УДК 004.051(43)(4'2) DOI: 10.30977/BUL.2219-5548.2022.96.0.81

АНАЛІЗ ЕФЕКТИВНОСТІ ТЕХНОЛОГІЙ РОЗРОБЛЕННЯ МОБІЛЬНИХ ЗАСТОСУНКІВ ДЛЯ ОС ANDROID

Федорченко В. М. [1], Поляков А. О. [1], Сєвєрінов О. В. [2]
[1] Харківський національний економічний університет ім. С. Кузнеця
[2] Харківський національний університет радіоелектроніки

Анотація. *Запропонована стаття резюмує результати проведеного авторами дослідження щодо вибору платформ розроблення мобільних додатків для ОС Android з урахуванням їхньої продуктивності. Пропонується метод аналізу продуктивності на основі профілювання додатку за пам'яттю, за завантаженням процесора, за кількістю коду та мережної активності. За результатами дослідження платформа Xamarin має обмеження з боку дозволів, потребує більше ресурсів як пам'яті, так енергії живлення, що може бути відчутним у великих застосунках.*

Ключові слова: *ОС Android, мобільна платформа, native-платформа, мультиплатформна розробка, кросплатформеність, Android profiler, платформа Xamarin.*

Вступ

Android – це мобільна операційна система (ОС) з відкритим вихідним кодом із найбільшою кількістю користувачів у світі [8]. Згідно зі статистикою використання різних мобільних операційних систем у світовому масштабі на грудень 2021 р. Android є системою, що використовують понад 70 % користувачів [9], але цей показник може відрізнятися залежно від регіону світу, що подано в табл. 1.

Таблиця 1 – Розподіл мобільних ОС залежно від регіону світу, Україна та Worldwide

Регіон	Android	IOS	Samsung
Europe	63,65 %	35,66 %	0,63 %
North America	44,12 %	55,54 %	0,29 %
Asia	82,09 %	17,09 %	0,41 %
Oceania	53,87 %	44,55 %	1,52 %
Africa	82,69 %	15,45 %	0,2 %
South America	87,52 %	12,2 %	0,22 %
Ukraine	77,9 %	21,77 %	0,22 %
Worldwide	70,05 %	29,21 %	0,43 %

Але не всі програми мають однакову продуктивність на кожному пристрої Android, оскільки кожен пристрій має різні обчислювальні можливості. Він має обмеження щодо потужності, пам'яті та ресурсів, які впливають на аспекти продуктивності [11]. У процесі розроблення застосунків для Android важливо враховувати продуктивність застосунку, оскільки кожна програма має різні характеристики та підходи. Як правило, продуктивність залежить від платформи, на якій розробляється та виконується застосунок. Необхідно ще наголосити на низці фреймворків, за допомогою яких можна розробити застосунки під ОС Android, але використання різних технологій, імовірно, обмежить можливості кінцевого продукту. Тому розвиток мобільних платформ вимагає чіткого розуміння, які платформи доцільно використовувати для вирішення завдань, що виникають у процесі розроблення.

Native-застосунки інтегруються з ОС пристрою так, що мають змогу працювати швидше та гнучкіше, ніж альтернативні типи застосунків, розроблені за іншими платформами. Наприклад, застосунок Facebook колись був написаний із використанням WebView віджета в HTML5, щоб застосовувати той самий код для iOS, Android та мобільної мережі. Проте програма була повільною для користувачів iOS, що привело розробників Facebook до створення окремого коду для iOS.

Отже, можна зробити висновок, що не кожний підхід буде однаково гарно вирішувати будь-яке завдання, що розробляється для мобільної OS. Щодо native-застосунків, то вони мають підтримувати всі native-технології та апаратні можливості конкретної платформи, у цьому дослідженні ОС Android [3].

Залежно від використаних технологій на вигляд однакові програми можуть споживати різну кількість ресурсів смартфона, тобто в них буде різна продуктивність. Застосунки можуть споживати неоднакову кількість пам'яті, процесорного часу, по-різному впливати на витрату батареї, завантажувати більше або менше даних по мережі. Зрештою все це впливає на "*Look and Feel*" і продук-

тивність вашої андроїд програми. Тому насамперед у проєктуванні самого застосунку треба думати про продуктивність за багатьма критеріями. Отже, потрібне наукове дослідження для порівняння кількох аспектів використання платформ мобільної розробки, щоб з'ясувати переваги та недоліки кожної з них.

Аналіз публікацій

Питання продуктивності застосунків постійно досліджуються в науковій спільноті. У роботі Doug Sillars [11] аналізує питання, пов'язані з продуктивністю мобільного застосунку в розрізі продажів та втримання користувачів застосунку; особливостями створення пулу пристроїв для максимізації покриття тестування інтерфейсу користувача, функціональності та продуктивності; моніторингу з витоків пам'яті та проблем із процесором, що впливають на продуктивність; особливостями роботи застосунку в мобільних пристроях.

Існують дослідження архітектури MVC та VIPER native Android застосунків у роботі В. Гуменюка [7]. Аналіз продуктивності архітектур MVC, MVP та MVVM у мобільних пристроях було виконано в роботі Т. Лоу [12]. У ній аналіз архітектур виконувався за методикою «Architecture Tradeoff Analysis Method» (ATAM), в якій три фактори якості – тестованість, можливість модифікації та продуктивність – розглядаються як стандарт для оцінювання. Також робиться висновок, що за продуктивністю архітектура MVC споживає більше пам'яті, ніж MVP і MVVM, але останні дві архітектури потребують більш розгорнутого аналізу. Такий аналіз проведено в роботі B. Wisnuadhi [6], де встановлено, що продуктивність архітектури MVVM краща у використанні ЦП і часу виконання, тоді як MVP краща щодо використання пам'яті.

Сучасні застосунки стають усе більш кросплатформні за рахунок використання проміжного шару адаптації коду, або на етапі компіляції, або під час виконання – і все це додатково знижує продуктивність щодо native Android застосунку. Отже, методи та підходи, що використовувалися для аналізу продуктивності архітектур, можуть бути застосовані під час аналізу продуктивності різних платформ Android для розроблення застосунків.

Мета та постановка завдання

Метою цього дослідження є огляд методів і засобів для створення native-застосунків, а завдання – розробити застосунок з однаковим функціоналом, але виконаний на різних платформах. Але за умови такого постійного зростання кількості конкурентів перед розробниками мобільних застосунків постає питання: який підхід і технології варто використовувати в розробленні, щоб нові програми не тільки мали змогу доступу до native-функцій, визначалися швидкодією та високою продуктивністю, а й були конкурентоспроможними. Наприклад, така можливість native-застосунків, як позначення геолокації дозволяє компаніям підлаштовувати свої програми таким чином, щоб споживачі могли отримувати повідомлення, коли вони перебувають біля фізичних магазинів.

Аналіз технологій платформ і мов розроблення мобільних застосунків для ОС Android

Беручи до уваги рейтинг мов програмування для мобільної розробки на 2021 р. з офіційного сайту спільноти програмістів DOU [2] (рис. 1), очевидно, що треба провести додаткову класифікацію для виявлення платформ, наближених до native Android.

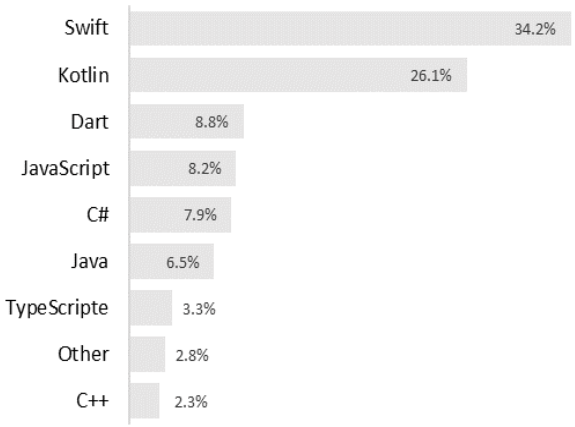

Рис. 1. Рейтинг мов програмування станом на 2021 р. з dou.ua

Окремо можна розглядати платформу Xamarin, яка реалізує застосунки, що мають ознаки native-коду. Загалом програми під Android можна розробляти практично будь-якою об'єктно орієнтованою мовою програмування. Найкраще для цього завдання підійдуть Java або Kotlin. У травні 2019 р. компанія Google оголосила Kotlin кращою мовою для Android-розробки [1]. З одного боку, Java все ще можна вважати офіційною

мовою андроїду, але вже існує низка факторів, здатних схилити чашу терезів на користь Kotlin, такі як: менша кількість коду; досить передбачувана перспектива розвитку, орієнтована саме на Android-розробку; позитивні особливості, які відсутні в Java. Дійсно, Java вимагає написання набагато більшої кількості коду, ніж Kotlin, тому існує більш високий ризик помилок. Узагальнювальна порівняльна інформація про мови Java та Kotlin наведена в таблиці 2.

Незважаючи на всі переваги Kotlin, його документація передбачає знання Java [8]. Так, з технічних аспектів різниця суттєва, але щоб займатися Android-розробкою, треба вчити обидві мови. Більшість популярних бібліотек підтримують зворотну сумісність із Kotlin, тобто код Java можна використовувати в Kotlin і навпаки, а ось щоб вирішити проблему або банально зрозуміти всі нюанси документації, варто знати Java.

Щодо Xamarin, то його можна використовувати за допомогою C# або F#. Xamarin.Android – бібліотека класів для C#, що надає розробнику доступ до Android SDK.

Тому оцінюючи мови, можна зробити висновок, що для порівняння найкращим варіантом буде обрати мови програмування Java і C# і використати технології Android SDK і Xamain відповідно.

Таблиця 2 – Порівняння мов програмування Java і Kotlin

Критерії / мова програмування	Kotlin	Java
Платформи / підтримка	open source конвертер Java в Kotlin підтримка ООП і ФП	open source (лише реалізація OpenJDK) ООП
Час компіляції	повільніше, ніж у Java (генерує додатковій байт-код)	швидко (native)
Об'єм коду початковий код	коротка (компактна)	надмірно багатослівна
Проміжний код (бай-код)	генерує додатковий байт-код для підтримки конструкцій мови	генерується компактний код (байт-код орієнтований для Java)
Швидкість розробки	висока	досить висока
Підтримка ком'юніті	зростаюча спільнота, слабкий канал ком'юніті	гігантська спільнота на GitHub, Reddit та StackOverflow
Безпека	більш безпечно (завдяки нульовій безпеці)	безпечно

Можна визначити два види платформ:

одноплатформні (single platform) – native-застосунки;

мультиплатформні (multiple platform) – гібридні або native-застосунки.

Одноплатформні: кожна операційна система має власний набір пакетів SDK для розроблення застосунків із швидким часом відгуку, графічною продуктивністю та підтримкою вбудованих сервісів (location, cellular, camera, sensors, Bluetooth тощо, рис. 2). Такі застосунки чутливі (responsive), мають високу швидкість, безперервну підтримку та широкий спектр функцій у SDK, але необхідно зазначити: якщо застосунок розроблений для понад однієї платформи, то підтримка двох різних кодових баз, відповідних команд розробників та синхронізація функцій між застосунками стає здебільшого слабким місцем.

Мови Java / Kotlin та Obective-C / Swift треба розглядати як native для відповідних ОС Android та iOS.

Платформа Xamarin

Xamarin – це платформа з відкритим кодом для створення сучасних та ефективних застосунків для iOS, Android та Windows за допомогою .NET. Ця платформа розширює платформу .NET спеціальними бібліотеками для iOS, macOS, Android тощо. Xamarin – це абстрактний рівень, який управляє зв'язком спільного коду з базовим кодом платформи. Xamarin працює в керованому середовищі, що забезпечує такі зручності, як виділення пам'яті та збір сміття [4]. Ця платформа дозволяє розробникам писати всю свою бізнес-логіку однією мовою і водночас досягати native-характеристик та вигляду на кожній платформі.

Фреймворк містить декілька основних компонентів:

Xamarin.IOS – бібліотека класів для мови C#, що надають розробнику доступ до платформи iOS SDK;

Xamarin.Android – бібліотека класів для мови C#, що надають розробнику доступ до Android SDK;

компілятори для iOS і Android;
IDE Xamarin Studio;
плагіни для Visual Studio.

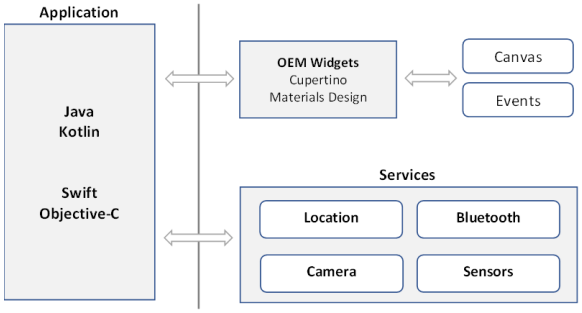

Рис. 2. OEM SDK – пряма інтеграція з native-функціями ОС

Щодо виконання застосунків, то між iOS і Android є одна ключова різниця – спосіб їхньої попередньої компіляції. Як відомо, для виконання застосунків на Android використовується регістр орієнтований JVM Dalvik або Android runtime (ART). Застосунки, що пишуться мовою Java, компілюються в проміжний байт-код, який інтерпретується Dalvik у команди процесора в момент виконання програми (тобто аналогічно тому, як працює CLR в .NET). Це так звана компіляція Just-in-time (компіляція на льоту). В iOS використовується інша модель компіляції – Ahead-of-Time (компіляція перед виконанням). Xamarin враховує цю різницю, надаючи окремі компілятори для кожної з цих платформ, що дозволяють на виході отримувати справжні native-застосунки, які можуть використовувати всі апаратні й програмні ресурси платформи.

Для Android у процесі компіляції програми відбувається переклад коду з C# у проміжний байт-код, зрозумілий віртуальній машині Mono, і сама ця віртуальна машина також додається в упакований застосунок. Компоненти Mono та Dalvik/ART написані на C і працюють поверх ядра Linux (варто зазначити, що Android основана на Linux). Під час запуску програми на Android обидві віртуальні машини починають працювати паралельно й обмінюються даними через спеціальний механізм wrapper-ів [4] (рис. 3).

Xamarin.Native (у цьому дослідженні Xamarin.Android) дозволяє досягти повторного застосування коду та максимально використати функціональність .NET під час створення мобільного застосунку. Використовуючи Xamarin, уся бізнес-логіка та серверний код є повністю спільними та мають повний доступ до базової платформи. Коли розробник починає писати свій застосунок на Xamarin, перед ним постає питання: Xamarin.Native або Xamarin.Forms. Зі знаннями .Net можна писати як з Xamarin.Native, так і Forms, але з XF інтерфейс створюється з простим у використанні XAML. З іншого боку, якщо в наявності команда з певним досвідом роботи з Android та iOS, тоді Xamarin.Native може окупитися в плані часу та якості.

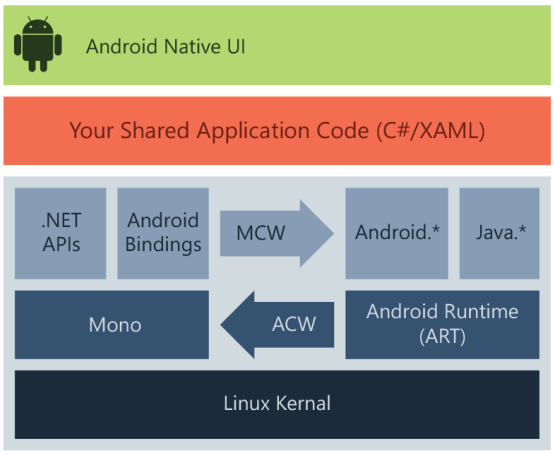

Рис. 3. Схема архітектури Xamarin.Android

Більшість власників бізнесу обирають платформу Xamarin, оскільки це скорочує час виходу на ринок (Time-To-Market) та інженерні витрати за рахунок спільного використання коду й застосування єдиного стека технологій.

Платформа Android SDK

Android SDK (Software Development Kit) – набір усіх компонентів від Google, пакет засобів для розроблення програм на Android – містить усе необхідне для створення, компіляції та запакування Android-програм. Ці програми здебільшого розробляються із застосуванням мови програмування Java. Усе, що ми робимо на Android за допомогою Java, залежить від Android SDK – якщо ми створюємо застосунок під певну версію, наприклад, для Android Nougat, то в нас мають бути встановлені відповідні інструменти SDK. Це треба враховувати в процесі розроблення [10].

Android SDK містить Android Debug Bridge, який є інструментом, що дозволяє підключитися до віртуального або реального пристрою для керування ним чи налагодження програми [5].

На сьогодні компанія Google пропонує тільки одне інтегроване середовище для роз-

роблення нових програм, і це Android Studio (інтегроване середовище розроблення для ОС Android від Google, створене на базі JetBrains IntelliJ IDEA та адаптоване під Android-розробку). Інше середовище розробки Andmore: Development Tools for Android (ADTA) базується на Eclipse IDE. ADTA – це набір компонентів (плагінів), які розширяють можливості Eclipse IDE для розроблення під операційну систему Android.

Обидва середовища містять увесь необхідний функціонал для створення, компіляції, налагодження та розгортання Android-програм. Вони також дозволяють розробнику створювати й запускати віртуальні пристрої Android для тестування. У SDK є плагіни для IDE, інструменти для збірки коду, інструменти для відлагодження, бібліотеки, емулятори.

Розглядаючи архітектуру Android-застосунків, можна побачити, що код поділений на два рівні: рівень даних містить DataManager і набір класів-помічників, рівень уявлення складається з класів Android SDK, таких як Activity, Fragment, ViewGroup тощо (рис. 4). *Helper-Class* (класи-помічники, третя колонка в діаграмі) мають дуже обмежені сфери відповідальності, і їх реалізують в послідовній манері. Наприклад, більшість проєктів мають класи для доступу до REST API, читання даних із DB або взаємодії з SDK від сторонніх виробників. У різних застосунків формується різний набір helper-class, але найбільш часто використовуваними будуть такі:

PreferencesHelper – працює з даними в SharedPreferences;

DatabaseHelper – працює з SQLite;

сервіси Retrofit, що виконують звернення до REST API.

DataManager є центральною частиною архітектури та широко використовує оператори RxJava для того, щоб комбінувати, фільтрувати й трансформувати дані, отримані від помічників. Завдання *DataManager* полягає в тому, щоб звільнити Activity і Fragments від роботи щодо оброблення даних – він буде виконувати всі потрібні трансформації всередині себе й віддавати назовні дані, готові до відображення.

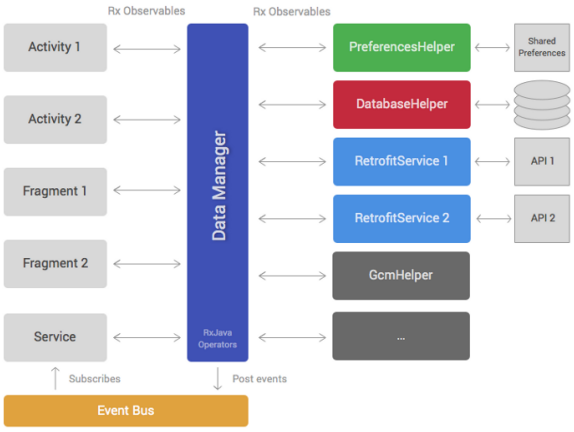

Рис. 4. Архітектура Android-застосунку

Мультиплатформні: застосунки підтримують декілька платформ з уніфікованим стилем і кодовою базою.

Мета та постановка завдання

Android SDK та Xamarin насамперед класифікуються як «Набір для розробки ПЗ» та «Платформа для кросплатформної розробки» відповідно. Порівняємо обрані технології за такими показниками: стек технологій, використання коду платформою, UI/UX (User Interface / User Expierence), продуктивність, можливості апаратних засобів та IDE. Результат порівняння наведено в таблиці 3.

Таблиця 3 – Порівняння Xamarin і Android SDK

Критерії / мобільна платформа розробки	Xamarin	Android SDK
Стек технологій	.Net framework + native бібліотеки	native-бібліотеки
Використання коду	спільне використання з іншою платформою до 96 % з Xamarin.Forms	єдиний код
UI/UX (User Interface / User Expierence)	можливе повне налаштування UI для платформи	платформозалежний UI
Продуктивність	близька до рідної	відмінна (native)
Можливості апаратних засобів	високі (Xamarin використовує платформозалежні API і підтримує зв'язок з native-бібліотеками)	високі (native-інструменти мають повну підтримку для можливостей системи)
IDE	Visual Studio	Android Studio

У разі створення застосунків зі складним інтерфейсом із використанням Xamarin кількість загального коду кардинально зменшується. Отже, кросплатформна розробка втрачає свою основну перевагу. Але це не робить застосунки на Xamarin менш якісними, просто трохи збільшує витрати. Проте жоден із інших кросплатформних інструментів не може мати такого самого рівня продуктивності, які пропонує платформа Xamarin.

Метою дослідження є розроблення двох native-застосунків, що мають такі функціонали:

− можливість відкрити телефонну книгу користувача;

− змога обрати контакт;

− відображення інформації про обраний контакт.

Застосунки мають бути розроблені з використанням таких технологій:

a) мови програмування: у прикладі показано:

- Java/Kotlin
- C#

b) інструменти / платформи:

- Android SDK
- Xamarin

c) середовища розробки:

- Android Studio
- Visual Studio

d) застосунки мають підтримувати версію Android 6.0.1 і пізнішу (що покриває майже 98 % використовуваних пристроїв).

Метою дослідження є порівняння таких показників, як вага застосунку, швидкодія та продуктивність.

Метод аналізу продуктивності профілюванням

Дуже зручно, що деякі середовища для розроблення навіть дозволяють відстежити всі основні метрики продуктивності вашого застосунку. Наприклад, якщо існує проєкт в Eclipse, можна відкрити перспективу DDMS. Там є змога профілювати програми та отримати уявлення про те, скільки часу займає кожен метод / що займає найбільше часу.

Також дуже зручний вбудований в Android Studio профілювальник дозволяє відстежити всі основні метрики продуктивності, такі як: обсяг пам'яті, завантаження процесора, використання мережі та споживання енергії (див. рис. 5).

Для того щоб обрати платформу, треба чітко знати завдання, що має виконувати застосунок, і яким саме функціоналом він має володіти. Оцінюючи технології розроблення мобільних застосунків, необхідно зазначити, що цей напрям вимагає вибору критеріїв оцінювання.

Показники оцінювання native-застосунків можна поділити на технічні та якісні. На швидкість та якість розроблення впливають такі якісні показники:

популярність мови;

наявність фахівців;

наявність повної документації та технічна підтримка;

зручність розроблення і налагодження (зручний синтаксис (синтаксичний цукор), наявність колекції (контейнерів), лямбда-вирази тощо).

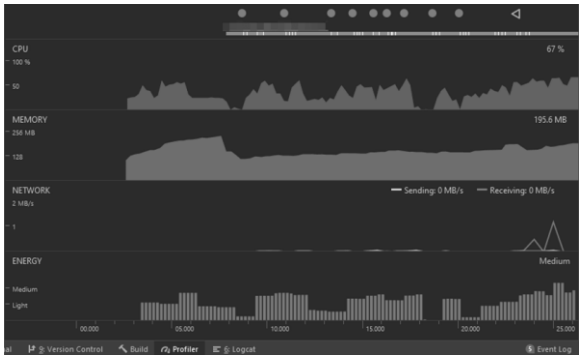

Рис. 5. Профілювальник застосунку в Android Studio

Щодо технічних вимог, то можна виокремити такі показники:

вага застосунку;

можливість доступу до native-функцій (безпроблемна взаємодія програми з мобільною ОС);

швидкість роботи й відгуку інтерфейсу;

швидкодія;

продуктивність (споживання пам'яті, процесорного часу, вплив на витрату батареї, завантаження даних по мережі);

надійність;

безпечність;

легкість підтримки / оновлення.

На основі порівняльної таблиці 3, а також якісних параметрів у межах роботи буде доцільно розглянути саме Xamarin і Android SDK. Для порівняння таких технічних критеріїв, як вага застосунку, швидкодія та продуктивність необхідно розробити два застосунки, які б задіяли native-функції, наприклад доступ до телефонної книги.

Результати аналізу ефективності мобільних застосунків

Аналіз застосунку, розробленого на Android SDK.

Застосунок на Java / Kotlin розроблений в Android Studio, яка містить зручний вбудований профілювальник, що дозволяє відстежити всі основні метрики продуктивності: пам'ять, процесор, використання мережі й споживання енергії (див. рис. 6).

Застосунок займає 22,01 МБ пам'яті пристрою. Варто зауважити, що цьому застосунку не потрібен дозвіл на використання даних.

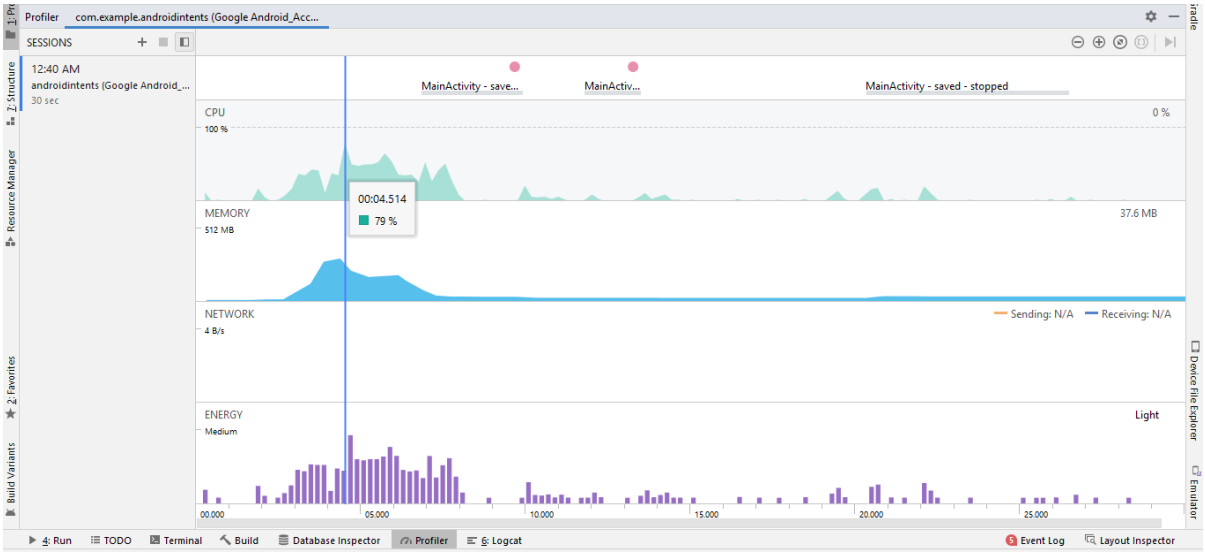

Рис. 6. Профілювання застосунку на Java

Аналіз застосунку на C#

Для застосунків на Xamarin існує Xamarin Profiler – графічний інтерфейс для профілювання застосунків Android та iOS на Windows. Але щоб скористуватися ним, потрібно бути передплатником Visual Studio Enterprise.

Також Android Studio містить засіб Android Profiler. Його можна використовувати для вимірювання продуктивності застосунку Xamarin Android, створеного за допомогою Visual Studio. Однак, на відміну від Xamarin Profiler, засіб Android Profiler не інтегровано в Visual Studio й може використовуватися лише для реєстрації APK, що було створено та імпортовано в Android Profiler.

Після завантаження APK-файлу Android Studio відображає інформацію про розмір застосунку та всіх його частин (див. рис. 7).

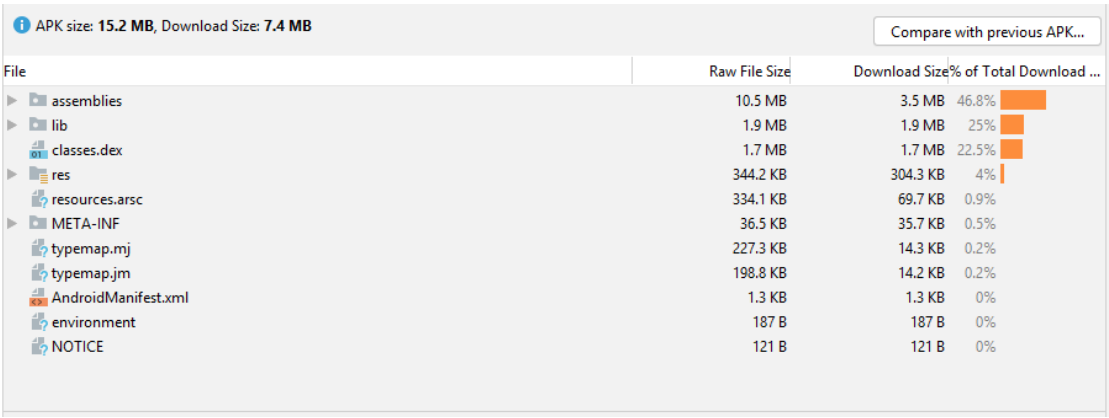

Рис. 7. Відображення інформації про розмір застосунку

Для того щоб отримати всі необхідні для аналізу показники, потрібно натиснути Profile 'назва_apk' у верхньому лівому кутку студії. Android Studio проаналізує застосунок, щоб відстежити всі основні метрики продуктивності (рис. 8).

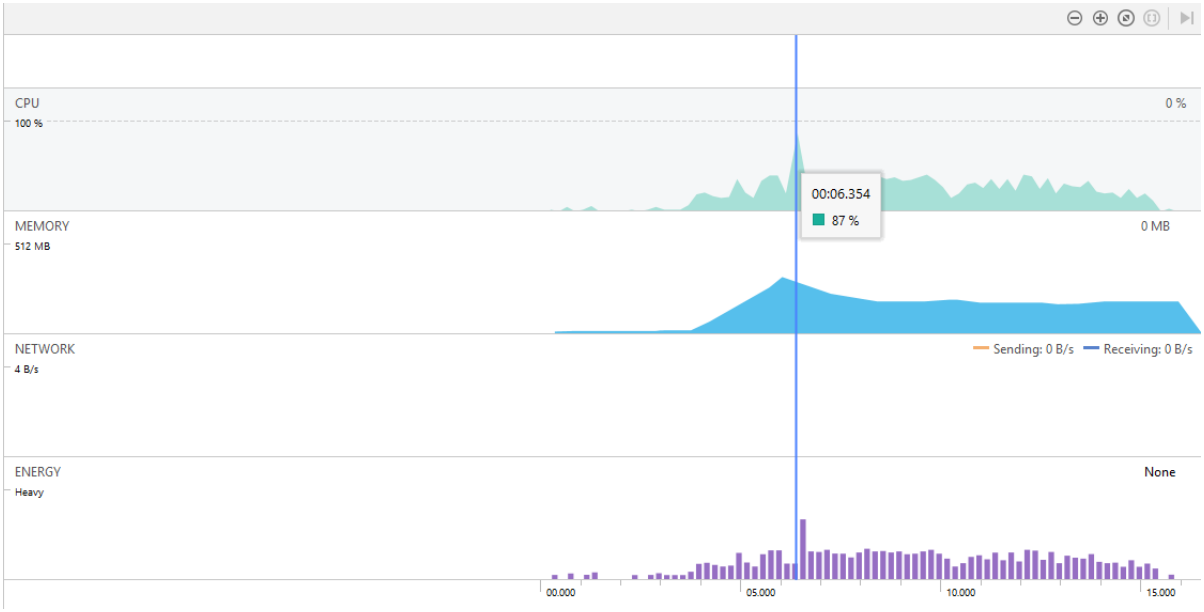

Рис. 8. Профілювання застосунку, розробленого на платформі Xamarin мовою C#

Варто зазначити, що застосунку на Xamarin необхідний дозвіл, щоб використовувати «Контакти пристрою», тоді як застосунок на Java просто використовує вже реалізовані можливості. Застосунок займає 28,78 МБ з пам'яті пристрою.

Аналіз застосунків

Застосунки встановлювалися й тестувалися на смартфоні з найменшою версією Android 6.0.1, версія прошивки ядра – Linux 3.10.49. Занесемо в таблицю 4 всі перелічені показники, а саме: розмір (size) застосунку, а також обсяг використовуваної пам'яті (memory Mb), процесорне завантаження та споживання енергії (рис. 6 і 8). Використання мережі залишилося стале в обох випадках. Energy Profiler відображає приблизне споживання енергії застосунком, що тестується.

Таблиця 4 – Порівняння результатів профілювання застосунків

	Застосунок на Xamarin (C#)	Застосунок на Java / Kotlin
Size (Mb)	28,78	22,01
CPU (%)	87	79
Memory (Mb)	304,7	273,2
Energy	00:04,513	00:01,434

Аналізуючи всі показники, можна зробити висновок, що використання native-технологій, а саме мови програмування Java / Kotlin і Android SDK, є найефективнішими в цьому дослідженні.

Висновки

У дослідженні розглянуті види та проведений аналіз типів мобільних застосунків і самої операційної системи Android. Крім того, розглянуто й проаналізовано технології для розроблення під Android. Також у роботі розглянуті основні компоненти архітектури мобільних застосунків для Android, що є блоками для побудови програми. За допомогою середовищ розробки Visual Studio і Android Studio було створено два застосунки, які можна встановити на мобільний пристрій та переглядати з нього інформацію про контакти користувача. Застосунки були розроблені мовами програмування Java і C#, використані Android SDK і Xamain відповідно.

Java і C# є популярними мовами програмування, тому важко сказати, яка з них краща за такими якісними показниками, як наявність повної документації, технічна підтримка й наявність фахівців. Обидві ці мови є C-подібними, тому мають схожий зрозумілий синтаксис, зручний у розробленні й налагодженні. Розглядаючи такі технічні характеристики, як безпроблемна взаємодія програми з мобільною ОС, варто зазначити, що платформи, що використовувалися в розробці, мають повний доступ до основних native-функцій. Як особливість варто зазначити, що під час використання застосунку на C#, програма запитувала доступ на використання контактів, тоді як застосунок мовою Java дозволу не потребує.

Для того, щоб дати однозначну відповідь на питання, яка з мов має вищу швидкість роботи та відгуку інтерфейсу, виконуються

багато тестів у «реальних умовах», де перевіряється швидкість значних обчислень або активних запитів до бази даних. У цьому дослідженні два застосунки експериментально порівнювалися за такими показниками: вага додатків та їхня продуктивність. У процесі аналізу native-застосунок, розроблений мовою Java, показав кращі результати. Проте не можна зробити однозначного вибору на користь однієї технології, адже під час створення того чи іншого мобільного застосунку необхідно чітко визначати мету, що й буде керувати кінцевим вибором технологій.

Як висновок можна зазначити, що для найбільш ефективних результатів, за умови наявності необхідних навичок, бюджету й особливо часу, доцільно використовувати повністю native-платформу. Якщо ж необхідний застосунок для декількох ОС, а його функціонал не вимагає на кожній операційній системі дуже високої продуктивності, то логічніше було б використовувати кросплатформний фреймворк. Він би забезпечив максимальне покриття за коротший час, ніж native-технології для кожної операційної системи окремо.

Література

1. Kotlin (programming language) // Wikipedia.: https://en.wikipedia.org/Kotlin_(programming_language) (дата звернення: 12.01.22).
2. Рейтинг мов програмування 2021: частка Python зменшується, а TypeScript обійшов C++ // ДОУ. URL: https://dou.ua/lenta/articles/language-rating-jan-2021/ (дата звернення: 08.01.22).
3. Android. Разработка приложений / Р. Роджерс, Д. Ломбардо, З. Медниекс, Б. Мейк. Москва: ЭКОМ Паблишерз, 2010. 400 с.
4. Что такое Xamarin? // Xamarin Microsoft. URL: https://docs.microsoft.com/ru-ru/xamarin/get-started/what-is-xamarin. (дата звернення: 10.12.21).
5. Янчук К. В., Федорченко В. М. Огляд платформ для створення мобільних додатків. Черкаси – Харків – Баку – Бельсько-Баяла, 2020. 64 с.
6. Wisnuadhi B., Munawar G., Wahyu U. Performance Comparison of Native Android Application on MVP and MVVM // Advances in Engineering Research. 2020. Vol. 198. P. 276–282.
7. Humeniuk V. Android Architecture Comparison: MVP vs. VIPER. 2019.
8. Gartner Says Worldwide Sales of Smartphones Recorded First Ever Decline During the Fourth Quarter of 2017 / Meulen R. van der // Gartner. URL: https://www.gartner.com/en/newsroom/press-releases/2018-02-22-gartner-says-worldwide-sales-of-smartphones-recorded-first-ever-decline-during-the-fourth-quarter-of-2017 (дата звернення: 16.12.21).
9. Mobile Operating System Market Share Worldwide // StatCounter Global Stats. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide. (дата звернення: 19.11.21).
10. SDK Platform release notes // Android Developers. URL: https://developer.android.com/studio/releases/platforms. (дата звернення: 20.11.21).
11. Sillars D. High performance Android apps: improve ratings with speed, optimizations, and testing. Beijing; Boston: O'Reilly, 2015. 247 p.
12. Tian Lou A Comparison of Android Native App Architecture – MVC, MVP and MVVM. Endhoven University of Technology, 2016.

Reference

1. Kotlin (programming language) // Wikipedia. URL: https://en.wikipedia.org/Kotlin_(programming_language) (accessed: 12.01.22).
2. Programming language rating 2021: Python share decreases and TypeScript bypasses C++ // DOU. URL: https://dou.ua/lenta/articles/language-rating-jan-2021/ (accessed: 08.01.22).
3. Android. Application Development / R. Rogers, D. Lombardo, Z. Mednieks, B. Meik. Moscow: ECOM Publishers, 2010. 400 p.
4. What is Xamarin? // Xamarin Microsoft. URL: https://docs.microsoft.com/ru-ru/xamarin/get-started/what-is-xamarin (accessed: 10.12.21).
5. Yanchuk K. V., Fedorchenko V. M. Review of platforms for creating mobile applications. Cherkasy – Kharkiv – Baku – Belsko-Bayala: 2020. 64 p.
6. Wisnuadhi B., Munawar G., Wahyu U. Performance Comparison of Native Android Application on MVP and MVVM // Advances in Engineering Research. 2020. Vol. 198. P. 276–282.
7. Humeniuk V. Android Architecture Comparison: MVP vs. VIPER. 2019.
8. Gartner Says Worldwide Sales of Smartphones Recorded First Ever Decline During the Fourth Quarter of 2017 / Meulen R. van der // Gartner. URL: https://www.gartner.com/en/newsroom/press-releases/2018-02-22-gartner-says-worldwide-sales-of-smartphones-recorded-first-ever-decline-during-the-fourth-quarter-of-2017 (accessed: 16.12.21).
9. Mobile Operating System Market Share Worldwide // StatCounter Global Stats. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide (accessed: 19.11.21).
10. SDK Platform release notes [Electronic Resource] // Android Developers. URL: https://developer.android.com/studio/releases/platforms (accessed: 20.11.21).
11. Sillars D. High performance Android apps: improve ratings with speed, optimizations, and testing. Beijing; Boston: O'Reilly, 2015. 247 p.

12. Tian Lou A Comparison of Android Native App Architecture – MVC, MVP and MVVM. Endhoven University of Technology, 2016.

Федорченко Володимир Миколайович, к.т.н., доц. каф. електронних обчислювальних машин, каф. безпеки інформаційних технологій, Харківський національний економічний університет ім. С. Кузнеця, тел. +38 099-640-52-78, Volodymyr.Fedorchenko@hneu.net,
Поляков Андрій Олександрович, к.т.н., доц. каф. інформаційних систем, Харківський національний економічний університет ім. С. Кузнеця, тел. +38 099-4151200, Andrii.Poliakov@hneu.net,
Сєвєрінов Олександр Васильович, к.т.н., доц. каф. безпеки інформаційних технологій, Харківський національний університет радіоелектроніки, тел. +38 (066) 175-86-68, oleksandr.sievierinov@nure.ua.

Analyzing the efficiency of technologies for developing mobile applications for Android OS

Abstract. Problem. *Mobile application development is quite a complex process that requires developers with high expertise in a particular development platform for Android. There are also several single-platform, cross-platform, Xamarin development platforms but there is no comparison of their performance as with native ones.* ***Methodology***. *As a method, it is proposed to use an experimental approach in which a mobile application is developed with the necessary components, and then the performance of the profiler on the metrics of CPU usage, memory, power consumption and network is investigated.* ***Results.*** *According to the study, the Xamarin platform has additional restrictions on permissions, requires more resources, both memory and power, which can be felt in large applications, but allows you to attract a large community of .Net developers.* ***Originality.*** *This approach to the analysis of productivity will reveal at a preliminary stage the problematic aspects of the mobile development platform, its integration with native Android services, as well as possible user preferences.* ***Practical value.*** *The proposed technique will introduce a preliminary analysis of the various components integrated into the Android OS. This approach will provide a more reasonable management decision for the full process of developing mobile applications for Android OS, considering such criteria as speed of development, professional level of developers to be involved in the development process, application integration with Android components, integration of new functionality to the application and others.*

Key words: *Android OS, mobile platform, native platform, multi-platform development, cross-platform, Android profiler, Xamarin platform*.

Fedorchenko Volodymyr, Ph.D., Assoc. Prof. Electronic Computers Department, and Information Systems Department, [1]Simon Kuznets Kharkiv National University of Economics, tel. +38 099-640-52-78, volodymyr.fedorchenko@nure.ua,
Poliakov Andrii, Ph.D., Assoc. Prof. Information Systems Department, Simon Kuznets Kharkiv National University of Economics, tel. +38 099-41-51-200, Andrii.Poliakov@m.hneu.edu.ua,
Sievierinov Oleksandr, Ph.D., Assoc. Prof. Department of Information Technology Security, Kharkiv National University of Radio Electronics, тел. +38 (066) 175-86-68, oleksandr.sievierinov@nure.ua.