DOI: https://doi.org/10.25140/2411-5363-2025-2(40)-275-283
УДК 004.415.2

Олег Миколайович Ілляшенко
аспірант кафедри комп'ютерної інженерії та інноваційних технологій
Міжнародний гуманітарний університет (Одеса, Україна)
E-mail: ctanok@gmail.com. **ORCID:** https://orcid.org/0009-0004-9184-1733

ПОРІВНЯЛЬНИЙ АНАЛІЗ РОЗМІРУ ІНСТАЛЯЦІЙНОГО ПАКЕТА І ЧАСУ ЗАПУСКУ МОБІЛЬНОГО ЗАСТОСУНКУ

У статті розглянуто три технологічні стеки розробки мобільних застосунків для ОС Android: Android (Native), Flutter та React Native. Представлено результати експериментального порівняння розміру інсталяційного пакета (APK) та часу запуску (TTID) при cold start сценарії. Дослідження новоствореного проєкту показало, що найменший розмір APK має Flutter з увімкненою оптимізацією R8, найшвидший запуск забезпечує Android (Native) підхід. React Native хоч і поступився за обома параметрами своїм конкурентам, але все одно показав гарні результати які не є підставою для відмови від його використання. Всі підходи можуть бути ефективно застосовані залежно від специфіки проєкту, доступних ресурсів і вимог до продуктивності.

Ключові слова: *мобільна розробка; Android; Android (Native); Flutter; React Native; APK; TTID; cold start.*
Табл.: 3. Бібл.: 22.

Актуальність теми дослідження. Мобільні пристрої інтегрувалися в наше повсякденне життя, ставши незамінними помічниками в багатьох сферах діяльності. Завдяки компактності та постійній доступності, вони дедалі частіше стають невід'ємною складовою комплексних програмних рішень, доповнюючи чи розширюючи їхні можливості. До найбільш поширених таких пристроїв можна віднести смартфони на базі операційних систем Android і iOS [1]. Згідно зі звітами Statista, кількість завантажень мобільних застосунків у світі щороку зростає [2]. Це зумовлено тим, що вони забезпечують зручний доступ до інформації, різноманітних сервісів, полегшують процес спілкування, інтегруються з IoT (Internet of things) та активно впроваджують ШІ (Штучний інтелект). Завдяки своєму формфактору вони вирішують вузькоспеціалізовані задачі, в яких використання десктопів чи ноутбуків менш зручне, або неможливе.

Постановка проблеми. Розділення ринку між Android і iOS позитивно впливає на їх розвиток і стимулює впровадження нових технологічних рішень. Така конкуренція створює сприятливі умови для інновацій та дозволяє кінцевим користувачам обирати ті продукти, які найкраще відповідають їхнім потребам, завданням чи особистим вподобанням. Водночас це змушує замовників програмного забезпечення піклуватися про користувачів обох платформ, аби гарантувати комфортне користування та якісний сервіс своїх продуктів.

Розробка нових програмних рішень ускладнюється низкою факторів, зокрема відмінностями у програмних та користувацьких інтерфейсах, різними середовищами розробки, а також необхідністю використання різних мов програмування — Kotlin для Android і Swift для iOS [3; 4]. Ці відмінності призводять до необхідності розробки й підтримки окремих проєктів, під кожну з платформ, що у свою чергу призводить до збільшення термінів розробки, дублювання бізнес-логіки, підвищення ймовірності виникнення помилок та додаткового навантаження на управління командою. У зв'язку з цим усе більшої популярності набувають підходи, що дозволяють реалізовувати мобільні застосунки одночасно для кількох платформ із мінімальними витратами ресурсів [5]. Разом із тим, ефективність таких рішень у порівнянні з нативною розробкою з погляду користувача потребує подальшого вивчення.

Аналіз останніх досліджень і публікацій. У статті "Cross-Platform Mobile Development: Comparing React Native and Flutter, and Accessibility in React Native" [6] проаналізовано два популярні кросплатформові фреймворки — React Native і Flutter. Особливу увагу приділено архітектурі, продуктивності та доступності. У висновках вказано, що Flutter забезпечує кращу продуктивність і плавність інтерфейсу, але React Native простіший для розробників знайомих із JavaScript.

У статті "A comparative analysis of the Flutter and React Native framework" [7] досліджено ефективність використання ресурсів мобільних застосунків, створених за допомогою Flutter і React Native. Основну увагу зосереджено на вимірюванні використання віртуальної пам'яті (VIRT), фізичної пам'яті (RES), спільної пам'яті (SHR), процесора (CPU), загального використання пам'яті (MEM).

У статті "A comparative analysis of native vs React Native mobile app development" [8] порівняно нативну розробку та React Native. Створено два ідентичні застосунки, проаналізовано продуктивність (час запуску, використання CPU, пам'яті), UX, витрати та масштабованість. Зазначено, що нативні застосунки мають кращу продуктивність та UX, але React Native дозволяє зменшити час розробки з деякими компромісами в оптимізації.

У статті "React Native evolution, native modules, and best practices" [9] розглянуто розвиток React Native, його архітектурні зміни, інтеграцію нативних модулів та кращі практики розробки. Проаналізовано переваги єдиної кодової бази для обох платформ, а також проблеми продуктивності, які виникають через міст між JavaScript та нативним кодом. Стаття підкреслює важливість балансування між використанням кросплатформових можливостей та нативних модулів для продуктивних завдань.

У статті "Comparison of Flutter and React Native platforms" [10] проведено порівняння Flutter та React Native як кросплатформових фреймворків, для розробки мобільних застосунків. Проаналізовано технічні особливості, популярність, мови програмування, підтримку бібліотек та продуктивність обох платформ. У висновках зазначено, що кожна з технологій має сильні і слабкі сторони: React Native відзначається зрілою екосистемою, та зручністю для розробників знайомих із JavaScript, а Flutter демонструє кращу продуктивність та менший розмір застосунків.

У статті "Comparative study of Android Native and Flutter app development" [11] порівняно нативну розробку та Flutter. Створено два однакові застосунки та проаналізовано їх продуктивність за показником використання CPU, пам'яті, енергоспоживанням та FPS. Результати показали, що нативні застосунки мають кращу оптимізацію та менший розмір інсталяційного пакета, тоді як Flutter демонструє вищий FPS і швидкість розробки.

У статті "Accelerating cross-platform development with Flutter framework" [12] досліджено переваги та недоліки використання Flutter для кросплатформової розробки. Розглянуто архітектуру Flutter, порівняно його з іншими фреймворками (React Native, Xamarin) за швидкістю розробки, продуктивністю та зручністю. У висновках виділено гнучкість Flutter, його переваги (hot reload, єдина кодова база, власний рендеринг) та недоліки (більший розмір застосунків, молодша екосистема).

У статті "A comparative study of Flutter, Kotlin Multiplatform, Jetpack Compose and React Native" [13] порівняно чотири сучасні фреймворки для Android: Flutter, Kotlin Multiplatform, Jetpack Compose та React Native. Проаналізовано їхню продуктивність, підхід до UI, швидкість розробки та інтеграцію з платформою. Висновки показують, що Flutter пропонує кращу кросплатформову узгодженість та продуктивність UI, Jetpack Compose забезпечує найкращу нативну інтеграцію для Android, React Native підходить для швидкого прототипування, а Kotlin Multiplatform добре балансує між спільною бізнес-логікою та нативним UI.

Виділення недосліджених частин загальної проблеми. Незважаючи на значну кількість досліджень, залишається недостатньо вивченим вплив використання різних фреймворків на розмір інсталяційного пакета (APK) та час запуску застосунку (TTID) в реальних умовах експлуатації. Зокрема, у наявних публікаціях не враховано впровадження нової архітектури React Native (TurboModules, Fabric, JavaScript Interface, далі — JSI), яка з'явилася у версії 0.70 (вересень 2022). Також поза увагою залишився Impeller engine [14], який з релізу Flutter 3.27 (грудень 2024) використовується за замовчуванням

не лише для iOS, а й для Android. Зважаючи на потенційний вплив цих змін на продуктивність, розмір пакета та час початкового запуску застосунку, зазначене питання потребує окремого емпіричного дослідження.

Мета дослідження. Порівняння розміру інсталяційного пакета, та часу запуску мобільного застосунку для Android, розробленого з використанням нативного підходу, а також React Native та Flutter. Це дозволить оцінити, наскільки ефективно оптимізовані кросплатформові рішення порівняно з нативною розробкою та виявити потенційні обмеження їх використання.

Виклад основного матеріалу.

Розмір APK. Зменшення розміру інсталяційного пакета є надзвичайно важливим для деяких ринків, оскільки напряму впливає на швидкість завантаження застосунку в умовах повільного інтернет-з'єднання, допомагає заощаджувати трафік при дорогих тарифах і суттєво підвищує ймовірність регулярного оновлення застосунку користувачами.

Android (Native) застосунок має компактний розмір завдяки відсутності додаткових шарів, який можна ще зменшити за допомогою R8 інструменту [15], котрий прийшов на заміну ProGuard і призначений для мініфікації та оптимізації коду. Він видаляє непотрібний код, проводить оптимізацію і перейменовує класи, методи та змінні на коротші назви. Також виконує обфускацію, що ускладнює читання коду сторонніми особами, і цим покращує безпеку та захищає інтелектуальну власність. Крім того, R8 дозволяє видаляти невикористані ресурси, які можуть з'являтися і накопичуватися під час розробки. Попри перелічені переваги, існує й певний недолік — для забезпечення коректної роботи проєкту необхідно виконувати додаткові налаштування, які залежать від використовуваних бібліотек, що може збільшити витрати часу на інтеграцію.

В основі Flutter лежить власний engine, який відповідає за рендеринг інтерфейсу, виконання коду на Dart та взаємодію з нативними API. Flutter відмальовує UI елементи використовуючи графічну бібліотеку Skia або Impeller engine, залежно від версії, пристрою та конфігурації проєкту. Impeller прийшов на заміну Skia бо забезпечує стабільнішу продуктивність рендерингу, ефективніше використовує GPU та усуває затримки, пов'язані з runtime-компіляцією шейдерів. Паралельно з цим існує Dart VM, яка дозволяє запускати Dart-код в одному з двох режимів: JIT (компіляція "на льоту" для Hot Reload) та AOT (компіляція наперед для максимальної оптимізації). Усі ці компоненти будуть додані в APK файл у вигляді libflutter.so.

React Native складається з двох основних частин: JavaScript шару, де працює логіка додатка та компоненти React, і нативного шару, який відповідає за відображення реальних UI-елементів. Обидві частини взаємодіють між собою за допомогою спеціального механізму, який залежить від використовуваної архітектури. JSI з'явився в новій архітектурі і прийшов на зміну Bridge, для вирішення обмежень старого підходу. Для виконання JavaScript використовується JavaScriptCore або Hermes [16], який забезпечує кращу продуктивність, пришвидшує запуск додатка (особливо на Android) та зменшує споживання пам'яті. Під час збірки застосунку з Hermes JavaScript-код попередньо компілюється у байт-код, що скорочує розмір JS bundle та знижує навантаження при старті. Разом з тим, сам Hermes додається в APK у вигляді libhermes.so файлу, що може збільшити розмір інсталяційного пакета, однак виграш у швидкодії часто це компенсує, а у випадку з новою архітектурою є необхідною умовою. В подальших дослідженнях доцільно використовувати нову архітектуру, адже вона має стати основним підходом у майбутньому і широко використовуватиметься в нових проєктах, або при оновленні старих.

Відкрита модель Android сприяла активному використанню системи багатьма виробниками, що зумовило появу значної кількості пристроїв із різноманітними апаратними конфігураціями. Починаючи із серпня 2021 року, для публікації нових додатків в Google

Play потрібно використовувати Android App Bundle (AAB) формат який прийшов на заміну APK для оптимізації такої фрагментації. AAB містить набір ресурсів і кодів для різних пристроїв: різні архітектури процесора, екрани, мови. Після завантаження в Google Play він буде розпакований і на його основі будуть згенеровані й підписані оптимізовані APK, які і будуть використовуватися для встановлення/оновлення застосунку на пристрої користувача. Такий механізм дозволяє зменшити розмір інсталяційного пакета завдяки використанню тільки необхідних компонентів для конкретного пристрою.

У роботі здійснено порівняльний аналіз розміру інсталяційного пакета, створеного із використанням різних технологічних стеків для ОС Android. Оскільки Google Play генерує оптимізовані APK для конкретних пристроїв, доцільно виконувати порівняння на основі одного ABI (Application Binary Interface). ARM64-v8a (64-бітних ARM-процесорів) наразі є найбільш поширеним серед Android-пристроїв згідно зі статистикою Google. Додатково для точності експерименту обмежимо збірку лише однією щільністю екрана — xxxhdpi. Для реалізації цього підходу були внесені відповідні налаштування splits.abi та splits.density до файлу build.gradle.kts.

Під час експерименту використовуватимуться такі версії середовищ: Flutter 3.29.2 (channel stable), React Native 0.78.0 з увімкненою новою архітектурою та Hermes, Android SDK рівня API 35 (Android 15), Gradle 8.10.2, AGP 8.8.2. Застосунки збиратимуться у release-режимі з увімкненим R8, а також у варіантах без нього для порівняння.

Для формалізації отриманих результатів найменший розмір APK (табл. 1) буде використано як референтне значення, що використовується як база для розрахунку відсоткового відхилення за формулою (1):

$$\Delta\% = \left(\frac{S - S_{\min}}{S_{\min}}\right) \times 100\%, \quad (1)$$

де S – розмір APK (MiB), S_{min} – найменший розмір APK (MiB), Δ% – відсоткове відхилення.

Таблиця 1 – Порівняння розмірів APK файлів (ARM64-v8a, xxxhdpi)

Платформа	R8 Увімкнено	Розмір APK (MiB)	Відхилення (%)
Android (Native)	так	7,11	7,4
Android (Native)	ні	11,33	71,15
Flutter	так	6,62	0,0
Flutter	ні	8,22	24,17
React Native	так	13,12	98,19
React Native	ні	15,36	132,02

Джерело: складено автором.

Проведений експеримент показав, що розмір інсталяційного пакета залежить не лише від обраної технології, але й від налаштувань збірки та інструментів оптимізації. Найменший розмір APK було отримано для Flutter з увімкненим R8, що навіть менше ніж у випадку нативного Android застосунку з активованим R8. На перший погляд, такий результат може виглядати неочікуваним, адже Flutter містить у собі власний engine, що додається в інсталяційний пакет, однак низка факторів дозволяє ефективно оптимізувати розмір базового проєкту. Водночас така перевага поступово нівелюється в міру масштабування застосунку. Додавання сторонніх плагінів (Firebase, Google Maps, Camera API тощо) призводить до швидшого зростання розміру, ніж у нативному Android, через інтеграцію додаткових нативних бібліотек та платформених каналів. За неформальними спостереженнями спільноти розробників, приріст розміру APK у Flutter може бути майже вдвічі швидшим, ніж у Kotlin-проєктах за умови однакової кількості залежностей. React Native продемонстрував найбільший розмір пакета серед усіх рішень, що зумовлено специфічною архітектурою платформи, зокрема необхідністю додавання Hermes та місткого JS bundle.

Якщо збереження компактного інсталяційного пакета в пріоритеті, найкращим рішенням буде Android (Native). Якщо ж цей параметр не є визначальним для проєкту, можна застосовувати будь-яку з розглянутих технологій, контролюючи збільшення розміру шляхом додавання лише необхідних залежностей.

App startup time. Швидкість запуску застосунку впливає на перше враження користувача та є складовою його загального задоволення програмним продуктом. Цей параметр може бути критично важливим для окремих категорій застосунків. Наприклад, для камери, якщо старт займає занадто багато часу, користувач ризикує втратити можливість зробити потрібний кадр у вирішальний момент. У мобільних застосунках є дві важливі метрики: TTID (time to initial display) і TTFD (time to fully drawn) [17].

TTID - це час від натискання на іконку застосунку, або від запуску через intent (на Android) до відображення першого кадру. У порівнянні з нативними застосунками у кросплатформових рішень є додаткові етапи під час ініціалізації, що потребує більше часу. У Flutter додаються: ініціалізація Flutter engine, запуск Dart VM, виклик runApp() і побудова widget tree. У React Native додаються: ініціалізація JS engine, парсинг та виконання JS bundle, виконання AppRegistry.registerComponent(), додаткові операції пов’язані з новою архітектурою.

Згідно з документацією ОС Android, запуск застосунку може відбуватися по одному з трьох сценаріїв: cold start (додаток завантажується з нуля), warm start (додаток у пам’яті, але activity було знищено), hot start (activity була призупинена, але вона в пам’яті).

Доцільним видається проведення порівняльного аналізу показника TTID при cold start із використанням різних технологічних стеків для ОС Android, оскільки цей сценарій найдовший із трьох можливих. Для цього в методі onCreate() класу Application було зафіксовано початковий час у мілісекундах. Після повного відображення першого екрану, зафіксуємо фінальний час та обчислимо різницю. Для отримання статистично надійних висновків проведено 50 замірів [18]. При цьому час витрачений системою до виклику onCreate(), класу Application, у розрахунках буде проігноровано. Таким чином отримаємо орієнтовний TTID для кожного технологічного стека. Цей підхід дозволить нам оцінити вплив різних технологій на швидкість запуску мобільного застосунку та зробити висновки щодо їхньої ефективності в контексті cold start.

Для обробки отриманих даних було застосовано мову програмування Python із залученням двох основних бібліотек: NumPy [19] та SciPy [20]. Бібліотека NumPy застосовувалась для базових числових операцій, таких як обчислення середнього значення, медіани, стандартного відхилення, мінімуму та максимуму. Для виконання статистичних тестів, зокрема розрахунку міжквартильного діапазону (IQR), перевірки нормальності розподілу за допомогою тесту Shapiro-Wilk, а також перевірки статистичної значущості між вибірками, використовувалась бібліотека SciPy (модуль stats).

Таблиця 2 – Статистичне порівняння TTID (Cold Start), мс

Показник	Android (Native)	Flutter	React Native
Середнє	123,42	187,92	235,14
Медіана	123,0	187,0	234,0
Стандартне відхилення	2,66	5,45	5,57
Мінімум	119	180	223
Максимум	130	206	248
IQR	3,5	6,0	6,5

Джерело: складено автором.

Як свідчать дані, наведені в табл. 2, спостерігається відмінність у показниках TTID (Cold Start) між досліджуваними підходами. Для обґрунтованого вибору статистичного методу порівняння необхідно попередньо перевірити, чи підпорядковуються вибірки нормальному розподілу. З цією метою було застосовано тест Shapiro-Wilk, результати якого подано в табл. 3.

Таблиця 3 – Результати перевірки на нормальність (Shapiro–Wilk)

Показник	Android (Native)	Flutter	React Native
p-значення	6.261698e-02	2.085780e-04	1.197405e-01
Нормальний розподіл	так	ні	так

Джерело: складено автором.

Оскільки принаймні одна з вибірок не має нормального розподілу, для порівняння груп було застосовано непараметричний критерій Kruskal-Wallis. Отримане p-значення (1.600934e−29) свідчить про статистично значущу різницю між реалізаціями в контексті ефективності TTID (Cold Start). Водночас отримані відмінності не є критичними з практичного погляду.

Проведений експеримент показав, що найшвидший запуск у Android (Native), але і кросплатформові рішення не критично поступаються в цьому показнику, і підтверджується можливість їх використання без істотного впливу на досвід користувача при старті. Для тесту був використаний середньобюджетний смартфон Google Pixel 3a на Android 12, що класифікується як застарілий пристрій [21] бо вже припинилася підтримка оновлень системи безпеки, однак технічно залишається сумісним із сучасними технологіями, зокрема підтримує Impeller у Flutter. Вибір застарілого смартфона був навмисним, для того щоб оцінити цей показник на слабкому за сучасними мірками процесорі, а саме Qualcomm Snapdragon 670 [22].

Слід зауважити, що на більш сучасних і продуктивних смартфонах час запуску додатків, ймовірно, буде ще кращим. Потужніші процесори, швидша оперативна пам’ять та покращена оптимізація нових версій Android сприяють зменшенню затримок при cold start. Водночас результати, отримані на застарілому пристрої, свідчать про те, що навіть за несприятливих умов показники запуску залишаються у межах прийнятного. Це підтверджує, що кросплатформові фреймворки можуть бути ефективно використані для розробки застосунків, орієнтованих на ринки з високим відсотком користувачів застарілих або бюджетних пристроїв, зберігаючи при цьому належний рівень користувацького досвіду.

TTFD - час до повної готовності UI, з яким користувач може почати взаємодіяти. Він складається з TTID і часу, що необхідний для повного завантаження та відображення даних. У межах цього дослідження TTFD не використовувався як порівняльна метрика через високу залежність від зовнішніх чинників (швидкість інтернет-з’єднання, затримки з боку сервера, алгоритми кешування і обробки даних). У клієнт-серверних застосунках затримки, пов’язані з мережею або серверною логікою, можуть нівелювати архітектурні переваги фреймворку. Крім того, в усіх розглянутих технологічних рішеннях існує можливість винесення важких операцій до нативного шару — наприклад, за допомогою Kotlin, Swift або C++ - що дає змогу здійснити оптимізацію часу повного завантаження незалежно від фреймворку.

Висновки. У межах проведеного дослідження було порівняно три технології розробки мобільних додатків для ОС Android: Android (Native), Flutter та React Native за показниками розміру APK і часу запуску (TTID) при cold start.

Найменший розмір APK вдалося отримати при використанні Flutter з увімкненою оптимізацією R8, що навіть менше, ніж у Android (Native). Це свідчить про високий рівень базової оптимізації Flutter, хоча зі зростанням складності проєкту його перевага по-

ступово нівелюється через архітектурні особливості при залежності від додаткових бібліотек. React Native, у свою чергу, показав найбільший розмір пакета, що пояснюється особливостями архітектури, зокрема необхідністю включення Hermes, JavaScript bundle.

Найшвидший запуск застосунку продемонстрував Android (Native), однак як Flutter, так і React Native забезпечили достатньо низькі значення TTID, значно нижчі за критичні межі, визначені в документації ОС Android. Такий час не спричинятиме суттєвого зниження користувацького досвіду, навіть на застарілих пристроях.

Незважаючи на те, що Android (Native) зберігає за собою певні переваги з погляду продуктивності, Flutter та React Native відкривають значні можливості для прискореної розробки і підтримці існуючих проєктів завдяки використанню єдиної кодової бази. За умови чіткого розуміння їхніх архітектурних обмежень та правильного застосування їхніх можливостей, ці фреймворки дозволяють створювати якісні та ефективні мобільні застосунки. Варто зазначити, що розробники кросплатформових рішень активно працюють над оптимізацією своїх технологій, що сприяє зростанню їх популярності серед розробників.

Отже, при виборі стека, для розробки мобільного застосунку, доцільно враховувати не лише технічні характеристики, а й контекст проєкту, доступність ресурсів для розробки, цільову аудиторію, вимоги до розміру застосунку, швидкість виходу на ринок та подальшу підтримку.

Список використаних джерел

1. StatCounter. (n.d.). *Operating system market share worldwide.* https://gs.statcounter.com/os-market-share.

2. Statista. (б. д.). *Number of mobile app downloads worldwide from 2016 to 2023.* https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads.

3. Apple. (n.d.). *Develop apps for Apple platforms.* Apple Developer Documentation. https://developer.apple.com/tutorials/app-dev-training.

4. Android Developers. (n.d.). *Build your first app | Get started with Android.* https://developer.android.com/get-started/overview.

5. JetBrains. (n.d.). *Popular cross-platform app development frameworks*. https://www.jetbrains.com/help/kotlin-multiplatform-dev/cross-platform-frameworks.html#popular-cross-platform-app-development-frameworks.

6. Goli, V. R. (2023). Cross-platform mobile development: Comparing React Native and Flutter, and accessibility in React Native. *International Journal of Innovative Research in Computer and Communication Engineering, 11*(3). https://doi.org/10.15680/ijircce.2023.1103002.

7. Markowski, M., & Smołka, J. (2023). A comparative analysis of the Flutter and React Native frameworks. *Journal of Computer Sciences Institute*, 29, 346–351. https://doi.org/10.35784/jcsi.3794.

8. Ramachandrappa, N. C. (2024). A comparative analysis of native vs React Native mobile app development. *International Journal of Computer Trends and Technology (IJCTT)*, 72(9), 33–37. https://doi.org/10.14445/22312803/IJCTT-V72I9P110.

9. Goli, V. R. (2021). React Native evolution, native modules, and best practices. *International Journal of Computer Engineering and Technology (IJCET),* 12(2), 67–75. https://doi.org/10.34218/IJCET_12_02_009.

10. Gülcüoğlu, A., Seyhan, R., Berk Ustun, A. (2021). Comparison of Flutter and React Native platforms. *Internet Applications and Management*, 12(2), 27–36. https://doi.org/10.34231/iuyd.888243.

11. Hussain, H., Khan, K., Farooqui, F., Arain, Q. A., & Siddiqui, I. F. (2021). Comparative study of Android Native and Flutter app development. *Proceedings of the 13th International Conference on Internet (ICONI), 99–102. KSII.* https://www.researchgate.net/publication/361208165.

12. Sattar, A. M., Soni, P., Ranjan, M. K., Kumar, A., Sahu, C., Saxena, S., & Chaudhari, P. (2023). Accelerating cross-platform development with Flutter framework. *Journal of Open Source Developments*, 10(2), 1–11. http://dx.doi.org/10.37591/joosd.v10i2.580.

13. Kumar, R. (2023). Evaluating modern Android framework: A comparative study of Flutter, Kotlin Multiplatform, Jetpack Compose and React Native. *International Journal of Scientific Research in Engineering and Management (IJSREM)*, 9(5), 1–9. https://doi.org/10.55041/IJSREM48732.

14. Flutter. (n.d.). *Flutter architectural overview.* https://docs.flutter.dev/resources/architectural-overview.

15. Android Developers. (n.d.). *Enable app optimization.* https://developer.android.com/topic/performance/app-optimization/enable-app-optimization.

16. React Native. (n.d.). *JavaScript environment – React Native.* https://reactnative.dev/docs/javascript-environment.

17. Android Developers. (n.d.). *App startup time.* https://developer.android.com/topic/performance/vitals/launch-time.

18. Illiashenko, O. (2025). *Заміри TTID (Cold Start), мс* [Data set]. Zenodo. https://doi.org/10.5281/zenodo.15774969.

19. NumPy. (n.d.). NumPy. https://numpy.org.

20. SciPy. (n.d.). SciPy. https://scipy.org.

21. Android Update Tracker. (n.d.). *Google Pixel 3a update history.* https://www.androidupdatetracker.com/p/google-pixel-3a.

22. NanoReview. (n.d.). *SoC rating list | Smartphone Processors Ranking.* https://nanoreview.net/en/soc-list/rating.

References

1. StatCounter. (n.d.). *Operating system market share worldwide.* https://gs.statcounter.com/os-market-share.

2. Statista. (б. д.). *Number of mobile app downloads worldwide from 2016 to 2023.* https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads.

3. Apple. (n.d.). *Develop apps for Apple platforms.* Apple Developer Documentation. https://developer.apple.com/tutorials/app-dev-training.

4. Android Developers. (n.d.). *Build your first app | Get started with Android.* https://developer.android.com/get-started/overview.

5. JetBrains. (n.d.). *Popular cross-platform app development frameworks*. https://www.jetbrains.com/help/kotlin-multiplatform-dev/cross-platform-frameworks.html#popular-cross-platform-app-development-frameworks.

6. Goli, V. R. (2023). Cross-platform mobile development: Comparing React Native and Flutter, and accessibility in React Native. *International Journal of Innovative Research in Computer and Communication Engineering, 11*(3). https://doi.org/10.15680/ijircce.2023.1103002.

7. Markowski, M., & Smołka, J. (2023). A comparative analysis of the Flutter and React Native frameworks. *Journal of Computer Sciences Institute*, 29, 346–351. https://doi.org/10.35784/jcsi.3794.

8. Ramachandrappa, N. C. (2024). A comparative analysis of native vs React Native mobile app development. *International Journal of Computer Trends and Technology (IJCTT)*, 72(9), 33–37. https://doi.org/10.14445/22312803/IJCTT-V72I9P110.

9. Goli, V. R. (2021). React Native evolution, native modules, and best practices. *International Journal of Computer Engineering and Technology (IJCET),* 12(2), 67–75. https://doi.org/10.34218/IJCET_12_02_009.

10. Gülcüoğlu, A., Seyhan, R., Berk Ustun, A. (2021). Comparison of Flutter and React Native platforms. *Internet Applications and Management*, 12(2), 27–36. https://doi.org/10.34231/iuyd.888243.

11. Hussain, H., Khan, K., Farooqui, F., Arain, Q. A., & Siddiqui, I. F. (2021). Comparative study of Android Native and Flutter app development. *Proceedings of the 13th International Conference on Internet (ICONI), 99–102. KSII.* https://www.researchgate.net/publication/361208165.

12. Sattar, A. M., Soni, P., Ranjan, M. K., Kumar, A., Sahu, C., Saxena, S., & Chaudhari, P. (2023). Accelerating cross-platform development with Flutter framework. *Journal of Open Source Developments*, 10(2), 1–11. http://dx.doi.org/10.37591/joosd.v10i2.580.

13. Kumar, R. (2023). Evaluating modern Android framework: A comparative study of Flutter, Kotlin Multiplatform, Jetpack Compose and React Native. *International Journal of Scientific Research in Engineering and Management (IJSREM)*, 9(5), 1–9. https://doi.org/10.55041/IJSREM48732.

14. Flutter. (n.d.). *Flutter architectural overview.* https://docs.flutter.dev/resources/architectural-overview.

15. Android Developers. (n.d.). *Enable app optimization.* https://developer.android.com/topic/performance/app-optimization/enable-app-optimization.

16. React Native. (n.d.). *JavaScript environment – React Native.* https://reactnative.dev/docs/javascript-environment.

17. Android Developers. (n.d.). *App startup time.* https://developer.android.com/topic/performance/vitals/launch-time.

18. Illiashenko, O. (2025). *Заміри TTID (Cold Start), мс* [Data set]. Zenodo. https://doi.org/10.5281/zenodo.15774969.

19. NumPy. (n.d.). NumPy. https://numpy.org.

20. SciPy. (n.d.). SciPy. https://scipy.org.

21. Android Update Tracker. (n.d.). *Google Pixel 3a update history.* https://www.androidupdatetracker.com/p/google-pixel-3a.

22. NanoReview. (n.d.). *SoC rating list | Smartphone Processors Ranking.* https://nanoreview.net/en/soc-list/rating.

Отримано 10.06.2025

UDC 004.415.2

***Illiashenko Oleh*[1]**

[1]PhD student, Faculty of Cybersecurity, software engineering, and computer science
International Humanitarian University (Odessa, Ukraine)
E-mail: ctanok@gmail.com. **ORCID:** https://orcid.org/0009-0004-9184-1733

COMPARATIVE ANALYSIS OF INSTALLATION PACKAGE SIZE AND STARTUP TIME OF A MOBILE APPLICATION

This article explores three technological stacks for mobile application development on the Android operating system: Android (Native), Flutter, and React Native. It presents the results of an experimental comparison focusing on two important parameters that directly influence the user's experience: the size of the installation package (APK) and the time to initial display (TTID) in a cold start scenario. To ensure objective comparison, identical conditions were used: the same processor architecture (ARM64-v8a), the same screen density (xxxhdpi), and the same build settings (release mode). The study of a newly created project showed that the smallest APK size was achieved using Flutter with the R8 code optimizer enabled. This indicates a high degree of default optimization and the absence of numerous dependencies at the initial stage. Android (Native) showed the best results in launch time (TTID), due to the lack of additional initialization layers. Although React Native lagged behind its competitors in both parameters, it still demonstrated solid performance that does not warrant its exclusion from consideration. TTID was measured on a mid-range smartphone, Google Pixel 3a, to evaluate this parameter on a processor that is considered weak by modern standards — the Snapdragon 670. The obtained results confirmed the feasibility of using cross-platform frameworks even on outdated devices without a significant loss in user experience quality. The article concludes that the choice of development stack should depend not only on technical characteristics but also on project specifics, team expertise, development timelines, maintenance demands, and desired time-to-market. Therefore, each of the reviewed approaches can be used under suitable conditions and is capable of providing a comfortable and stable user experience.

Keywords: *mobile development; Android; Android (Native); Flutter; React Native; APK; TTID; cold start.*
Table: 3. References: 22.

Ілляшенко О. М. Порівняльний аналіз розміру інсталяційного пакета і часу запуску мобільного застосунку. *Технічні науки та технології.* 2025. № 2(40). С. 275-283. DOI: https://doi.org/10.25140/2411-5363-2025-2(40)-275-283.