УДК 004.942
DOI https://doi.org/10.32782/2663-5941/2025.3.2/56

Філімончук Т.В.
Харківський національний університет радіоелектроніки

Колтун Ю.М.
Харківський національний університет радіоелектроніки

Климова І.М.
Харківський національний університет радіоелектроніки

Холобок В.І.
Харківський національний університет радіоелектроніки

МОДЕЛЬ ПІДХОДУ РОЗРОБКИ МОБІЛЬНИХ ЗАСТОСУНКІВ

У статті досліджується актуальність розробки мобільних застосунків у сучасних умовах цифрової трансформації з акцентом на вдосконалення моделі вибору між нативним та кросплатформним підходами до їх створення. Аналізуються ключові аспекти, що впливають на успішність мобільних рішень: архітектура застосунку, вибір операційної системи (Android, iOS), аудиторія та терміни реалізації проєкту. Дослідження базується на вивченні наукових публікацій, які розглядають інструменти розробки, принципи проєктування та порівняльний аналіз існуючих технологій. В статті наголошено на важливості застосування сучасних інструментів для масштабованості, надійності, зручності та ефективності мобільного застосунку. На основі проведеного аналізу існуючого стану запропонована оновлена модель розробки мобільних застосунків, яка включає додаткові елементи, що націлені на створення перспективного рішення для користувачів. Доцільність кожної складової в моделі детально аналізується з точці зору переваг та недоліків. Такий підхід дозволяє проєктувати більш гнучкі, функціональні та адаптовані до ринкових вимог програмні рішення.

Впровадження моделі в процес розробки мобільних застосунків сприяє створенню стабільних, масштабованих та користувацько-орієнтованих рішень, що відповідають сучасним вимогам цифрового ринку за рахунок підтримки стандартів розробки. Результати впровадження моделі наголошують, що нативна розробка забезпечує високу продуктивність та доступ до всіх можливостей пристрою, але потребує більших ресурсів, кросплатформні рішення економлять час та кошти, хоча мають обмеження в інтеграції з API, використання гібридних та PWA-рішень більш виправдане для веборієнтованих продуктів із меншими вимогами до продуктивності. Результати тестування використання моделі свідчать про необхідність гнучкого підходу до вибору технологій залежно від цілей проєкту та потреб аудиторії.

Ключові слова: *розробка мобільних застосунків, нативний та кросплатформний підхід, Android, iOS, UX, UI, безпека даних, API, хмарні сервіси, бази даних.*

Постановка проблеми. На даний час однією із найбільш динамічних галузей, що просувається, є розробка мобільних застосунків, які відкривають нові можливості для надання послуг та взаємодії з користувачами в реальному часі. Проте ключ до успіху мобільного рішення полягає не лише в його інноваційних функціях, а залежить його архітектури.

Дана робота присвячена вибору напряму розробки мобільних застосунків (нативного або кросплатформного), який допоможе розробнику прийняти рішення з урахуванням важливих компромісів. Дослідження включає перелік чітких критеріїв, яким слід приділити увагу при розгортанні застосунку.

У вік цифрових технологій мобільні застосунки виступають інструментом для багатьох індустрій, таких як транспорт, електронна комерція, онлайн-банкінг, подорожі, роздрібна торгівля та корпоративні послуги. Застосунки можуть бути орієнтовані на споживачів (B2C) або на бізнес (B2B). B2C-застосунки спрямовані на задоволення індивідуальних потреб користувачів та поділяються на контент-орієнтовані, маркетингові та сервісні. Контент-орієнтовані рішення надають користувачам інформацію, розваги, можливості для спілкування або підвищення продуктивності. Маркетингові, як правило, використовуються для реклами бренду, а сервісні дозволяють виконувати певні задачі, такі

як бронювання квитків або перевірка розкладу поїздів.

B2B-застосунки підтримують внутрішні бізнес-процеси компаній, наприклад, управління складом або автоматизацію продажів. Вони зазвичай мають обмежений доступ та призначені для використання лише певними співробітниками. Вибір операційної системи, яку слід підтримувати, є одним з ключових рішень у процесі розробки таких застосунків.

Ринок смартфонів значно виріс за останнє десятиріччя. Протягом цього часу використовувалися різні мобільні ОС, такі як Android, Blackberry, iOS, Symbian та Windows. Сьогодні на ринку домінують дві платформи – Android та iOS, але у майбутньому ситуація може змінитися. Компанії повинні враховувати ці невизначені обставини, приймаючи рішення про стратегію розробки. Вибір ОС для розробки мобільних застосунків залежить не тільки від частки ринку або орієнтованої індустрії. Виділяють чотири фактори для визначення, яка ОС найкраще підходить для вимог підприємства:

– аудиторія: Android має більшу частку ринку, і за 2023 рік Google Play Store отримав 110 мільярдів завантажень застосунків по всьому світу, в той час як Apple Application Store отримав лише 41,5 мільярдів;

– монетизація: хоча Android має більше завантажень, Apple майже в два рази випереджає за валовими витратами споживачів;

– термін реалізації проєкту: розробка застосунків для Android займає в середньому на 30–40% більше часу, ніж для iOS;

– бюджет: вартість розробки мобільного застосунку залежить від обсягу та складності проєкту, а також від підтримки пристроїв та версій ОС, маркетингових зусиль, участі команди та міждепартаментної взаємодії, а також підтримки та оновлення.

Аналіз останніх досліджень і публікацій. Під час вивчення сучасних підходів до розробки мобільних застосунків було проаналізовано низку наукових публікацій, присвячених архітектурі, інструментам та принципам проєктування мобільних застосунків.

У роботі [1] розглянуто модель мобільного застосунку, яка орієнтована на обробку даних. Основна увага у роботі приділяється взаємодії ключових компонентів, таких як інтерфейс користувача (UI), клієнтська та серверна частини, API, заходи безпеки, модулі аналітика та моніторингу, хмарні сервіси та модулі тестування. Автори натякають, що ретельно продумана модель мобільного застосунку є ключем до його успішної реалізації та конкурентоспроможності на ринку, забезпечує ефективну взаємодію компонентів, безпеку та продуктивність, що робить її придатною для сучасних технологічних вимог.

У роботі [2] розглянуто основні аспекти створення мобільних застосунків та вибір відповідних інструментів для їх розробки. Автори аналізують переваги та недоліки нативного та кросплатформного підходів. Використання першого забезпечує високу продуктивність та оптимізований UX, але їх розробка значно дорожча. Застосунки, які розроблено з орієнтацією на другий підхід, є більш економічними, проте можуть мати обмеження у продуктивності та гнучкості. Загалом в статті підкреслюється важливість усвідомленого підходу до вибору інструментів розробки, що сприяє створенню якісних, продуктивних та доступних мобільних рішень.

Метою роботи [3] є порівняльний аналіз трьох популярних крос-платформних рішень для створення мобільних застосунків: React Native, Flutter та Kotlin Multiplatform. Вибір між цими технологіями, як правило, залежить від вимог проєкту, кожна із розглянутих мов та технологій може бути використана тільки з точці зору аналізу завдань та сценаріїв використання. Фреймворк React Native підходить для реалізації швидких прототипів та веборієнтованих команд, Flutter є оптимальним для застосунків із складним UI та необхідністю однакового вигляду на всіх платформах, а Kotlin Multiplatform найбільше підходить для проєктів, які потребують високої продуктивності та інтеграції з нативними рішеннями. Таким чином, використання кросплатформних рішень дозволить розробнику зменшити витрати на розробку мобільних застосунків, проте кожне з рішень має свої переваги та обмеження, які необхідно враховувати при обиранні технології.

В роботі [4] розглянуто архітектуру платформи Android та структуру мобільного застосунку. Автори докладно розкривають сутність та призначення основних програмних компонентів: Activity, Service, Content Provider та Broatcast Resiver, наводять методи, які слід використовувати для побудови інтуїтивного інтерфейсу користувача. Особлива увага приділяється формам збереження та обробки даних на мобільному пристрої.

Метою роботи [5] є ретельний аналіз підходів та інструментальних засобів кросплат-

формної розробки мобільних застосунків, які дозволяють прискорити процес написання програмного коду. Автор проводить аналіз популярних фреймворків, які можливо використовувати для розробки кінцевого рішення за певними критеріями. При обиранні технології слід керуватися потребами застосунку, бюджетом, наявністю команди розробників та іншими чинниками. Автор пропонує використовувати фреймворк React Native, який задовольняє більшість потреб.

У роботі [6] проведено аналіз вимог до проєкту, проєктуванню, подальшої реалізації з використанням відповідного інструментарію, тестуванню та подальшої підтримки. Автор розглядає процес побудови мобільних застосунків, як набір окремих модулів, що дозволяє забезпечити гнучкість системи. Головними складовими мобільного застосунку є UX та UI, моделювання яких повинно розпочинатись на етапі проєктування. Також автор пропонує окрему увагу приділити зберіганню інформації та наводить переваги та недоліки використання існуючих на даний час БД.

Розглянуті роботи демонструють динамічний розвиток мобільних технологій та різноманітність підходів до створення застосунків. Вони висвітлюють ключові аспекти, починаючи від вибору моделі розробки до використання сучасних технологій, таких як React Native, Flutter, Kotlin Multiplatform, а також хмарних сервісів та локальних баз даних. Таким чином, розробка мобільних застосунків – це не лише актуальна, а й стратегічно важлива сфера розвитку сучасних цифрових технологій, яка формує майбутній досвід користувача, змінює способи ведення бізнесу та створює нові можливості для взаємодії у цифровому світі.

Постановка завдання. Метою даного дослідження є модифікація існуючої моделі, яка орієнтована на розробку мобільних застосунків, за рахунок додавання та розширення її складових. Запропонована модель дозволить створювати якісні, продуктивні та зручні продукти, що відповідають потребам користувачів.

Виклад основного матеріалу. Проведений аналіз виявив, що існуюча на даний момент модель, яка орієнтована на розробку мобільних застосунків має наступні складові (1): FE (front-end) – клієнтська частина; BE (back-end) – серверна частина; API (application programming interface) – інтерфейс програмування застосунків, який забезпечує взаємодію між клієнтською та серверною частинами; DS (data storage) – методи зберігання даних застосунку.

$$M=\{FE, BE, API, DS\}. \quad (1)$$

Наведена модель (1) підтримує просту взаємодію між кінцевим користувачем та мобільним застосунком, та має ряд недоліків:

– вона може бути менш не точною, оскільки не враховує деякі аспекти та взаємодії між окремими компонентами, що в свою чергу впливає на результати роботи застосунку та його ефективність;

– просту модель легше проєктувати та впроваджувати, але це обмежує її гнучкість та подальше удосконалення;

– просту модель не можливо використовувати в задачах, де потрібна складка або спеціалізована обробка.

Для усунення перелічених недоліків було вирішено модифікувати модель (1) за рахунок додавання та розширення її окремих складових. Запропонована модель (2) буде забезпечувати більш ефективну інтеграцію front-end та back-end компонентів, більш високий рівень продуктивності, безпеки та легкість масштабування за рахунок врахування ключових аспектів сучасної розробки:

$$M = \{OS, FE, BE, DS, UX, UI, PS, G, SDE, PN, TT, MAT\}, \quad (2)$$

де: OS (operating system) – нативна розробка на iOS або Andoid; UI (user interface) – інтерфейс користувача; UX (user experience) – досвід користувача; PS (payment systems) – інтеграція з платіжними системами; G (geolocation) – технологія визначення місцезнаходження; SDE (secure data exchange) – модуль безпечного обміну даними; PN (push notifications) – система повідомлень; TT (testing tools) – модуль інструментів тестування; MAT (monitoring and analytics tools) – інструменти для моніторингу та аналітики.

На даний час на ринку розробки мобільних застосунків конкурують дві платформи: iOS та Android. Головна їх відмінність полягає в використанні екосистеми. Як правило, платформа iOS має неповторну ідентичність, яка підтримується суворим контролем якості та дизайном. Платформа Android, надає розробникам більш гнучке середовище для розробки з використанням широкого спектру пристроїв. Кожна із платформ має свої переваги та недоліки, але орієнтуватись потрібно на цілі проєкту, доступні людські та грошові ресурси та очікувану продуктивність.

Front-end програмування відповідає за роботу клієнтської частини застосунку (веб або мобільний). На даному етапі розробки особлива увага приділяється розробці інтерфейсу між кінцевим користувачем та серверною частиною мобільного рішення. Складова FE запропонованої моделі відповідає за введення інформації від користувача, її первинну обробку, а також подальше її відправку на сервер за допомогою відповідного API.

Back-end програмування орієнтоване на розробку серверної частини програми, яка відповідає за передачу даних між користувачами. Складова BE моделі, що запропонована містить наступні інструменти (3):

$$BE = \{SA, API, AP, M\}, \quad (3)$$

де SA (server architecture) – серверна архітектура, яка містить алгоритми завантаження даних, методи авторизації, кешування, бази даних та інше; API (application programming interface) – інтерфейс прикладного програмування, який визначає функціональність серверної логіки; AP (admin panel) – адміністративна панель застосунку, функціонал якої розробляється виходячи з цілей та задач проєкту; M (metrics) – метрики, які оцінюють ефективність проєкту (статистичні дані щодо приросту користувачів, активності, щоденна відвідуваності проєкту в цілому та його окремих розділів, демографічні дані).

Складова DS (4) орієнтована на використання низки методів, які дозволяють здійснювати зберігання даних проєкту. Слід розуміти, що база даних є ключовим компонентом будь-якого сучасного застосунку, який забезпечує надійне зберігання та управління інформацією. У вебзастосунках база даних використовується, як правило, для зберігання контенту, даних користувача, а також журналів транзакцій. У мобільних застосунках вона часто виступає у ролі локального сховища для тимчасового зберігання даних або кешування, що забезпечує швидкий доступ до інформації навіть без підключення до мережі.

$$DS = \{DB, CS, FS, CM\}, \quad (4)$$

де DB (database) – організована структура для зберігання, зміни та обробки взаємозалежної інформації, переважно великих обсягів; CS (cloud storage) – хмарні сховища, орієнтовані на зберігання даних на віддалених серверах, доступ до яких здійснюється через інтернет; FS (file system) – файлова система, орієнтована на зберігання та управління даними на пристрої зберігання; C (caching mechanisms) – набір механізмів, що кешують.

В якості складової DB можливо використовувати:

– реляційні бази даних, які організовують дані у таблиці, що з'єднані між собою за допомогою ключів (MySQL, PostgreSQL, SQLite, MariaDB);

– NoSQL бази даних, які призначені для роботи з неструктурованими або слабо структурованими даними. NoSQL включають документоорієнтовані, колоночні, графові та інші типи баз даних (MongoDB, CouchDB, Firebase Firestore, Apache Cassandra, HBase, ScyllaDB, Neo4j, OrientDB);

– розподілені бази даних, де дані зберігаються на кількох серверах, що забезпечує високу доступність, відмовостійкість та масштабованість (CockroachDB, Google Cloud Spanner, TiDB).

Хмарні сховища (CS) – це сервіси, які дозволяють зберігати та керувати даними у віддалених дата-центрах, забезпечуючи доступність, безпеку та масштабованість. Перевагами використання хмарних сховищ є можливість резервного копіювання та захист даних, а також спільна робота з ними з будь-якого пристрою. Більшість хмарних сховищ, орієнтовані під використання конкретних цілей: для зберігання файлів (Google Drive, Dropbox, OneDrive), для застосунків (Amazon S3, Firebase, Cloud Storage), для баз даних (Cloud SQL, Firestore), для віртуальних машин (Amazon EBS, Azure Disks).

Файлова система (FS) – це структура, яка використовується операційною системою для розміщення та упорядкування даних, управління доступом, відстеження використання простору, веде процес журналювання (NTFS, ext4, XFS), здійснює захист від збоїв та відновлення даних збоїв (Btrfs, ZFS).

Файлові системи різняться за будовою та функціонуванням: локальні використовуються на жорстких дисках, SSD та флеш носіях (FAT32, exFAT, NTFS, APFS, ext4, XFS, Btrfs), мережні – для спільного доступу до мережі (NFS, SMB, AFP, WebDAV), розподілені – в кластерах та хмарних системах (GFS, HDFS, CephFS, GlusterFS). Вибір файлової системи залежить від конкретних потреб, сумісності з ОС та типу носія інформації.

Кешування – тимчасове зберігання даних, які часто використовуються у швидкому сховищі (оперативна пам'ять, SSD, диск) для прискорення

доступу та зниження навантаження на основне джерело даних.

Існують різні види кешування (CM):

– апаратне: кеш-пам'ять процесора, кеш-контролери жорстких дисків;

– програмне: в оперативній пам'яті, на диску;

– мережне: кешування у CDN, DNS-кеш.

Прикладами популярних інструментів кешування є Redis, Memcached, Varnish, NGINX FastCGI Cache, Cloudflare CDN, їх використання значно покращує швидкодію застосунків.

Інтерфейс користувача (UI) – це головний елемент мобільного застосунку. Використання існуючих на даний час фреймворків та спеціалізованих бібліотек спрощують його розробку.

Кросплатформні UI-фреймворки дозволяють писати один код та запускати його на платформах iOS та Android. Наприклад, Flutter дозволяє розробляти UI за допомогою віджетів (Material Design, Cupertino), має гнучку систему кастомізації інтерфейсу та високу продуктивність завдяки використання рушия Skia. Фреймворк React Native має компоненти для створення UI у стилі iOS та Android, підтримує сторонні UI-бібліотеки та дозволяє використовувати нативні модулі кастомізації.

Нативні UI-фреймворки використовуються для розробки під конкретну платформу. Наприклад, для розробки під Android можливо використовувати зв'язку XML та View System для класичного способу розмітки UI або Jetpack Compose для написання UI в декларативному стилі. Для iOS в якості основного фреймворку можливо використовувати UIKit або SwiftUI, які забезпечують декларативний підхід для організації швидкого прототипування.

Досвід користувача (UX) – це емоційні враження людини від взаємодії з кінцевим рішенням і при розробці слід дотримуватися принципів UX-дизайну:

– простота: мінімалізм в інтерфейсі та зрозуміла навігація;

– швидкодія: швидкість завантаження та чуйність критичні;

– інтуїтивність: логічна архітектура, зрозумілі жести та елементи управління;

– доступність: підтримка різних екранів, шрифтів, режимів;

– зворотній зв'язок: анімації, вібрації, підказки.

Складова PS моделі, що пропонується, забезпечує зручні та безпечні онлайн-платежі для мобільного застосунку та надає інтеграцію з платіжними системами, такими як Stripe, PayPal, Apple Pay та Google Pay. Ці системи дозволяють приймати платежі від клієнтів по всьому світу, використовуючи різні методи оплати. Варто зазначити, що доступність та умови використання платіжних систем можуть відрізнятися залежно від країни. Наприклад, у деяких певні платіжні системи можуть бути недоступні або мають обмеження.

Технологія визначення місцезнаходження користувача або пристрою (складова G) на основі GPS, Wi-Fi, мобільних мереж або IP-адреси використовується у мобільних застосунках, вебсайтах та корпоративних рішеннях для доставки персоналізованого контенту, логістики, картографії та навігації. На даний час технологія геолокації застосовується у:

– логістиці та доставці: трекінг посилок, моніторинг кур'єрів;

– навігації: GPS-карти для автомобілів, туристичні маршрути;

– локальних сервісах: пошук найближчих закладів, ресторанів та ін.;

– безпеці: відстеження викрадених гаджетів.

Модуль, який відповідає за безпечний обмін даних (SDE) – це комплекс заходів, що включає механізми шифрування, захищені канали зв'язку та автентифікацію (5). Чим більш чутливі дані передаються, тим більше рівнів захисту слід застосовувати при розробці застосунку. Впровадження ефективних практик безпеки є критичним для забезпечення конфіденційності, цілісності та доступності даних користувачів.

$$SDE = \{E, SCE, AV, SL\}, \quad (5)$$

де E (encryption) – механізми шифрування; SCC (secure communication channels) – використання захищених каналів зв'язку; AV (authentication + verification) – аутентифікація та авторизація; SL (social login) – аутентифікація через акаунти соціальних мереж.

Механізми шифрування (E) – це набір засобів, які дозволяють здійснювати перетворення даних у захищений формат, що можна прочитати лише за допомогою спеціального ключа. Шифрування використовується для захисту конфіденційної інформації під час її зберігання та передачі. На даний час існує декілька типів шифрування, які розрізняються за алгоритмами та областями застосування:

– симетричне шифрування: AES, DES, 3DES, Blowfish/Twofish;

– асиметричне шифрування: RSA, ECC, Diffie-Hellman;

– хешування: SHA, MD5, bcrypt / scrypt / Argon2;

– шифрування на транспортному рівні: TLS, SSL, End-to-End (E2EE);

– стеганографія.

Вибір типу шифрування залежить від задачі, яка вирішується. Наприклад, для захисту файлів слід обрати AES-256, для веббезпеки – TLS + RSA/ECC, для зберігання паролів – bcrypt або Argon2, для обміну ключами – Diffie-Hellman, для зашифрованих месенджерів – End-to-End (E2EE).

Використання надійних протоколів передачі даних (SCE) при розробці мобільних застосунків дозволить зробити кінцеве рішення надійним, стабільним та затребуваним у користувачів. Вибір протоколу передачі даних також слід орієнтувати на вирішення конкретних задач. Наприклад, для організації безпечних вебзастосунків та API слід обирати HTTPS або TLS, для фінансових транзакцій орієнтуватись на TLS, HMAC, OAuth.

Аутентифікація та авторизація (AV) – це важливі процеси у сфері інформаційної безпеки, які використовують разом, але вони виконують різні функції. Аутентифікація – перевіряє особу або систему, яка намагається отримати доступ до ресурсу (2FA/MFA). Аутентифікація підтверджує, що користувач є тим, за кого себе видає, за допомогою облікових даних, таких як логін та пароль, біометричні дані або інші методи. Авторизація – це визначення прав та привілеїв аутентифікованого користувача щодо доступу до певних ресурсів або виконання дій (OAuth, JWT, 2FA). Після успішної аутентифікації система перевіряє, які дії дозволені цьому користувачу.

Можливість входу через акаунти соціальних мереж (SL) дозволяє користувачам здійснювати аутентифікацію на вебсайтах або в мобільних застосунках, використовуючи облікові записи із соціальних мереж, таких як Facebook, Google, Twitter, GitHub, LinkedIn. Така аутентифікація спрощує процес реєстрації та входу для користувачів, оскільки їм не потрібно створювати новий обліковий запис або запам'ятовувати ще один пароль. Перевагами входу через акаунти соціальних мереж є швидка реєстрація (без паролю), зручність для користувачів (авто-вхід), безпека (OAuth 2.0 + шифрування) та підтримка різних соцмереж (Google, Facebook, Apple).

Технологія push-сповіщень (PN) дозволяє надсилати користувачам повідомлення навіть тоді, коли застосунок не активний або працює у фоновому режимі. Такі повідомлення використовуються для інформування про важливі оновлення, новини, акції чи події, залучаючи користувачів до взаємодії із застосунком. Push-сповіщення доставляються в режимі реального часу; можуть містити текст, зображення, кнопки та інші інтерактивні елементи; працюють через сервери повідомлень (наприклад, Firebase Cloud Messaging для Android та APNs для iOS); використовуються для підвищення залученості користувачів.

Процес тестування є обов'язковою складовою розробки кінцевого продукту, який забезпечує стабільність, надійність та якість програмного забезпечення. Тестування дозволяє виявляти та усувати помилки на ранніх етапах, що мінімізує ризик збоїв після впровадження змін чи масштабування застосунку, а також гарантує, що функціонал працює відповідно до заданих специфікацій, забезпечуючи якісний досвід користувача. У розробці слід обов'язково застосовувати різні види тестування (6):

$$TT = \{F, UA/UX, P, S, C, A\}, \quad (6)$$

де F (functional) – функціональне тестування, яке перевіряє роботу застосунку в цілому; UA/UX – тестування зручності та інтуїтивності інтерфейсу, роботи анімації, зрозумілості; P (productivity) – тестування продуктивності (швидкість відповіді, навантаження); S (security) – тестування безпеки (захист даних при пересиланні та зберіганні); C (compatibility) – тестування сумісності (робота на різних пристроях, із різними версіями ОС); A (automated) – автоматизоване тестування (з використанням скриптів).

Наявність інструментів аналітики та моніторингу (MAT) є важливими складовими сучасного застосунку, які забезпечують глибоке розуміння поведінки користувачів та продуктивності системи. Інтеграція інструментів аналітики дозволяє відстежувати активність користувачів, аналізувати взаємодію з різними елементами інтерфейсу та збирати ключові метрики, такі як конверсії, час сесій та залученість.

Моніторинг продуктивності системи забезпечує можливість своєчасного виявлення та вирішення потенційних проблем, таких як високий час відгуку сервера, перевантаження бази даних чи збої в роботі API. Використання таких інструментів, як Google Analytics, Firebase

Analytics, Mixpanel або Sentry, дозволяє автоматизувати процес збору та аналізу даних, забезпечуючи актуальну інформацію для прийняття відповідних рішень. Такий підхід підвищує якість досвіду користувача, дозволяє своєчасно реагувати на проблеми та адаптувати функціонал відповідно до потреб цільової аудиторії. Інтеграція аналітики та моніторингу уніфікує процес управління даними, сприяючи підвищенню ефективності розробки та підтримки застосунків.

Висновки. В ході аналізу підходів для проєктування та розгортання мобільних застосунків було розглянуто наукові публікацій, які пропонують використовувати різноманітні мобільні платформ та SDK з вказанням на їх переваги та недоліки. Наявність великої кількості інструментів розробки створюють унікальні проблеми, такі як вибір SDK, досвід користувача, стабільність фреймворка, легкість оновлень, вартість розробки для кількох платформ та час виходу застосунку на ринок.

Проведений аналіз показав, що вибір напряму розробки мобільних застосунків, як правило, залежить від цілей проєкту, доступних ресурсів та очікуваної продуктивності: використовувати нативні застосунки слід для складних рішень із високими вимогами до продуктивності, кросплатформні підходи для ситуацій, коли є потреба створювати мобільні рішення для широкої аудиторії, а гібридні та PWA-рішення для веборієнтованих додатків з меншими вимогами до продуктивності. Також слід зазначити, що нативні рішення розробляються окремо для кожної платформи, що потребує більше зусиль: як грошових, так і людських. В протилежність нативній розробці кросплатформна дозволяє створювати застосунок з використанням єдиного коду для кількох платформ.

Вибір кросплатформного підходу може здатися очевидним, оскільки можна мати єдину базу коду для кількох ОС, що робить розробку та підтримку швидшими, простішими та дешевшими. Однак це не завжди так. Кросплатформні фреймворки мають обмеження, які роблять їх менш здатними для розробки застосунків у порівнянні з нативним підходом і важко інтегруються з API пристрою. Також слід зазначити, що інструменти для кросплатформної розробки складніші для тестування та налагодження, ніж нативні.

При розробці мобільних застосунків можливо використовувати низку інструментів дизайну та прототипування для створення макетів, прототипів та інтерактивних інтерфейсів (Figma, Adobe XD, Sketch, InVision). Каркасні макети можуть допомогти з проєктуванням логіки взаємодії користувачів із продуктом (Miro, Whimsical, Balsamiq). Дизайн-системи та UI-кити – це готові компоненти, які прискорюють роботу та роблять дизайн одноманітним (Material Design, Apple HIG, Figma Community). Використання при розробці застосунків інструментів для дослідження та тестування UX допоможе проаналізувати поведінку користувачів, протестувати прототипи та зібрати зворотний зв'язок (Hotjar, Google Analytics, Maze, UserTesting, Lookback). Інструменти тестування доступності допоможуть перевірити, наскільки інтерфейс зручний для людей з обмеженими можливостями (Stark, Axe DevTools, WAVE).

Модель вибору підходу розробки мобільного застосунку має низку складових, які орієнтовано на використання різноманітних інструментів розробки. В залежності від потреб застосунку обирається інструментарій, який дозволить збалансувати продуктивність, швидкість розробки та витрати. Для складних і ресурсомістких застосунків доцільним є використання нативної розробки, яка забезпечує найвищу продуктивність і доступ до всіх можливостей пристрою. Кросплатформні фреймворки дозволяють пришвидшити розробку та зменшити витрати, проте можуть мати обмеження щодо інтеграції з API пристрою та продуктивності. Використання гібридних та PWA-рішень виправдане для веборієнтованих продуктів із меншими вимогами до продуктивності. Крім того, інструменти для дизайну, UX-досліджень і тестування забезпечують створення зручного, доступного та функціонального інтерфейсу. Запропонована модель підвищує консистентність коду, оскільки всі частини проєкту розроблятимуться в єдиному стилі та з використанням єдиних парадигм. Кожен із цих компонентів впливає на загальну структуру застосунку, створюючи стабільну, продуктивну та легко масштабовану архітектуру.

Список літератури:

1. Бешта В.С., Комаричев А.В., Філімончук Т.В., Баранєй Д.І. Модель мобільного додатку, яка орієнтована на обробку даних. *Системи управління, навігації та зв'язку*. Полтава: ПНТУ, 2024. Випуск. 3 (77). С. 80–83. DOI: 10.26906/SUNZ.2024.3.080.

2. Якимчук В.С., Гаврилюк В.С. Порівняння операційних систем iOS та Android. Переваги та недоліки розробки мобільних додатків для кожної з систем. *Біомедична інженерія і технологія*. Київ: КПІ ім. Ігоря Сікорського, 2019. Вип. 2. С 86–94.

3. Ічанська Н.В., Улько С.І. Основні аспекти створення мобільних додатків та вибір інструментів їх розробки. *Системи управління, навігації та зв'язку*. Полтава: ПНТУ, 2020. Вип. 1(59). С. 74–78. DOI: 10.26906/SUNZ.2020.1.074.

4. Шматко О.В., Поляков А.О., Федорченко В.М. Аналіз методів і технологій розробки мобільних додатків для платформи Android: навч. посібник. Харків: НТУ «ХПІ», 2018. 284 с.

5. Кислий О.І. Порівняльний аналіз інструментальних засобів для розробки мобільних додатків. *Наукові записки молодих учених*. Кропивницький: РВВ ЦДУ ім В. Винниченка, 2021. Вип. 8. URL: https://phm.cuspu.edu.ua/ojs/index.php/SNYS/article/view/1863/pdf (дата звернення 24.04.2025).

6. Сідельнікова Д.С. Етапи створення дизайну мобільного застосунку. *Мультимедійні технології в освіті та інших сферах діяльності:* матеріали XIII наук.-практ. конф. (Київ, 2 листопада 2021 р). Київ: НАУ, 2022. Секція № 2. С. 105–109.

Filimonchuk T.V., Koltun Yu.M., Klymova I.M., Kholobok V.I. MODEL OF THE MOBILE APPLICATION DEVELOPMENT APPROACH

The article explores the relevance of mobile application development in the current conditions of digital transformation, with a focus on improving the model for choosing between native and cross-platform approaches to app creation. It analyzes key aspects that influence the success of mobile solutions: application architecture, operating system selection (Android, iOS), target audience, and project implementation timelines. The research is based on a study of scientific publications that review development tools, design principles, and comparative analysis of existing technologies. The article emphasizes the importance of using modern tools for scalability, reliability, convenience, and efficiency of mobile applications. Based on the analysis of the current state, an updated model of mobile app development is proposed, which includes additional elements aimed at creating a forward-looking solution for users. The relevance of each component in the model is thoroughly analyzed in terms of its advantages and disadvantages. This approach allows for designing more flexible, functional, and market-adapted software solutions. Implementing the model in the mobile app development process contributes to the creation of stable, scalable, and user-oriented solutions that meet the modern demands of the digital market through compliance with development standards. The results of implementing the model highlight that native development provides high performance and full access to device capabilities but requires more resources; cross-platform solutions save time and money, though they have limitations in API integration; hybrid and PWA solutions are more suitable for web-oriented products with lower performance requirements. Testing results indicate the need for a flexible approach in choosing technologies based on the project's goals and the needs of the target audience.

Key words: *mobile app development, native and cross-platform approach, Android, iOS, UX, UI, data security, API, cloud services, databases.*