ПОБУДОВА МОБІЛЬНИХ ДОДАТКІВ З ВИКОРИСТАННЯМ КРОС-ПЛАТФОРМНОГО ПІДХОДУ

DOI 10.36994 / 2788-5518-2022-01-03-17

Гіоргізова-Гай В.Ш., *к.т.н. Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна. high.victoria@lll.kpi.ua*

Зарічний Я.С. *Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна. yarik120700@gmail.com*

Анотація: *Стаття присвячена аналізу сучасного стану засобів побудови мобільних застосунків з використанням крос-платформних фреймворків. Проведено порівняння переваг і обмежень поширених фреймворків: Kotlin Multiplatform Mobile (KMM), React Native, Flutter, Xamarin за рядом визначених критеріїв, серед яких продуктивність роботи, спосіб побудови користувацького інтерфейсу, підтримувані мови програмування та інші. Запропоновано рекомендації щодо доцільності вибору кожного з фреймворків залежно від задач проектування, а також наведено приклад побудови додатку з застосуванням KMM, який демонструє переваги прийнятого у фрейворку підходу.*

Ключові слова: *мобільні додатки, крос-платформні фреймворки, KMM, React Native, Flutter, Xamarin*

MOBILE APPS BUILDING USING A CROSS-PLATFORM APPROACH

Viktoriia Hiorhizova-Hai , *Ph.D., Ass..Prof. National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine. high.victoria@lll.kpi.ua*

Yaroslav Zarichnyi . *National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine. yarik120700@gmail.com*

Abstract: *A cross-platform approach to building mobile applications reduces developers' time and cost of developing and updating products. Now the needs to accelerate the release of software products on the market, their rapid upgrade and reduce the cost of development are a steadily growing. Therefore, the benefits of a cross-platform approach are becoming increasingly popular and cross-platform development tools are constantly improving. The article is devoted to the analysis of the current state of means of building mobile applications using cross-platform frameworks. A comparative description of common frameworks is carried out: Kotlin Multiplatform Mobile (KMM), React Native, Flutter, Xamarin on such criteria as performance, open source, developer support, user interface building methods, detailed documentation, supported programming languages. The advantages and limitations of each of them are analyzed. In general, applications based on cross-platform frameworks with the ability to create a user interface, compared to applications written for a specific operating system, are characterized by: lower execution and response performance; lower perfection of UX and UI design; difficulties with supporting some hardware device functions, as well as dependence on the framework on which the application is created. For exampl, after building programs using Xamarin, React Native or Flutter frameworks, it will not be possible to use the same code in another structure. The Kotlin Multiplatform Mobile platform differs from them in its approach to building an application, which involves reusing shared business logic and creating a platform-dependent user interface layer. Code written for reuse with KMM may be used to write platform-dependent apps for Android and iOS, while retaining the native performance of an app which was written in KMM. The article offers recommendations on the feasibility of choosing each of the considered frameworks depending on the design tasks and gives an example of building an application using the KMM framework. The article can be useful for developers of mobile applications, as well as students and teachers who are interested in the topic.*

Key words: *mobile apps, cross-platform frameworks, KMM, React Native, Flutter, Xamarin..*

Вступ

Нативна (рідна для конкретної операційної системи) платформа проти крос-платформності - це нескінченна дискусія, яка протягом багатьох років хвилює спільноту розробників мобільних застосунків.

Як нативні, так і крос-платформні технології розробки додатків перебувають у постійному стані розвитку. Тому доцільно час від часу переглядати відомі та нові варіанти крос-платформних рішень, щоб орієнтуватись в поточних пропозиціях ринку і обирати найбільш ефективний інструментарій відповідно до задач проектування.

Крос-платформні фреймворки дозволяють створити програму, яка охоплює якомога більшу кількість послідовників бренду та велику кількість кінцевих пристроїв під час розробки.

Основні переваги крос-платформних фреймворків:

- Економія часу. Розробники зможуть швидше розгортати свої програми за допомогою мобільних крос-платформних платформ завдяки гібридним додаткам, що пришвидшують випуск на ринок.
- Економія витрат. Розробка крос-платформних додатків базується на концепціях «пиши один раз, запускай скрізь». Код багаторазового використання та гнучка розробка додатків за допомогою інструментарію може зменшити витрати на розробку. Використовуючи мобільний крос-платформний фреймворк, розробники можуть досягти значної економії не тільки часу, а й коштів, які могли б витрачатись на підтримку вдвічі більшої команди, якщо рахувати по одному розробнику на основні платформи (iOS та Android).
- База з одним кодом. Ці фреймворки можуть дозволити розробникам використовувати одну базу коду у кількох операційних системах та додатках, можуть допомогти розробникам у найкращій підтримці кодової бази додатка.
- Максимальний вплив на цільову аудиторію. Використовуючи мобільний крос-платформний підхід, можливо створювати програми та розгортати їх на різних платформах, включаючи веб-додатки. Це означає, що створюючи один додаток, можна покрити обидві платформи iOS та Android, максимізуючи охоплення додатка.
- Краще обслуговування та розгортання. Оновлення можна негайно синхронізувати на всіх платформах та пристроях, тим самим заощаджуючи час і кошти. Більше того, якщо помилка виявлена в загальній кодовій базі, її слід виправити тільки один раз.
- Скорочення часу виходу на ринок і оновлення. Як вже згадувалось вище, «пиши один раз, запускай скрізь» - це концепція, яка підтримується при створенні на різних платформах. Крім того, якщо потрібно перетворити або знайти програму, розробники легко вводять незначні зміни в одне місце коду. Це також допомагає виводити продукцію на ринок швидше, ніж конкуренти, покращуючи залучення клієнтів.
- Уніфікований дизайн. Користувачі можуть розпізнавати елементи користувацького інтерфейсу та розуміти їх роботу на різних платформах. Тому досвід користувача (UX, user experience/досвід взаємодії) - це важлива річ, яку слід зберігати для будь-яких додатків.

Ще зовсім недавно розробка між-платформних додатків була обмежена створенням простих мобільних застосунків та ігор. З часом нові технології зробили крос-платформну розробку більше пристосованою, потужною та гнучкою. А оскільки потреби у прискоренні випуску програмних продуктів на ринок, їх швидкому оновленні і зниженні собівартості розробки неухильно зростають, переваги крос-платформного підходу стають все більше затребуваними.

Однак через платформний розвиток крос-платформна розробка все ще стикається з такими проблемами, як:

- збій у продуктивності через непослідовне спілкування між власниками платформ та нерідними компонентами гаджетів;
- проблеми загальної продуктивності застосунків, які можуть призвести до погіршення роботи користувачів;
- проблеми використання деяких нативних для платформ бібліотек.

Метою статті є аналіз засобів побудови мобільних застосунків з використанням крос-платформних фреймворків, порівняльна характеристика сучасних фреймворків, висновки щодо доцільності їх вибору залежно від задачі проектування.

Виконання досліджень

На сьогодні серед найпопулярніших фреймворків для побудови мобільних додатків можна виділити: Kotlin Multiplatform Mobile, React Native, Flutter, Xamarin [2 - 5].

У якості критеріїв порівняння було обрано продуктивність роботи фреймворку, чи має фреймворк відкритий вихідний код, спосіб побудови інтерфейсу користувача (UI, user interface/інтерфейс користувача). Також треба звертати увагу на зрілість фреймворку, тобто на термін його існування, адже ми прагнемо користуватися надійним фреймворком, який буде мати підтримку в майбутньому. Крім зрілості самого фреймворку, якщо він з відкритим вихідним кодом, важливо, щоб фреймворк мав велику спільноту, яка ним користується та розвиває його. До важливих критеріїв також відносяться наявність докладної документації, підтримувані мови програмування та вартість використання цього фреймворку.

Нижче наведено порівняльну табл.1 по заданим критеріям

У випадках, коли створення продукту має обмежені терміни, варто обирати крос-платформні рішення, які дозволяють будувати уніфікований користувацький інтерфейс UI, щоб заощадити час на створення додатку. Такий підхід в цілому є виправданим, але варто пам’ятати, що уніфікований інтерфейс не завжди є добрим рішенням. По перше,

через візуальну «не нативність» додатку, по-друге через можливе зниження продуктивності роботи додатку.

Розглянемо кожен з наявних фреймворків більш докладно.

React Native

React Native використовує інший підхід, ніж інші гібридні та мобільні веб-моделі. Замість того, щоб імітувати продуктивність як у нативного додатку, він бере фактичні блоки власного інтерфейсу користувача UI і збирає їх за допомогою спеціальної бібліотеки JavaScript від React. Розробники мають можливість писати та вбудовувати власний код, а також писати на суміші платформо-залежних компонент, бібліотек та React коду, щоб отримати точну бажану функцію, зберігаючи оригінальний вигляд UI елементів, яку чекають прихильники тієї чи іншою ОС [6].

Теоретично розробники створюють свій код один раз на JavaScript, а React Native має дбати про створення версій для певної платформи. Насправді переклад між операційними системами не є ідеальним, але існує значна частина кодової бази, яка використовується між платформами.

Основним недоліком React Native є продуктивність. Вона краща, ніж у інших гібридних інструментів та веб-програм, але не можна обійти велику накладну структуру, яка сповільнює продуктивність порівняно з нативними програмами. Для простих програм і створення MVP (наприклад, для тестування бізнес-гіпотез) зниження продуктивності не настільки помітне, щоб мати значний вплив. Використання React Native для більш складного застосування може означати вплив на досвід взаємодії користувача UX.

Програми React Native більші за нативні програми. Це має кілька неприємних побічних ефектів. Користувачі старіших або економічних моделей телефонів можуть відчути незручність через більший розмір додатків.

Як згадувалося раніше, девіз «пиши один раз, запускай скрізь» не зовсім точний. Розробники повинні налаштувати додаток для кожної платформи. Розмір цього додаткового фрагмента коду залежить від функцій програми та відповідної операційної системи. Наприклад, для ОС Android ReactNative більш зручний, і додатки для цієї ОС виходять меншого розміру. На практиці від 60 до 90% кодової бази можна повторно використовувати. Хоча це значно скорочує час розробки, однак означає, що React Native не є ідеальним рішенням, що не залежить від платформи [6].

Flutter

В цілому Flutter має такі ж переваги, як і ReactNative. Цей потужний фреймворк, що дозволяє створювати багатофункціональні програми, підходить як для новачків, так і для досвідчених програмістів. Його перевагами є висока швидкість, зручність і простота. Однак він має деякі недоліки, особливо під час створення програм в наступних ситуаціях.

1. Необхідно використовувати рідкісні рідні для ОС бібліотеки.

Найважливіші бібліотеки вже є, і постійно додаються нові. Наприклад, у другій половині 2018 року в публічний репозиторій проекту Flutter були додані численні бібліотеки (очевидно, в рамках підготовки до першого стабільного релізу), причому найважливіші, такі як Google Analytics, Firebase, Maps тощо, вже існують. Однак, якщо розробник має намір використовувати певну рідну бібліотеку, ще недоступну в репозиторії Flutter під час розробки програми, це потребуватиме чимало додаткових зусиль і часу. Для цього розробникам доведеться впровадити налаштовані канали платформи (абстракції, які дозволяють спілкуватися між собою частині додатку на Flutter та частині на рідних фреймворках) окремо для Android та для iOS.

2. Flutter не завжди є найкращим рішенням для програм, які напряму спілкуються з IoT пристроями через Bluetooth.

Якщо за допомогою Flutter потрібно розробити такий додаток, можна піти одним з двох шляхів. Або розробляти такі функції окремо для iOS і Android. Після чого можна додати їх до основної програми Flutter через канали платформи. Але сумнівно, чи цей процес заощадить більше часу для розробника, ніж фактична розробка двох рідних програм. Або можна спробувати створити цю функціє додатку для платформ iOS і Android одночасно за допомогою FlutterBle – плагіна Bluetooth для Flutter. Але в такому випадку додаток буде залежати від стороннього плагіну, тому не всі проблеми, які можуть виникнути с виходом нової версії ОС, будуть швидко вирішені [7].

Xamarin

Xamarin також позиціонує себе, як «пиши один раз, запускай скрізь», але має аналогічні недоліки, серед яких розмір додатку, труднощі роботи з рідними бібліотеками платформ. Коли випускаються оновлення для Android або iOS, розробники Xamarin не можуть використовувати їх одразу. Їм потрібен деякий час, щоб належним чином інтегрувати нові функції в екосистему, що спричиняє затримку оновлення і може вплинути на репутацію компанії. Також важко створювати привабливі рішення за допомогою Xamarin із складними графічними елементами або анімацією. Отже, ця платформа не є найкращим вибором для створення програм із розширеною візуалізацією або ігрових програм [8].

Kotlin Multiplatform Mobile

З наведеної вище таблиці стає зрозумілим, що більшість крос-платформних фреймворків, які надають змогу побудови інтерфейсу користувача, мають недоліки з продуктивністю виконання додатка, а також не зручного використання API платформи, коли потрібно працювати, наприклад, з геолокацією або камерою на девайсі [9].

Фреймворк Kotlin Multiplatform Mobile має дещо інший підхід до створення додатку, а саме повторне використання спільної бізнес логіки та створення платформозалежного шару інтерфейсу користувача.

При виборі Kotlin Multiplatfrom Mobile, як фреймворку для крос-платформного рішення потрібно усвідомити його посил та структуру. Спробуємо розглянути фактори мотивації для цього:

- прагнення розробити додатки для Android та iOS, але не робити роботу двічі;
- зберігання принципу DRY- не повторюйся (Don't repeat yourself);
- прагнення збереження нативного інтерфейсу користувача для кожної с платформ.

Сучасність вимагає сучасних додатків, а сучасні додатки - сучасних архітектур. І екосистема Android, і iOS перебувають у фазі, коли вони погано поєднуються із шаблонами архітектури стереотипів. Існують такі поняття, як кешування, підтримка стану перегляду, стеження за змінами життєвого циклу тощо, а схема на рисунку 1 пояснює, яким шаром програми можна поділитися. Це дає розуміння, де саме можна використовувати концепції KMM.

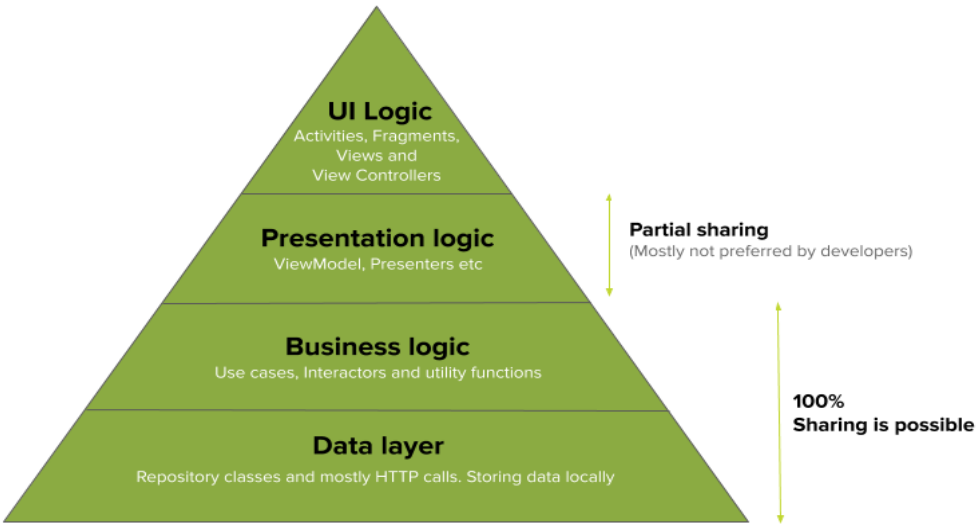

Рис. 1. Концепція використання KMM

На рис. 1 показано, що крос-платформним на 100% може бути рівень бізнес логіки і рівень даних, який має класи, відповідальні за здійснення мережевих викликів і запити до бази даних. Ці можливості KMM надає фреймворк Ktor, який дозволяє здійснювати асинхронні мережеві виклики. Також є інший, сумісний з KMM фреймворк під назвою SQLDelight, який дозволяє писати код для виконання деяких операцій з базою даних SQLite [10]. На рисунку 2 показано, як можна розділити структуру майбутнього додатку на три основні шари, тобто спільний код, власний код iOS та власний код Android. Спільний код матиме спільну реалізацію бізнес-логіки навколо мережевих викликів та інших основних функцій типу утиліти.

Приклад застосування КММ

Розглянемо використання КММ на прикладі додатку, який дозволяє будувати пішохідні маршрути, створювати позначки на карті тощо.

Через використання Kotlin Multiplatform Mobile існує можливість повторно використати шар сервісів, створюючи їх інтерфейс загальним для iOS застосунку, і мати можливість використати такий самий інтерфейс вже при розробці Android версії застосунку. Також, варто відмітити, що повторного використання можна домогтись на модельному шарі додатку. Це надасть змогу синхронізувати моделі на усіх платформах iOS та Android. Важливо відзначити, що побудова моделей на мові Kotlin дасть змогу в подальшому використовувати ці моделі на серверній частині при побудові серверної логіки за допомогою мови програмування Kotlin та фреймворку Ktor або Spring Boot.

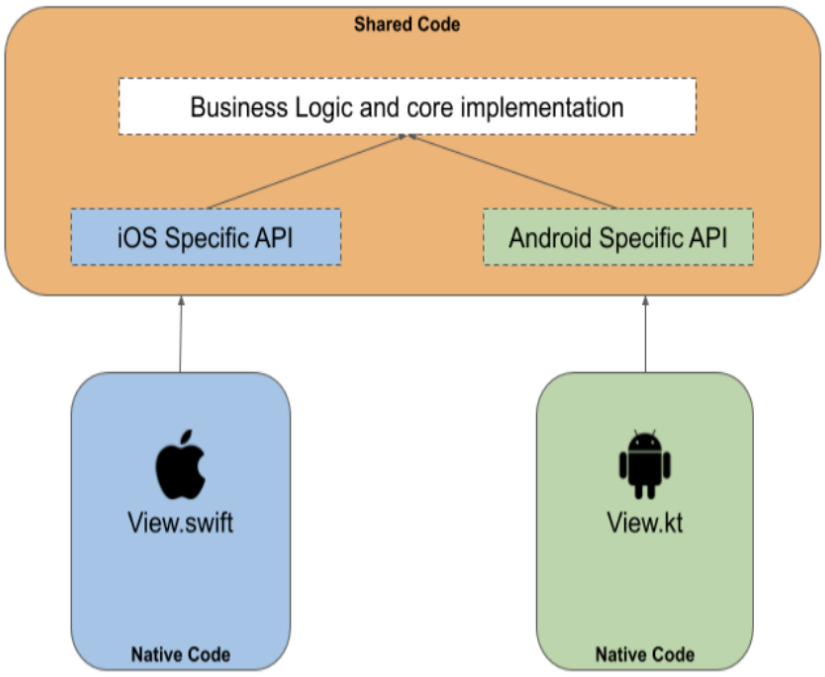

Рис. 2. Архітектура КММ

На рис. 3 відображено, що в спроектованому додатку усі моделі та сервіси можуть повторно використовуватись в Android та в iOS версіях. Самі сервіси поділені за принципом єдиної відповідальності та при необхідності будуть взаємодіяти через медіатор у вигляді MapViewController.

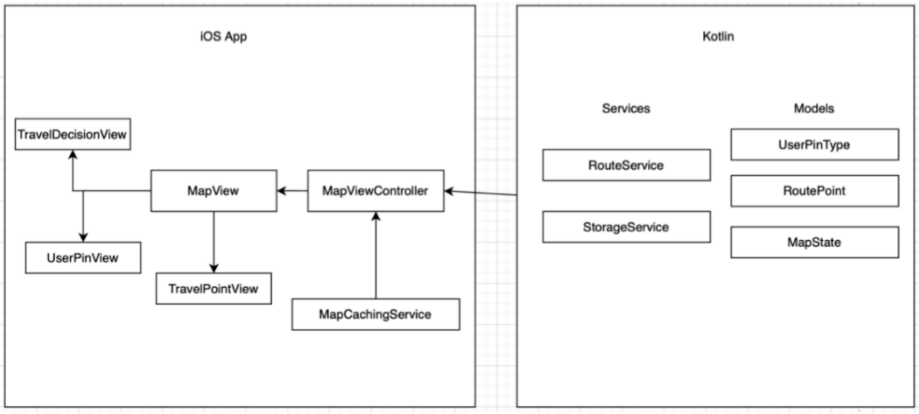

Рис. 3.Архітектура мобільного застосунку

Висновки

Розробка крос-платформних додатків поки не стала ідеальним рішенням у порівнянні з додатками, створеним за допомогою нативних для платформи бібліотек та рішень. В той же час, крос-платформні рішення мають багато суттєвих переваг. Крос-

платформна розробка швидше, дешевше і простіше. Таки переваги роблять цей підхід єдино можливим рішенням, наприклад, для певних типів стартапів. Це ідеальний інструмент, щоб перевірити ідею стартапу, або представити її інвесторам перш ніж виділяти кошти на розробку нативного додатка. Також даний підхід добре підходить для ряду корпоративних застосунків чи додатків, на функціональність яких не суттєво впливають розглянуті вище недоліки того чи іншого фреймворку.

Загалом, в якості недоліків крос-платформних мобільних додатків у порівнянні з платформо-залежними слід зазначити зазвичай нижчу швидкість роботи та реагування, менш досконалий UX (user experience/досвід взаємодії) та UI (user interface/інтерфейс користувача) дизайн та складності з підтримкою деяких апаратних функцій пристрою. Також часто критикується невіддільність додатка від фреймворку, на якому він був створений. Наприклад, після створення програми за допомогою Xamarin, React Native або Flutter не буде можливості використовувати той самий код в іншій структурі.

Проте напрямок крос-платформної розробки активно розвивається і шукає нові підходи для подолання існуючих обмежень і недоліків. Наприклад, код написаний з використанням підходу KMM, можливо повторно використати при написанні платформо-залежного додатку під ОС Android та iOS, зберігши при цьому нативну продуктивність додатка, який був написаний на KMM.

Також розробникам не слід забувати про інші, крім крос-платформного, підходи до побудови сучасних мобільних додатків. Наприклад, реалізація гібридних додатків [11] є дешевою, хоча і не дуже підходять для інтеграції апаратних функцій, а прогресивні веб-програми [12] добре підходять для електронної комерції.

Література

1. Cross platform app frameworks in 2021. URL: https://www.netsolutions.com/insights/cross-platform-app-frameworks-in-2021
2. Документація Kotlin Multiplatform Mobile. URL: https://kotlinlang.org/docs/multiplatform.html
3. Документація ReactNative. URL: https://reactnative.dev
4. Документація Flutter. URL: https://flutter.dev
5. Документація Xamarin. URL: https://docs.microsoft.com/en-us/xamarin/
6. Why you should (or shouldn’t) use React Native. URL: https://www.conceptatech.com/blog/why-you-should-or-shouldnt-use-react-native
7. What is Flutter. URL: https://lanars.com/blog/what-is-flutter
8. Why use Xamarin? URL: https://www.sam-solutions.com/blog/xamarin-cross-platform-development/
9. Why You Should Consider Kotlin Multiplatform. URL: https://x-team.com/blog/kotlin-multiplatform/
10. KMM: common code for iOS and Android, DOU. URL: https://medium.com/globant/kotlin-multiplatform-mobile-kmm-code-sharing-between-android-and-ios-9a9af66e2655
11. Гібридні мобільні додатки та їх переваги. URL: https://web4u.in.ua/blog/g-bridn-mob-l-n-dodatki-ta-h-perevagi-18
12. PWA, або Прогресивні веб-додатки. URL: https://smile-ukraine.com/ua/pwa/introduction

References

1. Cross platform app frameworks in 2021. URL: https://www.netsolutions.com/insights/cross-platform-app-frameworks-in-2021
2. Kotlin Multiplatform Mobile documentation. URL: https://kotlinlang.org/docs/multiplatform.html
3. ReactNative documentation. URL: https://reactnative.dev
4. Flutter documentation. URL: https://flutter.dev
5. Xamarin documentation. URL: https://docs.microsoft.com/en-us/xamarin/
6. Why you should (or shouldn’t) use React Native. URL: https://www.conceptatech.com/blog/why-you-should-or-shouldnt-use-react-native
7. What is Flutter. URL: https://lanars.com/blog/what-is-flutter
8. Why use Xamarin? URL: https://www.sam-solutions.com/blog/xamarin-cross-platform-development/
9. Why You Should Consider Kotlin Multiplatform. URL: https://x-team.com/blog/kotlin-multiplatform/
10. KMM: common code for iOS and Android, DOU. URL: https://medium.com/globant/kotlin-multiplatform-mobile-kmm-code-sharing-between-android-and-ios-9a9af66e2655
11. Hybrid mobile applications and their benefits. URL: https://web4u.in.ua/blog/g-bridn-mob-l-n-dodatki-ta-h-perevagi-18
12. PWA, or Progressive Web Applications. URL: https://smile-ukraine.com/ua/pwa/introduction