УДК 004.41

Маковій А.В.

КРИТЕРІЇ ВИБОРУ ТЕХНОЛОГІЧНОГО СТЕКУ ДЛЯ РЕАЛІЗАЦІЇ МОБІЛЬНИХ ПРОЄКТІВ

Вінницький національний технічний університет

Анотація
У статті було розглянуто критерії вибору технологічного стеку для реалізації мобільних проєктів.
Ключові слова: програмне забезпечення, стек, кросплатформні застосунки, нативна розробка.

Abstract
The article discusses the criteria for selecting a technology stack for implementing mobile projects.
Keywords: software, stack, cross-platform applications, native programming.

Вступ

Стрімкий розвиток мобільних комп'ютерних систем вимагає від інженерів застосування ретельно продуманих з математичної та архітектурної точок зору методів при виборі технологічного стеку. Традиційно процес вибору платформи обмежувався простим вибором між нативним та кросплатформним програмуванням. Проте сьогодні головним критерієм вибору стала не економічна вигода, а технічна спроможність архітектури забезпечити необхідну частоту оновлення зображення, ефективне управління оперативною пам'яттю та мінімальну затримку при виконанні системних завдань.

Основна відмінність між доступними платформами полягає в тому, як програмний код взаємодіє з операційною системою пристрою. Вибір конкретного рішення безпосередньо впливає на архітектуру додатка, алгоритми обробки даних та механізми відображення графічного інтерфейсу користувача.

Основна частина

Нативна розробка означає створення програмного забезпечення з використанням мов та інструментів, які є «нативними» для конкретної операційної системи: Swift для iOS та Kotlin для Android. Такий підхід забезпечує прямий доступ до системних інтерфейсів прикладного програмування без необхідності використання проміжних мостів або віртуальних машин для обробки користувацького інтерфейсу [1].

Нативна архітектура є єдиним практичним варіантом для систем, що вимагають інтенсивних обчислень на рівні пристрою. До таких рішень належать програми для обробки аудіо та відео в режимі реального часу, складні системи 3D-моделювання, а також програми, що вимагають глибокої та безперервної інтеграції з апаратними датчиками (гіроскопами, акселерометрами, модулями Bluetooth). Системні виклики в нативному середовищі виконуються за мінімальний час із ізоляцією пам'яті на рівні системи, що забезпечує найвищий рівень продуктивності [2].

З іншого боку, кросплатформені рішення забезпечують баланс між продуктивністю та швидкістю розробки завдяки використанню абстракцій над системними API. Найпопулярнішим прикладом такого підходу є React Native, який дозволяє розробникам створювати графічні інтерфейси за допомогою JavaScript, зберігаючи при цьому візуальну цілісність програми.

Історично слабким місцем цього фреймворку був асинхронний міст, який відповідав за передачу даних між потоком JavaScript та нативним потоком шляхом перетворення об’єктів у текстовий формат JSON. Ця операція спричиняла значні затримки під час швидких оновлень інтерфейсу користувача.

Щоб подолати ці обмеження, було впроваджено нову архітектуру Fabric Renderer та JavaScript Interface. JSI – це синхронний інтерфейс C++, який дозволяє коду безпосередньо отримувати доступ до нативних модулів без серіалізації даних. Рендеринг інтерфейсу тепер обробляється двигуном Fabric, який синхронно будує тіньове дерево компонентів у пам'яті, обчислює розміри елементів за допомогою двигуна Yoga та застосовує до екрану лише необхідні зміни [3].

На відміну від React Native, який спирається на компоненти операційної системи, Flutter використовує власний графічний рушій для візуалізації кожного пікселя на екрані. Код, написаний на Dart, компілюється безпосередньо в нативний машинний код, що забезпечує високу швидкість виконання [4].

Довгий час стандартним графічним рушієм Flutter був Skia, який компілював шейдери під час виконання програми. Це іноді призводило до пропусків кадрів під час початкової візуалізації складних анімацій. Сучасний Flutter використовує новий рушій Impeller, який переносить компіляцію графічних інструкцій на етап побудови додатка. Усі об’єкти в графічному конвеєрі створюються заздалегідь, що повністю усуває затримки. Impeller безпосередньо використовує сучасні графічні інтерфейси смартфонів, забезпечуючи стабільну частоту оновлення екрану до 120 кадрів на секунду.

Kotlin Multiplatform (KMP) пропонує принципово інший підхід. Замість того, щоб намагатися уніфікувати інтерфейс користувача, KMP зосереджується на спільній використанні бізнес-логіки. Мережеві запити, алгоритми шифрування та взаємодія з локальними базами даних пишуться один раз на Kotlin і компілюються в нативний двійковий код для iOS та Android.

Поточна версія Kotlin/Native використовує вдосконалену модель управління пам'яттю. Об'єкти зберігаються у спільній пам'яті та можуть безпечно модифікуватися з будь-якого потоку. Ключовою особливістю KMP є його безшовна інтеграція з колектором пам'яті операційної системи iOS. Компілятор здатний інтегрувати свій граф об'єктів із системою в iOS, що мінімізує навантаження на пам'ять та запобігає перевантаженню процесора [5].

Окрім технічних характеристик, при виборі технології важливу роль відіграють економічні фактори та швидкість розробки. Зокрема, варто звернути увагу на досвід розробника, оскільки він суттєво впливає на продуктивність команди та якість кінцевого продукту. Сучасні кросплатформені фреймворки ретельно оптимізовані саме для спрощення розробки. Наприклад, такі механізми, як «hot reload», дозволяють миттєво бачити результати змін у коді без повного перезапуску додатка, що значно прискорює ітеративний процес. React Native особливо привабливий для розробників з досвідом роботи з веб-технологіями, оскільки використовує звичну екосистему JavaScript та декларативний підхід до побудови інтерфейсів [3]. Flutter, у свою чергу, пропонує уніфіковане середовище з передбачуваним рендерингом та потужними інструментами UI, що зменшує кількість несподіваних помилок, пов’язаних з відмінностями платформ [4]. Нативна розробка, хоча й забезпечує максимальний контроль над системою, має вищий поріг входу через необхідність глибокого розуміння особливостей кожної операційної системи, інструментів розробки та життєвого циклу додатка. Водночас вона пропонує найкращі можливості для тонкої оптимізації та глибокої інтеграції з апаратним забезпеченням [5].

Питання вартості та швидкості розробки є одними з ключових при виборі технологічного стеку, оскільки вони безпосередньо впливають на життєздатність проєкту. Нативна розробка передбачає створення окремих додатків для кожної платформи, що означає необхідність утримання двох команд або фахівців з різними технологічними стеками. Це збільшує витрати як на етапі розробки, так і під час подальшої підтримки. Водночас такий підхід забезпечує максимальну оптимізацію та стабільність, що є надзвичайно важливим для складних або високозавантажених систем. З іншого боку, кросплатформені рішення значно скорочують час виходу на ринок, оскільки більша частина коду використовується повторно. Це робить їх особливо привабливими для стартапів, де швидкість розробки важливіша за бездоганну продуктивність. Однак у довгостроковій перспективі витрати можуть зрости через необхідність вирішувати специфічні для платформи проблеми, інтегрувати нативні модулі та усувати технічний борг.

React Native найкраще підходить для додатків, у яких ключове значення мають комплексний стан даних, та відображення динамічного контенту. Оскільки цей фреймворк використовує стандартні системні компоненти для відображення кнопок або списків, він ідеально підходить для рішень, де основна увага приділяється контенту (соціальні мережі, новинні стрічки, торгові майданчики, додатки для бронювання тощо), а інтерфейс складається зі звичних карток та кнопок. Архітектура Fabric дозволяє ефективно керувати великими масивами текстових даних та швидко оновлювати екран при отриманні нової інформації з сервера, зберігаючи при цьому звичний вигляд системних елементів керування.

Власний графічний рушій Flutter робить його оптимальним рішенням для додатків із високо персоналізованим дизайном, які повинні виглядати однаково на різних операційних системах. Цей стек ідеально підходить для фінансових інформаційних панелей зі складними діаграмами, інтерактивних

навчальних програм та додатків із нестандартною типографікою та складними анімаціями. Він гарантує ідеальне відображення додатка з точністю до пікселя незалежно від версії операційної системи пристрою.

В свою чергу Kotlin Multiplatform найкраще підходить для великих систем, що характеризуються складною обчислювальною логікою на стороні клієнта. Цей стек обирають для навігаційних систем, банківських рішень зі складною логікою шифрування, або для додатків, що працюють у фоновому режимі, збирають дані з датчиків «розумного будинку», чи взаємодіють з портативними пристроями через нестандартні протоколи зв'язку. Відсутність проміжних рівнів абстракції дозволяє точно контролювати потоки ресурсів і запобігає раптовому завершенню роботи додатка операційною системою через перевищення лімітів оперативної пам'яті. Також додатки, які застосовують фільтри до відео в режимі реального часу, обробляють багатоканальний аудіосигнал або здійснюють потокову передачу у високій роздільній здатності, потребують прямого доступу до апаратних кодеків пристрою. У цьому випадку підходить лише нативний стек, оскільки тільки він здатний забезпечити алгоритмічну обробку масивів даних за постійний час.

Висновок

Вибір технологічного стеку для розробки мобільних додатків не повинен базуватися виключно на суб'єктивних уподобаннях чи короткострокових трендах. Це технічне рішення, яке вимагає аналізу середовища розробки, механізмів взаємодії з апаратним забезпеченням, методів візуалізації графіки та методів управління пам'яттю. Нативний підхід залишається золотим стандартом для завдань низького рівня. Flutter домінує у сфері високоспеціалізованої графіки завдяки своєму графічному рушію. React Native забезпечує ефективне управління станом для інтерфейсів з великою кількістю контенту, тоді як Kotlin Multiplatform пропонує безкомпромісну продуктивність для рішень з високим обчислювальним навантаженням.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Android Mobile App Developer Tools [Електронний ресурс] – Режим доступу: https://developer.android.com/
2. Apple Developer [Електронний ресурс] – Режим доступу: https://developer.apple.com/
3. Introduction · React Native [Електронний ресурс] – Режим доступу: https://reactnative.dev/docs/getting-started
4. Flutter - Build apps for any screen [Електронний ресурс] – Режим доступу: https://flutter.dev/
5. What is Kotlin Multiplatform [Електронний ресурс] – Режим доступу: https://kotlinlang.org/docs/multiplatform/kmp-overview.html

Маковій Андрій Васильович – студент групи 5ПІ-22Б, факультет інформаційних технологій і комп'ютерної інженерії, Вінницький національний технічний університет, Вінниця, e-mail: makoviystud@gmail.com

Науковий керівник – Бабюк Наталя Петрівна, к. т. н., доцент кафедри програмного забезпечення, Вінницький національний технічний університет, м. Вінниця, e-mail: babiuk@vntu.edu.ua

Makovii Andii V. – student of group 5PI-22B, Faculty of Information Technologies and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: makoviystud@gmail.com

Supervisor – Babiuk Natalia Petrivna, Candidate of Technical Sciences, Associate Professor of the Department of Software, Vinnytsia National Technical University, Vinnytsia, e-mail: babiuk@vntu.edu.ua