

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ МУЗИЧНОГО
ПЛЕЄРА

SOFTWARE DEVELOPMENT OF MUSIC PLAYER MOBILE
APPLICATION

Виконав: студент(ка) 4 курсу,
групи КНТ-139

Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ДОЛЖЕНКО Ф.І.

(прізвище та ініціали)

Керівник ФЕДОРОНЧАК Т.В.

(прізвище та ініціали)

Рецензент КОРОТКИЙ О.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій Субботін
“ ” 2023 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ДОЛЖЕНКА Філіппа Ігоровича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка мобільного застосунку музичного плеєра.
Software Development of Music Player Mobile Application

керівник проєкту (роботи) к.т.н., доцент, ФЕДОРОНЧАК Тетяна Василівна,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “ 04 ” квітня 2023 року
№ 87

2. Строк подання студентом проєкту (роботи) червня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно
розробити) 1. Аналіз проблеми та постановка завдань дослідження.
2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та
експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ФЕДОРОНЧАК Т.В., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., старший викладач		

7. Дата видачі завдання « 04 » квітня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Філіпп ДОЛЖЕНКО
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Тетяна ФЕДОРОНЧАК
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 107 с., 31 рис., 5 табл., 3 дод., 11 джерел.

МОБІЛЬНИЙ ЗАСТОСУНОК, ANDROID, МУЗИКА, МУЗИЧНИЙ ПЛЕЄР, KOTLIN.

Об'єкт дослідження – програмні засоби для прослуховування музики.

Предмет дослідження – мобільні застосунки для прослуховування музики.

Мета роботи – створення Android-застосунку для прослуховування музики офлайн з пам'яті пристрою з легким для розуміння інтерфейсом.

Матеріали, методи та технічні засоби: функціональне та об'єктно-орієнтоване програмування, мобільний пристрій з операційною системою Android, мова програмування Kotlin.

Результати. Створено програмний Android-застосунок, що дозволяє користувачам слухати музику з пам'яті пристрою.

Висновки. Було розроблено програмний Android-застосунок, що виконує функції музичного плеєра. Він дозволяє користувачам слухати музику з пам'яті пристрою.

Галузь використання – повсякденне життя в умовах відсутності інтернет-зв'язку. Може супроводжувати працю у побуті, використання приватного чи громадського транспорту, фізичні вправи, вільний відпочинок чи будь-які інші дії.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor:
107 pages, 31 figures, 5 tables, 3 appendixes, 11 sources.

MOBILE APPLICATION, ANDROID, MUSIC, MUSIC PLAYER,
Kotlin.

The object of study is a software for listening to music.

The subject of study are apps for listening to music.

The purpose of the work is to create an Android application for listening to music offline from the device's memory with a user friendly interface.

Materials, methods, and tools: functional and object-oriented programming, a mobile device with an Android operating system, Kotlin programming language.

Results. The Android application that allows users to listen to music from the device's memory is created.

Conclusions. The Android application for listening to music was developed. It allows users to listen to music from the device's memory.

The field of use is everyday life in the absence of an internet connection. It can accompany work at home, the use of private or public transport, exercise, leisure or any other activity.

ЗМІСТ

	С.
ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК	7
ВСТУП	8
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	9
1.1 Знайомство з предметною областю	9
1.2 Огляд існуючих рішень	9
1.3 Висновки за розділом 1	14
2 ВИБІР ТА ОБҐРУНТУВАННЯ СТРУКТУРИ СИСТЕМИ, ЯКА ПРОЕКТУЄТЬСЯ.....	15
2.1 Вибір інструментарію розробки програмного забезпечення	15
2.2 Файлова структура Android-застосунку	24
3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ....	26
3.1 Реалізація шаблонів проектування та вибір інструментів програмування.....	26
3.2 Огляд компонентів та розроблених класів.....	28
4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЗАСТОСУНКУ	44
4.1 Процес роботи застосунку	44
4.2 Тестування застосунку	54
4.3 Висновки за розділом 4	55
ВИСНОВКИ.....	56
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ	57
ДОДАТОК А Технічне завдання	58
ДОДАТОК Б Текст програми	62
ДОДАТОК В Слайди презентації.....	102

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

IDE – Integrated Development Environment;

UI – user interface;

ОС – операційна система,

Свайп – метод керування сенсорним екраном, який полягає у проведенні по ньому пальцем у певному напрямку;

ВСТУП

Музика завжди супроводжувала життя людини протягом усієї історії людства. Вона допомагала підтримувати моральний дух, пісні супроводжували як роботу, так і відпочинок. Тому не дивно, що й зараз, за часів розвитку технологій, повсюдного поширення інтернету: люди найрізноманітніших вікових категорій і верств населення продовжують слухати музику з тими самими цілями в найрізноманітніших ситуаціях.

На сьогодні найчастіше користувачі слухають музику онлайн за допомогою стрімінг-сервісів. Оскільки у більшості квартир користувачів, а також у більшості їхніх робочих або навчальних приміщень є домашній кабельний інтернет і Wi-Fi роутер, офлайн-прослуховування музики за допомогою персонального комп'ютера не є затребуваним і не потребує появи нових рішень. Але офлайн-прослуховування музики залишається доцільним для користувачів, які користуються смартфонами. Користувачі часто носять їх з собою в місцях, де немає Wi-Fi, а використання мобільного інтернету для прослуховування музики не завжди є доцільним, оскільки швидкість підключення до мережі може бути низькою, а також потокова трансляція музики може виявитися дуже витратною для мобільного трафіку і заряду акумулятора пристрою. Крім цього, покриття мобільного зв'язку та мобільного інтернету присутнє не скрізь, наприклад, воно часто відсутнє на маршрутах залізниць між містами. Нарешті, не можна забувати про те, що і Wi-Fi, і мобільний інтернет перестають працювати в разі появи проблем з електропостачанням, які, на жаль, нерідко трапляються у наш час. Усі ці проблеми вирішуються за допомогою музичного офлайн-плеєра, функціонал якого реалізовано у Dolzhify.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Знайомство з предметною областю

Основні функції застосунку – це сканування пам'яті пристрою на наявність аудіофайлів, створення з них аудіотеки та можливість прослуховування композицій. Зробивши дослідження на наявність аналогів, які вирішують схожі проблеми, можна зробити висновок, що кожен з них відрізняється як за інтерфейсом, так і за функціоналом.

1.2 Огляд існуючих рішень

Spotify [1] – це стримінговий сервіс потокового аудіо, що дозволяє прослуховувати музичні композиції та подкасти. Цей сервіс надає послуги легального онлайнового стримінгу аудіозаписів основних світових і незалежних лейблів, в тому числі BBC, Sony, EMI, Warner Music Group та Universal. «Spotify» є першим стримінговим сервісом, який надає можливість слухати музику онлайн, не завантажуючи її на пристрій. Сервіс є доступним у всіх країнах Америки (крім Куби), Європи й Океанії, а також у більшості країн Азії й Африки. Сервіс також доступний і в Україні. Сервіс має онлайн-версію, а також доступний у вигляді Desktop-застосунку та Android-застосунку. Більшість функціонала сервісу доступна лише за Premium-підпискою. Інтерфейс Spotify для ОС Android наведено на рис. 1.1.

Samsung Music [2] – це застосунок-музичний плеєр, оптимізований для Android-пристроїв Samsung, що забезпечує потужну функціональність відтворення музики та зручний користувацький інтерфейс. Він підтримує відтворення різних звукових форматів, таких як MP3, AAC та FLAC, але підтримувані формати файлів можуть відрізнятися залежно від пристрою. Цей застосунок допомагає ефективно керувати списками композицій за категоріями (доріжка, альбом, виконавець, жанр, директорія, композитор).

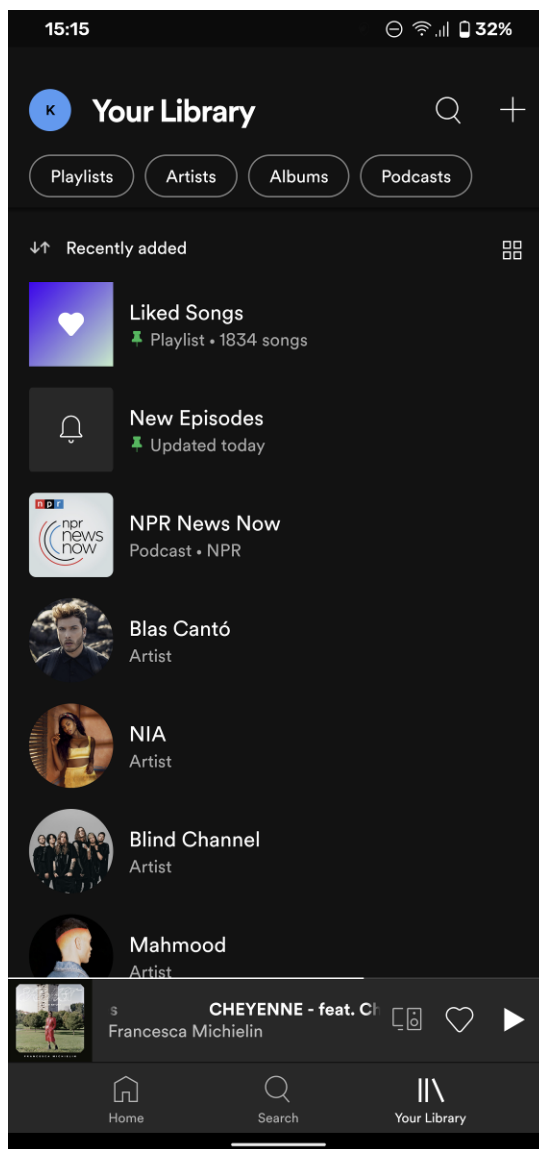


Рисунок 1.1 – Інтерфейс Android-застосунку Spotify

Він підтримує користувацький інтерфейс, який легко взаємодіє з інтелектуальними пристроями Samsung, як-от планшет, телевізор (за допомогою технології DLNA або за допомогою дублювання екрана) та переносні пристрої. Також Samsung Music інтегрований зі Spotify – він може показувати рекомендації плейлистів з цього сервісу, але тільки в тих країнах, де працює Spotify. Головна вада Samsung Music полягає в його ексклюзивності для пристроїв Samsung, тобто він не може бути встановлений на пристрої будь-яких інших виробників. Також у цьому плеєрі наявна

реклама, яку не можна відключити, хоча застосунок і не потребує інтернет-з'єднання. Інтерфейс Samsung Music наведено на рис. 1.2.

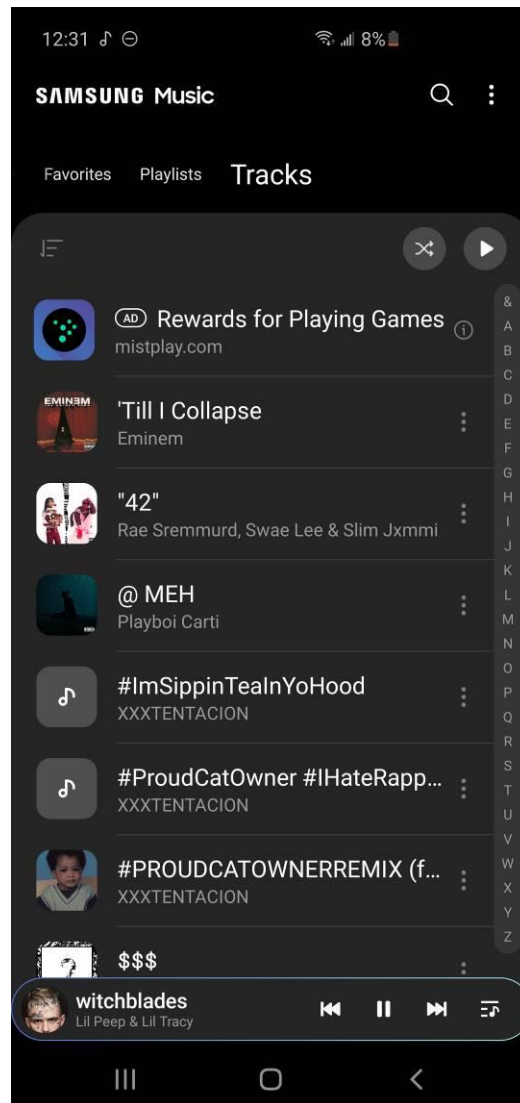


Рисунок 1.2 – Інтерфейс Android-застосунку Samsung Music

VLC for Android [3] – це порт безкоштовного крос-платформного мультимедійного плеєра VLC на платформу Android. Він який може відтворювати будь-які відео- та аудіофайли, включаючи MKV, MP4, AVI, MOV, Ogg, FLAC, TS, M2TS, WAV та AAC. Усі кодеки включено до складу програми, їх не потрібно завантажувати окремо. Застосунок підтримує багатодоріжковий звук, субтитри, телетекст і закриті субтитри, автоповорот, регулювання співвідношення сторін і жести для керування гучністю,

яскравістю і пошуком. Він також містить віджет для керування звуком, підтримує керування аудіо-гарнітурами, обкладинки та повну аудіо-медіа-бібліотеку.

VLC для Android також може відтворювати мережеві потоки (включно з адаптивним потоком), DVD ISO, як і настільна версія VLC. Він також підтримує спільний доступ до дисків.

VLC для Android має базу даних, еквалайзер та фільтри для аудіозаписів, медіатеку для аудіо- та відеофайлів, а також дозволяє переглядати папки безпосередньо. VLC є безкоштовним, в ньому відсутня реклама та мікротранзакції. Застосунок розробляється спільнотою волонтерів та має відкритий вихідний код. Інтерфейс VLC for Android наведено на рис. 1.3.

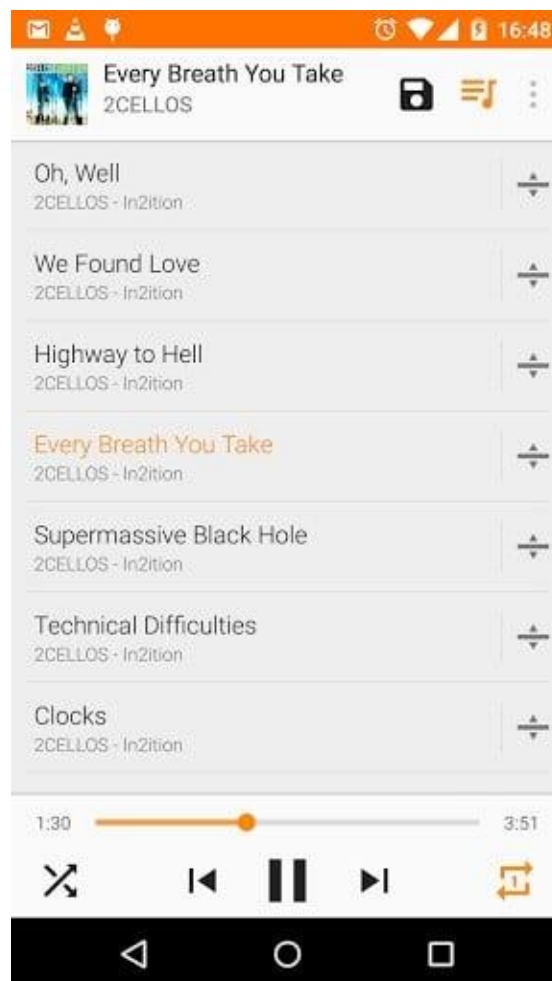


Рисунок 1.3 – Інтерфейс застосунку VLC for Android

Порівняння особливостей аналогів застосунку, що розроблюється, наведено у табл. 1.1.

Таблиця 1.1 – Порівняння програмних продуктів

Критерій оцінки	Назва застосунку		
	Spotify	Samsung Music	VAC for Android
1	2	3	4
Інтуїтивна зрозумілість	+++	++	+
Зручність	++	+	+
Сучасність дизайну	+++	+++	+
Платний функціонал	Підписка Premium (10\$)	Немає	Немає
Наявність реклами	Є, якщо нема підписки	Є	Немає
Споживання пам'яті	Від 134 МБ	Від 58 МБ	Від 104 МБ

Можна зазначити, що Spotify має сучасний, приємний та інтуїтивно зрозумілий інтерфейс, але при цьому більша частина його функціонала є платною, присутня реклама для користувачів, що не мають підписки, а також він споживає дуже багато пам'яті пристрою.

Samsung Music повністю безкоштовний, але підтримується лише пристроями Samsung та також містить рекламу.

Найкращим з функціоналу є VAC for Android, але при цьому він виглядає не дуже сучасно, його інтерфейс не є інтуїтивно зрозумілим та зручним.

1.3 Висновки за розділом 1

У розділі 1 було проведено аналіз предметної області розробки, а також було проведено детальний аналіз трьох найпопулярніших застосунків, було виявлено переваги та недоліки кожного з них та було описано вирішення їх у застосунку, що розроблюється. Його розробку можна вважати доцільною, тому що у ньому реалізовано усі функції, що потрібні для зручного прослуховування музики.

2 ВИБІР ТА ОБҐРУНТУВАННЯ СТРУКТУРИ СИСТЕМИ, ЯКА ПРОЕКТУЄТЬСЯ

У цьому розділі розглянуто основні особливості мов програмування, вибір інструментарію та огляд структури застосунку.

2.1 Вибір інструментарію розробки програмного забезпечення

2.1.1 Вибір операційної системи, для якої розроблюється застосунок

Станом на квітень 2023 року найпопулярнішими операційними системами для смартфонів [4] є Android (68,61% пристроїв) та iOS (30,61% пристроїв), частка пристроїв з іншими ОС складає менше ніж 0,78%, тож їх можна не враховувати, і вибір доводиться робити між Android та iOS. Діаграму зі статистикою популярності мобільних ОС наведено на рис. 2.1.

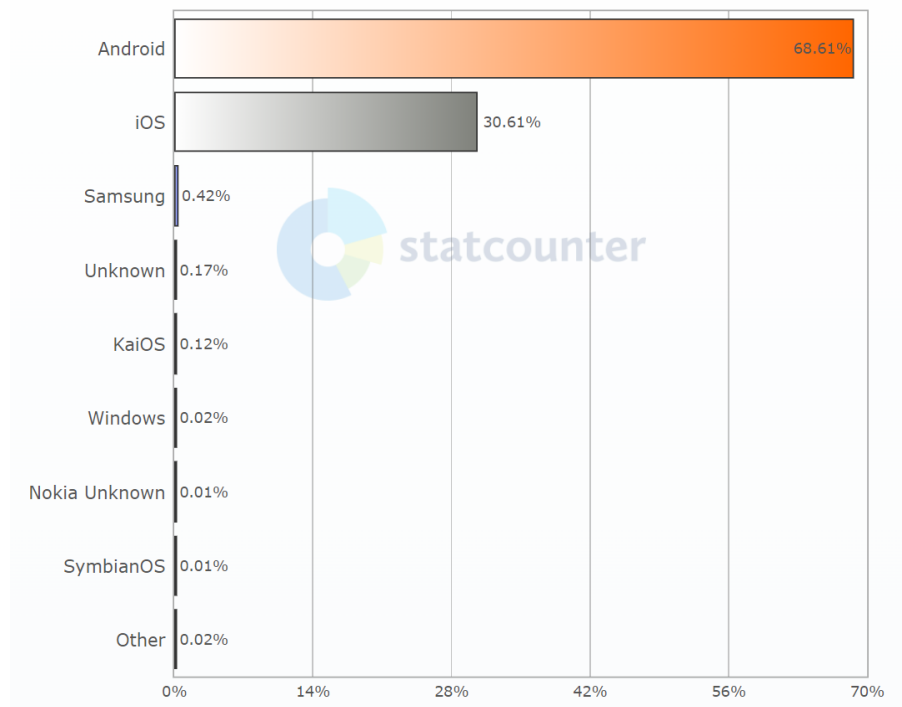


Рисунок 2.1 – Статистика глобального ринку мобільних операційних систем станом на квітень 2023 року

Android [5] – це найпопулярніша операційна система у світі, з понад 2,5 мільярдами активних користувачів у більш ніж 190 країнах. Створена Енді Рубіном як альтернатива iPhone та Palm OS з відкритим вихідним кодом, Android швидко стала улюбленою операційною системою для більшості виробників мобільних пристроїв на початку 2010-х років. Спочатку вона розроблялася компанією Android, Inc., яку потім придбала Google. Згодом Google ініціювала створення альянсу Open Handset Alliance (ОНА), який займається підтримкою та подальшим розвитком платформи.

Android заснована на ядрі Linux та на власній реалізації віртуальної машини Java під назвою Dalvik [6]. Android дає змогу запускати Java-застосунки, що керують пристроєм через розроблені Google бібліотеки. Android Native Development Kit дає змогу портувати бібліотеки та компоненти застосунків, написані мовою програмування C та іншими мовами.

Завдяки тому, що такі титани індустрії, як Samsung, LG, Motorola і HTC, випустили телефони під управлінням Android, вона швидко стала найпопулярнішою мобільною ОС, досягнувши понад мільярд активних користувачів до 2014 року. Android є домінуючою платформою в більшості країн, хоча в Японії та Сполучених Штатах вона має проблеми з випередженням Apple. У таких країнах, як Бразилія, Індія, Індонезія, Іран та Туреччина, вона займає понад 85% ринку. Вбудований в Android магазин застосунків Google Play значно зріс за останнє десятиліття, досягнувши 38,6 мільярда доларів доходу у 2020 році. У 2020 році в магазині було доступно понад 2,9 мільйона застосунків, які завантажили 108 мільярдів разів.

Розробники активно продовжують розвивати Android, регулярно виходять нові версії цієї ОС. Остання з них на поточний момент, Android 13 Tiramisu, була випущена 15 серпня 2022 року.

При розробці застосунку для операційної системи Android наявні як переваги, так і недоліки.

Перевагами є великий обсяг технічної документації та засобів розробки, також для розробки не обов'язково мати пристрій з операційною системою Android, тому що існують офіційні емулятори, завдяки яким можна відлагоджувати застосунок, використовуючи лише персональний комп'ютер.

До недоліків Android як платформи для розробки застосунків можна віднести часті оновлення систем, які можуть стати причиною проблем з сумісністю застосунку. Для підтримки різних версій Android обсяг коду застосунку можна значно збільшуватися, також це ускладнює процес розробки та підтримки застосунків.

Вигляд домашньої сторінки Android наведено на рис. 2.2.

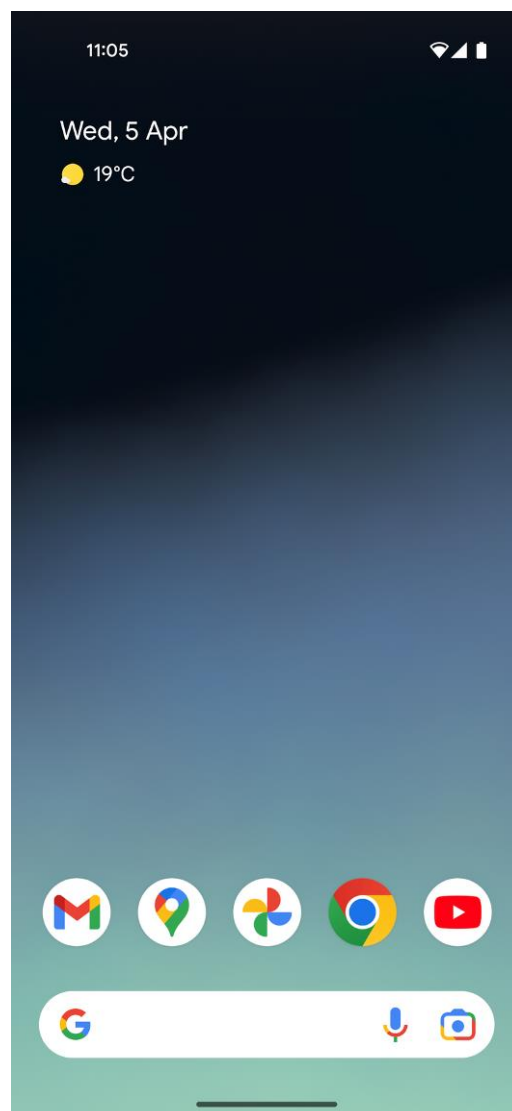


Рисунок 2.2 – Домашня сторінка Android 13 Tiramisu

iOS [7] – це власницька мобільна операційна система від Apple, заснована на архітектурі OS X. Спочатку вона була розроблена для iPhone під назвою iPhone OS, згодом вона також була вдосконалена для використання на iPad (до літа 2019, коли на конференції Apple WWDC було представлено нову OS для iPad — iPadOS), iPod Touch та Apple TV (до 9 вересня 2015, коли на спеціальному заході Apple було представлено tvOS). Apple не дозволяє роботу ОС на мобільних телефонах інших фірм.

Користувацький інтерфейс iOS заснований на концепції прямої маніпуляції з використанням жестів Мультитач. Елементи інтерфейсу управління складаються з повзунків, перемикачів і кнопок. Він призначений для безпосереднього контакту користувача з екраном пристрою. Внутрішній акселерометр використовуються деякими програмами для реагування на струшування пристрою, яке є також загальною командою скасування, або обертання пристрою у трьох вимірах, що є загальною командою перемикання між книжковим та альбомним режимами.

Станом на 2019 рік інтернет-магазин App Store містить понад 2 мільйони застосунків для iOS, які були завантажені понад 15 мільярдів разів. Розробники iOS також активно підтримують розвиток цієї операційної системи, остання версія iOS 16 вийшла 7 квітня 2023 року.

При розробці застосунку для операційної системи iOS наявні як переваги, так і недоліки.

Серед переваг розробки застосунків для iOS можна виділити більші можливості для монетизації застосунку у порівнянні з Android-розробкою, а також меншу конкуренцію з іншими застосунками у App Store, магазині застосунків iOS, у порівнянні з Google Play, магазином застосунків Android.

Головним недоліком iOS-розробки є необхідність розробнику мати продукцію Apple та особистий акаунт у сервісах Apple.

Вигляд домашньої сторінки iOS наведено на рис. 2.2, порівняння особливостей операційних систем iOS та Android наведено у табл. 2.1.



Рисунок 2.3 – Домашня сторінка iOS 16

Таблиця 2.1 – Порівняння мобільних операційних систем

Критерій оцінки	Назва операційної системи	
	Android	iOS
1	2	3
Процент ринку (станом на квітень 2023)	68,61%	30,61%
Конкуренція у магазинах застосунків	Велика	Помірна
Наявність Desktop-емулятора	+	-
Необхідність мати особистий акаунт для розробки	-	+

Платформою для розробки нашого застосунку було обрано операційну систему Android, оскільки вона більш ніж удвічі популярніша, а отже, охоплення користувачів також буде набагато більшим. Крім цього на вибір платформи вплинуло те, що Android має більше технічної документації та допоміжних матеріалів для розробників.

2.1.2 Вибір версії операційної системи, для якої розроблюється застосунок

Чим старіша версія ОС Android, тим більше пристроїв підтримуватиме наш застосунок. Однак чим новіша версія, тим більше нових функцій Android буде підтримуватися в застосунку. При цьому виборі підтримувати баланс між зручністю розробки, функціональністю застосунку та його сумісністю зі старими версіями Android – це дуже непросте завдання.

При розробці застосунку можна обрати рівень API, який відповідає певній версії ОС Android. У табл. 2.2 подано їх список разом з приблизним процентом пристроїв, сумісних з кожним з рівнів API станом на 6 січня 2023 року.

Таблиця 2.2 – Список версій Android API

Версія Android		Версія API	Процент сумісних пристроїв
Версія	Назва версії		
1	2	3	4
4.4	KitKat	19	
5.0	Lollipop	21	99,3%
5.1	Lollipop	22	99,0%
6.0	Marshmallow	23	97,2%

Продовження таблиці 2.2

Версія Android		Версія API	Процент сумісних пристроїв
Версія	Назва версії		
1	2	3	4
7.0	Nougat	24	94,4%
7.1	Nougat	25	92,5%
8.0	Oreo	26	90,7%
8.1	Oreo	27	88,1%
9.0	Pie	28	81,2%
10	Q	29	68,0%
11	R (робоча назва – Red Velvet Cake)	30	48,5%
12	S (робоча назва – Snow Cone)	31	24,1%
13	T (робоча назва – Tiramisu)	33	5,2%

Було вирішено обрати мінімальну версію API 28, яка відповідає версії Android 9.0 Pie, щоб охопити найбільшу аудиторію користувачів без значної втрати функціональності.

Також було вирішено обрати максимальну версію API 33, яка відповідає версії Android 13.0 T для сумісності з найбільш новим функціоналом бібліотек Android.

2.1.3 Вибір середовища розробки та мови програмування для розробки мобільного застосунку

Існує офіційне програмне забезпечення для Android-розробки – Android Studio. Вперше воно було представлено 16 травня 2013 року на

конференції Google I/O, через п'ять років після виходу першої версії Android. На поточний час Android Studio активно розвивається компанією JetBrains. Також Android-розробка підтримується в ряді інших середовищ розробки [8].

Порівняння особливостей середовищ розробки застосунків для операційної системи Android наведено у табл. 2.3.

Таблиця 2.3 – Порівняння середовищ розробки для Android

Назва середовища	Мови програмування			Аудиторія	Ліцензія	Ціна
	Java	Kotlin	C++			
1	2	3	4	5	6	7
Android Studio	+	+	+	Досвідчена	Freeware	\$0
Eclipse	+	-	+	Професійні розробники	Eclipse Public License	\$0
Visual Studio (з Xamarin)	-	-	+	Досвідчена	Proprietary	до \$2,999
IntelliJ IDEA	+	+	-	Професійні Java-розробники	Proprietary	до \$499 за рік
NetBeans	+	-	+	Професійні розробники	CDDL 1.0 та GPL2	\$0
Komodo	+	-	-	Професійні розробники	Proprietary	до \$394
AIDE	+	-	+	Початківці	Proprietary	\$0

Мова програмування – це лише інструмент у руках розробника, призначений для реалізації необхідних алгоритмів. І все ж вибір конкретної

мови програмування є найважливішою частиною розроблення застосунку, оскільки важливо переконатися в наявності актуальних бібліотек для реалізації усього необхідного функціоналу проєкту. У наш час для розроблення Android-застосунків використовуються такі мови, як Java, Kotlin, C++, C#, JavaScript і Python. Найпоширенішими та найзручнішими для розробки серед них є Java та Kotlin.

Java є офіційною мовою розробки системи Android і підтримується середовищем розробки Android Studio. Її основи беруться до уваги, щоб навчитися ефективно програмувати застосунки для обраної системи. Однак часи змінюються, і на зміну їй приходить новіша та досконаліша мова програмування під назвою Kotlin [9].

Kotlin – обгортка для мови Java, яка була створена компанією Jet Minds, щоб забезпечити більш стислий і безпечний за типом опис коду, ніж Java. Вона розробляється з 2010 року і була офіційно представлена в липні 2011 року. Код, написаний цією мовою, спочатку компілюється в код Java, а потім весь результат передається компілятору Java. Схему цього процесу наведено на рис. 2.4.

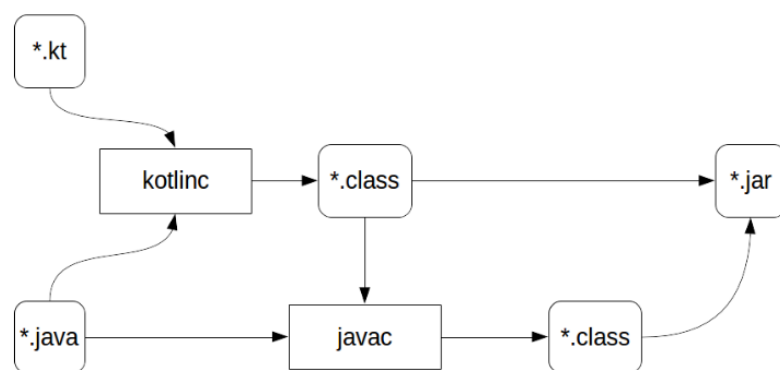


Рисунок 2.4 – Схема компілювання коду Kotlin та Java

Було вирішено обрати середовище розробки Android Studio та мову програмування Kotlin через їхню популярність та зручність використання.

2.2 Файлова структура Android-застосунку

Всі Android-застосунки повинні мати чітко визначену структуру розподілу файлів. У корні проєкту повинні знаходитися файли Gradle, їх призначення – описувати процес збірки застосунку у бінарний файл для встановлення у систему Android. Також у корні проєкту мають бути наявні модулі. При створенні нового проєкту Android Studio автоматично створює головний модуль із назвою «app» («application»), який є точкою входу у застосунок. При розробці Android-застосунку існує багато типів модулів, які можна створити. Їх перелік наведено на рис. 2.5.

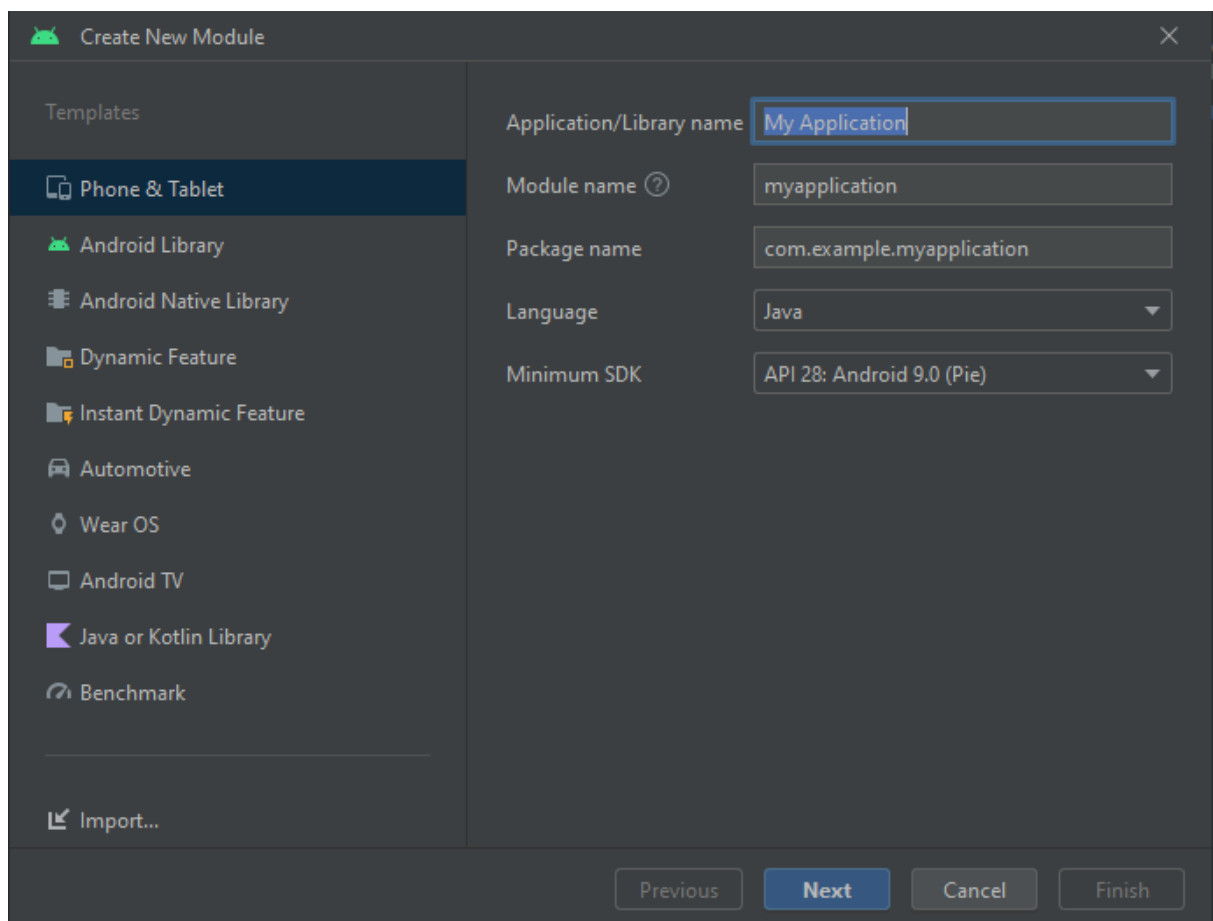


Рисунок 2.5 – Перелік шаблонів-модулів в Android Studio

Модуль – це контейнер, який зберігає вихідний код програми, файли ресурсів та конфігурацій. Кожен з модулів можна компілювати незалежно від

інших модулів, тому при розробці проєктів з великим обсягом кодової бази підхід «багатомодульність» спрощує процес розробки, а також покращує зручність читання коду.

Для розробки застосунку для прослуховування музики потребуються модулі «Phone & Tablet», «Android Library» та «Java or Kotlin Library». Інші модулі можуть бути корисними, якщо є потреба розробити застосунок, який буде взаємодіяти зі смарт-годинниками, телевізорами або автомобільними пристроями [10].

Модулі «Phone & Tablet» та «Android Library» містять схожі файли маніфесту та ресурсів, але «Phone & Tablet» компілюється у файл APK, а «Android Library», на відміну від нього, компілюється у файл Android Archive (AAR). Це дає можливість підключати такий модуль до інших проєктів як залежність [11].

У модулі «Java or Kotlin Library» відсутні файли ресурсів та маніфест, він може містити лише файли з кодом Java або Kotlin. Цей тип модулів призначений для відокремлення бізнес-логіки, розділення абстракцій та реалізацій, а також для надання доступу через інтерфейси за допомогою модифікаторів доступу Java та Kotlin.

У проєкті завжди повинні бути наявні наступні директорії:

- manifest – директорія, яка містить файл AndroidManifest.xml, що є описом програми та переліком усіх компонентів Android: Activity, Service, Receiver, ContentProvider;
- java – директорія, яка містить код застосунку;
- res – директорія, яка містить файли ресурсів, що можуть бути використані під час роботи застосунку.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

У даному розділі розглянуто шаблони проектування, що реалізовані в розроблюваному застосунку, а також абстракції та їх реалізації, алгоритми та рішення щодо дизайну інтерфейсу.

3.1 Реалізація шаблонів проектування та вибір інструментів програмування

3.1.1 Шаблони проектування, що реалізовані в застосунку

У застосунку реалізовано шаблони проектування (software design patterns) «адаптер» та «singleton».

Адаптер – структурний шаблон проектування, призначений для організації використання функцій об'єкта, недоступного для модифікації, через спеціально створений інтерфейс.

Singleton – шаблон проектування, який гарантує, що клас матиме тільки один екземпляр, та забезпечує легкий доступ до цього екземпляру.

У мові програмування Kotlin цей шаблон реалізовано у структурі об'єкт-компаньйон (companion object), яка є альтернативою статичним об'єктам Java. Об'єкт-компаньйон – це спеціальний об'єкт, який прив'язаний до визначення класу. Доступ до нього можна отримати, використовуючи ім'я класу, що охоплює його. Об'єкт-компаньйон ініціалізується при першому завантаженні класу, що його містить, навіть якщо об'єкт-компаньйон не використовується.

У застосунку, що розроблюється, шаблон «адаптер» реалізовано у класах, що зберігаються у директорії adapters, а саме AlbumAdapter, SongAdapter, SongAlbumAdapter та ViewPagerAdaper. Шаблон «singleton» у вигляді об'єктів-компаньйонів Kotlin реалізовано у класах MainActivity,

PlayerActivity, SongAdapter, SongAlbumAdapter, SongsFragment, AlbumsFragment, NotificationCenter та OnSwipeTouchListener.

3.1.2 Вибір інструменту реалізації багатопотоковості

У сучасному програмуванні широко використовується доктрина багатопотоковості, яка дозволяє виконувати декілька задач паралельно. Реалізація багатопотоковості необхідна в нашому застосунку для програвання композицій в окремому потоці, щоб користувач мав змогу користуватися інтерфейсом застосунку під час програвання композицій.

Віртуальна машина Java дозволяє програмі мати декілька потоків виконання, що виконуються одночасно. Основою кожного Android-застосунку є його головний потік (Main-потік). Система Android за замовчуванням має три потоки:

- Default;
- IO;
- Main (UI).

З головного потоку (Main-потіку) створюються інші потоки та оновлюється інтерфейс користувача. Але для роботи довготривалих чи важких для пам'яті задач доцільно використовувати інші існуючі потоки або ж створювати нові.

Кожен потік має пріоритет. Потоки з вищим пріоритетом виконуються в першу чергу перед потоками з нижчим пріоритетом. Кожен потік може бути позначений або не позначений як демон (daemon). Коли код, що виконується у якомусь потоці, створює новий об'єкт Thread, новий потік має пріоритет, який спочатку дорівнює пріоритету потоку, що його створив, і є потоком-демоном тоді і тільки тоді, коли потік, що його створив, є демоном.

Для роботи з потоками використовується функціонал класу Thread, що знаходиться у пакеті java.lang.Thread бібліотеки Java, а також класів Handler та Message, що знаходяться у пакетах android.os.Handler та android.os.Message бібліотеки Android.

Handler – це клас, який надає можливість доставляти повідомлення (Message) та виконувати програми (Runnable) до черги повідомлень потоку (Thread) та виконувати їх. Кожен екземпляр Handler при створенні прив’язується до одного потоку та до його черги повідомлень.

Існує два основних способи використання Handler:

- планування повідомлень і виконуваних файлів для виконання у певний момент у майбутньому;
- створення черги на виконання дії на іншому потоці, відмінному від вашого власного.

Thread – це клас, який реалізує сам потік. Він створюється за допомогою переданого у конструктор об’єкту Runnable, що зберігає опис роботи, що буде виконуватись. Після цього достатньо викликати метод start() у створеного об’єкту Thread. Для зупинки потоку рекомендується використовувати метод interrupt(), а довготривалі операції обгорнути в блоки try/catch. Для отримання результату від потоку використовують функції зворотного виклику (callback).

Під час розробки застосунку було вирішено використовувати класи Thread та Handler для реалізації багатопотоковості.

3.2 Огляд компонентів та розроблених класів

3.2.1 Огляд створених файлів та структури застосунку, що розроблюється

На рис. 3.1 наведено відображення файлової структури проєкту в Android Studio.

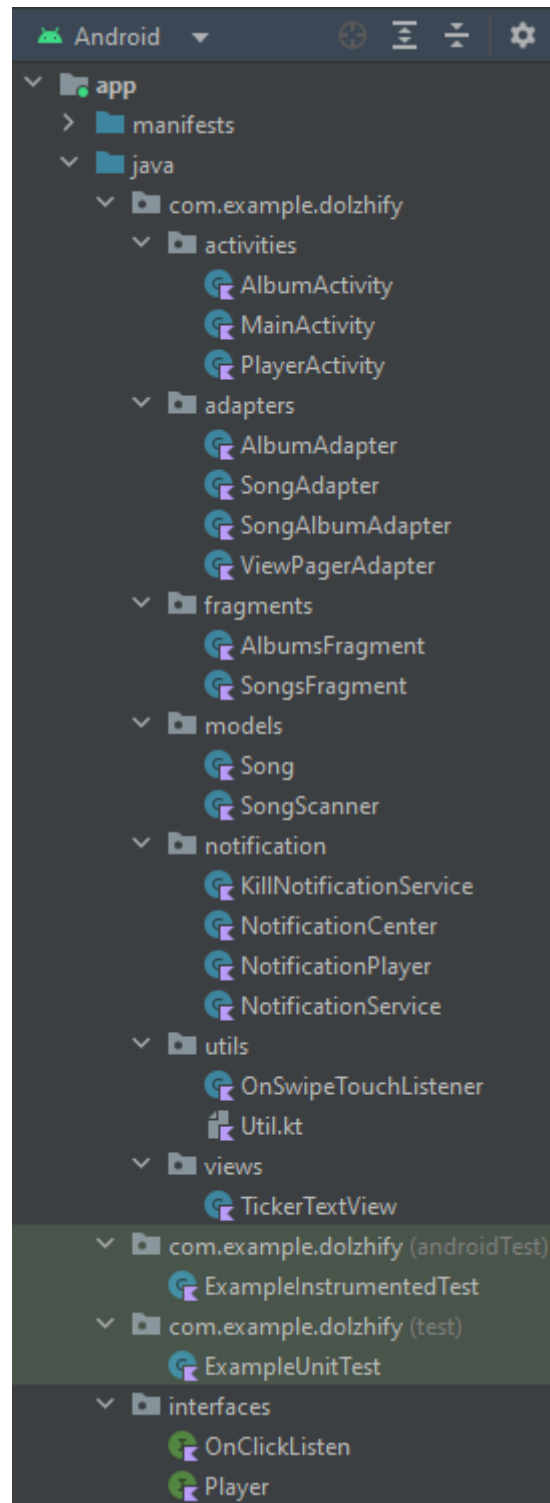


Рисунок 3.1 – Модулі застосунку

Під час розробки застосунку було використано створено лише один основний модуль «Phone & Tablet» з назвою app. Модуль app складається з чотирьох пакетів (packages):

- com.example.dolzhify – основний пакет;

- `com.example.dolzhify (androidTest)` – автоматично згенерований пакет для тестування;
- `com.example.dolzhify (test)` – автоматично згенерований пакет для тестування;
- `interfaces` – пакет, що містить інтерфейси.

Основний пакет складається з декількох пакетів, які містять класи та файли с різним призначенням:

- `Activities`;
- `Adapters`;
- `Fragments`;
- `Models`;
- `Notification`;
- `Utils`;
- `Views`.

Класи активностей (`Activities`), фрагментів активностей (`Fragments`) і адаптерів (`Adapters`) потрібні для реалізації відображення візуального інтерфейсу в Android-застосунках. Класи моделей (`Models`) та обробки повідомлень (`Notification`), а також допоміжні файли (`Utils`) використовуються для реалізації алгоритмів, що забезпечують функціонування застосунку. Класи видів (`Views`) потрібні для реалізації власних елементів користувацького інтерфейсу.

3.2.2 Інтерфейси

Було розроблено два інтерфейси:

- `OnClickListen`;
- `Player`;

Інтерфейс `OnClickListen` описує класи, яким потрібно обробляти натиснення та містить тільки метод `onClick()` для оброблення натиснення.

Інтерфейс `Player` описує класи, яким потрібно програвати музику. Він містить поля `playIcon`, `pauseIcon`, `prevIcon`, `nextIcon`, що містять ідентифікатори іконок, які зберігаються у директорії `res`. Також інтерфейс містить метод `setPlayIcon()` для оновлення іконки у візуальному відображенні плеєра.

Діаграму зв'язків інтерфейсів з класами наведено на рис 3.2.

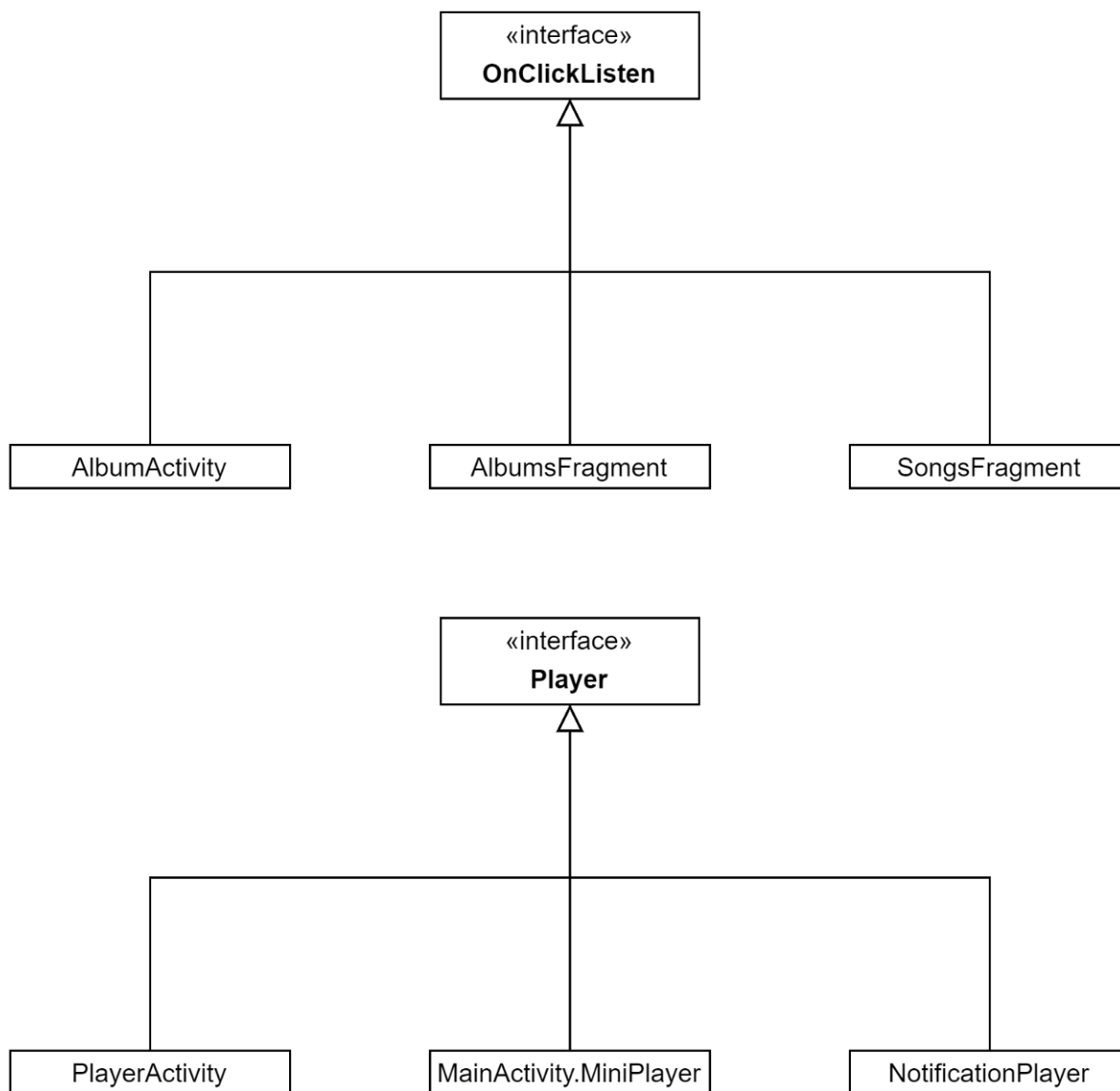


Рисунок 3.2 – Діаграма зв'язків інтерфейсів з класами

3.2.3. Класи активностей (Activities)

Було розроблено три класи активностей:

- MainActivity;
- PlayerActivity;
- AlbumActivity.

3.2.3.1 Клас MainActivity

Клас MainActivity є початковою активністю застосунку, яка запускається першочергово і згодом запускає інші активності. Клас успадковує клас AppCompatActivity. Під час запуску клас ініціалізує візуальний інтерфейс, заснований на макеті (layout) activity_main.xml, зокрема:

- ініціалізує плеєр у повідомленнях NotificationPlayer;
- ініціалізує візуальний інтерфейс, заснований на макеті (layout) activity_main.xml, зокрема менеджер макетів (layout manager) типу ViewPager, що є контейнером для фрагментів SongsFragment і AlbumsFragment, які представляють списки композицій і альбомів відповідно.
- створює та записує в поле класу об'єкт MainActivity.MinisPlayer;
- запускає сесію MediaSessionCompat, що реалізує програвання музики;
- запитує дозвіл на доступ до пам'яті пристрою, якщо цього дозволу ще немає; у разі успіху сканує пам'ять пристрою на предмет наявності аудіофайлів і формує аудіотеку зі знайдених файлів.

Клас MainActivity містить підклас MainActivity.MinisPlayer, який реалізує інтерфейс Player та при створенні ініціалізує міні-плеєр з кнопками, що дозволяють запускати методи класу PlayerActivity для керування

програванням музики («попередня композиція», «увімкнути/зупинити», «наступна композиція»). Підклас MainActivity.MinisPlayer містить:

- поле playIcon, яке при зверненні повертає іконку програвання або паузи залежно від стану поля isPlaying у класі PlayerActivity;
- поле pauseIcon, яке при зверненні повертає стандартну або спеціальну іконку паузи в залежності від стану глобальної змінної catStrokes;
- метод setPlayIcon(), який оновлює іконку міні-плеєра вмістом поля playIcon.

Ще клас MainActivity містить допоміжні методи та поля для реалізації вищеописаного функціоналу, а також об'єкт-компаньйон (companion object), у якому є:

- поле albumList, що містить посилання на аудіотеку, представлену у вигляді списку (ArrayList) альбомів, кожен з яких є списком (ArrayList) композицій, представлених у вигляді об'єктів класу Song;
- поле instance, що містить посилання на єдиний екземпляр класу MainActivity. Воно використовується іншими активностями для доступу до даних MainActivity.

3.2.3.2 Клас PlayerActivity

Клас PlayerActivity є активністю, яка реалізує функціонал плеєра. Клас успадковує клас AppCompatActivity, а також реалізує інтерфейс Player. Під час запуску клас ініціалізує візуальний інтерфейс, заснований на макеті (layout) activity_player.xml, зокрема:

- ImageView, що містить обкладинку альбому (якщо наявна);
- об'єкти TextView з назвою композиції та ім'ям виконавця;
- представлення прогресу прослуховування у вигляді стрічки прогресу SeekBar, а також у вигляді двох TextView з поточним часом прослуховування та з загальною тривалістю композиції;

- кнопки керування програванням: «попередня композиція», «увімкнути/зупинити», «наступна композиція».

Це реалізовано за допомогою створення об'єкта класу `MediaPlayer` та встановлення для нього прослуховувачів `OnPreparedListener` та `OnCompletionListener`. Також створюється потік (`Thread`), у якому кожні 1000 мілісекунд викликається метод класу `sendMessage()`, яка оновлює представлення прогресу прослуховування у вигляді стрічки `SeekBar` та `TextView` з поточним часом прослуховування та загальної тривалості композиції.

Клас містить методи керування програванням композиції:

- `prev()` – перейти до попередньої композиції;
- `next()` – перейти до наступної композиції;
- `play()` – почати/поновити програвання композиції;
- `pause()` – призупинити програвання композиції.

Ще клас містить метод `handleNotificationActions()`, який оброблює рядкову команду від `NotificationPlayer` типу `String` та в залежності від її змісту викликає один з методів керування програванням композиції.

Також клас містить поле типу `Handler`, яке містить об'єкт класу `Handler`, що використовується для керування потоком (`Thread`).

Окрім цього, у класі наявний об'єкт-компаньйон (companion object), який містить:

- метод `initiateSeekBar()`, яка ініціює представлення прогресу прослуховування у вигляді стрічки прогресу `SeekBar`;
- поля, що містять елементи візуального інтерфейсу та об'єкт класу `MediaPlayer`;
- поле `isPlaying`, у якому зберігається стан програвання композиції (програвается або призупинена);
- допоміжні поля для реалізації вищеописаного функціоналу.

3.2.3.3 Клас AlbumActivity

Клас AlbumActivity є активністю, яка реалізує функціонал перегляду окремого альбому. Клас успадковує клас AppCompatActivity, а також реалізує інтерфейс OnClickListener. Під час запуску клас ініціалізує візуальний інтерфейс, заснований на макеті (layout) activity_album.xml, зокрема:

- ImageView, що містить велику обкладинку альбому (якщо наявна);
- об'єкти TextView з назвою альбому та ім'ям виконавця;
- об'єкт RecyclerView, який містить список композицій альбому.

Клас містить реалізацію методу onClick() інтерфейсу OnClickListener. Реалізація цього методу у AlbumActivity після натиснення на композицію зі списку композицій альбому викликає метод startPlayerActivity() з пакету utils, який запускає PlayerActivity та починає програвання музики.

3.2.4 Класи фрагментів (Fragments)

Було розроблено два класи фрагментів:

- AlbumsFragment;
- SongsFragment.

3.2.4.1 Клас AlbumsFragment

Клас AlbumsFragment успадковує клас Fragment та реалізує функціонал фрагменту зі списком альбомів. Він містить поле з посиланням на RecyclerView, який використовується для розміщення списку альбомів, а також поле з посиланням на менеджер макету (layout manager) RecyclerView, який ініціалізується як GridLayoutManager, тобто макет у вигляді сітки з

елементами альбомів AlbumAdapter. Клас має об'єкт-компаньйон, що містить посилання на об'єкт AlbumAdapter.

Також клас реалізує інтерфейс OnClickListen та його метод OnClick(). Реалізація цього методу у AlbumsFragment після натиснення на альбом зі списку альбомів викликає метод startAlbumActivity() з пакету utils, який запускає AlbumActivity.

3.2.4.2 Клас SongsFragment

Клас SongsFragment успадковує клас Fragment та реалізує функціонал фрагменту зі списком альбомів. Він містить поле з посиланням на RecyclerView, який використовується для розміщення списку альбомів, а також поле з посиланням на менеджер макету (layout manager) RecyclerView, який ініціалізується як LinearLayoutManager, тобто макет у вигляді послідовного списку композицій SongAdapter. Клас має об'єкт-компаньйон, що містить посилання на об'єкт SongAdapter, а також функцію пошуку, яка звертається до поля filter класу SongAdapter.

Також клас реалізує інтерфейс OnClickListen та його метод OnClick(). Реалізація цього методу у SongsFragment після натиснення на композицію зі списку композицій викликає метод startPlayerActivity() з пакету utils, який запускає PlayerActivity та починає програвання музики.

3.2.5 Класи адаптерів (Adapters)

Було розроблено чотири класи адаптерів:

- AlbumAdapter;
- SongAdapter;
- SongAlbumAdapter;
- ViewPagerAdapter.

3.2.5.1 Клас AlbumAdapter

Клас AlbumAdapter реалізує функціонал адаптеру для відображення альбому у AlbumsFragment. Він успадковує клас RecyclerView.Adapter <AlbumAdapter.ViewHolder> та перевизначає його методи onCreateViewHolder(), onBindViewHolder() та getItemCount().

Також клас AlbumAdapter має підклас ViewHolder, який успадковує клас RecyclerView.ViewHolder(itemView), а також реалізує інтерфейс View.OnClickListener та його метод onClick(), що повертає позицію адаптеру.

3.2.5.2 Клас SongAdapter

Клас SongAdapter реалізує функціонал адаптеру для відображення композиції у SongFragment. Він успадковує клас RecyclerView.Adapter <AlbumAdapter.ViewHolder> та перевизначає його методи onCreateViewHolder(), onBindViewHolder() та getItemCount().

Також клас SongAdapter має підклас ViewHolder, який успадковує клас RecyclerView.ViewHolder(itemView), а також реалізує інтерфейс View.OnClickListener та його метод onClick(), що повертає позицію адаптеру.

Також клас SongAdapter реалізує інтерфейс Filterable та його метод getFilter(), реалізація якого у SongAdapter який повертає об'єкт класу Filter, що зберігається у полі SongAdapter. У цьому об'єкті було перевизначено метод performFiltering(), який здійснює пошук по списку композицій, та метод publishResults(), який повертає результати пошуку.

3.2.5.3 Клас SongAlbumAdapter

Клас SongAdapter реалізує функціонал адаптеру для відображення композиції у AlbumActivity, також він використовується в PlayerActivity. Він

успадковоює клас `RecyclerView.Adapter <SongAlbumAdapter.ViewHolder>` та перевизначає його методи `onCreateViewHolder()`, `onBindViewHolder()` та `getItemCount()`

Також клас `SongAlbumAdapter` має підклас `ViewHolder`, який успадковує клас `RecyclerView.ViewHolder(itemView)`, а також реалізує інтерфейс `View.OnClickListener` та його метод `OnClick()`, що повертає позицію адаптеру.

Також клас `SongAlbumAdapter` реалізує інтерфейс `Filterable` та його метод `getFilter()`, реалізація якого у `SongAlbumAdapter` повертає `null`.

Клас має об'єкт-компаньйон, що містить посилання на список композицій класу `ArrayList<Song>`.

3.2.5.4 Клас `ViewPagerAdapter`

Клас `ViewPagerAdapter` реалізує функціонал адаптеру для відображення фрагментів `AlbumsFragment` та `SongsFragment` у `ViewPager`. Він успадковує клас `FragmentPagerAdapter` та перевизначає його методи `getItem()`, `getPageTitle()` та `getItemCount()`, а також має метод `addFragment()`, за допомогою якого можна додати новий фрагмент до `ViewPagerAdapter`.

3.2.6 Класи моделей (Models)

Було розроблено два класи моделей:

- `Song`;
- `SongScanner`.

3.2.6.1 Клас Song

Клас Song є моделлю композиції. Він реалізує інтерфейс Comparable<Song> та його метод compareTo(), реалізація якого у класі Song порівнює два об'єкта класу Song за полем title («назва композиції»). Клас містить:

- рядкові поля типу String з назвою композиції, назвою альбому, до якого належить композиція (якщо він є) та іменем виконавця, а також поле типу Spanned з відформатованою назвою композиції;
- поля з уніфікованими ідентифікаторами аудіофайлу композиції та обкладинки альбому (якщо наявна), а також шляхом до аудіофайлу композиції;
- конструктор, який ініціалізує ці поля, та допоміжні методи для ініціалізації цих полів.

3.2.6.2 Клас SongScanner

Клас SongScanner є моделлю сканування аудіофайлів з пам'яті пристрою.

Він містить:

- метод scanDevice(), який сканує пам'ять пристрою та знаходить аудіофайли з коректно заповненим полем «ім'я виконавця», формує з них об'єкти класу Song та повертає список композицій у вигляді об'єкта класу ArrayList<Song>;
- список виконавців у вигляді об'єкта класу HashMap<String, MutableList<Song>> у приватному полі artists;
- список альбомів у вигляді об'єкта класу HashMap<String, MutableList<Song>> у публічному полі albums.

3.2.7 Класи обробки повідомлень (Notifications)

Було розроблено чотири класи обробки повідомлень:

- NotificationPlayer;
- NotificationService;
- NotificationCenter;
- KillNotificationService.

3.2.7.1 Клас NotificationPlayer

Клас NotificationPlayer реалізує інтерфейс Player та впроваджує функціонал плеєра в повідомленнях. Під час створення клас створює канал для повідомлень NotificationChannel і запускає службу контролю повідомлень KillNotificationChannel;

Також клас містить:

- метод update(), що формує та надсилає повідомлення в наявний канал, тим самим оновлюючи плеєр у повідомленнях;
- допоміжний метод playbackAction(), що формує PendingIntent, який потрібний для передання рядкових команд кнопками керування програванням музики у плеєру в повідомленнях;
- поле, що містить ID каналу для повідомлень;
- поле, що містить об'єкт NotificationManager;
- поля типу String, що містять рядкові команди для плеєра в повідомленнях.

3.2.7.2 Клас NotificationService

Клас NotificationService є сервісом повідомлень, який успадковує клас Service та перевизначає його методи onStartCommand() та onBind().

3.2.7.3 Клас NotificationCenter

Клас NotificationCenter успадковує клас Application та перевизначає його метод onCreate(). Він використовується для синхронізації плеєра у повідомленнях з PlayerActivity.

3.2.7.4 Клас KillNotificationService

Клас KillNotificationService є сервісом, який успадковує клас Service та перевизначає його методи onStartCommand(), onBind(), та onTaskRemoved(), реалізація якого у KillNotificationService закінчує роботу NotificationService після закриття застосунку.

3.2.8 Допоміжні класи та файли (Utils)

Було розроблено допоміжний клас OnSwipeTouchListener та файл Util, що містить корисні функції.

3.2.8.1 Клас OnSwipeTouchListener

Клас OnSwipeTouchListener є допоміжним класом, який використовується для реалізації керування застосунком за допомогою свайпів. Він має відкриті методи onSwipeRight(), onSwipeLeft(), onSwipeUp() та onSwipeDown(), а також підклас GestureListener, який успадковує клас SimpleOnGestureListener та перевизначає його методи onSingleTapUp(), onDown() та onFling().

Клас OnSwipeTouchListener реалізує інтерфейс OnTouchListener та його метод onTouch(), реалізація якого у OnSwipeTouchListener звертається

до методу `onTouchEvent` об'єкту класу `GestureDetector`, якому передано об'єкт класу `GestureListener`.

3.2.8.2 Файл Util

Файл `Util` містить корисні статичні функції, які використовуються у застосунку:

- `compareSongData()` – перевіряє композицію у вигляді об'єкта класу `Song` на наявність тексту пошукового запиту в її метаданих;
- `compare()` – приватна функція, що перевіряє рядок на наявність підрядку;
- `createTimeLabel()` – форматує час у мілісекундах у форматі «хвилини:секунди»;
- `startPlayerActivity()` – запускає `PlayerActivity` та починає програвання музики;
- `startAlbumActivity()` – запускає `AlbumActivity`;
- `petTheCat()` – реалізує додатковий функціонал;
- `toast()` – спрощене створення `Toast`-повідомлення;
- `glideAlbumArt()` – відображає обкладинку альбому або зафарбоване стандартне зображення у об'єкті `ImageView`;
- `glideCat()` – відображає анімоване зображення кішки;
- `getBitmapArtwork()` – повертає обкладинку альбому у форматі `Bitmap`.

Також файл містить приватну змінну `catStrokes` типу `Int` та публічну змінну `catInvasion` типу `Boolean`.

3.2.9 Класи видів (Views)

Було розроблено лише один клас видів `TickerTextView`.

3.2.9.1 Клас `TickerTextView`

Клас `TickerTextView` успадковує клас `AppCompatActivity`, це клас текстового елементу UI. Він реалізує рухомий рядок.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЗАСТОСУНКУ

У цьому розділі надано опис процесу роботи застосунку, а також результати його тестування на різних пристроях з різними версіями операційної системи Android.

4.1 Процес роботи застосунку

Щоб запустити застосунок, користувач натискає на іконку «Dolzhify». Система проводить первинну ініціалізацію компонентів і запуск служб, після чого відображається вікно MainActivity з кнопкою пошуку, макетом вибору фрагментів та неактивним міні-плеєром, який відображає вітальний текст. Якщо у застосунку немає дозволу на читання пам'яті пристрою, то відображається інтерфейс запиту дозволів (рис. 4.1, рис 4.2).

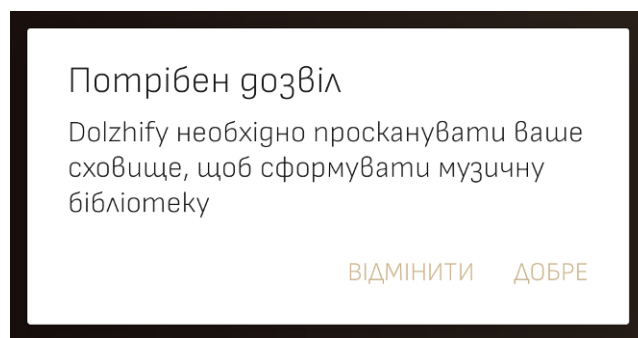


Рисунок 4.1 – Інтерфейс запиту дозволів Dolzhify

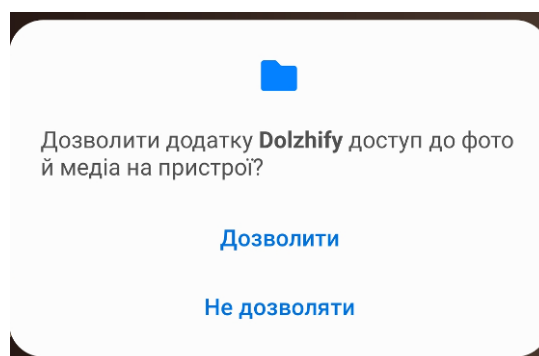


Рисунок 4.2 – Інтерфейс запиту дозволів Dolzhify

Якщо дозвіл отримано або якщо дозвіл вже є, то пам'ять пристрою сканується на предмет аудіофайлів і з них формується аудіотека. Після цього фрагмент списку композицій заповнюється композиціями (рис. 4.3), для кожної з яких відображається ім'я виконавця, назва композиції та обкладинка альбому (якщо наявна). Фрагмент списку композицій можна гортати за допомогою свайпів.

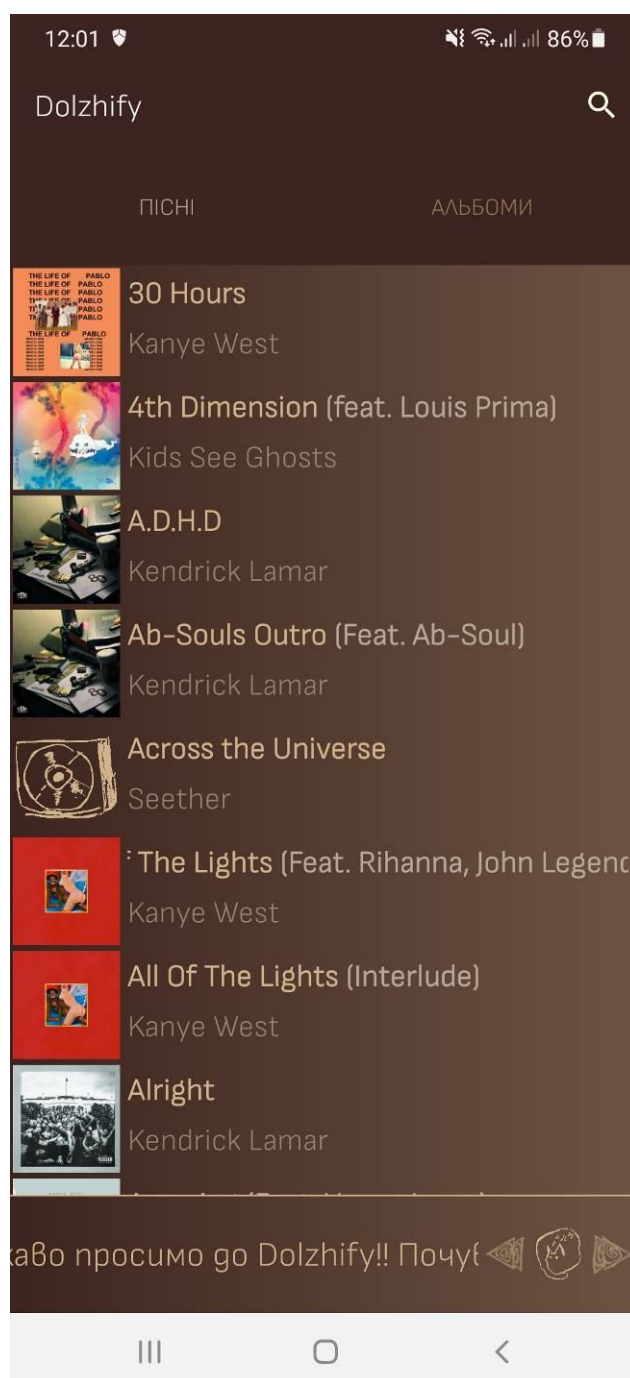


Рисунок 4.3 – Список композицій

Якщо користувач натискає на кнопку пошуку, то замість неї з'являється поле, в яке можна ввести пошуковий запит і кнопка «назад», а також автоматично відкривається віртуальна клавіатура Android. Після кожного символу, який користувач вводить у поле пошуку, в списку композицій залишаються лише ті композиції, в яких ім'я виконавця, назва композиції або назва альбому містять текст пошукового запиту (рис. 4.4). Список композицій повертається до первісного повного вигляду після натискання кнопки «назад» у застосунку або ж кнопки «назад» Android.

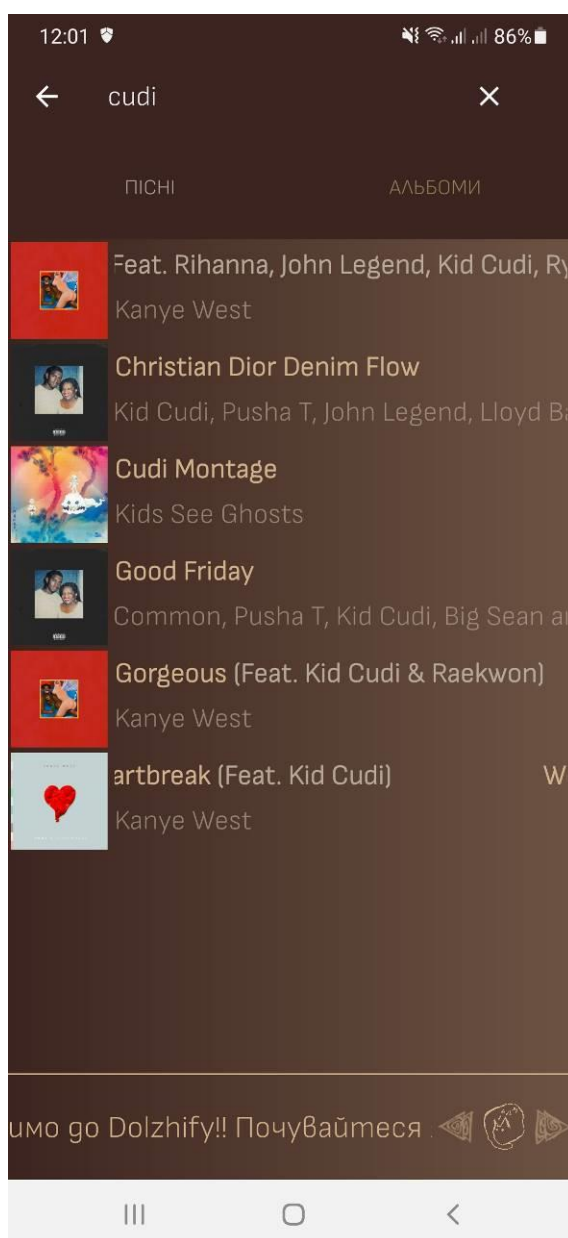


Рисунок 4.4 – Пошук

Якщо користувач натискає на кнопку «Альбоми» в макеті виділення фрагмента або свайпає праворуч, відображається фрагмент списку альбомів (рис. 4.5). Для кожного альбому відображаються ім'я виконавця, назва альбому та обкладинка альбому (якщо наявна). Список альбомів можна гортати за допомогою свайпів. Щоб повернутися до списку композицій, користувачеві необхідно натиснути на кнопку «Пісні» в макеті вибору фрагментів або свайпнути ліворуч.



Рисунок 4.5 – Список альбомів

Якщо користувач натискає на відображення будь-якого з альбомів у списку альбомів, відображається вікно AlbumActivity з великою обкладинкою альбому (якщо наявна), назвою альбому та ім'ям виконавця, а також нумерованим списком композицій альбому (рис. 4.6), який можна гортати за допомогою свайпів (рис. 4.7).



Рисунок 4.6 – Альбом

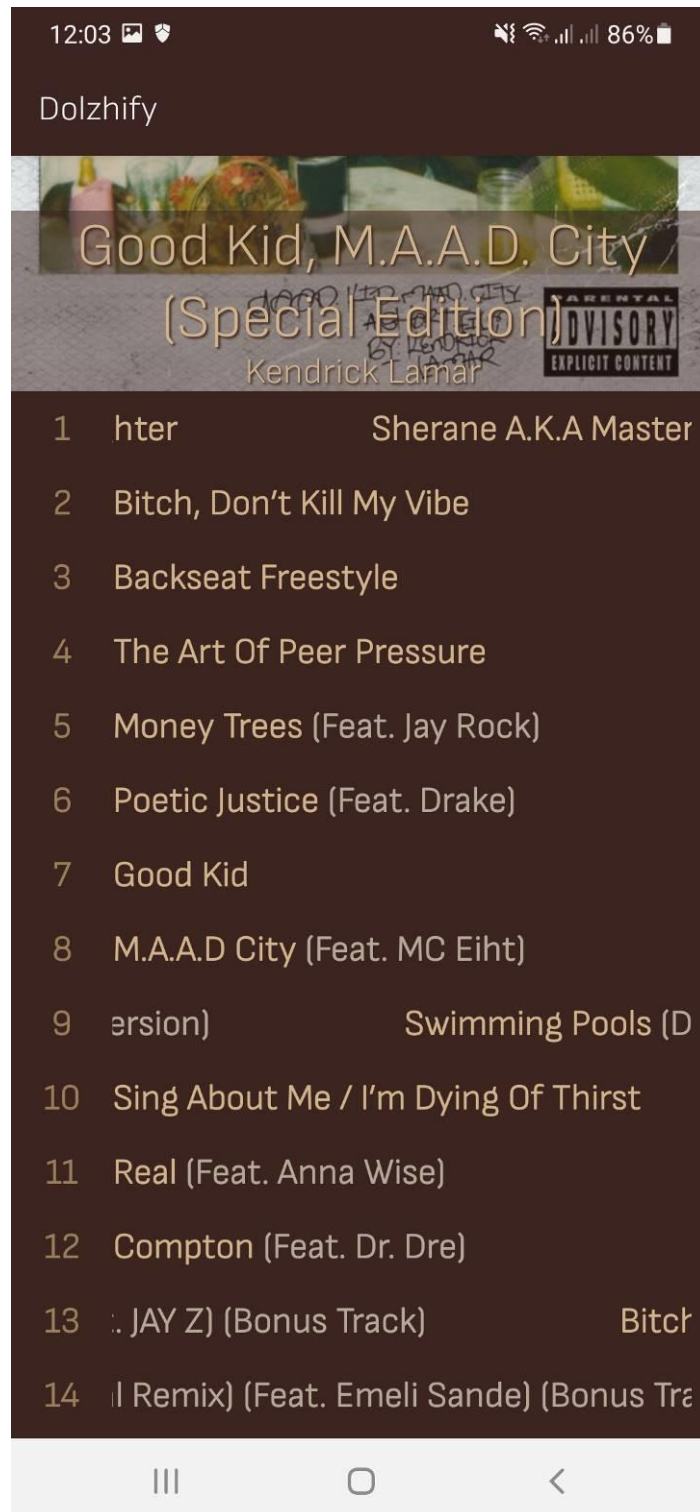


Рисунок 4.7 – Гортання альбому

Щоб почати програвання композиції, користувач може натиснути на відображення композиції у фрагменті списку композицій MainActivity або у списку композицій AlbumActivity.

Після цього відкривається вікно PlayerActivity (рис. 4.8). У ньому відображається велика обкладинка альбому (якщо наявна), назва композиції та ім'я виконавця, а також представлення прогресу прослуховування у вигляді стрічки прогресу, відображення поточного часу прослуховування та відображення загальної тривалості композиції. Нижче розташовані кнопки керування програванням: «попередня композиція», «увімкнути/зупинити», «наступна композиція». Користувач може керувати програванням композиції за допомогою цих кнопок та стрічки прогресу. Щоб згорнути PlayerActivity, користувач може свайпнути угору або натиснути кнопку «назад» Android.

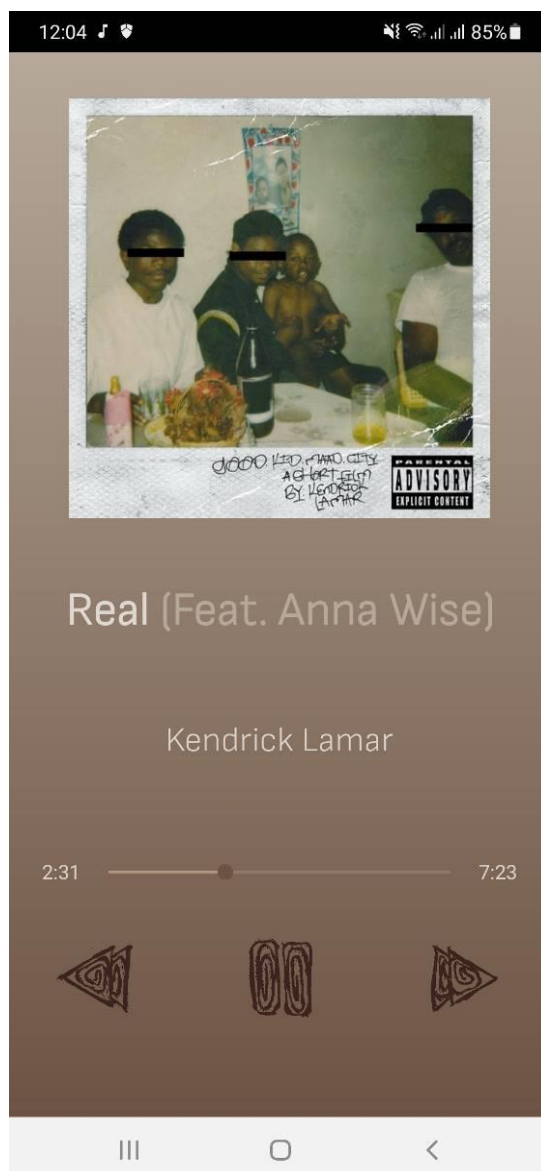


Рисунок 4.8 – Плеєр

Після того, як користувач почав програвання будь-якої композиції, міні-плеєр MainActivity активується. Починаючи з цього моменту в його рядковому полі відображається назва поточної композиції, а натиснення на його кнопку «увімкнути/зупинити» вмикає чи зупиняє програвання поточної композиції (рис. 4.9).

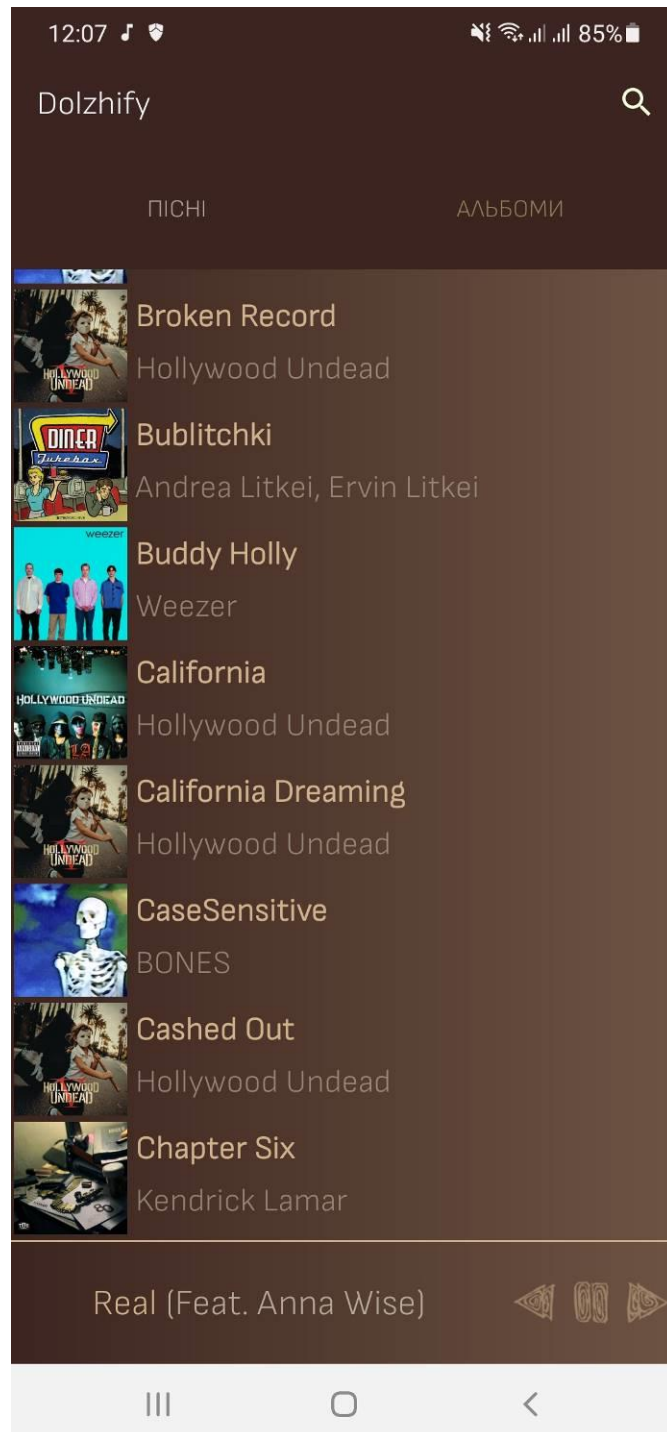


Рисунок 4.9 – Міні-плеєр

Окрім цього, після того, як користувач почав програвання будь-якої композиції, створюється перманентне повідомлення Android (рис. 4.10), що відображує назву композиції, ім'я виконавця та обкладинку альбому (якщо наявна). Повідомлення також містить кнопки керування програванням: «попередня композиція», «увімкнути/зупинити», «наступна композиція», що функціонують аналогічно кнопкам PlayerActivity. За допомогою цього плеєра в повідомленнях користувач має можливість керувати програванням композицій навіть коли застосунок згорнуто та він перебуває у фоновому режимі.

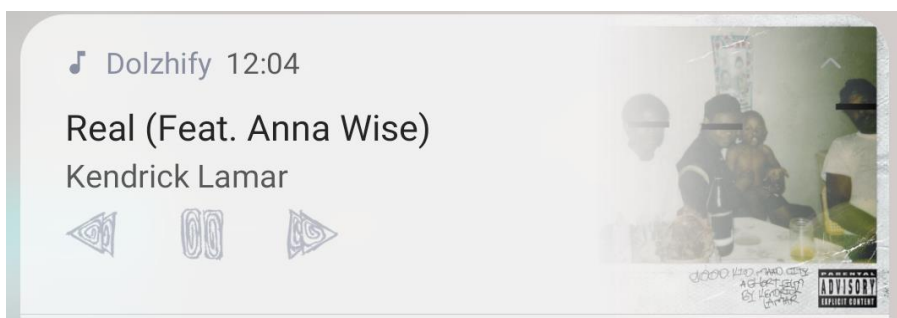


Рисунок 4.10 – Плеєр у повідомленнях

Після тридцяти одного натиснення на неактивну кнопку паузи в міні-плеєрі замість усіх обкладинок альбомів застосунок зображає анімоване кошеня (рис 4.11), а вітальний текст у міні-плеєрі замінюється на текст, у якому вказано авторів застосунку.

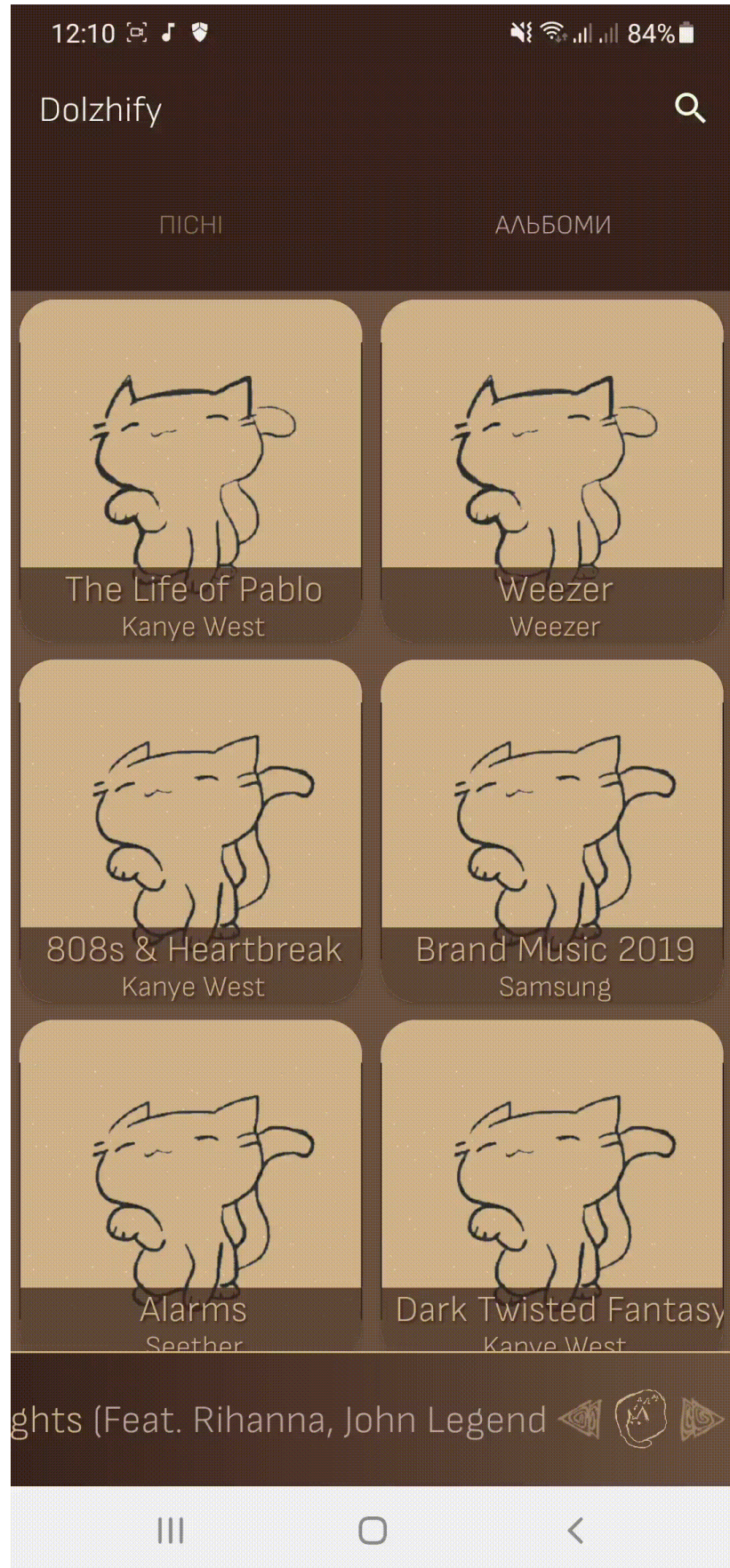


Рисунок 4.11 – Відображення анімованого кошеня замість обкладинок альбомів

4.2 Тестування застосунку

Було проведено ряд тестувань серед декількох пристроїв із різними версіями Android. Були розглянуті наступні критерії:

- швидкодія застосунку;
- коректність формування аудіотеки з пам’яті пристрою;
- коректність відображення інтерфейсу;
- коректність програвання музики;
- коректність функціонування пошуку;
- коректність функціонування повідомлень.

Результати тестування занесено у таблицю 4.1.

Таблиця 4.1 – Результати проведених тестувань

Критерій тестування	Назва пристрою та версія Android		
	Samsung A50, Android 13.0 T	Oppo Reno4 Lite, Android 10.0 Q	Redmi 8A, Android 10.0 Q
1	2	3	4
Швидкодія застосунку	Відмінно	Відмінно	Відмінно
Коректність формування аудіотеки з пам’яті пристрою	Коректне	Коректне	Коректне
Коректність відображення інтерфейсу	Коректне	Коректне	Коректне
Коректність програвання музики	Коректне	Коректне	Коректне
Коректність функціонування пошуку	Коректне	Коректне	Коректне

Продовження таблиці 4.1

Критерій тестування	Назва пристрою та версія Android		
	Samsung A50, Android 13.0 T	Oppo Reno4 Lite, Android 10.0 Q	Redmi 8A, Android 10.0 Q
1	2	3	4
Коректність функціонування повідомлень	Коректне	Коректне	Коректне

Тестування було пройдено успішно на всіх пристроях. Усі інші операції, що передбачені функціоналом застосунку, виконувались коректно та без проблем.

4.3 Висновки за розділом 4

У цьому розділі було наведено опис процесу роботи застосунку та його основних функцій. Також було наведення результати тестування застосунку на різних пристроях із різними версіями ОС Android. Можна зробити висновок, що застосунок працює стабільно та виконує всі поставлені завдання.

ВИСНОВКИ

Під час виконання дипломної кваліфікаційної роботи було розроблено застосунок для прослуховування музики. Був проведений огляд аналогів та методів вирішення завдання, що вже існують, проаналізовано переваги та недоліки кожного з них і розроблено функціонал власного програмного продукту.

Для реалізації програмного забезпечення було використано мову програмування Kotlin.

Створений застосунок дозволив реалізувати такі можливості:

- формування аудіотеки з аудіофайлів, що зберігаються у пам'яті пристрою;
- подання аудіотеки у вигляді повного списку композицій або ж у вигляді альбомів;
- програвання композицій з аудіотеки та керування цим процесом (перехід до попередньої або наступної композиції, пауза, призупинення і відновлення програвання, перемотування);
- відображення метаданих аудіофайлів (номер композиції, ім'я виконавця, назва композиції, версія композиції, назва альбому, обкладинка альбому);
- пошук композицій в аудіотеці за назвою або за ім'ям виконавця.

Також було проведене тестування застосунку, що підтвердило його працездатність.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Spotify [Електронний ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/Spotify>
2. Samsung Music [Електронний ресурс]. – Режим доступу: <https://play.google.com/store/apps/details?id=com.sec.android.app.music&hl=ru&gl=US>
3. VLC for Android [Електронний ресурс]. – Режим доступу: <https://play.google.com/store/apps/details?id=org.videolan.vlc&hl=ru&gl=US>
4. Mobile System Operating System Market Share Worldwide [Електронний ресурс]. – Режим доступу: <https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-202304-202304-bar>
5. Android Statistics (2023) [Електронний ресурс]. – Режим доступу: <https://www.businessofapps.com/data/android-statistics/>
6. Android (operating system) [Електронний ресурс]. – Режим доступу: [https://en.wikipedia.org/wiki/Android_\(operating_system\)](https://en.wikipedia.org/wiki/Android_(operating_system))
7. iOS [Електронний ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/IOS>
8. 11 Best Android IDEs for Developers of 2022 [Електронний ресурс]. – Режим доступу: <https://www.developer.com/mobile/top-android-ides-for-developers/>
9. Kotlin (programming language) [Електронний ресурс]. – Режим доступу: [https://en.wikipedia.org/wiki/Kotlin_\(programming_language\)](https://en.wikipedia.org/wiki/Kotlin_(programming_language))
10. Add a module for a new device [Електронний ресурс]. – Режим доступу: <https://developer.android.com/studio/projects/add-app-module#CreatingAModule>
11. Create an Android library [Електронний ресурс]. – Режим доступу: <https://developer.android.com/studio/projects/android-library.html>

ДОДАТОК А
Технічне завдання

Вступ

Найменування продукту – «Dolzhify». Застосунок призначений для офлайн-прослуховування аудіофайлів з пам'яті пристрою.

A.1 Підстави для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Розробка мобільного застосунку музичного плеєра», затверджене наказом Національного університету «Запорізька політехніка» № 87 від 4 квітня 2023 р.

A.2 Призначення розробки

Метою розробки програмного забезпечення є створення Android-застосунку для прослуховування музики офлайн з пам'яті пристрою з легким для розуміння інтерфейсом.

A.3 Вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Вимоги до функціональних характеристик застосунку:

- створення аудіотеки з аудіофайлів, що зберігаються у пам'яті пристрою;
- групування композицій в аудіотеці за альбомами, зазначеними в метаданих аудіофайлів;
- надання можливості прослуховування композицій з аудіотеки та можливості керування цим процесом (перехід до попередньої або наступної композиції, призупинення та відновлення прослуховування).

A.3.2 Вимоги до надійності

Вимоги до надійності роботи застосунку:

- стабільне функціонування;
- обробка помилок для запобігання несанкціонованого закриття програми.

A.3.3 Умови експлуатації

Умови експлуатації співпадають з умовами експлуатації девайсу, на якому відбувається функціонування застосунку.

A.3.4 Вимоги до складу й параметрів технічних засобів

Технічні вимоги до обладнання:

- наявність екрану;
- не менше 1 ГБ ОЗУ;
- вільний простір не менш 100 МБ.

A.3.5 Вимоги до інформаційної й програмної сумісності

Програмні вимоги:

- сервіс віртуальної клавіатури;
- версія Android від 9.0.

A.3.6 Вимоги до маркування й упакування

Програма повинна мати маркування імені застосунку – «Dolzlify». Програма може бути записана у повному обсязі на компакт-диск не менше 100МБ.

A.3.7 Вимоги до транспортування й збереження

Вимоги до транспортування і зберігання аналогічні вимогам, що пред'являються до лазерних дисків або карт пам'яті.

A.4 Вимоги до програмної документації

Програмна документація повинна включати пояснювальну записку на розроблений продукт та вихідний код програми.

A.5 Техніко-економічні показники

Монетизація застосунку передбачає його публікацію у магазин Android-застосунків Google Play.

A.6 Порядок контролю й приймання

Передбачено захист дипломного проєкту у встановлені керівником проєкту строки.

ДОДАТОК Б
Текст програми

```

package com.example.dolzhify.activities

import android.Manifest
import android.annotation.SuppressLint
import android.content.DialogInterface
import android.content.pm.PackageManager
import android.os.Bundle
import android.support.v4.media.session.MediaSessionCompat
import android.view.Menu
import android.view.inputmethod.EditorInfo
import android.widget.ImageButton
import android.widget.TextView
import android.widget.Toast
import androidx.appcompat.app.AlertDialog
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.widget.SearchView
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import androidx.viewpager.widget.ViewPager
import com.example.dolzhify.R
import com.example.dolzhify.adapters.SongAdapter
import com.example.dolzhify.adapters.ViewPagerAdapter
import com.example.dolzhify.fragments.AlbumsFragment
import com.example.dolzhify.fragments.SongsFragment
import com.example.dolzhify.fragments.SongsFragment.Companion.search
import com.example.dolzhify.models.Song
import com.example.dolzhify.models.SongScanner
import com.example.dolzhify.notification.NotificationPlayer
import com.example.dolzhify.utils.catInvasion
import com.example.dolzhify.utils.petTheCat
import com.google.android.material.tabs.TabLayout
import interfaces.Player

class MainActivity : AppCompatActivity() {
    private val storagePermissionCode = 1
    private var viewPagerAdapter: ViewPagerAdapter? = null

```

```

private var mediaSession: MediaSessionCompat? = null
var notificationPlayer: NotificationPlayer? = null
var miniPlayer: MiniPlayer? = null

@SuppressLint("ClickableViewAccessibility")
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    instance = this
    setContentView(R.layout.activity_main)

    // init notifications
    notificationPlayer = NotificationPlayer(this)

    // init viewpager
    val viewPager = findViewById<ViewPager>(R.id.view_Pager)
    viewPagerAdapter = ViewPagerAdapter(supportFragmentManager)
    viewPagerAdapter!!.addFragment(SongsFragment(),
resources.getString(R.string.songs))
    viewPagerAdapter!!.addFragment(AlbumsFragment(),
resources.getString(R.string.albums))
    viewPager!!.adapter = viewPagerAdapter

    findViewById<TabLayout>(R.id.table_Layout)!!.setupWithViewPager(viewPager)

    // init actionbar
    supportActionBar!!.elevation = 0f

    // init mini player
    miniPlayer = MiniPlayer(this)

    // init media
    mediaSession = MediaSessionCompat(this, "tag")
    if (ContextCompat.checkSelfPermission(
        this@MainActivity,
        Manifest.permission.READ_EXTERNAL_STORAGE
    ) == PackageManager.PERMISSION_DENIED

```

```

        ) requestStoragePermission() else startScan()
    }

    private fun requestStoragePermission() {
        if (ActivityCompat.shouldShowRequestPermissionRationale(
            this,
            Manifest.permission.READ_EXTERNAL_STORAGE
        )
    ) {
        AlertDialog.Builder(this).setTitle(R.string.permission_needed)
            .setMessage(R.string.needs_to_scan)
            .setPositiveButton(R.string.okay) { _: DialogInterface?, _:
Int ->
                ActivityCompat.requestPermissions(
                    this@MainActivity, arrayOf(
                        Manifest.permission.READ_EXTERNAL_STORAGE
                    ), storagePermissionCode
                )
            }
            .setNegativeButton(R.string.cancel) { dialog:
DialogInterface, _: Int -> dialog.dismiss() }
            .create().show()
        } else ActivityCompat.requestPermissions(
            this,
            arrayOf(Manifest.permission.READ_EXTERNAL_STORAGE),
            storagePermissionCode
        )
    }

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

```

```

        if (requestCode == storagePermissionCode) {
            if (grantResults.isNotEmpty() && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                Toast.makeText(this, R.string.permission_granted,
Toast.LENGTH_SHORT).show()
                startScan()
            } else Toast.makeText(this, R.string.permission_denied,
Toast.LENGTH_SHORT).show()
        }
    }

    private fun startScan() {
        val songScanner = SongScanner(this)
        SongAdapter.songs = ArrayList()
        SongAdapter.songs.add(Song())
        val songs = songScanner.scanDevice
        songs.sort()
        SongAdapter.songs = songs
        val albums: HashMap<String, MutableList<Song>> = songScanner.albums
        val it: MutableIterator<*> = albums.entries.iterator()
        while (it.hasNext()) {
            val (_, value) = it.next() as Map.Entry<*, *>
            albumList.add(value as ArrayList<Song>)
            it.remove()
        }
    }

    override fun onCreateOptionsMenu(menu: Menu): Boolean {
        val inflater = menuInflater
        inflater.inflate(R.menu.search_menu, menu)
        val searchView = menu.findItem(R.id.action_search).actionView as
SearchView?
        searchView!!.imeOptions = EditorInfo.IME_ACTION_DONE
        searchView.setOnQueryTextListener(object :
SearchView.OnQueryTextListener {
            override fun onQueryTextSubmit(query: String): Boolean {

```

```

        return false
    }
    override fun onQueryTextChange(newText: String): Boolean {
        search(newText)
        return false
    }
})
return true
}

```

```

class MiniPlayer(mainActivity: MainActivity): Player {
    private val activity = mainActivity
    private var playButton: ImageButton? = null
    var tv: TextView? = null

    init {
        tv = activity.findViewById(R.id.mini_player_title)
        tv!!.text = activity.getString(R.string.welcome)
        playButton = activity.findViewById(R.id.mini_player_play_button)
        playButton!!.setOnClickListener {
            PlayerActivity.instance?.play()
            if (PlayerActivity.instance == null)
                petTheCat(activity)
        }

        activity.findViewById<ImageButton>(R.id.mini_player_prev_button).setOnClickListener {
            PlayerActivity.instance?.prev()
        }

        activity.findViewById<ImageButton>(R.id.mini_player_next_button).setOnClickListener {
            PlayerActivity.instance?.next()
        }
    }
}

```

```

        override fun setPlayIcon() {
            playButton?.setBackgroundResource(if (PlayerActivity.isPlaying)
pauseIcon else playIcon)
        }

        override val pauseIcon: Int
            get() = if (catInvasion) R.drawable.music_cat else
super.pauseIcon
    }

    companion object {
        var albumList = ArrayList<ArrayList<Song>>>()

        @JvmStatic
        @SuppressWarnings("StaticFieldLeak")
        var instance: MainActivity? = null
        private set
    }
}

package com.example.dolzlify.activities

import android.animation.LayoutTransition
import android.annotation.SuppressLint
import android.media.MediaPlayer
import android.os.Bundle
import android.os.Handler
import android.os.Looper
import android.os.Message
import android.view.View
import android.widget.ImageButton
import android.widget.ImageView
import android.widget.LinearLayout
import android.widget.SeekBar
import android.widget.SeekBar.OnSeekBarChangeListener
import android.widget.TextView

```

```

import androidx.appcompat.app.AppCompatActivity
import com.example.dolzhify.R
import com.example.dolzhify.adapters.SongAdapter
import com.example.dolzhify.adapters.SongAlbumAdapter
import com.example.dolzhify.models.Song
import com.example.dolzhify.notification.NotificationCenter
import com.example.dolzhify.notification.NotificationPlayer
import com.example.dolzhify.utils.OnSwipeTouchListener
import com.example.dolzhify.utils.createTimeLabel
import com.example.dolzhify.utils.glideAlbumArt
import interfaces.Player

class PlayerActivity : AppCompatActivity(), Player {
    private val handler: Handler = object : Handler(Looper.getMainLooper()) {
        override fun handleMessage(msg: Message) {
            val currentPosition = msg.what
            mSeekBar!!.max =
                mMediaPlayer!!.duration / 1000
            mSeekBar!!.progress = currentPosition / 1000
            println(mSeekBar!!.progress)
            curTime!!.text = createTimeLabel(currentPosition)
            totTime!!.text = createTimeLabel(msg.arg1)
        }
    }

    private var notificationPlayer: NotificationPlayer? = null
    private var position = 0
    private var oldPos = -1
    private var songs: ArrayList<Song>? = null
    private var songTitle: TextView? = null
    private var artistName: TextView? = null
    private var imagePlayer: ImageView? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_player)
        mSeekBar = findViewById(R.id.seek)
    }

```

```

songTitle = findViewById(R.id.song_title)
artistName = findViewById(R.id.artist_name)
totTime = findViewById(R.id.total_time)
pause = findViewById(R.id.play)
curTime = findViewById(R.id.current_time)
imagePlayer = findViewById(R.id.image_player)

// set smooth text animations
val l = LayoutTransition()
l.enableTransitionType(LayoutTransition.CHANGING)
(findViewById<View>(R.id.player_names_layout) as
LinearLayout).layoutTransition = l

// set notifications
notificationPlayer = MainActivity.instance!!.notificationPlayer
NotificationCenter()

// fetch intent
fetchIntent(intent.extras!!)

// set song controls
pause!!.setOnClickListener { play() }
findViewById<ImageButton>(R.id.prev).setOnClickListener { prev() }
findViewById<ImageButton>(R.id.next).setOnClickListener { next() }
findViewById<View>(R.id.player_layout).setOnTouchListener(object :
OnSwipeTouchListener(
    instance
) {
    override fun onSwipeLeft() {
        next()
    }
    override fun onSwipeRight() {
        prev()
    }
    override fun onSwipeDown() {
        finish()
    }

```

```

        }
    })
}

private fun fetchIntent(bundle: Bundle) {
    position = bundle.getInt("index")
    if (instance != null && !bundle.getBoolean("from")) {
        songTitle!!.text = instance!!.songTitle!!.text
        artistName!!.text = instance!!.artistName!!.text
        totTime!!.text = totTime!!.text
        curTime!!.text = curTime!!.text
        imagePlayer!!.setImageDrawable(instance!!.imagePlayer!!.drawable)
        initiateSeekBar()
        songs = instance!!.songs
        position = instance!!.position
    } else {
        if (mMediaPlayer != null) mMediaPlayer!!.stop()
        instance = this
        songs = if (bundle.getInt("val") == 1) SongAlbumAdapter.albumSong
else SongAdapter.songs
        initPlayer(position)
    }
}

private fun initPlayer(pos: Int) {
    if (mMediaPlayer != null && mMediaPlayer!!.isPlaying)
mMediaPlayer!!.reset()

    // set song name
    val title = songs!![pos].coloredTitle
    songTitle!!.text = title
    MainActivity.instance!!.miniPlayer!!.tv!!.text = title

    // set artist and album art, if it's the first song or not the same
artist
    val artist = songs!![pos].artist

```

```

        if (oldPos < 0 || songs!![oldPos].artist != artist) artistName!!.text
= artist

        if (oldPos < 0 || songs!![oldPos].album != songs!![pos].album)
            glideAlbumArt(applicationContext, songs!![pos],
R.drawable.ic_note, imagePlayer!!, true)

        // set up mediaplayer
        mMediaPlayer = MediaPlayer.create(
            applicationContext, songs!![pos].audioUri
        )
        mMediaPlayer!!.setOnPreparedListener {
            totTime!!.text = createTimeLabel(mMediaPlayer!!.duration)
            mSeekBar!!.max = mMediaPlayer!!.duration / 1000
            isPlaying = false
            play()
        }
        mMediaPlayer!!.setOnCompletionListener { initPlayer((pos + 1) %
songs!!.size) }
        initiateSeekBar()
        Thread {
            while (mMediaPlayer != null) {
                try {
                    if (mMediaPlayer!!.isPlaying) sendMessage()
                } catch (e: InterruptedException) {
                    e.printStackTrace()
                }
            }
        }.start()
    }

    private fun sendMessage() {
        val msg = Message()
        msg.what = mMediaPlayer!!.currentPosition
        msg.arg1 = mMediaPlayer!!.duration
        handler.sendMessage(msg)
    }

```

```

        Thread.sleep(1000)
    }

    fun handleNotificationActions(action: String) {
        when (action) {
            notificationPlayer?.actionPause -> if (isPlaying) pause() else
play()
            notificationPlayer?.actionNext -> next()
            notificationPlayer?.actionPrev -> prev()
        }
    }

    fun prev() {
        oldPos = position
        position = if (position == 0) songs!!.size - 1 else position - 1
        initPlayer(position)
    }

    fun next() {
        oldPos = position
        position = (position + 1) % songs!!.size
        initPlayer(position)
    }

    fun play() {
        if (mMediaPlayer != null && !mMediaPlayer!!.isPlaying) {
            isPlaying = true
            mMediaPlayer!!.start()
            setPlayIcon()
        } else pause()
    }

    private fun pause() {
        if (mMediaPlayer!!.isPlaying) {
            isPlaying = false
            mMediaPlayer!!.pause()
        }
    }

```

```

        setPlayIcon()
    }
}

override fun setPlayIcon() {
    pause!!.setBackgroundResource(if (isPlaying) R.drawable.ic_pause else
R.drawable.ic_play)
    MainActivity.instance!!.miniPlayer!!.setPlayIcon()
    notificationPlayer!!.update(
        songs!![position],
        isPlaying
    )
}

companion object {
    var isPlaying = false
    private set

    @SuppressWarnings("StaticFieldLeak")
    private var mSeekBar: SeekBar? = null

    @SuppressWarnings("StaticFieldLeak")
    var instance: PlayerActivity? = null
    private set

    @SuppressWarnings("StaticFieldLeak")
    private var curTime: TextView? = null

    @SuppressWarnings("StaticFieldLeak")
    private var totTime: TextView? = null

    @SuppressWarnings("StaticFieldLeak")
    private var pause: ImageButton? = null

    private var mMediaPlayer: MediaPlayer? = null
    fun initiateSeekBar() {

```

```

        mSeekBar!!.setOnSeekBarChangeListener(object :
OnSeekBarChangeListener {
            override fun onProgressChanged(seekBar: SeekBar, progress:
Int, fromUser: Boolean) {
                if (fromUser) mMediaPlayer!!.seekTo(progress * 1000)
            }
            override fun onStartTrackingTouch(seekBar: SeekBar) {}
            override fun onStopTrackingTouch(seekBar: SeekBar) {}
        })
    }
}

```

```
package com.example.dolzhify.activities
```

```

import android.os.Bundle
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
import androidx.core.content.ContextCompat
import androidx.recyclerview.widget.DividerItemDecoration
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.example.dolzhify.R
import com.example.dolzhify.adapters.SongAlbumAdapter
import com.example.dolzhify.utils.glideAlbumArt
import com.example.dolzhify.utils.startPlayerActivity
import interfaces.OnClickListen

```

```
class AlbumActivity : AppCompatActivity(), OnClickListen {
```

```

    private var recyclerView: RecyclerView? = null
    var position = 0

```

```

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

```

```

        // fetch intent
        val bundle = intent.extras!!
        position = bundle.getInt("index")

        // set recyclerview
        recyclerView = findViewById(R.id.album_recycler)
        recyclerView?.setHasFixedSize(true)
        recyclerView?.layoutManager = LinearLayoutManager(this)
        recyclerView?.adapter =
SongAlbumAdapter(MainActivity.albumList[position], this)

        // set album, artist and cover
        val song = MainActivity.albumList[position][0]
        findViewById<TextView>(R.id.album_view_album).text = song.album
        findViewById<TextView>(R.id.album_view_artist).text = song.artist
        glideAlbumArt(this, song, R.drawable.ic_note,
findViewById(R.id.album_view_art), true)

        // set vertical decoration
        val verticalDecoration = DividerItemDecoration(this,
DividerItemDecoration.HORIZONTAL)
        verticalDecoration.setDrawable(ContextCompat.getDrawable(this,
R.drawable.line_divider)!!)
        recyclerView?.addItemDecoration(verticalDecoration)
    }

    override fun onClick(position: Int) {
        startPlayerActivity(position, 1)
    }
}

package com.example.dolzhiify.adapters

import android.content.Context
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.dolzlify.R
import com.example.dolzlify.activities.MainActivity
import com.example.dolzlify.utils.glideAlbumArt
import interfaces.OnClickListen

class AlbumAdapter(private val alClickListen: OnClickListen) :
    RecyclerView.Adapter<AlbumAdapter.ViewHolder>() {
    init {
        val inflater = MainActivity.instance
            ?.getSystemService(Context.LAYOUT_INFLATER_SERVICE) as
LayoutInflater
    }

    override fun onCreateViewHolder(viewGroup: ViewGroup, i: Int): ViewHolder
{
        val view = LayoutInflater.from(viewGroup.context)
            .inflate(R.layout.adapter_album_layout, viewGroup, false)
        return ViewHolder(view, alClickListen)
    }

    override fun onBindViewHolder(vh: ViewHolder, i: Int) {
        val song = MainActivity.albumList[i][0]
        vh.album.text = song.album
        vh.artist.text = song.artist
        glideAlbumArt(
            MainActivity.instance!!,
            song,
            R.drawable.ic_album,
            vh.albumArtwork,
            false,
            R.color.bistre
        )
    }
}

```

```

        )
    }

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

    class ViewHolder(itemView: View, onClickListen: OnClickListen) :
        RecyclerView.ViewHolder(itemView), View.OnClickListener {
        val album: TextView
        val artist: TextView
        val albumArtwork: ImageView
        private val onClickListen: OnClickListen

        init {
            album = itemView.findViewById(R.id.album_name)
            artist = itemView.findViewById(R.id.album_artist)
            albumArtwork = itemView.findViewById(R.id.album_image)
            this.onClickListen = onClickListen
            itemView.setOnClickListener(this)
        }

        override fun onClick(v: View) {
            onClickListen.onClick(bindingAdapterPosition)
        }
    }
}

package com.example.dolzlify.adapters

import android.annotation.SuppressLint
import android.app.Activity
import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup

```

```

import android.widget.Filter
import android.widget.Filterable
import android.widget.ImageView
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.example.dolzhify.R
import com.example.dolzhify.models.Song
import com.example.dolzhify.utils.compareSongData
import com.example.dolzhify.utils.glideAlbumArt
import interfaces.OnClickListen
import java.util.Locale

class SongAdapter(
    private val context: Activity,
    argSongs: ArrayList<Song>,
    onClickListen: OnClickListen
) : RecyclerView.Adapter<SongAdapter.ViewHolder>(), Filterable {

    private var myOnClickListen: OnClickListen
    private val allSongs: ArrayList<Song> = ArrayList(songs)

    private val filter: Filter = object : Filter() {
        override fun performFiltering(constraint: CharSequence):
FilterResults {
            val filteredList: MutableList<Song> = ArrayList()
            if (constraint.isEmpty()) filteredList.addAll(allSongs) else {
                val filterPattern =
                    constraint.toString().lowercase(Locale.getDefault()).trim
{ it <= ' ' }
                for (iSong in allSongs) if (compareSongData(iSong,
filterPattern)) filteredList.add(
                    iSong
                )
            }
            val results = FilterResults()
            results.values = filteredList

```

```

        return results
    }

    @SuppressWarnings("NotifyDataSetChanged")
    override fun publishResults(constraint: CharSequence, results:
FilterResults) {
        songs.clear()
        songs.addAll(results.values as ArrayList<Song>)
        notifyDataSetChanged()
    }
}

init {
    val inflater =
context.getSystemService(Context.LAYOUT_INFLATER_SERVICE) as LayoutInflater
    myOnClickListen = onClickListen
    songs = ArrayList(argSongs)
}

override fun onCreateViewHolder(viewGroup: ViewGroup, i: Int): ViewHolder
{
    val v = LayoutInflater.from(viewGroup.context)
        .inflate(R.layout.adapter_view_layout, viewGroup, false)
    return ViewHolder(v, myOnClickListen)
}

override fun onBindViewHolder(vh: ViewHolder, i: Int) {
    vh.songTitle.text = songs[i].coloredTitle
    vh.artistName.text = songs[i].artist
    glideAlbumArt(context, songs[i], R.drawable.ic_track, vh.songArtwork)
}

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

```

```

        override fun getFilter(): Filter {
            return filter
        }

class ViewHolder(itemView: View, onClickListen: OnClickListener) :
    RecyclerView.ViewHolder(itemView), View.OnClickListener {
    var songArtwork: ImageView
    var songTitle: TextView
    var artistName: TextView
    private var onClickListen: OnClickListener

    init {
        songArtwork = itemView.findViewById(R.id.imageView)
        songTitle = itemView.findViewById(R.id.tv_song_title)
        artistName = itemView.findViewById(R.id.tv_artist)
        artistName.text = ""
        this.onClickListen = onClickListen
        itemView.setOnClickListener(this)
    }

    override fun onClick(v: View) {
        onClickListen.onClick(bindingAdapterPosition)
    }
}

companion object {
    var songs: ArrayList<Song> = ArrayList()
}

}

package com.example.dolzhiify.adapters

import android.annotation.SuppressLint
import android.content.Context
import android.view.LayoutInflater
import android.view.View

```

```

import android.view.ViewGroup
import android.widget.Filter
import android.widget.Filterable
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.example.dolzhify.R
import com.example.dolzhify.activities.MainActivity
import com.example.dolzhify.models.Song
import interfaces.OnClickListen

class SongAlbumAdapter(albumSong: ArrayList<Song>?, private val
onClickListen: OnClickListen) :
    RecyclerView.Adapter<SongAlbumAdapter.ViewHolder>(), Filterable {
    init {
        MainActivity.instance
            ?.getSystemService(Context.LAYOUT_INFLATER_SERVICE) as
LayoutInflater
        Companion.albumSong = albumSong
    }

    override fun onCreateViewHolder(viewGroup: ViewGroup, i: Int): ViewHolder
{
        val view = LayoutInflater.from(viewGroup.context)
            .inflate(R.layout.album_song_layout, viewGroup, false)
        return ViewHolder(view, onClickListen)
    }

    @SuppressWarnings("SetTextI18n")
    override fun onBindViewHolder(viewHolder: ViewHolder, i: Int) {
        val song = albumSong!![i]
        if (i < 100) viewHolder.albumSongNumber.text = (i + 1).toString() +
""
        viewHolder.albumSongTitle.text = song.coloredTitle
    }

    override fun getItemCount(): Int {

```

```

        return albumSong!!.size
    }

    override fun getFilter(): Filter? {
        return null
    }

    class ViewHolder(itemView: View, onClickListen: OnClickListen) :
        RecyclerView.ViewHolder(itemView), View.OnClickListener {
        var albumSongNumber: TextView
        var albumSongTitle: TextView
        private var onClickListen: OnClickListen

        init {
            albumSongNumber = itemView.findViewById(R.id.album_song_number)
            albumSongTitle = itemView.findViewById(R.id.album_song_title)
            this.onClickListen = onClickListen
            itemView.setOnClickListener(this)
        }

        override fun onClick(v: View) {
            onClickListen.onClick(bindingAdapterPosition)
        }
    }

    companion object {
        @JvmField
        var albumSong: ArrayList<Song>? = null
    }
}

package com.example.dolzhiify.adapters

import androidx.fragment.app.Fragment
import androidx.fragment.app.FragmentManager
import androidx.fragment.app.FragmentPagerAdapter

```

```

class ViewPagerAdapter(fm: FragmentManager?) : FragmentPagerAdapter(
    fm!!,
    BEHAVIOR_RESUME_ONLY_CURRENT_FRAGMENT
) {
    private val fragments: MutableList<Fragment> = ArrayList()
    private val titles: MutableList<String> = ArrayList()
    override fun getItem(i: Int): Fragment {
        return fragments[i]
    }

    override fun getPageTitle(position: Int): CharSequence {
        return titles[position]
    }

    override fun getCount(): Int {
        return titles.size
    }

    fun addFragment(fragment: Fragment, title: String) {
        fragments.add(fragment)
        titles.add(title)
    }
}

```

```
package com.example.dolzhify.fragments
```

```

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import androidx.recyclerview.widget.GridLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.example.dolzhify.R
import com.example.dolzhify.adapters.AlbumAdapter

```

```

import com.example.dolzlify.utils.startAlbumActivity
import interfaces.OnClickListen

class AlbumsFragment : Fragment(), OnClickListen {
    private var v: View? = null
    private var recyclerView: RecyclerView? = null
    private var layoutManager: RecyclerView.LayoutManager? = null

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        v = inflater.inflate(R.layout.albums_fragment, container, false)
        recyclerView = v?.findViewById(R.id.albums_recycle_view)
        recyclerView?.setHasFixedSize(true)
        layoutManager = GridLayoutManager(context, 2)
        albumAdapter = AlbumAdapter(this)
        recyclerView?.layoutManager = layoutManager
        recyclerView?.adapter = albumAdapter
        return v
    }

    override fun onClick(position: Int) {
        startAlbumActivity(position)
    }

    companion object {
        private var albumAdapter: AlbumAdapter? = null
    }
}

package com.example.dolzlify.fragments

import android.annotation.SuppressLint
import android.os.Bundle

```

```

import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.example.dolzhify.R
import com.example.dolzhify.activities.MainActivity
import com.example.dolzhify.adapters.SongAdapter
import com.example.dolzhify.utils.startPlayerActivity
import interfaces.OnClickListen

class SongsFragment : Fragment(), OnClickListen {

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val v = inflater.inflate(R.layout.songs_fragment, container, false)
        val recyclerView = v.findViewById<RecyclerView>(R.id.recycle_view)
        recyclerView.setHasFixedSize(true)
        val manager: RecyclerView.LayoutManager =
LinearLayoutManager(context)
        songAdapter = MainActivity.instance?.let { SongAdapter(it,
SongAdapter.songs, this) }
        recyclerView.layoutManager = manager
        recyclerView.adapter = songAdapter
        return v
    }

    override fun onClick(position: Int) {
        startPlayerActivity(position, if (position > 0) 0 else 2)
    }

    companion object {

```

```

        @SuppressWarnings("StaticFieldLeak")
        private var songAdapter: SongAdapter? = null
        @JvmStatic
        fun search(text: String?) {
            songAdapter!!.filter.filter(text)
        }
    }
}

package com.example.dolzlify.models

import android.content.ContentUri
import android.net.Uri
import android.text.Spanned
import androidx.core.text.HtmlCompat

class Song: Comparable<Song> {
    var title: String? = null
        private set
    var coloredTitle: Spanned? = null
    var artist: String? = null
    var album: String? = null

    // file uris
    private var artworkUri: Uri? = null
    var audioUri: Uri? = null
    var artworkUriString: String? = null
    var path: String? = null
    //var albumID: Long = 0

    constructor()

    constructor(path: String, title: String, album: String?, artist: String,
albumID: Long) {
        this.album = album
        this.artist = artist

```

```

        this.title = title
        this.path = path

        // set colored title for ui
        coloredTitle = getColoredTitle(title)

        // set uris from album id and path
        artworkUri = Uri.parse("content://media/external/audio/albumart")
        audioUri = Uri.parse(path)
        artworkUriString = ContentUris.withAppendedId(
            artworkUri!!, albumID
        ).toString()
    }

    // crops the number in the beginning of the song name
    private fun cropNumber(name: String): String {
        val x = name.split("\\.".toRegex(), limit = 2).toTypedArray()
        return if (x.size == 2 && x[0].matches("[0-9]+".toRegex())) x[1] else
name
    }

    private fun getColoredTitle(title: String): Spanned {
        // try to split title to name and version
        val x = title.split("\\(".toRegex(), limit = 2).toTypedArray()
        val songName = cropNumber(x[0])
        // try to set version if there's one
        val songVersion = if (x.size == 2 && x[1].indexOf('(') != -1)
"("+x[1].substring(
            0,
            x[1].lastIndexOf('(')+1
        ) else ""
        // set colored title for ui
        return HtmlCompat.fromHtml(
            "$songName<font color=#B6A99A>$songVersion</font>",
            HtmlCompat.FROM_HTML_MODE_LEGACY
        )
    }

```

```

    }

    override fun compareTo(other: Song): Int {
        return title!!.compareTo(other.title!!)
    }
}

package com.example.dolzhify.models

import android.content.Context
import android.provider.MediaStore

class SongScanner
    (private val context: Context) {
    private val artists = HashMap<String, MutableList<Song>>()
    var albums = HashMap<String, MutableList<Song>>()

    val scanDevice: ArrayList<Song>
        get() {
            val songs = ArrayList<Song>()
            val c = context.contentResolver.query(
                MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,
                arrayOf(
                    MediaStore.Audio.AudioColumns.DATA,
                    MediaStore.Audio.AudioColumns.TITLE,
                    MediaStore.Audio.AudioColumns.ALBUM,
                    MediaStore.Audio.ArtistColumns.ARTIST,
                    MediaStore.Audio.AudioColumns.ALBUM_ID
                ),
                null,
                null,
                null
            )
            if (c != null) {
                while (c.moveToNext()) {
                    val song = Song(

```

```

        c.getString(0), // path
        c.getString(1), // name
        c.getString(2), // album
        c.getString(3), // artist
        c.getLong(4) // album id
    )
    if (song.artist != "<unknown>") {
        songs.add(song)
        albums.computeIfAbsent(song.album!!) { ArrayList() }
        albums[song.album!!]?.add(song)
        artists.computeIfAbsent(song.artist!!) { ArrayList() }
    }

    artists[song.artist!!]?.add(song)
}
}
c.close()
}
return songs
}
}

```

```
package com.example.dolzhify.notification
```

```

import android.Manifest
import android.app.Notification
import android.app.NotificationChannel
import android.app.NotificationManager
import android.app.PendingIntent
import android.content.Intent
import android.content.pm.PackageManager
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import androidx.core.app.NotificationCompat
import com.example.dolzhify.R
import com.example.dolzhify.activities.MainActivity
import com.example.dolzhify.models.Song

```

```

import com.example.dolzlify.utils.getBitmapArtwork
import interfaces.Player

class NotificationPlayer(mainActivity: MainActivity) : Player {

    private val channelId = "10031"
    private var notificationManager: NotificationManager? = null
    private var activity: MainActivity = mainActivity

    val actionPause = "com.example.dolzlify.action_pause"
    val actionNext = "com.example.dolzlify.action_next"
    val actionPrev = "com.example.dolzlify.action_prev"

    init {
        notificationManager =
this.activity.getSystemService(AppCompatActivity.NOTIFICATION_SERVICE) as
NotificationManager
        notificationManager!!.createNotificationChannel(
            NotificationChannel(
                channelId,
                "PLAYER",
                NotificationManager.IMPORTANCE_DEFAULT
            )
        )
        this.activity.startService(Intent(this.activity,
KillNotificationService::class.java))
    }

    fun update(song: Song, isPlaying: Boolean) {
        val notification = NotificationCompat.Builder(activity, channelId)
            .setSmallIcon(R.drawable.ic_note_little)
            .setContentTitle(song.title)
            .setContentText(song.artist)
            .setLargeIcon(getBitmapArtwork(song, activity.resources))
            .addAction(prevIcon, "Previous", playbackAction(actionPrev))
    }

```

```

        .addAction(if (isPlaying) pauseIcon else playIcon, "Pause",
playbackAction(actionPause))
        .addAction(nextIcon, "Next", playbackAction(actionNext))
        .setContentIntent(
            PendingIntent.getActivity(
                activity,
                0,
                Intent(activity, MainActivity::class.java),
                PendingIntent.FLAG_IMMUTABLE
            )
        ).setStyle(
            androidx.media.app.NotificationCompat.MediaStyle()
        )
        .setPriority(NotificationManager.IMPORTANCE_MAX)
        .build()
notification.flags =
    notification.flags or (Notification.FLAG_NO_CLEAR or
Notification.FLAG_ONGOING_EVENT)
    if (ActivityCompat.checkSelfPermission(
        activity, Manifest.permission.POST_NOTIFICATIONS
    ) == PackageManager.PERMISSION_GRANTED
    ) notificationManager!!.notify(1, notification)
}

```

```

private fun playbackAction(s: String): PendingIntent? {
    val playbackAction = Intent(activity,
NotificationService::class.java)
    playbackAction.action = s
    return PendingIntent.getService(
        activity, 31, playbackAction, PendingIntent.FLAG_IMMUTABLE
    )
}
}

```

```

package com.example.dolzhify.notification

```

```

import android.app.Service
import android.content.Intent
import android.os.IBinder
import com.example.dolzhify.activities.PlayerActivity

class NotificationService : Service() {

    override fun onStartCommand(intent: Intent, flags: Int, startId: Int):
Int {
        if (intent.action != null)

PlayerActivity.instance?.handleNotificationActions(intent.action!!)
        return START_NOT_STICKY
    }

    override fun onBind(intent: Intent): IBinder? {
        return null
    }
}

package com.example.dolzhify.notification

import android.app.Application
import android.app.NotificationChannel
import android.app.NotificationManager

class NotificationCenter : Application() {
    override fun onCreate() {
        super.onCreate()

        getSystemService(NotificationManager::class.java).createNotificationChannel(
            NotificationChannel("10032", "Song",
NotificationManager.IMPORTANCE_LOW)
        )
    }
}

```

```

}

package com.example.dolzlify.notification

import android.app.NotificationManager
import android.app.Service
import android.content.Intent
import android.os.IBinder

class KillNotificationService : Service() {
    override fun onTaskRemoved(rootIntent: Intent) {
        super.onTaskRemoved(rootIntent)
        (getSystemService(NOTIFICATION_SERVICE) as
NotificationManager).cancelAll()
    }

    override fun onStartCommand(intent: Intent, flags: Int, startId: Int):
Int {
        return super.onStartCommand(intent, flags, startId)
    }

    override fun onBind(intent: Intent): IBinder? {
        return null
    }
}

package com.example.dolzlify.utils

import android.annotation.SuppressLint
import android.content.Context
import android.view.GestureDetector
import android.view.GestureDetector.SimpleOnGestureListener
import android.view.MotionEvent
import android.view.View
import android.view.View.OnTouchListener
import kotlin.math.abs

```

```

open class OnSwipeTouchListener(context: Context?) : OnTouchListener {
    private val gestureDetector: GestureDetector

    init {
        gestureDetector = GestureDetector(context, GestureListener())
    }

    @SuppressLint("ClickableViewAccessibility")
    override fun onTouch(view: View, motionEvent: MotionEvent): Boolean {
        return gestureDetector.onTouchEvent(motionEvent)
    }

    open fun onSwipeRight() {}
    open fun onSwipeLeft() {}
    open fun onSwipeUp() {}
    open fun onSwipeDown() {}

    private inner class GestureListener : SimpleOnGestureListener() {
        override fun onSingleTapUp(event: MotionEvent): Boolean {
            return true
        }

        override fun onDown(e: MotionEvent): Boolean {
            return true
        }

        override fun onFling(
            e1: MotionEvent,
            e2: MotionEvent,
            velocityX: Float,
            velocityY: Float
        ): Boolean {
            val swipeThreshold = 100
            val swipeVelocityThreshold = 100
            val result = false

```

```

        try {
            val diffY = e2.rawY - e1.rawY
            val diffX = e2.rawX - e1.rawX
            if (abs(diffX) - abs(diffY) > swipeThreshold) {
                if (abs(diffX) > swipeThreshold && abs(velocityX) >
swipeVelocityThreshold) {
                    if (diffX > 0) onSwipeRight() else onSwipeLeft()
                }
            } else {
                if (abs(diffY) > swipeThreshold && abs(velocityY) >
swipeVelocityThreshold) {
                    if (diffY > 0) onSwipeDown() else onSwipeUp()
                }
            }
        } catch (ignored: Exception) {
        }
        return result
    }
}

```

```
package com.example.dolzhify.utils
```

```

import android.app.Activity
import android.content.Context
import android.content.Intent
import android.content.res.Resources
import android.graphics.Bitmap
import android.graphics.BitmapFactory
import android.media.MediaMetadataRetriever
import android.media.MediaPlayer
import android.widget.ImageView
import android.widget.Toast
import androidx.appcompat.content.res.AppCompatResources
import androidx.core.graphics.drawable.DrawableCompat
import com.bumptech.glide.Glide

```

```

import com.bumptech.glide.load.engine.DiskCacheStrategy
import com.bumptech.glide.load.resource.drawable.DrawableTransitionOptions
import com.example.dolzhify.R
import com.example.dolzhify.activities.AlbumActivity
import com.example.dolzhify.activities.MainActivity
import com.example.dolzhify.activities.PlayerActivity
import com.example.dolzhify.models.Song
import java.util.Objects
import kotlin.system.exitProcess

var catInvasion = false
private var catStrokes = 0

fun compareSongData(i: Song, s: String): Boolean {
    return compare(i.title!!, s) || compare(i.artist!!, s) ||
    compare(i.album!!, s)
}

private fun compare(a: String, b: String): Boolean {
    return Objects.requireNonNull(a).lowercase().contains(b.lowercase())
}

fun createTimeLabel(duration: Int): String {
    val min = duration / 1000 / 60
    val sec = duration / 1000 % 60
    var timeLabel = "$min:"
    if (sec < 10) timeLabel += "0"
    timeLabel += sec
    return timeLabel
}

fun startPlayerActivity(position: Int, `val`: Int) {
    MainActivity.instance?.startActivity(
        Intent(
            MainActivity.instance, PlayerActivity::class.java

```

```

        ).putExtra("index", position).putExtra("val", `val`).putExtra("from",
true)
    )
}

```

```

fun startAlbumActivity(position: Int) {
    MainActivity.instance?.startActivity(
        Intent(MainActivity.instance, AlbumActivity::class.java).putExtra(
            "index", position
        )
    )
}

```

```

fun petTheCat(act: MainActivity) {
    catStrokes++
    when (catStrokes) {
        1 -> toast(act, R.string.egg01)
        2 -> toast(act, R.string.egg02)
        11 -> toast(act, R.string.egg11)
        21 -> toast(act, R.string.egg21)
        31 -> {
            catInvasion = true
            MediaPlayer.create(act, R.raw.weezer_lick).start()
            toast(act, R.string.egg31, Toast.LENGTH_LONG)
            act.miniPlayer!!.tv!!.text = act.getString(R.string.authors)
        }

        310 -> toast(act, R.string.egg310)
        320 -> exitProcess(-1)
    }
}

```

```

fun toast (act: Activity, stringID: Int, length: Int = Toast.LENGTH_SHORT) {
    Toast.makeText(act, stringID, length).show()
}

```

```

// ui utils
fun glideAlbumArt(
    context: Context,
    song: Song,
    placeholder: Int,
    iv: ImageView,
    onlyOnError: Boolean = false,
    placeholderTint: Int = R.color.LightCoffee
) {
    if (catInvasion)
        glideCat(context, iv)
    else {
        val d = DrawableCompat.wrap(AppCompatResources.getDrawable(context,
placeholder))!!.mutate()
        .also {
            DrawableCompat.setTint(
                it, context.resources.getColor(placeholderTint,
context.theme)
            )
        }
        Glide.with(context)
            .load(song.artworkUriString)
            .placeholder(if (!onlyOnError) d else null)
            .error(if (onlyOnError) d else null)
            .diskCacheStrategy(DiskCacheStrategy.NONE)
            .skipMemoryCache(true)
            .transition(
                DrawableTransitionOptions().crossFade()
            ).into(iv)
    }
}

fun glideCat(context: Context, iv: ImageView) {
    Glide.with(context)
        .asGif()
        .load(R.raw.dancing_cat)

```

```

        .diskCacheStrategy(DiskCacheStrategy.NONE)
        .skipMemoryCache(true)
        .transition(
            DrawableTransitionOptions().crossFade()
        ).into(iv)
        iv.setBackgroundColor(context.resources.getColor(R.color.lightCoffee,
context.theme))
    }

```

```

fun getBitmapArtwork(song: Song, res: Resources): Bitmap? {
    val retriever = MediaMetadataRetriever()
    retriever.setDataSource(song.path)
    val art = retriever.embeddedPicture
    return try {
        BitmapFactory.decodeByteArray(art, 0, art!!.size)
    } catch (_: Exception) {
        BitmapFactory.decodeResource(res, R.drawable.ic_track)
    }
}

```

```

package com.example.dolzlify.views

```

```

import android.content.Context
import android.text.TextUtils
import android.util.AttributeSet
import androidx.appcompat.widget.AppCompatTextView

```

```

class TickerTextView(context: Context, attributeSet: AttributeSet) :
    AppCompatTextView(context, attributeSet) {

    init {
        ellipsize = TextUtils.TruncateAt.MARQUEE
        marqueeRepeatLimit = -1
        isSingleLine = true
        isSelected = true
    }
}

```

}

ДОДАТОК В
Слайди презентації

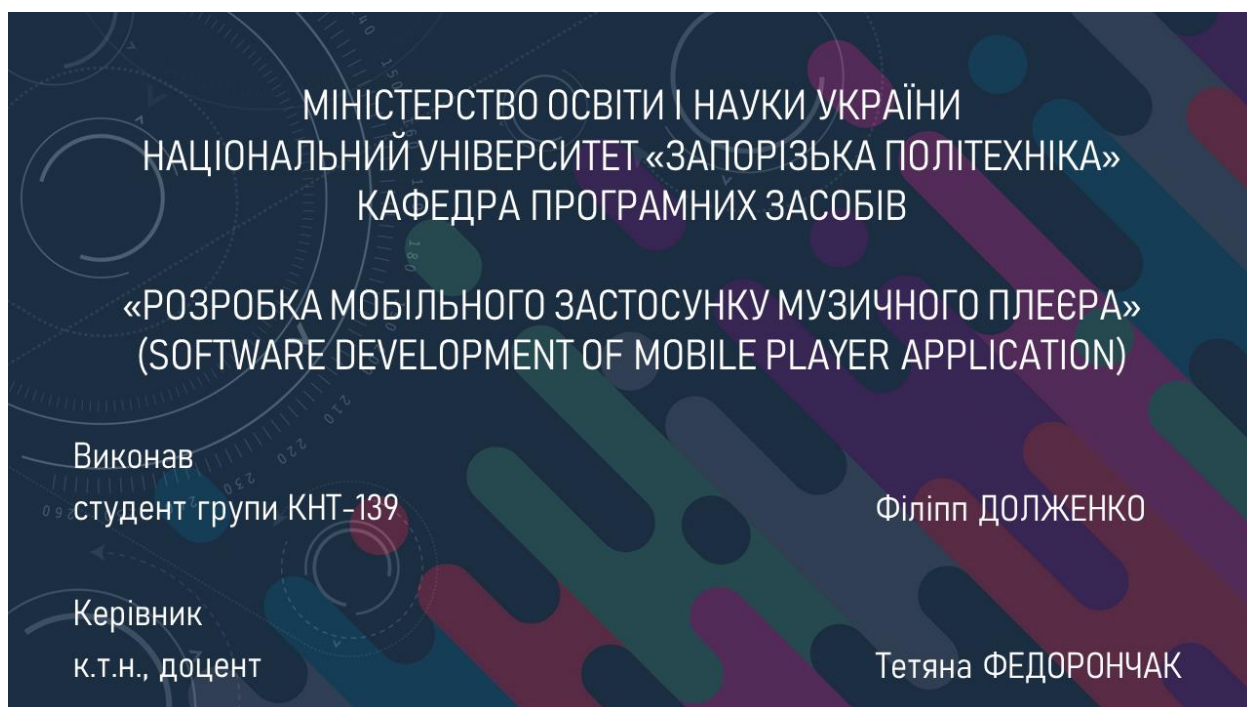


Рисунок В.1 – Слайд 1

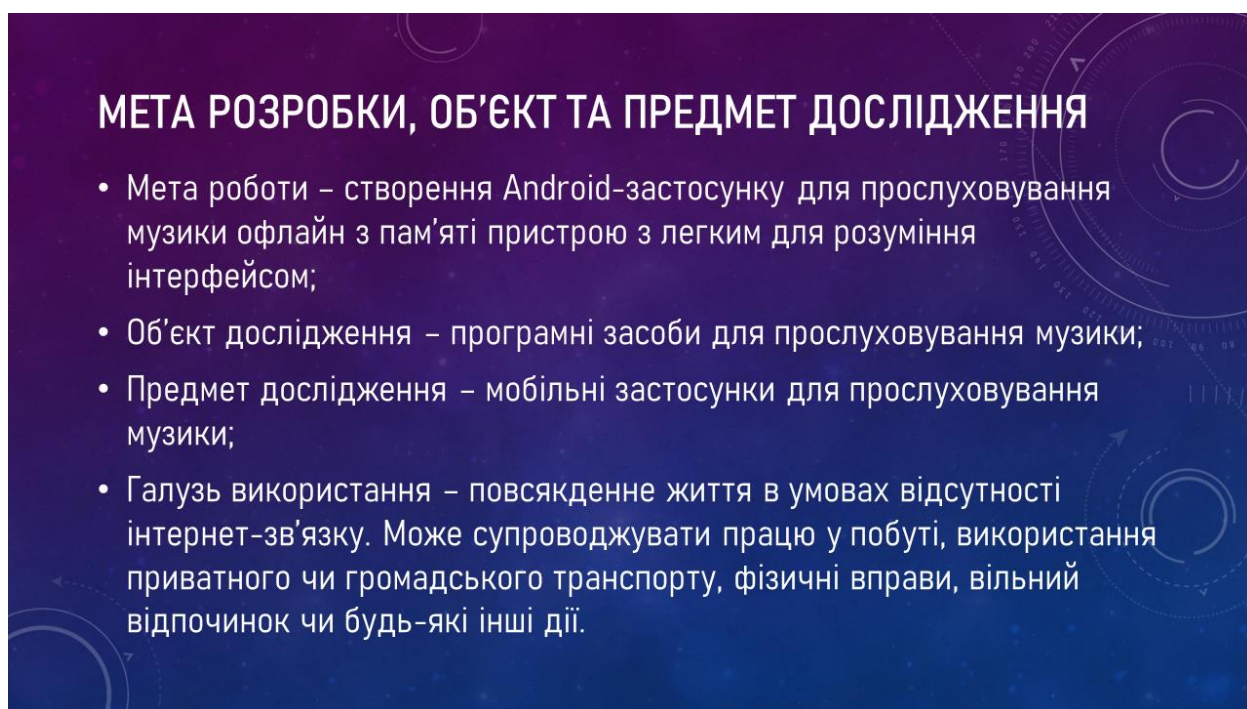


Рисунок В.2 – Слайд 2

ЗАДАЧІ РОБОТИ

Аналіз предметної області та аналогів застосунку, що розроблюється

Вибір платформи та її версії, для якої буде розроблюватися застосунок

Вибір мови програмування та середовища розробки

Розробка застосунку

Тестування застосунку на різних пристроях та версіях платформи

Рисунок В.3 – Слайд 3

ПОРІВНЯННЯ З ІСНУЮЧИМИ АНАЛОГАМИ

Критерій оцінки	Назва застосунку		
	Spotify	Samsung Music	VAC for Android
Інтуїтивна зрозумілість	+++	++	+
Зручність	++	+	+
Сучасність дизайну	+++	+++	+
Платний функціонал	Підписка Premium (\$10)	Немає	Немає
Наявність реклами	Є, якщо нема підписки	Є	Немає
Споживання пам'яті	Від 134 МБ	Від 58 МБ	Від 104 МБ

Рисунок В.4 – Слайд 4

ОСНОВНІ ФУНКЦІЇ ЗАСТОСУНКУ

- Створення аудіотеки з аудіофайлів, що зберігаються у пам'яті пристрою;
- Групування композицій в аудіотеці за альбомами, зазначеними в метаданих аудіофайлів;
- Надання можливості прослуховування композицій з аудіотеки та можливості керування цим процесом (перехід до попередньої або наступної композиції, призупинення та відновлення прослуховування).

Рисунок В.5 – Слайд 5

ВИБІР МОВИ ПРОГРАМУВАННЯ

Критерій порівняння	Назва пристрою та версія Android	
	Java	Kotlin
Популярність	Більш популярна та поширена мова	Активно набирає популярність
Стислість коду	Менш стислий код	Більш стислий код
Безпечність коду	Менш безпечний код	Більш безпечний код

Для розробки застосунку було обрано мову програмування Kotlin.

Рисунок В.6 – Слайд 6

ІНТЕРФЕЙС ЗАСТОСУНКУ. ГОЛОВНИЙ ЕКРАН

Інтерфейс заснований на кольоровій гамі з бежевих та коричневих кольорів (бістр і колір світлої кави). На головному екрані можна переглянути список композицій та альбомів, також наявний міні-плеєр.

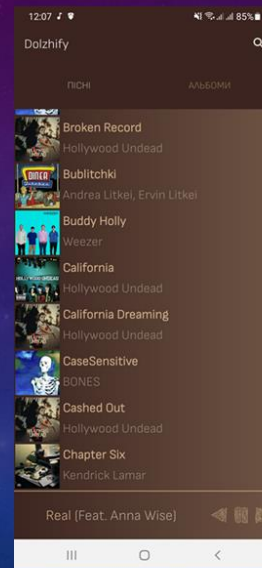


Рис. 1 – Список композицій



Рис. 2 – Список альбомів

Рисунок В.7 – Слайд 7

ІНТЕРФЕЙС ЗАСТОСУНКУ. ЕКРАНИ ПЛЕЄРА ТА АЛЬБОМУ

Програвання музики можна керувати за допомогою міні-плеєра на головному екрані, окремого екрану плеєра або плеєра у повідомленнях.

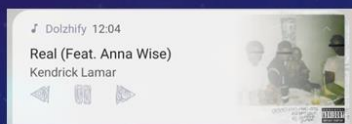


Рис. 3 – Плеєр у повідомленнях



Рис. 4 – Екран плеєра



Рис. 5 – Екран альбому

Рисунок В.8 – Слайд 8

РЕЗУЛЬТАТИ ТЕСТУВАННЯ ЗАСТОСУНКУ

Критерій тестування	Назва пристрою та версія Android		
	Samsung A50, Android 13.0 T	Oppo Reno4 Lite, Android 10.0 Q	Redmi 8A, Android 10.0 Q
Швидкодія застосунку	Відмінно	Відмінно	Відмінно
Формування аудіотеки	Коректне	Коректне	Коректне
Відображення інтерфейсу	Коректне	Коректне	Коректне
Програвання музики	Коректне	Коректне	Коректне
Функціонування пошуку	Коректне	Коректне	Коректне
Функціонування повідомлень	Коректне	Коректне	Коректне

Рисунок В.9 – Слайд 9

ВИСНОВКИ

Під час виконання дипломної кваліфікаційної роботи було розроблено застосунок-музичний плеєр зі зручним та сучасним користувацьким інтерфейсом. Застосунок вирішує актуальну проблему прослуховування музики при відсутності стабільного інтернет-зв'язку.

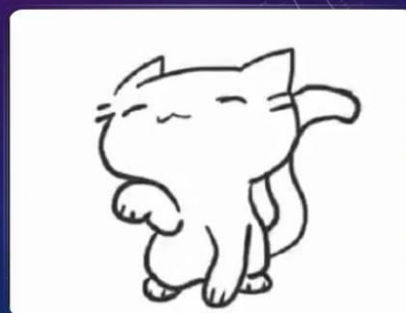


Рисунок В.10 – Слайд 10