DOI: https://doi.org/10.33216/1998-7927-2022-275-5-10-15

УДК 004.41:004.9

ДЕКЛАРАТИВНИЙ ПІДХІД ПРИ СТВОРЕННІ МУЛЬТИПЛАТФОРМНИХ ДОДАТКІВ

Козуб Г.О., Козуб Ю.Г.

DECLARATIVE METHOD FOR CREATING MULTIPLATFORM APPLICATIONS

Kozub H.O., Kozub Yu.H.

У роботі надано аналіз сучасних аналогів розробки кросплатформних та мультиплатформних додатків, досліджено інструменти розробки Kotlin Multiplatform та Jetpack Compose. Описано бізнес-логіку та методику створення інтерфейсу користувача під декілька платформ, що сприяє зменшенню вартості продукту та прискорення його розробки. Представлено методику розробки мультиплатформного додатку для операційних систем Android, Windows, Linux та macOS, який дозволяє створювати фонові хвилі у звуковому форматі. Досліджено принципи декларативного програмування та фреймворки для мультиплатформної розробки. Для програмної реалізації окремих нативних додатків, обрано нативні рішення. Такому рішенню сприяє використовування фреймворків Kotlin Multiplatform, що дозволяє створювати універсальний код логіки додатку, у поєднанні з UI фреймворком Compose Multiplatform. Їх застосування дає можливість написання єдиного коду логіки та інтерфейсу додатку для декількох платформ одночасно, що допомагає економити час і уникати значної кількості помилок. Запропоновано методику розробки архітектури компонентів Android-додатку. Модульна структура архітектури проєкту поділена на дві частини: на common модуль, який містить у собі основну логіку додатку, а також платформні реалізації компонентів, та платформні модулі, що виконують ініціалізацію та запуск додатку на певній платформі. Описано контракт бізнес-логіки компонентів, що реалізується у інтерфейсах Store. Показано формування формату мелодії додатку з трьох класів: Song, Track та Chord.. Для відтворення звуків створено спеціальну утиліту. Вона виконує завантаження звуків з ресурсів додатку та відтворює з наданням необхідного музичного тону. Звуки зберігаються в ресурсах спільного модуля у вигляді MP3 файлів. Для налагодження доступу до репозиторіїв з компонентів бізнес-логіки додатку, використовується впровадження залежностей. Логіка впровадження залежностей описується у файлах-модулях Koin. Запропоновано методику, що узагальнює методологію розробки мультиплатформних додатків на мові програмування Kotlin та використання декларативного підходу для розробки інтерфейсів користувача під декілька платформ.

Ключові слова: *MVIKotlin, Decompose, Android, Kotlin, Store,Common, Jetpack Compose, Kotlin Multiplatform.*

Вступ. В світі щорічно з'являється мільйони стартапів, користувачі стають все вимогливішими і час виведення програмного продукту на ринок стає набагато важливіше вартості, тому розробники обирають нативні рішення. Однак програмна реалізація окремих мобільних додатків під декілька платформ сприяє збільшенню витрат на їх підтримку. Серед розробників мобільних додатків, спостерігається тенденція вміння швидко вносити зміни у додаток на потребу ринку. Внаслідок чого виникає необхідність визначення ефективних підходів до розробки конкурентних мобільних мультиплатформних додатків.

На швидкість розробки, впровадження нових функціональностей, якість та зручності використання програмного додатку впливає застосування кросплатформних інструментів. Найвідомішими кросплатформними інструментами є: Xamarin, React Native, Flutter.

Xamarin – платформа від Microsoft для створення додатків під Android, iOS, Windows, Linux, macOS, watchOS та tvOS. Xamarin включає єдину загальну кодову базу C# і надає можливість тестувати додатки на декількох пристроях з використанням Xamarin Cloud. Xamarin має два основних інструменти: Xamarin.Android, Xamarin.iOS і Xamarin.Forms. По частині кросплатформної розробки Xamarin пропонує використовувати єдиний API Xamarin.Essentials. Xamarin.Android і Xamarin.iOS наділяють додаток тими ж можливостями і інтерфейсом, які є у нативних рішень. У випадку Xamarin.iOS програма компілюється безпосередньо в машинний код (AOT-компіляція), тоді як в Xamarin.Android спочатку відбувається компіляція в байт-код, який потім інтерпретується віртуальною машиною (JIT-компіляція). Для прискорення процесу написання коду, краще використовувати Xamarin.Forms, в якому майже всі елементи повністю сумісні з будь-якими платформами [1].

React Native розроблений компанією Facebook, дозволяє писати додатки для IOS, Android, Windows, Web, Windows Phone, VR, Android TV, macOS, tvOS на мові JavaScript. Середовище поставляється з ве-

ликим набором готових компонентів, однак вони не завжди адаптуються під різні платформи, що вимагає додаткових коригувань в коді. В гонитві за продуктивністю розробники віддають перевагу саме цьому фреймворку. Native дозволяє використовувати кастомні модулі на мовах для нативної розробки, але їх необхідно писати окремо для кожної платформи [2].

Безкоштовний кросплатформний фреймворк Flutter від Google з відкритим вихідним кодом для швидкої розробки додатків під Android, iOS, Windows, Linux, macOS, Web та Google Fuchsia, використовує об'єктно-орієнтовану мову програмування DartFlutter використовує один і той же код для всіх платформ, перевершує конкурентів і демонструє найвищу продуктивність завдяки власному движку рендерингу і сучасній мові Dart, яка була розроблена Google. Flutter включає сторонні SDK, API для 2D, анімації, власні віджети Material Design і надає можливість повторно використовувати існуючий код Java, Swift та Objective-C [3].

Перевагами косплатформної розробки є висока швидкість та низька вартість розробки. До недоліків відносяться: відносно низька продуктівність, у порівнянні з нативними додатками; займає більше місця у пам'яті; складна підтримка низкорівневих платформних функцій; нові платформні можливості з'являються пізніше ніж у нативних додатках;

Щоб вирішити ці недоліки, використовується декларативний підхід програмування з використанням фреймворків Kotlin Multiplatrofm та Jetpack Compose.

Мета роботи – дослідження підходів декларативного програмування при створенні мультиплатформних додатків під декілька платформ, з використанням сучасних інструментів та методів.

Викладення основних матеріалів. Для створення мульти-платформних додатків використовується мова програмування Kotlin та середовища розробки IntelliJ IDEA або Android Studio, за технологією, яка дозволяє використовувати один раз написаний код на безлічі платформ одночасно є фреймворк Kotlin Multiplatrofm, де система збірки Gradle підтримує синтаксис groovy і kotlin script (kts).

У Kotlin Multiplatform є поняття targets – цільові платформи. На рис. 1 показано, як у даних блоках налаштовуються необхідні нам операційні системи, в нативний код, де і компілюється код на Kotlin [4].

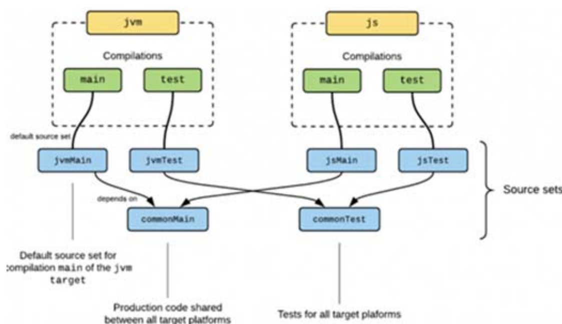

Рис. 1. Реалізація jvm та js таргетів [5]

Відповідна структура коду має вигляд:

```
kotlin{
  Jjvm()
  Js()
  source Sets {
     val commonMain by getting {}
     val commonTest by getting {}
     val jsMain by getting {}
     val js Test by getting {}
  }
}
```

Аналогічним чином реалізується підтримка інших платформ.

Source sets – це вихідні коди для платформ. Є загальний набір вихідних кодів і платформні (їх стільки, скільки в проєкті таргетів, за цим стежить IDE). Для реалізації цієї особливості використовується механізм expect-actual (рис. 2).

Рис. 2. Механізм expect-actual [5]

Expect-actual дозволяє із загального модуля звертатися до платформозалежного коду. Можна оголосити expect декларацію в Common модулі і реалізувати її в платформних модулях. Структура мульти-платформних проєктів складається зі спільного модуля та модулів платформ (рис. 3).

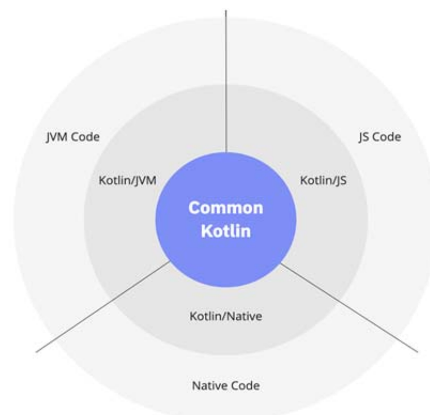

Рис. 3. Структура Kotlin Multiplatform

На цій схемі кожен шар (платформа) має позначення:

– Common Kotlin - спільний код для всіх платформ. Він включає в себе основну логіку додатку, спільні бібліотеки та інструменти. За допомогою мультиплатформних бібліотек Kotlin можна повторно використовувати мульти-платформну логіку в загальному і специфічному для платформи коді. Зага-

льний код може покладатися на набір бібліотек, які охоплюють повсякденні завдання, такі як HTTP-запити, серіалізація та управління співпрограмами;

– Kotlin/JVM, Kotlin/JS, Kotlin/Native – для взаємодії з платформами використовуються версії Kotlin. Вони мають розширення мови Kotlin, а також бібліотеки та інструменти для конкретних платформ (JVM, JS і Native).

За допомогою Kotlin Multiplatform витрачається менше часу на написання та підтримку одного й того ж коду для різних платформ і використовується механізм спільного коду [5].

Незважаючи, на те, що фреймворк знаходиться на стадії розробки і не вистачає готових рішень, він має свої суттєві переваги:

– єдиний код, який можна в будь-який час дописувати і змінювати;

– час на додавання нового функціоналу для декількох платформ скоротився на 30-50%;

– поділ бізнес-логіки між платформами зі збереженням власного коду для кожного клієнтського інтерфейсу;

– помилки в разі виникнення з’являються відразу на всіх платформах, що полегшує їх знаходження;

– будь-які зміни та коригування відбуваються одночасно на всіх платформах.

Декларативний підхід передбачає надання специфічної для предметної області мови (domain-specific language, DSL) для вираження того, що хоче користувач, і захисту її від низькорівневих конструкцій (циклів, умовних позначень, призначень), які матеріалізують бажаний кінцевий стан. Декларативне програмування можна додатково розділити на програмування обмежень, логічне програмування та програмування логіки обмежень [6].

Одним з прикладів використання декларативного програмування та DSL є UI фреймворк Jetpack Compose. Це фреймворк для створення інтерфейсу користувача для Android-додатків, має адаптації під Desktop (Windows, Linux, macOS) та Web. Він спрощує і прискорює розробку UI, зменшує необхідну кількість коду для вирішення задач та має потужні інструменти і інтуїтивно зрозумілі API-інтерфейси Kotlin [7].

При написанні коду програмного забезпечення для якого потрібна постійна підтримка, необхідно мінімізувати зв'язаність і максимізувати його цілісність. Це добре відомий принцип розробки програмного забезпечення Separation of concerns, і полягає в тому, щоб згрупувати якомога більше зв'язаного коду разом, щоб код можна було легко підтримувати та масштабувати, коли додаток зростає. Jetpack Compose заснований на функціональному підході до програмування. Для опису інтерфейсу користувача використовуються Composable функції:

```
@Composable
fun App(appData: AppData) {
  val derivedData = compute(appData)
  Header()
  if (appData.isOwner) {
    EditButton()
  }
  Body {
    for (item in derivedData.items) {
      Item(item)
    }
  }
}
```

Це означає, що при виклику інших Composable-функцій, вони відображають структуру UI. Для використання надаються всі примітиви Kotlin, включаючи оператори if і цикли for для управління структурою UI, щоб вирішувати більш складну логіку інтерфейсу користувача.

Однією з особливостей Jetpack Compose є інкапсуляція стану в Composable функціях. Стан в додатку – це будь-яке значення, яке може змінюватися з плином часу. Composable функція може мати інкапсульований в неї стан, який зберігається між її викликами. Це прагматичне рішення, яке спрощує його сигнатуру та дозволяє виконувати деякі оптимізації, що особливо зручно для анімацій та інших речей, які не потрібно змінювати або спостерігати зовні [7]. Переваги Jetpack Compose: незалежність від конкретних версій цільової платформи; вся робота з UI виконується за допомогою мови програмування Kotlin; використання композиції замість спадкування. UI-компонент описується у вигляді функції з анотацією Composable, яка відповідає тільки за обмежений функціонал, тобто без зайвої логіки; однонаправленість потоку даних; зменшення кількості коду для UI-логіки.

Результати досліджень. Для розробки додатків використано мову програмування Kotlin, це статично типізована мова програмування, що компілюється в JavaScript та працює поверх JVM [8]. Для опису логіки інтерфейсу користувача при використанні Jetpack Compose найкраще підходить патерн MVI [9]. У MVI взаємодія з інтерфейсом користувача обробляється бізнес-логікою, що вносить зміни в стан, який впливає на відображення інтерфейсу користувача. Це призводить до односпрямованого і циклічного потоку даних. Для реалізації реактивного отримання оновлених даних, використовуються засоби Kotlin, такі як Flow, StateFlow та Channel [10].

Спосіб написання спільного коду з використанням шаблону MVI для Kotlin Multiplatform надає фреймворк MVIKotlin. Він також включає в себе потужні інструменти налагодження, такі як логування та “ time travel ”.

MVIKotlin не приносить і не застосовує жодної конкретної архітектури. Його відповідальність описується наступним чином: надає єдине джерело істини для стану; забезпечує абстракцію для UI з ефективними оновленнями; забезпечує прив’язку до життєвого циклу між входами та виходами.

Основою MVIKotlin є Store, що представляє собою модель від MVI, яка містить бізнес-логіку (рис.4).

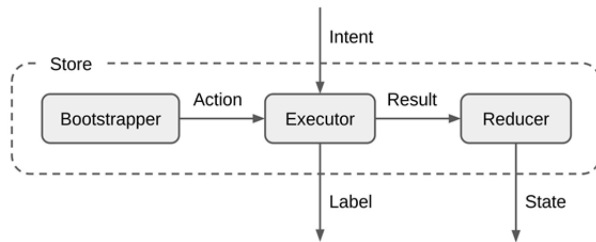

Рис. 4. Діаграма Store [11]

Store має такі компоненти: Intent – вхідна подія, яка приходить з UI; Label – одноразова вихідна подія, яка повертається до UI; State – стан, який відображається на UI; Bootstrapper – компонент, який виконує ініціалізацію Store, та при необхідності відправляє події до Executor; Action – події, які відправляє Bootstrapper до Executor; Executor – виконавець бізнес-логіки та усіх асинхронних операцій. Під час виконання або після завершення операцій, Executor може відправити Label до UI та Result до Reducer; Result – результат виконання операції, який передається до Reducer; Reducer – компонент, який приймає Result та змінює State [11].

Наступний код демонструє опис контракту бізнес-логіки компонентів реалізуємого у інтерфейсах Store з розробленого програмного застосування " Composer " [12]:

```
interface TestStore : Store<Intent, State, Nothing> {
  //Опис подій
  sealed class Intent {
    data class TabChanged(val newTab: Tab) : Intent()
  }
  //Опис стану
  data class State(
    val selectedTab: Tab = Tab.AllMusic
  )
  //Допоміжні сутності стану
  enum class Tab {
    AllMusic, AllPlaylists, AllSamples
  }
}
```

Реалізація контрактів Store виконується у класах StoreProvider.

Для роботи з мульти-платформними модулями, проведено налаштування системи збору проєктів Gradle та виконано у файлах: build.gradle.kts модуля common; build.gradle.kts модуля android; build.gradle.kts модуля desktop.

Архітектура проєкту поділена на дві частини: Common модуль містить у собі основну логіку додатку, а також платформні реалізації компонентів; платформні модулі - виконують ініціалізацію та запуск додатку на певній платформі (рис. 5).

Модуль спільної логіки Common складається з 5 частин:

– core пакет - містить класи типів даних додатку;

– data пакет - реалізує роботу з базою даних та надає класи-репозиторії для доступу до даних з feature пакету;

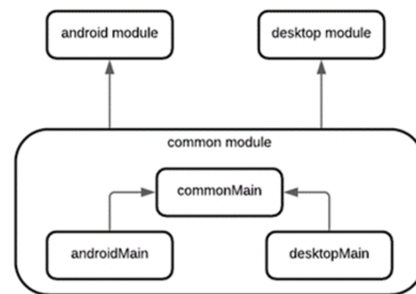

Рис. 5. Модульна структура додатку

– feature пакет - містить компоненти-екрани додатку з бізнес-логікою та UI частиною, а також описує навігацію між компонентами;

– di пакет - виконує налаштування dependency injection у додатку;

– база даних SQL Delight.

Платформні модулі виконують ініціалізацію головного компоненту додатку зі спільного модуля, вмикають DI та запускають додаток.

Ініціалізація додатку виглядає наступним чином:

– Вмикається dependency injection у класі App:

– Створюється головний компонент та використовується при створенні головного вікна додатку.

Формат мелодії додатку складається з трьох класів: Song, Track та Chord.

Класс Song містить у собі ідентифікатор, назву, ім'я композитора, назву альбому, ім'я переписувача, довжину мелодії у мілісекундах, час створення та редагування та список треків.

Класс Track описує сутність треку. Він має ідентифікатор, інструмент, тон мелодії, BPM, гучність та список акордів.

Класс Chord описує сутність акорду та містить час на звуковій доріжці у мілісекундах та список ідентифікаторів нот.

Для відтворення звуків створено спеціальну утиліту. Вона виконує завантаження звуків з ресурсів додатку та відтворює з наданням необхідного музичного тону. Звуки зберігаються в ресурсах спільного модуля у вигляді MP3 файлів.

Контракт утиліти описано у інтерфейсі SoundMachine:

```
interface SoundMachine {
    suspend fun loadSounds(soundsPaths: List<String>):
List<Sound>
    suspend fun playSound(sound: Sound, pitch: Double = 1.0)
}
```

Для налагодження доступу до репозиторіїв з компонентів бізнес-логіки додатку, використовується впровадження залежностей Dependency Injection. Логіка впровадження залежностей описується у файлах-модулях Koin. DatabaseModule містить модулі розробленого об'єкту бази даних для кожної платформи окремо: Common модулі;

PreferencesModule; RepositoryModule; SoundMashineModule. Всі модулі впровадження залежностей ініціюються у функції initKoin.

Висновки. В роботі розглянуто принципи декларативного підходу для розробки інтерфейсів користувача під декілька платформ. Проведено дослідження та впровадження сучасних інструментів та методик розробки мультиплатформних додатків: Compose Multiplatform, Jetpack Compose, MVIKotlin, Dependency Injection. За запропонованою методикою, що узагальнює методологію розробки мультиплатформних додатків на мові програмування Kotlin, спроєктовано структуру додатку та створено архітектурний патерн MVI, який найкраще підходить під декларативний стиль мульти-платформного фреймворку Compose Multiplatform. Розроблено проєкт “Composer”, що дозволяє створювати музичні мелодії у новому форматі, а також проводити з ними різні маніпуляції.

Література

1. Xamarin documentation. URL: https://docs.microsoft.com/en-us/xamarin/ (дата звернення: 25.12.2022).
2. React Native. URL: https://reactnative.dev/ (дата звернення: 25.12.2022).
3. Flutter. URL: https://flutter.dev/ (дата звернення: 25.12.2021).
4. Как Kotlin Multiplatform помогает сократить время разработки приложений. URL: https://habr.com/ua/ post/525888/ (дата звернення: 25.12.2022).
5. Почему мы выбрали Kotlin одним из целевых языков компании. Часть 2: Kotlin Multiplatform. URL: https://habr.com/ru/company/domclick/blog/499820/ (дата звернення: 25.12.2022).
6. Declarative Programming: Is It A Real Thing?. URL: https://www.toptal.com/software/declarative-programming (дата звернення: 25.12.2022).
7. Understanding Jetpack Compose – part 1 of 2. URL: https://medium.com/androiddevelopers/understanding-jetpack-compose-part-1-of-2-ca316fe39050 (дата звернення: 25.12.2022).
8. Kotlin Programming Language. URL: https://kotlinlang.org (дата звернення: 25.12.2022).
9. Козуб, Г., Козуб Ю., Могильний Г., Жуков А. «Розробка мобільного Android-додатку з застосуванням принципів Clean Architecture». *Вісник Східноукраїнського національного університету імені Володимира Даля*, вип. 5 (269), Вересень 2021, с. 5–10, doi:10.33216/1998-7927-2021-269-5-5-10.
10. Android MVI-Reactive Architecture Pattern. URL: https://abhiappmobiledeveloper.medium.com/android-mvi-reactive-architecture-pattern-74e5f1300a87 (дата звернення: 25.12.2022).
11. MVIKotlin Overview. URL: https://arkivanov.github.io/ MVIKotlin/ (дата звернення: 25.12.2022).
12. Жуков А. В., Козуб Г.О. Застосування фреймворку Jetpack Compose у багатомодульному Android-додатку. *Вітчизняна наука на зламі епох: проблеми та перспективи розвитку :* Зб. наук. праць. Переяслав, 2021. Вип. 67. С.109–111.

References

1. Xamarin documentation. URL: https://docs.microsoft.com/en-us/xamarin/ (data zvernennja: 25.12.2022).
2. React Native. URL: https://reactnative.dev/ (data zvernennja: 25.12.2022).
3. Flutter. URL: https://flutter.dev/ (data zvernennja: 25.12.2021).
4. Kak Kotlin Multiplatform pomogaet sokratit' vremja razrabotki prilozhenij. URL: https://habr.com/ua/post/525888/ (data zvernennja: 25.12.2022).
5. Pochemu my vybrali Kotlin odnim iz celevyh jazykov kompanii. Chast' 2: Kotlin Multiplatform. URL: https://habr.com/ru/company/domclick/blog/499820/ (data zvernennja: 25.12.2022).
6. Declarative Programming: Is It A Real Thing?. URL: https://www.toptal.com/software/declarative-programming (data zvernennja: 25.12.2022).
7. Understanding Jetpack Compose – part 1 of 2. URL: https://medium.com/androiddevelopers/understanding-jetpack-compose-part-1-of-2-ca316fe39050 (data zvernennja: 25.12.2022).
8. Kotlin Programming Language. URL: https://kotlinlang.org (data zvernennja: 25.12.2022).
9. Kozub, G., Kozub Ju., Mogil'nij G., Zhukov A. 2021. Rozrobka mobil'nogo Android-dodatku z zastosuvannjam principiv Slean Architecture. *Visnik Shidnoukraïns'kogo nacional'nogo universitetu imeni Volodimira Dalja,* 5(269), 5–10, doi:10.33216/1998-7927-2021-269-5-5-10.
10. Android MVI-Reactive Architecture Pattern. URL: https://abhiappmobiledeveloper.medium.com/android-mvi-reactive-architecture-pattern-74e5f1300a87 (data zvernennja: 25.12.2022).
11. MVIKotlin Overview. URL: https://arkivanov.github.io/MVIKotlin/ (data zvernennja: 25.12.2022).
12. Zhukov A. V., Kozub G.O. 2021. Zastosuvannja frejmvorku Jetpack Compose u bagatomodul'nomu Android-dodatku. *Vitchiznjana nauka na zlami epoh: problemi ta perspektivi rozvitku : Zb. nauk. prac'. Perejaslav*, 2021, 67, 109–111.

Kozub H. O., Kozub Yu. H. Declarative method for creating multiplatform applications

The work provides an analysis of modern analogues of cross-platform and multi-platform application development, the Kotlin Multiplatform and Jetpack Compose development tools are investigated. The business logic and methodology of creating a user interface for several platforms are described, which contributes to reducing the cost of the product and accelerating its development. The method of developing a multi-platform application for Android, Windows, Linux and macOS operating systems is presented, which allows you to create background waves in sound format. The principles of declarative programming and frameworks for multi-platform development are studied. For the software implementation of individual native applications, native solutions have been chosen. This solution is facilitated by the use of Kotlin Multiplatform frameworks, which allows you to create universal application logic code, in combination with the Compose Multiplatform UI framework. Their use makes it possible to write a single logic code and application interface for several platforms at the same time, which helps to save time and avoid a significant number of errors. A methodology for developing the architecture of Android application components is proposed. The modular structure of the project architecture is divided into two parts: a common module, which contains the main logic of the application, as well as

platform implementations of components, and platform modules that perform initialization and launch of the application on a certain platform. The business logic contract of components implemented in Store interfaces is described. The formation of the melody format of the application from three classes is shown: Song, Track and Chord.. A special utility has been created to reproduce sounds. It downloads sounds from the application's resources and plays them with the necessary musical tone. Sounds are stored in the shared module resources as MP3 files. Dependency implementation is used to establish access to repositories from application business logic components. The logic of implementing dependencies is described in Koin module files. A methodology is proposed that summarizes the methodology of developing multi-platform applications in the Kotlin programming language and the use of a declarative approach for the development of user interfaces for several platforms.

***Keywords**: MVIKotlin, Decompose, Android, Kotlin, Store,Common, Jetpack Compose, Kotlin Multiplatform.*

Козуб Галина Олександрівна – к. т. н., доцент, доцент кафедри інформаційних технологій і систем Луганського національного університету імені Тараса Шевченка (м. Полтава), galina14kz@gmail.com
Козуб Юрій Гордійович – д. т. н., доцент, завідувач кафедри фізико-технічних систем та інформатики Луганського національного університету імені Тараса Шевченка (м. Полтава), kosub.yg@gmail.com

Стаття подана 09.10.2022.