ISSN 2224-087X. Електроніка та інформаційні технології. 2023. Випуск 21. С. 24–33
Electronics and information technologies. 2023. Issue 21. P. 24–33

УДК 004.932 DOI: https://doi.org/10.30970/eli.21.3

АНАЛІЗ ЕФЕКТИВНОСТІ ВИКОРИСТАННЯ ТЕХНОЛОГІЇ KOTLIN MULTIPLATFORM MOBILE ДЛЯ СТВОРЕННЯ КРОСПЛАТФОРМНИХ ДОДАТКІВ

І. Оленич, Р. Коростенський

Львівський національний університет імені Івана Франка,
вул. Драгоманова, 50, 79005 Львів, Україна
igor.olenych@lnu.edu.ua

Розглянуто особливості застосування технології Kotlin Multiplatform Mobile (KMM) для створення кросплатформних мобільних додатків на прикладі сумісних з операційними системами Android та iOS застосунків різної структурної складності. Виявлено зменшення сумарного обсягу кодової бази Android та iOS версій мобільних додатків у результаті винесення бізнес логіки програмних продуктів у спільний KMM модуль. Досліджено зміну часу збірки мобільних додатків і KMM модулів з різним обсягом кодової бази на платформах ОС Android та iOS. Продемонстровано підвищення ефективності розробки кросплатформних мобільних додатків завдяки використанню спільного KMM модуля.

Ключові слова: мобільний додаток, кросплатформна розробка, структура програмного забезпечення.

1. Вступ

Розробка мобільних додатків – надзвичайно динамічна область програмування, що визначається стрімким розвитком різноманітних мобільних пристроїв, робота яких пов’язана з різними мережевими платформами, стандартами, протоколами та технологіями [1–3]. Популярність мобільних додатків зумовлена можливістю задовольнити майже всі інформаційні потреби користувачів. Завдяки своїй гнучкості, такі застосунки можуть поєднати у сучасних мобільних пристроях різні функціональні елементи та забезпечити простоту їх керування для великої кількості користувачів, що є вирішальною перевагою мобільних програм.

Щоб максимізувати присутність програмного продукту на ринку, необхідно забезпечити можливість його роботи на якомога більшій кількості мобільних пристроїв з різними операційними системами (ОС). Одне з можливих рішень пов’язано з нативною розробкою мобільних додатків на кожній з існуючих платформ з використанням інтегрованого середовища розробки (IDE), яке надає необхідні інструменти для створення та налагодження програмних продуктів. Кожна платформа має свої нативні мови, “рідні” для цієї ОС, які дають змогу максимально використовувати можливості мобільної платформи та її апаратного забезпечення [4–7]. Зокрема, щоб створити програмні продукти для двох найпопулярніших мобільних ОС – Android та iOS, зазвичай використовують мови програмування Java або Kotlin та Objective C або ж Swift, відповідно.

Нативні додатки характеризуються високою продуктивністю та ефективністю, адже завдяки прямій інтеграції мобільних додатків з операційною платформою досягається максимальна відповідність можливостей програмного забезпечення з апаратними можливостями пристроїв, що підвищує швидкість і стабільність роботи програми. Однак, нативні додатки потребують більших витрат на розробку та підтримку, оскільки для кожної цільової платформи необхідно створювати подібні за функціональністю програми з адаптованими до ОС бізнес-логікою, інтерфейсом тощо. Враховуючи велику кількість версій і різновидів ОС, створювати окреме рішення для кожної платформи складно і недоцільно.

Інший підхід до розробки мобільних додатків базується на кросплатформних рішеннях, які адаптують програмне забезпечення до багатьох ОС [8]. Мультиплатформний підхід дає змогу оптимізувати процеси розробки програмного забезпечення на одній кодовій базі і навіть однією мовою програмування та суттєво підвищити їхню ефективність. У результаті компіляції кросплатформного додатку можна одержати виконувані файли для різних ОС. Крім економії бюджету на створення мобільного застосунку до переваг кросплатформних рішень можна віднести високу швидкість і простоту розгортання на різних пристроях, однаковий зовнішній вигляд та інтерфейс, що створює додаткові зручності для користувачів. Проте не завжди вдається адаптувати мобільний додаток для максимально ефективного використання можливостей кожної ОС, що призводить до зниження його гнучкості та продуктивності. Незважаючи на це, кросплатформні мобільні додатки доволі популярні, а пошук шляхів підвищити ефективність та продуктивність їх розробки є актуальною задачею.

Крім того, постійний розвиток і розширення інструментарію для розробки мобільних додатків потребує їх аналіз та уніфікацію [9]. Зараз на ринку є чимало фреймворків для спрощення розробки кросплатформних проєктів. Перспективною є технологія Kotlin Multiplatform, яка допомагає економити час та кошти розробників, зберігаючи при цьому гнучкість і переваги нативного програмування [10]. Kotlin Multiplatform Mobile (KMM) [11], як набір засобів розробки (software development kit, SDK) кросплатформних мобільних додатків, забезпечує спільне використання коду мобільними ОС Android та iOS, зменшуючи зусилля та час, затрачені на написання та підтримку тієї самої кодової бази (наприклад, для бізнес-логіки, роботи в мережі, зберігання даних тощо) для різних платформ. Реалізація користувацького інтерфейсу (user interface, UI) здійснюється за допомогою нативних для конкретної платформи інструментів завдяки створенню різних компіляторів для мови програмування Kotlin, сумісних з відповідними ОС.

У роботі створено сумісні з ОС Android та iOS мобільні застосунки різної структурної складності з метою дослідження особливостей застосування технології KMM для створення кросплатформних додатків. Особлива увага зосереджена на пошуку оптимальних шляхів розробки програмних компонентів, які забезпечать збільшення продуктивності розробки завдяки повторному використанні коду, а також збереження функціональності додатків, їх надійності та зручності для користувачів.

2. Методи та засоби реалізації

Для аналізу ефективності використання технології KMM для створення та підтримки кроспратформних додатків було реалізовано три мобільні додатки різної складності та різним обсягом кодової бази. Найпростіший мобільний додаток (МД1) дає

змогу відслідковувати запуски ракет компанії SpaceX, а саме - отримувати інформацію про минулі та майбутні запуски, отримувати детальну інформацію про ракетоносій, відображати статус запуску. Для отримання необхідної інформації додаток використовує сторонній API [12]. Схему роботи додатку МД1 ілюструє рис. 1.

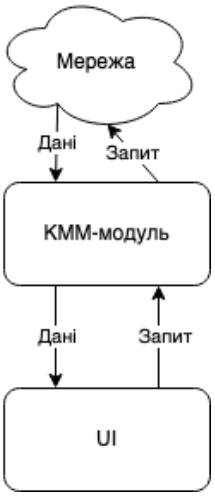

Рис. 1. Схема роботи мобільного додатку МД1 для відслідковування запусків ракет компанії SpaceX.

Наступний мобільний додаток для планування завдань (МД2) є дещо складнішим за попередній, бо, окрім завантаження даних з мережі, передбачає також їх зберігання (кешування) для використання додатку без доступу до мережі Інтернет. Функціонал мобільного додатку, який схематично проілюстровано на рис. 2, забезпечує завантаження даних з мережі Інтернет, локальне зберігання завантажених даних, зчитування та обробку даних, введених користувачем, візуалізацію існуючих даних та оновлення свого стану.

Найскладніший за структурою мобільний додаток (МД3), який використовувався для досліджень, дає змогу редагувати короткі відеозаписи та аналізувати їх за допомогою методів машинного навчання. Він містить усі елементи, що були у попередніх додатках, а також додаткові інструменти FFmpeg для обробки відеозаписів і Tensorflow Lite та CoreML для їх аналізу на платформах Android та iOS, відповідно. На рис. 3 показано функціональну схему кросплатформного додатку МД3, відповідно до якої користувач має змогу редагувати короткі відеозаписи та зберігати результат у файловій системі. Після обробки відео потрапляє на аналіз до відповідного ОС модуля машинного навчання (Tensorflow Lite або CoreML) для пошуку заданих об’єктів. Результати аналізу виводяться користувачу, а також зберігаються у локальній базі даних та вивантажуються на хмарний сервіс. Слід зазначити, що зі зростанням складності розроблених мобільних додатків збільшується обсяг їх кодової бази.

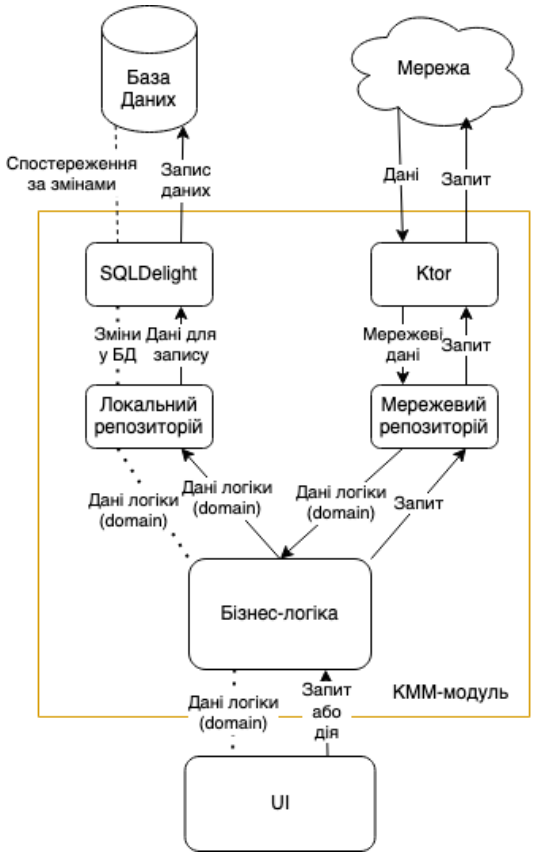

Рис. 2. Схема роботи мобільного додатку МД2 для планування завдань.

Розробка мобільних додатків здійснювалась за допомогою мови програмування Kotlin, яка має декілька компіляторів і дає змогу реалізувати мультиплатформність програмних продуктів. Винесення спільної логіки в окремий модуль здійснювалось за допомогою технології KMM. Для роботи над програмними продуктами використовувалось середовище розробки Android Studio, а для збірки iOS, iPadOS, watchOS і macOS версій створених додатків використовувалось середовище розробки Xcode. Користувацький інтерфейс для Android був реалізований за допомогою технології Jetpack Compose, а для платформ iOS, macOS, watchOS, tvOS було використано технологію SwiftUI. Тестування розроблених мобільних додатків проводилося на пристроях Google Pixel 4a з ОС Android 13 та iPhone 11 з iOS 16.1.

3. Результати та їх аналіз

Основною ідеєю технології Kotlin Multiplatform є винесення спільної логіки у окремий модуль, де вона може працювати незалежно від реалізацій платформи. Такий підхід добре узгоджується з сучасними принципами побудови програмних продуктів [13], згідно яких архітектуру програмного забезпечення розділяють на ієрархічні

компоненти чи шари. Внутрішні шари архітектури описують бізнес правила, за якими працює програмний продукт, а зовнішні виконують визначені функції. Зміна будь-якого елемента зовнішнього шару не впливатиме на роботу внутрішніх шарів. Це дозволяє доволі легко масштабувати та підтримувати програмне забезпечення.

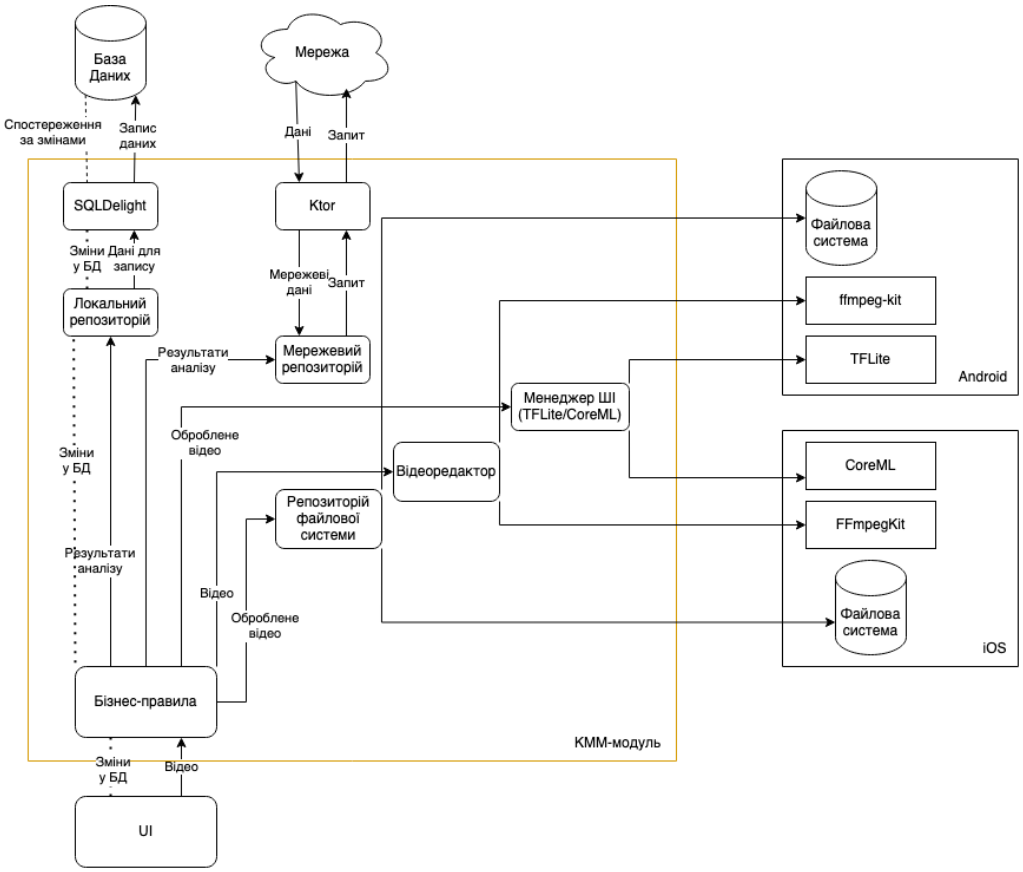

Рис. 3. Схема роботи мобільного додатку МДЗ для обробки та аналізу коротких відеозаписів.

Модуль-ядро розроблених мобільних додатків, де описано їхню поведінку, компілюється та збирається у декілька артефактів, які в свою чергу, використовуються як “рідна” бібліотека при збірці кінцевого продукту для кожної з платформ. Наприклад, для роботи з мережею Інтернет технологія KMM використовує механізм expect та actual компонентів. Перший компонент відповідає за оголошення функції у спільному модулі, а другий – за її реалізацію на різних платформах. Використовуючи бібліотеку Ktor, яка містить увесь необхідний функціонал для роботи з мережею, було створено екземпляр класу HttpClient на основі технології OkHttp для ОС Android і технології URLSession для iOS. Перелік бібліотек, які використовувались у спільних KMM-модулях розроблених мобільних додатків і модулях, пов’язаних тільки з платформою Android або iOS,

наведено у табл. 1. Загалом, набір бібліотек KMM-модуля визначається тим, яку кількість логіки розробники хочуть поширити між платформами.

Таблиця 1. Перелік бібліотек, використаних для розробки мобільних додатків.

Мобільний додаток	KMM-модуль	Android-модуль	iOS-модуль
МД1	Ktor Kotlinx-DateTime Kotlin Coroutines Kotlinx.serialization Koin Nappier	Koin Kotlin Coroutines Jetpack Compose	SwiftUI Combine Swinject
МД2	SQLDelight Ktor Kotlinx-DateTime Kotlin Coroutines Kotlinx.serialization Koin Nappier	Koin Kotlin Coroutines Jetpack Compose	SwiftUI Combine Swinject IQKeyboardManager
МД3	SQLDelight Ktor Kotlinx-DateTime Kotlin Coroutines Kotlinx.serialization Koin Nappier	Koin Kotlin Coroutines Jetpack Compose CameraX Tensorflow Lite FFmpeg-kit	SwiftUI Combine Swinject IQKeyboardManager AVFoundation CoreML FFmpegKit

У мобільному додатку МД1 у спільний модуль було винесено бізнес-правила, роботу з мережею та кешування. Структура спільних KMM-модулів мобільних додатків МД2 і МД3 зображена на рис. 2 і рис. 3, відповідно. Формування спільних модулів дало змогу зменшити сумарну кодову базу (кількість рядків програмного коду) обох Android та iOS версій мобільних додатків приблизно на 19–28%, як це можна побачити на рис. 4. Причому, максимальне збільшення продуктивності процесу розробки програмного забезпечення (тобто, найбільший відсоток зменшення сумарної кодової бази) завдяки повторному використанні коду спостерігалося для структурно складніших додатків. Обсяг KMM-модуля найпростішого додатку МД1 становить 637 рядків програмного коду, а кодова база Android та iOS модулів зменшилась приблизно на 44% порівняно з вихідними версіями додатку. У складніших за структурою додатках МД2 та МД3 кодова база спільних KMM-модулів збільшується та становить 1212 і 2347 рядків, відповідно. Крім того, збільшується співвідношення між обсягом вихідних нативних версій мобільних додатків і обсягом Android та iOS модулів відповідних кросплатформних застосунків. Зокрема, кодова база Android та iOS модулів мобільного додатку МД3 зменшилась на більш ніж 50% порівняно з вихідними версіями. Слід зазначити, що при використанні спільних KMM-модулів не спостерігалося ніяких змін у продуктивності розроблених додатків, а саме їх функціональності, швидкодії, стійкості та надійності.

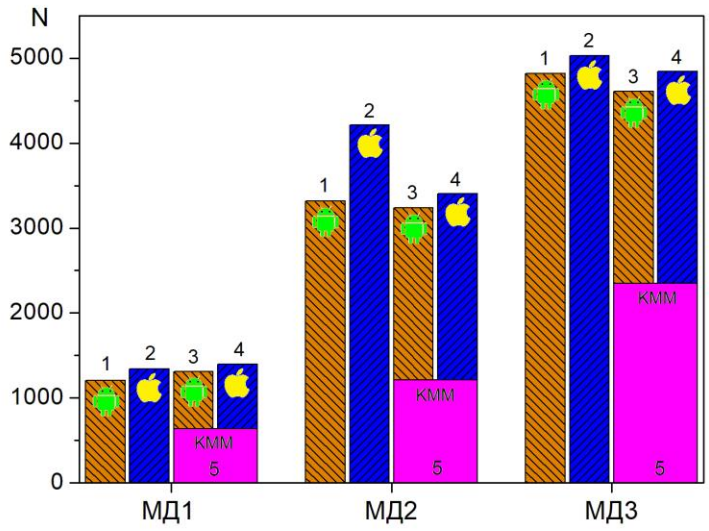

Рис. 4. Обсяг кодової бази вихідних Android (1) та iOS (2) версій розроблених мобільних додатків, а також їх Android (3), iOS (4) і KMM (5) модулів.

Загалом, усі розглянуті мобільні додатки незалежно від їх структурної складності характеризувалися більшим обсягом кодової бази їх iOS версій. Це може бути пов'язано з особливостями використаних фреймворків Jetpack Compose та SwiftUI для створення користувацьких інтерфейсів мобільних додатків, а також з різними технологіями штучного інтелекту TensorFlow Lite та CoreML, використаних у Android та iOS версіях додатку МД3, відповідно.

Іншим параметром, який досліджували у роботі, є час збірки розроблених мобільних додатків, а також їх KMM модулів. На основі серії вимірювань було визначено середній час збірки програмних продуктів, який наведено у табл. 2. Перед кожним вимірюванням часу збірки попередньо згенеровані файли були видалені.

Таблиця 2. Середній час збірки мобільних додатків та їх KMM модулів.

Мобільний додаток	Час збірки Android версії додатку, с			Час збірки iOS версії додатку, с		
	Вихідний додаток	Додаток з KMM модулем	KMM модуль	Вихідний додаток	Додаток з KMM модулем	KMM модуль
МД1	15,50	16,58	8,09	15,34	17,39	10,54
МД2	28,52	32,31	11,32	29,50	36,44	15,45
МД3	44,59	48,45	20,48	43,25	51,25	24,33

Час збірки мобільних додатків прогнозовано зростав із збільшенням їх складності і, відповідно, обсягу кодової бази. Крім того, було виявлено збільшення часу збірки

кросплатформних додатків у разі використання спільного KMM модуля. Причому час, на який збільшилась тривалість збірки розроблених застосунків з KMM модулем, залежав як від їх складності, так і від платформи реалізації. Зокрема, збірка Android версій мобільних додатків у разі використання спільного модуля стала тривалішою на 1–3 с, а час збірки iOS версій збільшився на 2–8 с. Незважаючи на збільшення тривалості збірки мобільних додатків, винесення бізнес-логіки, роботи з мережею та кешування у спільний KMM модуль забезпечує повторне використання коду і, як наслідок, підвищення ефективності процесу розробки кросплатформних додатків.

Варто зазначити, що час збірки спільних KMM модулів кросплатформних додатків був менший у випадку їх реалізації на платформі Android. Це зумовлено особливостями компіляції програмного коду для ОС Android та iOS. Технологія KMM використовує звичну для Android застосунків систему збірки Gradle, яка передбачає компіляцію вихідного коду та необхідних Android бібліотек у проміжний байт-код та виконувальні файли [14]. Процес збірки iOS версії кросплатформного застосунку з KMM модулем дещо складніший за процес збірки його Android версії. Він потребує окрему компіляцію програмних компонентів за допомогою різних компіляторів. Зокрема, iOS модулі компілюють за допомогою "рідних" для них компіляторів Swiftc або Clang, а компілятор Kotlin/Native застосовують до програмного коду KMM модуля. Отримані у результаті компіляції KMM модулів проміжні IR (intermediate representation) файли перетворюють у необхідний формат за допомогою технології LLVM (Low Level Virtual Machine) [15]. Тільки після цього всі згенеровані файли збирають за допомогою програми-компонувальника у виконуваний файл iOS версії мобільного додатку. Зазначений підхід є доволі надійним, проте характеризується більшим часом процесу збірки кросплатформних мобільних додатків.

4. Висновки

У роботі досліджено ефективність використання технології KMM у процесі розробки кросплатформних мобільних додатків. Для цього реалізовано Android та iOS версії трьох застосунків різної структурної та функціональної складності. На основі аналізу обсягу кодової бази і часу збірки програмних продуктів визначено основні особливості винесення бізнес-логіки, роботи з мережею та кешування у спільний KMM модуль. Виявлено зменшення сумарного обсягу кодової бази Android та iOS версій мобільних додатків на 19–28% залежно від їх структурної складності. Крім того, виявлено збільшення приблизно на 7–23% часу збірки мобільних додатків внаслідок використання спільного KMM модуля. Причому час збірки KMM модуля мобільних додатків на платформі iOS був більший ніж на платформі Android, що зумовлено особливостями компіляції програмного коду на цих платформах і використанням додаткових засобів у випадку iOS. Незважаючи на це, застосування Kotlin Multiplatform дає змогу підвищити ефективність процесу розробки кросплатформних мобільних додатків.

Список використаних джерел

[1] *Ngai E.W.T., Gunasekaran A.* A review for mobile commerce research and applications // Decision Support Systems. – 2007. – Vol. 43. – P. 3–15.

[2] *Islam Md.R., Islam Md.R., Mazumder T.A.* Mobile Application and Its Global Impact // International Journal of Engineering and Technology. - 2010. - Vol. 10. - P. 104-111.

[3] *Chandi L., Silva C., Martínez D., Gualotuna T.* Mobile application development process: A practical experience // 12th Iberian Conference on Information Systems and Technologies. - 2017. DOI:10.23919/CISTI.2017.7975825.

[4] *Deshmukh R.K., Markandey S., Sahu P.* Mobile Application Development with Android // International Journal of Advances in Applied Sciences. – 2018. – Vol. 7. – P. 317–321.

[5] Android Developer Platform. – 2019. [Online]. Available: https://developer.android.com/guide/platform

[6] *Beaton T.* Introduction to iOS Development. – 2016. [Online]. Available: https://learn.adafruit.com/introduction-to-ios-development

[7] Apple Developer Swift. – 2023. [Online]. Available: https://developer.apple.com/swift/

[8] *Xanthopoulos S., Xinogalos S.* A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications // 6th Balkan Conference in Informatics. - 2013. DOI:10.1145/2490257.2490292.

[9] *Jabangwe R., Edison H., Duc A.N.* Software engineering process models for mobile app development: A systematic literature review // Journal of Systems and Software. – 2018. – Vol. 145. – P. 98–111.

[10] Kotlin Multiplatform. – [Online]. Available: https://kotlinlang.org/docs/multiplatform.html

[11] Kotlin Multiplatform Mobile Goes Alpha. – 2023. [Online]. Available: https://blog.jetbrains.com/kotlin/2020/08/kotlin-multiplatform-mobile-goes-alpha/

[12] SpaceX REST API. – 2023. [Online]. Available: https://github.com/r-spacex/SpaceX-API

[13] *Robert C. Martin*. Clean Architecture: A Craftsman's Guide to Software Structure and Design. – Prentice Hall, 2018.

[14] Configure your build. – [Online]. Available: https://developer.android.com/build

[15] The LLVM Compiler Infrastructure. – [Online]. Available: https://llvm.org/

ANALYSIS OF THE EFFECTIVENESS OF USING KOTLIN MULTIPLATFORM MOBILE TECHNOLOGY FOR CREATING CROSS-PLATFORM APPLICATIONS

I. Olenych, R. Korostenskyi

Ivan Franko National University of Lviv,
50 Drahomanov St., UA–79005 Lviv, Ukraine
igor.olenych@lnu.edu.ua

Given the popularity of various mobile applications, the search for ways to improve the efficiency and productivity of their development is an actual task. One possible approach to mobile application development is based on cross-platform solutions that adapt the software to multiple operating systems (OS). A multi-platform approach optimizes software development processes on one code base that significantly increases their efficiency. In particular, Kotlin Multiplatform Mobile (KMM) technology makes it possible to implement cross-platform projects by providing code sharing between Android and iOS while maintaining the flexibility and advantages of native programming.

The Android and iOS versions of three applications with different structural and functional complexity have been implemented to study the effectiveness of using the KMM technology in the process of developing cross-platform mobile applications. The main features of transferring

business logic into a shared KMM module are determined on the basis of the analysis of the code base volume and the build time of software components. A decrease in the total volume of the code base of Android and iOS versions of mobile applications was revealed by 19–28%, depending on their structural complexity. In addition, an approximately 7–23% increase in the build time of mobile applications due to the use of shared KMM modules was found. It has been established that the build time of the shared KMM modules of the developed mobile applications on the iOS platform was longer than on the Android platform due to the use of additional tools for the compilation. No changes were found in the performance of the developed applications, namely their functionality, speed, stability, and reliability when using shared KMM modules. Therefore, the KMM technology provides an increase in the efficiency of the process of developing cross-platform mobile applications.

Key words: mobile application, cross-platform development, software structure.

Стаття надійшла до редакції 26.03.2023
Прийнята до друку 29.03.2023