylistFile
import com.example.music_app.R

class PlaylistAdapter(
    private val playlist: ArrayList<PlaylistFile>,
    private val onClickListener: (PlaylistFile) -> Unit,
    private val onDeleteClickListener: (PlaylistFile) -> Unit
): RecyclerView.Adapter<PlaylistAdapter.PlaylistViewHolder>() {

    class PlaylistViewHolder(itemView: View) :
RecyclerView.ViewHolder(itemView) {
        val titleTextView: TextView =
itemView.findViewById(R.id.playlist_name)
        val deleteButton: ImageView =
itemView.findViewById(R.id.button_delete)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
PlaylistViewHolder {
        val view =
LayoutInflater.from(parent.context).inflate(R.layout.playlist_item, parent,
false)
        return PlaylistViewHolder(view)
    }

    override fun onBindViewHolder(holder: PlaylistViewHolder, position: Int)
{
        val playlistFile = playlist[position]
        holder.titleTextView.text = playlistFile.thisName
        holder.itemView.setOnClickListener { onClickListener(playlistFile) }
        holder.deleteButton.setOnClickListener {
```

```
onDeleteClickListener(playlistFile) }
    }

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

DBHelper.kt

```
package com.example.music_app.helpers

import android.content.ContentValues
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper
import android.util.Log
import com.example.music_app.MusicFiles
import com.example.music_app.PlaylistFile

class DBHelper(context: Context):SQLiteOpenHelper(context, DATABASE_NAME,
null, DATABASE_VERSION) {

    companion object {
        const val DATABASE_VERSION = 2
        const val DATABASE_NAME = "MusicDatabase"
        const val TABLE_MUSIC_FILES = "MusicFiles"
        const val TABLE_PLAYLISTS = "Playlists"
        const val TABLE_PLAYLIST_MUSIC = "PlaylistMusic"
        const val KEY_ID = "_id"
        const val KEY_PATH = "path"
        const val KEY_TITLE = "title"
        const val KEY_ARTIST = "artist"
        const val KEY_ALBUM = "album"
        const val KEY_DURATION = "duration"
        const val KEY_PLAYLIST_ID = "playlist_id"
        const val KEY_MUSIC_ID = "music_id"
        const val KEY_PLAYLIST_NAME = "playlist_name"
    }

    override fun onCreate(db: SQLiteDatabase?) {
        db?.execSQL("CREATE TABLE $TABLE_MUSIC_FILES ($KEY_ID INTEGER PRIMARY
KEY, $KEY_PATH TEXT, $KEY_TITLE TEXT, $KEY_ARTIST TEXT, $KEY_ALBUM TEXT,
$KEY_DURATION TEXT)")
        db?.execSQL("CREATE TABLE $TABLE_PLAYLISTS ($KEY_ID INTEGER PRIMARY
KEY, $KEY_PLAYLIST_NAME TEXT)")
        db?.execSQL("CREATE TABLE $TABLE_PLAYLIST_MUSIC ($KEY_ID INTEGER
PRIMARY KEY, $KEY_PLAYLIST_ID INTEGER, $KEY_MUSIC_ID INTEGER, FOREIGN
KEY($KEY_PLAYLIST_ID) REFERENCES $TABLE_PLAYLISTS($KEY_ID), FOREIGN
KEY($KEY_MUSIC_ID) REFERENCES $TABLE_MUSIC_FILES($KEY_ID))")
    }

    override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion:
Int) {
        db?.execSQL("DROP TABLE IF EXISTS $TABLE_MUSIC_FILES")
        db?.execSQL("DROP TABLE IF EXISTS $TABLE_PLAYLISTS")
        db?.execSQL("DROP TABLE IF EXISTS $TABLE_PLAYLIST_MUSIC")
        onCreate(db)
    }

    //For table TABLE_MUSIC_FILES
```

```
    fun addMusicFile(musicFile: MusicFiles) {
        val db = this.writableDatabase
        val values = ContentValues().apply {
            put(KEY_PATH, musicFile.thisPath)
            put(KEY_TITLE, musicFile.thisTitle)
            put(KEY_ARTIST, musicFile.thisArtist)
            put(KEY_ALBUM, musicFile.thisAlbum)
            put(KEY_DURATION, musicFile.thisDuration)
        }
        Log.d("MusicDatabaseHelper", "Adding music file to database:
$values")
        db.insert(TABLE_MUSIC_FILES, null, values)
        db.close()
    }

    fun getMusicFileIdByTitle(title: String): Int {
        val db = this.readableDatabase
        val cursor: Cursor? = db.rawQuery("SELECT $KEY_ID FROM
$TABLE_MUSIC_FILES WHERE $KEY_TITLE=?", arrayOf(title))
        var musicFileId = -1
        cursor?.use {
            if (it.moveToFirst()) {
                musicFileId = it.getInt(it.getColumnIndexOrThrow(KEY_ID))
            }
        }
        cursor?.close()
        return musicFileId
    }

    fun getAllMusicFiles(): ArrayList<MusicFiles> {
        val musicFilesList = ArrayList<MusicFiles>()
        val selectQuery = "SELECT * FROM $TABLE_MUSIC_FILES"
        val db = this.readableDatabase
        val cursor: Cursor? = db.rawQuery(selectQuery, null)
        cursor?.use {
            if (it.moveToFirst()) {
                val pathIndex = it.getColumnIndex(KEY_PATH)
                val titleIndex = it.getColumnIndex(KEY_TITLE)
                val artistIndex = it.getColumnIndex(KEY_ARTIST)
                val albumIndex = it.getColumnIndex(KEY_ALBUM)
                val durationIndex = it.getColumnIndex(KEY_DURATION)

                do {
                    val path = if (pathIndex != -1) it.getString(pathIndex)
else ""
                    val title = if (titleIndex != -1)
it.getString(titleIndex) else ""
                    val artist = if (artistIndex != -1)
it.getString(artistIndex) else ""
                    val album = if (albumIndex != -1)
it.getString(albumIndex) else ""
                    val duration = if (durationIndex != -1)
it.getString(durationIndex) else ""

                    val musicFile = MusicFiles(path, title, artist, album,
duration)
                    musicFilesList.add(musicFile)
                } while (it.moveToNext())
            }
        }
        cursor?.close()
        return musicFilesList
    }
```

```
    fun deleteMusicFile(musicFile: MusicFiles) {
        val db = this.writableDatabase
        db.delete(TABLE_MUSIC_FILES, "$KEY_PATH=?",
arrayOf(musicFile.thisPath))
        db.close()
    }

    fun clearMusicFiles() {
        val db = this.writableDatabase
        db.delete(TABLE_MUSIC_FILES, null, null)
        db.close()
    }

    //For table TABLE_PLAYLIST_FILES

    fun addPlaylist(playlist: PlaylistFile) {
        val db = this.writableDatabase
        val values = ContentValues().apply {
            put(KEY_PLAYLIST_NAME, playlist.thisName)
        }
        db.insert(TABLE_PLAYLISTS, null, values)
        db.close()
    }

    fun addMusicToPlaylist(playlistId: Int, musicId: Int) {
        val db = this.writableDatabase
        val values = ContentValues().apply {
            put(KEY_PLAYLIST_ID, playlistId)
            put(KEY_MUSIC_ID, musicId)
        }
        db.insert(TABLE_PLAYLIST_MUSIC, null, values)
        db.close()
    }

    fun getAllPlaylists(): ArrayList<PlaylistFile> {
        val playlists = ArrayList<PlaylistFile>()
        val db = this.readableDatabase
        val cursor: Cursor? = db.rawQuery("SELECT * FROM $TABLE_PLAYLISTS",
null)
        cursor?.use {
            if (it.moveToFirst()) {
                do {
                    val name =
it.getString(it.getColumnIndexOrThrow(KEY_PLAYLIST_NAME))
                    playlists.add(PlaylistFile(name))
                } while (it.moveToNext())
            }
        }
        cursor?.close()
        return playlists
    }

    fun getPlaylistIdByName(playlistName: String): Int {
        val db = this.readableDatabase
        val cursor: Cursor? = db.rawQuery("SELECT $KEY_ID FROM
$TABLE_PLAYLISTS WHERE $KEY_PLAYLIST_NAME=?", arrayOf(playlistName))
        var playlistId = -1
        cursor?.use {
            if (it.moveToFirst()) {
                playlistId = it.getInt(it.getColumnIndexOrThrow(KEY_ID))
            }
```

```
        }
        cursor?.close()
        return playlistId
    }

    fun getPlaylistFiles(playlistId: Int): ArrayList<MusicFiles> {
        val musicFiles = ArrayList<MusicFiles>()
        val db = this.readableDatabase
        val cursor: Cursor? = db.rawQuery("SELECT mf.* FROM
$TABLE_MUSIC_FILES mf INNER JOIN $TABLE_PLAYLIST_MUSIC pm ON mf.$KEY_ID =
pm.$KEY_MUSIC_ID WHERE pm.$KEY_PLAYLIST_ID = ?",
arrayOf(playlistId.toString()))
        cursor?.use {
            if (it.moveToFirst()) {
                do {
                    val id = it.getInt(it.getColumnIndexOrThrow(KEY_ID))
                    val path =
it.getString(it.getColumnIndexOrThrow(KEY_PATH))
                    val title =
it.getString(it.getColumnIndexOrThrow(KEY_TITLE))
                    val artist =
it.getString(it.getColumnIndexOrThrow(KEY_ARTIST))
                    val album =
it.getString(it.getColumnIndexOrThrow(KEY_ALBUM))
                    val duration =
it.getString(it.getColumnIndexOrThrow(KEY_DURATION))
                    musicFiles.add(MusicFiles(path, title, artist, album,
duration, id))
                } while (it.moveToNext())
            }
        }
        cursor?.close()
        return musicFiles
    }

    fun deletePlaylist(playlistId: Int) {
        val db = this.writableDatabase
        db.delete(TABLE_PLAYLISTS, "$KEY_ID=?",
arrayOf(playlistId.toString()))
        db.delete(TABLE_PLAYLIST_MUSIC, "$KEY_PLAYLIST_ID=?",
arrayOf(playlistId.toString()))
        db.close()
    }

    fun deleteMusicFromPlaylist(playlistId: Int, musicId: Int) {
        val db = this.writableDatabase
        db.delete(TABLE_PLAYLIST_MUSIC, "$KEY_PLAYLIST_ID=? AND
$KEY_MUSIC_ID=?", arrayOf(playlistId.toString(), musicId.toString()))
        db.close()
    }


    fun clearDatabase() {
        val db = this.writableDatabase
        db.delete(TABLE_MUSIC_FILES, null, null)
        db.delete(TABLE_PLAYLISTS, null, null)
        db.delete(TABLE_PLAYLIST_MUSIC, null, null)
        db.close()
    }

}
```

ConvertVideoToAudio.kt

```
package com.example.music_app

import android.content.Context
import android.content.Intent
import android.media.MediaExtractor
import android.media.MediaFormat
import android.media.MediaMuxer
import android.provider.MediaStore
import androidx.activity.result.ActivityResultLauncher
import androidx.activity.result.contract.ActivityResultContracts
import java.io.File
import java.io.IOException
import java.nio.ByteBuffer

class ConvertVideoToAudio() {

    private lateinit var resultLauncher: ActivityResultLauncher<Intent>

    fun convertVideoToAudio(context: Context, inputFilePath: String,
outputFilePath: String) {
        val extractor = MediaExtractor()
        try {
            extractor.setDataSource(inputFilePath)
        } catch (e: IOException) {
            throw RuntimeException("Error setting data source: ${e.message}")
        }

        val numTracks = extractor.trackCount
        var audioTrackIndex = -1
        var format: MediaFormat? = null

        for (i in 0 until numTracks) {
            format = extractor.getTrackFormat(i)
            val mime = format.getString(MediaFormat.KEY_MIME)
            if (mime != null && mime.startsWith("audio/")) {
                audioTrackIndex = i
                break
            }
        }

        if (audioTrackIndex == -1) {
            throw IllegalArgumentException("Audio track not found in
$inputFilePath")
        }

        extractor.selectTrack(audioTrackIndex)

        val output = File(outputFilePath)
        val muxer = MediaMuxer(output.absolutePath,
MediaMuxer.OutputFormat.MUXER_OUTPUT_MPEG_4)
        val outputTrackIndex = muxer.addTrack(format!!)
        muxer.start()

        val buffer = ByteBuffer.allocate(1024 * 1024)
        val bufferInfo = android.media.MediaCodec.BufferInfo()

        try {
            while (true) {
                bufferInfo.offset = 0
                bufferInfo.size = extractor.readSampleData(buffer, 0)
                if (bufferInfo.size < 0) {
                    bufferInfo.size = 0
                    break
```

```
                } else {
                    bufferInfo.presentationTimeUs = extractor.sampleTime
                    bufferInfo.flags = extractor.sampleFlags
                    muxer.writeSampleData(outputTrackIndex, buffer,
bufferInfo)
                    extractor.advance()
                }
            }
        } catch (e: Exception) {
            throw RuntimeException("Error converting video to audio:
${e.message}")
        } finally {
            muxer.stop()
            muxer.release()
            extractor.release()
        }
    }

}
```

PlaylistDetailActivity.kt

```
package com.example.music_app

import android.os.Bundle
import android.util.Log
import android.widget.ImageView
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.example.music_app.MusicFiles
import com.example.music_app.R
import com.example.music_app.adapters.MusicAdapter
import com.example.music_app.helpers.DBHelper

class PlaylistDetailActivity : AppCompatActivity() {

    private lateinit var recyclerView: RecyclerView
    private lateinit var backBtn: ImageView
    private lateinit var titlePlaylist: TextView
    private lateinit var dbHelper: DBHelper
    private lateinit var musicAdapter: MusicAdapter
    private var playlistId: Int = -1
    private var playlistName: String = ""
    private lateinit var musicFiles: ArrayList<MusicFiles>

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_playlist_detail)

        backBtn = findViewById(R.id.back_btn)
        titlePlaylist = findViewById(R.id.playlist_name)
        recyclerView = findViewById(R.id.recyclerView)
        recyclerView.setHasFixedSize(true)
        recyclerView.layoutManager = LinearLayoutManager(this)
        dbHelper = DBHelper(this)

        playlistId = intent.getIntExtra("playlistId", -1)
        playlistName = intent.getStringExtra("playlistName").toString()

        backBtn.setOnClickListener{
            returnToPreviousActivity()
```

```
        }
        titlePlaylist.text = playlistName
        musicFiles = dbHelper.getPlaylistFiles(playlistId)
        musicAdapter = MusicAdapter(this, musicFiles, isInPlaylist = true,
playlistId = playlistId)
        recyclerView.adapter = musicAdapter
    }

    private fun returnToPreviousActivity() {
        finish()
    }
}
```

PlaylistFile.kt

```
package com.example.music_app

class PlaylistFile(private val name: String) {

    val thisName: String
        get() = name
}
```

SettingsActivity.kt

```
package com.example.music_app

import android.content.SharedPreferences
import android.os.Bundle
import android.widget.ImageView
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat

class SettingsActivity : AppCompatActivity() {

    private lateinit var sharedPreferences: SharedPreferences

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_settings)

        sharedPreferences = getSharedPreferences("AppSettings", MODE_PRIVATE)
        applyTheme()

        setContentView(R.layout.activity_settings)

        val buttonLightTheme =
findViewById<ImageView>(R.id.button_theme_light)
        val buttonDarkTheme = findViewById<ImageView>(R.id.button_theme_dark)
        val backBtn = findViewById<ImageView>(R.id.back_btn)

        buttonLightTheme.setOnClickListener {
            setTheme("light")
        }

        buttonDarkTheme.setOnClickListener {
            setTheme("dark")
        }

        backBtn.setOnClickListener(){
```

```
            returnToPreviousActivity()
        }
    }

    private fun setTheme(theme: String) {
        val editor = sharedPreferences.edit()
        editor.putString("AppTheme", theme)
        editor.apply()

        recreate() // Перезапуск активности для применения темы
    }

    private fun applyTheme() {
        when (sharedPreferences.getString("AppTheme", "light")) {
            "light" -> setTheme(R.style.LightTheme)
            "dark" -> setTheme(R.style.DarkTheme)
        }
    }

    private fun returnToPreviousActivity() {
        finish()
    }
}
```