

Міністерство освіти і науки України
Державний заклад
«Луганський національний університет імені Тараса Шевченка»

Навчально-науковий інститут математики та інформаційних технологій

Кафедра інформаційних технологій та систем

Лошакова Ірина Вадимівна

**КРОСПЛАТФОРМНА РОЗРОБКА:
ПОРІВНЯЛЬНИЙ АНАЛІЗ МЕТОДІВ ПРОГРАМУВАННЯ
МОБІЛЬНИХ ДОДАТКІВ**

кваліфікаційна робота

здобувача вищої освіти другого (магістерського) рівня

освітньої програми «Мультимедійні системи»

за спеціальністю 121 Інженерія програмного забезпечення

Особистий підпис

Ірина ЛОШАКОВА

Науковий керівник

Галина КОЗУБ,
кандидат педагогічних наук, доцент
кафедри інформаційних технологій
та систем

Завідувач кафедри

Микола СЕМЕНОВ,
кандидат педагогічних наук, доцент
кафедри інформаційних технологій
та систем

АНОТАЦІЯ

Дипломна робота присвячена дослідженню сучасних підходів до розробки мобільних додатків із використанням кросплатформних технологій. Актуальність теми зумовлена стрімким поширенням мобільних пристроїв і зростанням вимог до програмних продуктів, які мають стабільно працювати на різних операційних системах за умов обмежених ресурсів часу та бюджету.

У роботі розглянуто особливості нативної та кросплатформної розробки мобільних додатків, а також проаналізовано їхні сильні та слабкі сторони з позиції практичного використання. Основну увагу приділено сучасним кросплатформним фреймворкам React Native, Flutter та .NET MAUI, які на сьогодні активно застосовуються у реальних програмних проєктах. Проведено їх порівняння за такими критеріями, як продуктивність, архітектурні підходи, зручність розробки та якість користувацького інтерфейсу.

Практична частина роботи передбачає реалізацію прикладного кросплатформного мобільного застосунку на основі .NET MAUI та аналіз результатів його роботи. У межах дослідження було оцінено доцільність використання обраного підходу з урахуванням вимог до продуктивності та користувацького досвіду.

Отримані результати дозволяють сформулювати практичні рекомендації щодо вибору методів і технологій мобільної розробки залежно від особливостей проєкту, цілей та поставлених перед розробниками завдань.

Ключові слова: мобільні додатки, кросплатформна розробка, нативна розробка, React Native, Flutter, .NET MAUI.

ANNOTATION

This diploma thesis focuses on the study of modern approaches to mobile application development with the use of cross-platform technologies. The relevance of the topic is driven by the rapid spread of mobile devices and the growing expectations for software products that are expected to work reliably across different operating systems while being developed under limited time and budget constraints.

The paper examines the key characteristics of native and cross-platform mobile development and analyzes their strengths and weaknesses from a practical perspective. Particular attention is given to widely used cross-platform frameworks such as React Native, Flutter, and .NET MAUI, which are actively applied in real-world software projects today. These frameworks are compared based on performance, architectural approaches, development convenience, and the quality of the user interface.

The practical part of the work involves the implementation of a sample cross-platform mobile application using .NET MAUI, followed by an analysis of its performance and behavior. Within the scope of the study, the suitability of the selected approach is evaluated with regard to performance requirements and user experience considerations.

The results obtained make it possible to formulate practical recommendations for selecting mobile development methods and technologies depending on project specifics, goals, and the tasks faced by developers.

Key words: mobile applications, cross-platform development, native development, React Native, Flutter, .NET MAUI.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API	- інтерфейс прикладного програмування (Application Programming Interface);
AI / ІІІ	- штучний інтелект;
IT / ІТ	- інформаційні технології;
HTML	- мова розмітки гіпертексту (HyperText Markup Language);
CSS	- каскадні таблиці стилів (Cascading Style Sheets);
.NET MAUI (MAUI)	- багатоплатформний інтерфейс користувача (Multi-platform App UI);
PWA	- прогресивні веб-додатки (Progressive Web Applications);
UI	- інтерфейс користувача (User Interface);
СУБД	- система управління базами даних;
IT	- інформаційні технології;
UX	- користувацький досвід (User Experience);
MVVM	- архітектурний шаблон Model–View–ViewModel;
SDK	- набір засобів розробки програмного забезпечення (Software Development Kit);
SQL	- мова структурованих запитів (Structured Query Language);
GPS	- глобальна система позиціонування (Global Positioning System)
OS	- операційна система (Operating System)
CI/CD	- безперервна інтеграція та безперервне розгортання (Continuous Integration / Continuous Deployment)
DevOps	- сукупність практик, що поєднують розробку та експлуатацію програмного забезпечення;
GDPR	- Загальний регламент захисту даних Європейського Союзу (General Data Protection Regulation).

ЗМІСТ

ВСТУП.....	6
РОЗДІЛ 1. Огляд сучасного стану проблеми кросплатформної розробки мобільних додатків	
1.1. Сучасний стан та тенденції розвитку методів розробки мобільних додатків	9
1.2. Переваги та недоліки кросплатформних підходів	14
1.3. Огляд основних фреймворків та технологій кросплатформної розробки.....	17
1.4. Висновки до розділу 1	21
РОЗДІЛ 2. Теоретична розробка та порівняльний аналіз методів програмування мобільних додатків	
2.1. Аналіз нативної розробки, особливості та приклади	24
2.2. Особливості кросплатформного програмування (React Native, Flutter, .NET MAUI)	28
2.3. Порівняння продуктивності та якості користувацького досвіду.....	33
2.4. Інструменти тестування та відлагодження	37
2.5. Висновки до розділу 2	42
РОЗДІЛ 3. Програмна реалізація та дослідження ефективності кросплатформної розробки	
3.1. Постановка задачі та опис прикладного проєкту	45
3.2. Архітектура та структура застосунку	50
3.3. Реалізація додатку за допомогою кросплатформного фреймворку.....	55
3.4. Порівняння результатів роботи з нативними рішеннями	62
3.5. Висновки до розділу 3	66
ВИСНОВКИ	69
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	73
ДОДАТКИ	79

ВСТУП

Смартфони та планшети виконують функції персональних асистентів, платформ для комунікації, навчання, розваг, контролю здоров'я та бізнес-процесів. Фактично мобільні пристрої перетворилися на універсальні інструменти, без яких складно уявити сучасне цифрове середовище. Розвиток мобільних технологій є складовою загального процесу цифровізації суспільства, у межах якого мобільні застосунки відіграють важливу роль у повсякденному житті [49]. Це пояснює потребу у швидкій, якісній та ефективній розробці програмних додатків, здатних функціонувати на різних операційних системах.

Актуальність теми обумовлена необхідністю оптимізації процесів створення мобільних додатків. Традиційна нативна розробка, яка передбачає використання окремих інструментів і мов програмування для Android та iOS, потребує значних ресурсів і часу. Для вирішення цих викликів було розроблено кросплатформні технології, що дають змогу створювати єдину кодову базу для кількох операційних систем. Такі рішення скорочують витрати, прискорюють вихід продукту на ринок і підвищують доступність програмних рішень для малого та середнього бізнесу. Водночас постає питання порівняння продуктивності, якості користувацького досвіду, гнучкості й надійності кросплатформних методів в порівнянні з нативними. Вивчення цього питання є важливим як для наукової спільноти, так і для практиків. Оскільки воно впливає на стратегічний вибір технологій у сфері мобільного програмування.

Але, незважаючи на це, сьогодні існує потреба у дослідженні, яке б узагальнило, систематизувало існуючі відомості з даної проблеми.

Враховуючи все вищесказане, нами і була обрана тема магістерської роботи: "Кросплатформна розробка: порівняльний аналіз методів програмування мобільних додатків".

Об’єкт дослідження – процес розробки мобільних додатків для різних операційних систем.

Предмет дослідження – методи кроссплатформного програмування мобільних додатків та їх порівняння з нативними підходами.

Мета роботи полягає у проведенні порівняльного аналізу методів програмування мобільних додатків на основі кроссплатформних фреймворків і технологій, а також у визначенні їхніх можливостей, переваг і обмежень у практичній реалізації.

Відповідно до мети були визначені наступні **завдання**:

- 1) Проаналізувати сучасний стан і тенденції розвитку мобільних технологій та методів розробки додатків;
- 2) розглянути переваги й недоліки кроссплатформних підходів у порівнянні з нативними рішеннями;
- 3) здійснити огляд основних кроссплатформних фреймворків (.NET MAUI, Flutter, React Native) та оцінити їхні особливості;
- 4) дослідити архітектурні принципи побудови мобільних додатків і визначити оптимальні підходи до їх реалізації;
- 5) розробити прикладний проєкт мобільного додатку на базі обраного кроссплатформного інструменту;
- 6) провести експериментальне порівняння продуктивності, стабільності та якості користувацького досвіду кроссплатформного рішення й нативних аналогів;
- 7) сформулювати рекомендації щодо доцільності застосування кроссплатформних методів у різних сценаріях розробки.

У процесі виконання роботи було використано методи аналізу та узагальнення наукових джерел, порівняльного аналізу, а також методи практичної програмної реалізації й експериментального тестування.

Наукова новизна дослідження полягає в тому, що автором запропоновано підхід до порівняльного аналізу сучасних кроссплатформних технологій із урахуванням їх продуктивності, якості користувацького досвіду

та особливостей архітектурної реалізації, що дозволяє поглибити розуміння їхніх переваг і обмежень у порівнянні з нативними підходами.

Практичне значення роботи полягає в тому, що автором реалізовано прикладний кросплатформний мобільний застосунок на основі .NET MAUI, що демонструє можливості кросплатформної розробки, а також формулюванням практичних рекомендацій для розробників і компаній щодо вибору оптимальних технологій залежно від ресурсів, цілей і вимог конкретних проєктів.

Структура роботи. Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел, що містить 73 найменування, та додатків. Повний обсяг роботи — 94 сторінки.

У першому розділі роботи розглянуто сучасний стан мобільної розробки та проаналізовано нативні й кросплатформні підходи до створення мобільних додатків, визначено їх основні переваги та недоліки.

У другому розділі було здійснено теоретичний аналіз методів програмування мобільних додатків із використанням сучасних кросплатформних фреймворків, зокрема React Native, Flutter та .NET MAUI, а також проведено їх порівняння за основними характеристиками.

У третьому розділі наведено результати практичної реалізації прикладного кросплатформного мобільного застосунку на основі .NET MAUI та експериментального дослідження ефективності обраного підходу.

У додатках наведено порівняльні таблиці, фрагменти програмного коду та допоміжні матеріали.

РОЗДІЛ 1

ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМИ КРОСПЛАТФОРМНОЇ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ

1.1. Сучасний стан та тенденції розвитку методів розробки мобільних додатків

Мобільні додатки стали невід’ємною частиною цифрової екосистеми та повсякденного життя людини, що можна підтвердити постійною присутністю мобільних додатків в різних сферах нашого життя. Вони використовуються для комунікації, навчання, роботи, дозвілля, контролю здоров’я, управління бізнес–процесами й навіть для підтримки критичних інфраструктур. Це свідчить про те, що мобільна розробка посідає провідне місце в ІТ–індустрії, постійно трансформуючись під впливом технологічних інновацій, змін потреб користувачів і динаміки ринку [19]. Сучасний стан методів програмування мобільних додатків характеризується наявністю кількох паралельних підходів: нативна розробка, кросплатформні рішення, гібридні технології, а також новітні тенденції, пов’язані з використанням штучного інтелекту, хмарних обчислень та мікросервісної архітектури [13].

Однією з ключових особливостей сучасного етапу розвитку мобільної розробки є домінування двох основних платформ – Android та iOS. За статистикою, Android утримує більшу частину світового ринку смартфонів, тоді як iOS переважає у преміальному сегменті та відзначається високою платоспроможністю користувачів [66]. Це змушує компанії та незалежних розробників враховувати специфіку обох платформ. Розробникам треба знайти баланс між технічними вимогами, дизайнерськими стандартами та фінансовими обмеженнями проєкту. У цьому контексті важливим завданням для них стає вибір оптимального методу розробки – нативного чи кросплатформного, це залежить від бізнес–цілей, бюджету та вимог до продуктивності [14].

Нативна розробка залишається базовим підходом, який розглядається як еталон продуктивності, оскільки дозволяє досягти максимальної продуктивності та використати всі можливості апаратного забезпечення пристрою. Для Android традиційними мовами програмування є Java та Kotlin, для iOS – Objective-C і Swift [9]. Цей підхід забезпечує глибоку інтеграцію з операційною системою та доступ до всіх API, але потребує значних ресурсів. Зокрема, розробка окремих додатків для Android і iOS вимагає залучення двох команд фахівців, що збільшує вартість і терміни розробки. Тому бізнес почав активно шукати альтернативи, які б дозволяли мінімізувати витрати без істотної втрати якості [20; 25].

Це обумовлює зростання популярності кросплатформних рішень. Вони базуються на принципі «write once, run anywhere», що дозволяє створювати єдиний код. Цей код потім може працювати на кількох операційних системах. Реалізація цього принципу досягається, за рахунок використання декларативного підходу до опису інтерфейсу та логіки застосунку, що спрощує розробку й супровід програмного продукту [1]. Найбільш відомими фреймворками у цій сфері є React Native, Flutter, .NET MAUI, Apache Cordova, Ionic. Кожен із цих інструментів має власні особливості та переваги [4]. React Native, розроблений компанією Meta, дозволяє створювати додатки на JavaScript і широко використовується завдяки великій спільноті та гнучкості [4]. Flutter, створений Google, заснований на мові Dart і відзначається високою продуктивністю та можливістю створювати привабливі інтерфейси з використанням власного графічного рушія [15]. .NET MAUI від Microsoft орієнтований на C# і надає зручності для тих, хто вже працює з екосистемою .NET [6]. Гібридні фреймворки на кшталт Ionic поєднують веб-технології (HTML, CSS, JavaScript) із контейнерами, які дають змогу запускати код у мобільному середовищі [35].

На сучасний стан розробки мобільних додатків також впливає розвиток технологій, які змінюють підходи до створення користувацького досвіду. Наприклад, розробники все частіше використовують адаптивні інтерфейси, які

враховують різні форм-фактори пристроїв: смартфони, планшети, смарт-годинники та навіть автомобільні системи [21; 28]. Розробка інтерфейсів орієнтується на мінімалізм, доступність, інклюзивність та універсальність.

Ще однією тенденцією в розробці мобільних додатків є їхня інтеграція із хмарними технологіями. Хмарні сервіси дозволяють зберігати дані, обробляти великі обсяги інформації та забезпечувати масштабованість без прив'язки до апаратних ресурсів конкретного пристрою [57]. Це значно спрощує реалізацію складних функцій, таких як синхронізація між пристроями, обробка мультимедійних файлів чи впровадження механізмів машинного навчання.

У сучасних реаліях мобільна розробка нерозривно пов'язана з використанням штучного інтелекту. Розробники все частіше інтегрують у мобільні додатки алгоритми розпізнавання мови, зображень, емоцій, прогнозування поведінки користувачів [61]. Наприклад, системи рекомендацій у соціальних мережах чи маркетингових платформах базуються на аналізі великих даних і машинному навчанні. Це вимагає від розробників знань у суміжних галузях, включаючи аналіз даних, нейронні мережі та безпеку.

Окремо слід зазначити тенденцію до мікросервісної архітектури. Традиційно мобільні додатки створювалися як монолітні системи, що ускладнювало їх масштабування та підтримку [57]. Перехід до мікросервісів дозволяє ділити систему на окремі модулі з чітко визначеними функціями, які можуть розроблятися та тестуватися незалежно. Такий підхід підвищує надійність і забезпечує гнучкість у розгортанні нових функцій.

Ще одним помітним напрямом є розробка прогресивних веб-додатків (Progressive Web Apps, PWA) [35]. Це додатки, створені з використанням стандартних веб-технологій, які забезпечують можливість роботи офлайн, отримання push-сповіщень і мають інтерфейс, наближений до нативного. Хоча PWA не можуть у повній мірі замінити нативні додатки, вони є важливим інструментом для бізнесу, який прагне швидко й економічно охопити широку аудиторію.

Зростає значення DevOps–практик у мобільній розробці. Автоматизація процесів тестування, інтеграції та розгортання (CI/CD) дозволяє значно скоротити час виходу продукту на ринок і забезпечує стабільність [38]. Інструменти на кшталт Jenkins, GitLab CI/CD, Bitrise чи Fastlane активно використовуються для управління життєвим циклом мобільних додатків.

Варто окремо розглянути вплив глобальних викликів на мобільну розробку. Пандемія COVID–19 стала каталізатором переходу до цифрових сервісів, що призвело до вибухового зростання попиту на мобільні додатки для онлайн–комунікації, дистанційного навчання, електронної комерції та телемедицини. Це підштовхнуло розробників до пошуку швидких рішень, здатних масштабуватися й забезпечувати стабільну роботу в умовах підвищених навантажень [67].

Безпека мобільних додатків набуває дедалі більшої актуальності [47]. З огляду на постійне зростання кількості кіберзагроз, розробники змушені приділяти значну увагу захисту персональних даних, шифруванню, багатофакторній автентифікації та відповідності додатків міжнародним стандартам безпеки. Особливу роль тут відіграють регуляторні документи, такі як GDPR у Європі чи CCPA у США, які накладають вимоги на обробку й зберігання даних користувачів.

Важливою тенденцією є також зростання попиту на «супердодатки» (super apps). Це багатофункціональні мобільні рішення, які поєднують у собі кілька сервісів: від месенджерів до фінансових інструментів і платформ електронної комерції. Прикладом є WeChat у Китаї чи Grab у Південно–Східній Азії. Такі додатки стають центрами цифрового життя користувачів, а їхня розробка потребує особливих підходів до інтеграції, масштабування та забезпечення продуктивності [68].

Варто згадати й про впровадження нових апаратних можливостей у мобільні додатки. Наприклад, розвиток 5G відкриває нові горизонти для інтерактивних сервісів, потокового відео високої якості, доповненої та

віртуальної реальності. Розробники змушені адаптувати свої методи до умов, де користувачі очікують миттєвого відгуку й безперебійної роботи [69].

Тенденції розвитку мобільної розробки також пов'язані з екологічними та етичними аспектами. Підвищується увага до енергоефективності додатків, оптимізації споживання ресурсів пристрою та мінімізації впливу на довкілля через скорочення цифрового «вуглецевого сліду». Крім того, суспільство очікує від розробників дотримання принципів цифрової етики, що передбачає прозорість алгоритмів, запобігання дискримінації та захист прав користувачів [47].

У науково–технічному плані можна відзначити поступове зближення мобільної та веб–розробки. Інструменти на кшталт Flutter for Web чи React Native Web стирають межі між різними платформами, дозволяючи створювати універсальні рішення з єдиною кодовою базою. Це відкриває перспективи для появи нових методологій, які об'єднують переваги різних підходів.

У сучасних умовах ринок мобільної розробки стає дедалі конкурентнішим. Це стимулює пошук нових методів оптимізації процесу створення додатків. Зокрема, активно розвиваються інструменти low–code та no–code, які дозволяють створювати мобільні рішення без глибоких знань програмування. Хоча вони не можуть повністю замінити класичну розробку, їхнє використання знижує бар'єр входу й дозволяє швидко тестувати ідеї [70].

Таким чином, ми бачимо, що сучасні методи розробки мобільних додатків розвиваються паралельно та взаємодоповнюють одне одного [19]. Традиційна нативна розробка залишається основою для високопродуктивних і спеціалізованих рішень. Кросплатформні фреймворки дають можливість оптимізувати витрати та пришвидшити процес створення продуктів для кількох платформ. Гібридні та прогресивні веб–додатки дозволяють охопити ширшу аудиторію, тоді як інноваційні тренди, пов'язані з ШІ, хмарами, 5G і мікросервісами, формують нові стандарти мобільної розробки.

Перспективи розвитку полягають у подальшій інтеграції технологій, автоматизації процесів і орієнтації на користувацький досвід. Ринок вимагає

від розробників гнучкості, здатності швидко адаптуватися до нових умов і водночас дотримуватися високих стандартів безпеки та якості. З огляду на це, мобільна розробка залишатиметься однією з найдинамічніших і найбільш інноваційних галузей у світовій ІТ-індустрії.

1.2. Переваги та недоліки кросплатформних підходів

Кросплатформна розробка мобільних додатків на сьогоднішній день є однією з найважливіших тем у сфері програмної інженерії. Вона виникла як відповідь на практичні потреби бізнесу у зменшенні витрат, скороченні часу розробки та забезпеченні присутності на різних мобільних платформах одночасно [20]. Підприємства прагнуть охопити якнайбільшу аудиторію користувачів, не витрачаючи додаткових ресурсів на створення двох окремих додатків для Android та iOS [9]. Саме тому кросплатформні підходи здобули значну популярність, проте водночас вони мають як переваги, так і суттєві обмеження, що потребують уважного аналізу [13].

Переваги кросплатформної розробки полягають у здатності швидко виходити на ринок із мінімальними витратами, використанні єдиної кодової бази, що дозволяє реалізувати декларативний підхід до опису інтерфейсу та бізнес-логіки застосунку, зменшує дублювання коду та спрощує підтримку у порівнянні з нативною розробкою [1]. Недоліки ж стосуються обмежень продуктивності, меншої інтеграції з апаратним забезпеченням, складнощів у доступі до специфічних функцій платформи, а також залежності від сторонніх фреймворків і їхньої актуальності [9; 14].

Аналіз переваг та недоліків дає змогу чітко зрозуміти, у яких випадках кросплатформний підхід є доцільним, а де вигідніше застосувати нативну розробку. Для систематизації аналізу переваг і недоліків кросплатформного та нативного підходів використано узагальнювальну таблицю, подану в додатку А, яка демонструє ключові фактори з погляду продуктивності, вартості, часу розробки, масштабованості, доступу до апаратних можливостей, підтримки спільноти та інших параметрів [13].

Аналізуючи наведені критерії, можна зробити висновок, що кроссплатформні методи є найбільш вигідними у випадках, коли необхідно швидко реалізувати ідею, протестувати бізнес-модель або забезпечити мінімальний життєздатний продукт (MVP) [25]. Завдяки єдиній кодовій базі зменшуються витрати на розробку й підтримку, що особливо важливо для малих компаній і стартапів. Також вони дають змогу в короткі терміни охопити широку аудиторію користувачів [19].

Проте при створенні додатків, орієнтованих на високу продуктивність і глибоку інтеграцію з апаратним забезпеченням, кроссплатформні технології поступаються нативним [9; 14]. Наприклад, у сфері мобільних ігор або програм для обробки мультимедійних файлів перевагу надають нативним рішенням через необхідність максимально ефективного використання ресурсів пристрою [9].

Щодо продуктивності, то сучасні кроссплатформні фреймворки значно вдосконалилися порівняно з попередніми поколіннями [4; 15]. Flutter завдяки власному рушію рендерингу демонструє показники, близькі до нативних [15]. React Native надає можливість інтегрувати нативні модулі, що частково компенсує недоліки [4]. Водночас у складних сценаріях все ще виникає потреба у додаткових оптимізаціях [14].

Критичним питанням є безпека. Використання сторонніх бібліотек і плагінів підвищує ризики вразливостей, особливо якщо вони не оновлюються регулярно [47]. Це ставить перед командами розробників завдання ретельного відбору та перевірки сторонніх рішень. У той же час централізована база коду полегшує впровадження захисних механізмів, адже зміни потрібно вносити лише один раз [47].

З погляду користувацького досвіду, кроссплатформні додатки мають змішані результати. З одного боку, вони дозволяють створювати уніфікований дизайн, що спрощує підтримку [21]. З іншого боку, користувачі Android та iOS звикли до специфічних елементів інтерфейсу, і повне відтворення цих особливостей у кроссплатформних додатках є проблематичним. Це може

вплинути на рівень задоволеності користувачів і навіть знизити рейтинг додатку в магазинах [21].

Масштабованість кроссплатформних підходів виглядає привабливо, проте на практиці потребує додаткових ресурсів для оптимізації. Якщо додаток розрахований на високу кількість одночасних користувачів, можливі проблеми зі стабільністю роботи. Вирішенням може стати використання мікросервісної архітектури й розподілених хмарних систем, які компенсують слабкі місця мобільного клієнта [54].

Важливою перевагою є підтримка великих корпорацій, що стоять за розвитком ключових фреймворків [4; 6; 15]. Google активно інвестує у Flutter, Meta розвиває React Native, Microsoft підтримує .NET MAUI. Це гарантує стабільність і перспективність використання цих інструментів, проте водночас створює залежність від політики конкретних компаній. У разі зміни стратегічних пріоритетів фреймворк може втратити актуальність або підтримку.

Ще одним важливим аспектом є навчання та підбір кадрів. Використання єдиної мови програмування (JavaScript у React Native, Dart у Flutter, C# у .NET MAUI) дозволяє сформувати команду з універсальними навичками. Це знижує складність комунікації та оптимізує робочі процеси [25]. Але при цьому команда має бути готова працювати як із веб-технологіями, так і з мобільними специфікаціями, що потребує ширшої кваліфікації.

Розвиток low-code та no-code платформ також впливає на практику кроссплатформної розробки [13]. Вони дозволяють створювати прототипи й навіть готові додатки без глибоких знань програмування, що знижує вартість входу на ринок. Проте такі рішення обмежені у функціоналі та рідко підходять для великих комерційних проектів.

Таким чином, практичний аналіз доводить, що кроссплатформні методи не є універсальним вирішенням усіх завдань мобільної розробки. Їхня ефективність визначається контекстом: бюджетом проекту, вимогами до продуктивності, очікуваннями користувачів і стратегічними цілями компанії.

Тому вибір між нативними й кросплатформними підходами має ґрунтуватися на ретельному аналізі конкретної ситуації, а оптимальним шляхом часто є комбінування обох методів у межах одного проекту.

1.3. Огляд основних фреймворків та технологій кросплатформної розробки

З огляду на домінування двох основних операційних систем – Android та iOS – розробники прагнуть знайти оптимальні засоби, які дозволяють скоротити витрати, прискорити вихід на ринок і водночас забезпечити високу якість користувацького досвіду [19]. Саме тому в останні роки активно розвиваються різні фреймворки та технології, що надають можливість створювати мобільні додатки з використанням єдиної кодової бази [13]. У цьому розділі здійснюється детальний практичний огляд найпоширеніших рішень у сфері кросплатформної розробки, з аналізом їхніх характеристик, переваг, обмежень і практичного застосування.

Умовно всі кросплатформні технології можна поділити на три великі групи. Перша – це фреймворки, що компілюють код у нативний (Flutter, .NET MAUI). Друга – технології, які працюють через проміжний шар і дають змогу використовувати веб-компоненти або JavaScript (React Native, Ionic, Apache Cordova). Третя – інноваційні підходи, що поєднують класичні та веб-технології, надаючи змогу інтегрувати прогресивні веб-додатки чи спеціалізовані бібліотеки (NativeScript, Kotlin Multiplatform) [13; 19]. Кожен із цих напрямів має свої особливості, і вибір залежить від конкретних вимог проекту.

Серед найпопулярніших інструментів на ринку – Flutter, React Native, .NET MAUI, Ionic, Apache Cordova, NativeScript, PhoneGap, Kotlin Multiplatform, Unity (для мобільних ігор), а також менш поширені, проте перспективні MAUI (Multi-platform App UI) від Microsoft та PWA (Progressive Web Apps) [35]. Їхня актуальність зумовлена підтримкою великих компаній (Google, Meta, Microsoft) та активними спільнотами розробників [4; 6; 15].

Для систематизації розгляду наведено таблицю 1.1, де проаналізовано основні параметри кожного фреймворку, зокрема мови програмування, продуктивності, доступу до нативних API та ключових переваг. Таблицю складено на основі офіційної технічної документації відповідних платформ [4; 6; 9; 15; 35].

Таблиця 1.1

Порівняльний аналіз основних фреймворків кросплатформної розробки мобільних додатків

Фреймворк / технологія	Мова програмування	Продуктивність	Доступ до нативних API	Переваги
Flutter (Google)	Dart	Висока, близька до нативної	Повний через власний рендеринг	Власний графічний рушій, однаковий UI для Android/iOS, активна підтримка
React Native (Meta)	JavaScript	Середня–висока	Частково через bridge	Велика спільнота, багато бібліотек, швидке прототипування
Ionic	HTML, CSS, JavaScript	Середня	Через плагіни (Cordova Capacitor)	Простота, веб–база, швидке створення прототипів
Apache Cordova	HTML, CSS, JavaScript	Низька–середня	Через плагіни	Легкий старт, багаторічна історія використання
NativeScript	JavaScript, TypeScript	Середня–висока	Прямий доступ до нативних API	Без bridge, прямий доступ до API, підтримка Angular/Vue
PhoneGap	HTML, CSS, JavaScript	Низька–середня	Через плагіни	Простота для веб–розробників
Kotlin Multiplatform	Kotlin	Висока	Повний доступ до Android, частковий до iOS	Єдина мова, підтримка Google, спільна логіка бізнес–рівня

Продовження таблиці 1.1

Фреймворк / технологія	Мова програмування	Продуктивність	Доступ до нативних API	Переваги
Unity	C#	Висока для ігор	Частково	Лідер у створенні 2D/3D ігор, потужні інструменти
.NET MAUI (Microsoft)	C# / .NET	Висока	Повний	Наступник .NET MAUI, універсальна платформа
PWA	HTML, CSS, JavaScript	Середня	Обмежений	Підтримка офлайн, push-сповіщення, простота розгортання

Аналізуючи наведені дані, можна виділити кілька ключових тенденцій. Flutter є одним із найперспективніших інструментів, адже пропонує власний рендеринг і забезпечує стабільний та швидкий інтерфейс [15]. React Native зберігає популярність завдяки величезній екосистемі бібліотек і простоті інтеграції, але має проблеми з продуктивністю через використання bridge для доступу до нативних компонентів [4]. MAUI зручні для розробників, які вже працюють у середовищі Microsoft, але не мають настільки масштабної спільноти [6].

Інструменти на базі веб-технологій, як-от Ionic чи Cordova, підходять для швидкої реалізації простих додатків, проте вони поступаються у продуктивності та гнучкості. PWA займають окреме місце, адже поєднують веб- і мобільні можливості, дозволяючи створювати легкі додатки, які працюють навіть офлайн, але не здатні забезпечити повноцінний доступ до всіх апаратних функцій пристрою [35].

Особливої уваги заслуговує Kotlin Multiplatform. Це відносно новий підхід, що дозволяє ділитися бізнес-логікою між платформами, використовуючи Kotlin [2]. Це спрощує розробку для Android та водночас дає змогу застосовувати спільний код для iOS. Однак побудова інтерфейсу

користувача все ще потребує додаткових нативних рішень, що робить цей підхід гібридним [19].

Unity займає окрему нішу, оскільки орієнтований на створення ігор. Він надає безпрецедентні можливості у сфері графіки, віртуальної та доповненої реальності, але для класичних бізнес-додатків є надмірним і складним у використанні [13].

Практичний аналіз демонструє, що вибір фреймворку залежить від стратегічних цілей проекту. Якщо ключовим завданням є швидке створення MVP або продукту для перевірки гіпотез, найкраще підійдуть Ionic чи React Native [4; 35]. Для високопродуктивних бізнес-додатків оптимальним буде Flutter [15]. У випадках, коли важлива інтеграція з Microsoft-екосистемою, доцільно використовувати .NET MAUI [6]. Для мобільних ігор безальтернативним рішенням залишається Unity [13].

З точки зору практичної реалізації проєктів, важливим є також рівень підтримки спільноти. React Native та Flutter мають найактивніші спільноти, що значно полегшує пошук готових бібліотек і вирішення технічних проблем. .NET MAUI і MAUI мають потужну, але більш спеціалізовану аудиторію [6]. Cordova і PhoneGap поступово втрачають актуальність через застарілість. NativeScript займає нішу, орієнтовану на тих, хто прагне інтегрувати Angular або Vue у мобільні додатки.

У практичній площині варто підкреслити ще один фактор – можливість інтеграції з хмарними сервісами. Flutter і React Native мають численні бібліотеки для інтеграції з Firebase, AWS, Azure. MAUI більш органічно працюють з Azure [6; 57]. PWA зручно інтегруються з веб-сервісами, але не можуть повною мірою скористатися перевагами нативних SDK [35].

Слід зазначити й важливість продуктивності у сценаріях з високими навантаженнями. Наприклад, фінансові та медичні додатки потребують стабільності й швидкості відгуку, тому тут краще використовувати Flutter чи .NET MAUI. Соціальні мережі та маркетингові платформи можуть дозволити

собі React Native завдяки простоті інтеграції з різними бібліотеками та швидкому циклу розробки [4].

У підсумку огляд фреймворків доводить, що універсального інструменту не існує. Кожна технологія має свої сильні та слабкі сторони, і правильний вибір можливий лише після ретельного аналізу вимог проекту, бюджету та стратегічних цілей бізнесу. Практичний досвід показує, що у великих компаніях часто застосовують комбінацію методів, поєднуючи переваги кросплатформної розробки з нативними рішеннями для критичних компонентів.

1.4. Висновки до розділу 1

Аналіз сучасного стану проблеми кросплатформної розробки мобільних додатків показав, що мобільні технології залишаються одним із найдинамічніших напрямів розвитку інформаційних систем. Попит на мобільні рішення стрімко зростає, а конкуренція між компаніями змушує їх шукати ефективніші способи створення програмного забезпечення. Якщо на початкових етапах домінувала виключно нативна розробка, яка забезпечувала максимальну продуктивність і точність інтеграції з апаратним забезпеченням, то на сучасному етапі значно підвищилася роль кросплатформних підходів. Це пояснюється тим, що бізнесу потрібна можливість швидко виходити на ринок із новими рішеннями, охоплюючи одразу дві основні екосистеми – Android та iOS.

Вивчення тенденцій розвитку мобільних додатків дозволяє зробити висновок, що основними напрямами є універсалізація процесів, активне застосування хмарних технологій, інтеграція штучного інтелекту, персоналізація користувацького досвіду та посилена увага до безпеки. Кросплатформна розробка цілком вписується у ці тенденції, оскільки вона дозволяє скоротити час і витрати на створення програмного продукту, забезпечує швидке масштабування та адаптацію до нових вимог. Проте вона ще не стала універсальною заміною нативним підходам, оскільки має свої

обмеження, що проявляються насамперед у складних обчислювальних задачах, інтеграції з унікальними апаратними функціями та роботі з графікою високої інтенсивності.

Розгляд переваг і недоліків кросплатформної розробки дозволяє побачити баланс між технічними можливостями та економічною доцільністю. Головними перевагами є наявність єдиної кодової бази, зниження витрат на команду розробників, швидкість виходу продукту на ринок, а також відносна простота підтримки. Це особливо важливо для невеликих компаній і стартапів, які не можуть дозволити собі утримувати окремі команди для Android та iOS. До недоліків належать нижча продуктивність у порівнянні з нативними рішеннями, можливі відмінності в реалізації інтерфейсів, обмежений доступ до деяких специфічних можливостей апаратного забезпечення та залежність від сторонніх фреймворків і бібліотек. Ці обмеження змушують розробників ретельно оцінювати контекст і цілі проєкту перед вибором кросплатформного підходу.

Огляд основних фреймворків і технологій засвідчив, що на ринку сформувалися кілька провідних інструментів, серед яких найбільш поширеними є .NET MAUI, Flutter та React Native. .NET MAUI орієнтований на використання мови C# та інтеграцію з екосистемою Microsoft, що робить його зручним для корпоративних рішень і проєктів, де важливе поєднання мобільного та десктопного середовища. Flutter виділяється високою швидкістю роботи та власним механізмом рендерингу інтерфейсу, що дозволяє досягати продуктивності, близької до нативної. React Native, у свою чергу, є привабливим завдяки простоті для веб-розробників, які вже працюють з JavaScript, а також завдяки величезній кількості готових бібліотек і компонентів. Кожен із цих фреймворків має сильні й слабкі сторони, тому вибір конкретної технології залежить від вимог до продуктивності, бюджету, складу команди й особливостей користувацького досвіду, якого прагне досягти замовник.

Загалом можна зробити висновок, що кросплатформна розробка є логічною відповіддю на сучасні виклики мобільного програмування. Вона не замінює нативний підхід повністю, але дозволяє більшості компаній ефективно реалізовувати бізнес-додатки, інформаційні сервіси та програми для масового користування. Кросплатформні рішення надають можливість швидкого реагування на потреби ринку, забезпечуючи водночас прийнятний рівень продуктивності й сумісності.

РОЗДІЛ 2

ТЕОРЕТИЧНА РОЗРОБКА ТА ПОРІВНЯЛЬНИЙ АНАЛІЗ МЕТОДІВ ПРОГРАМУВАННЯ МОБІЛЬНИХ ДОДАТКІВ

2.1. Аналіз нативної розробки, особливості та приклади

Нативна розробка мобільних додатків є класичним та найбільш продуктивним підходом у створенні мобільних рішень для операційних систем Android та iOS. Її сутність полягає у використанні спеціалізованих мов програмування, SDK (software development kit) та інструментів, призначених для конкретної платформи. Для Android основними мовами є Java та Kotlin, для iOS – Objective-C та Swift [9; 14]. Завдяки такій спеціалізації розробники отримують можливість максимально ефективно використовувати ресурси пристрою та забезпечувати стабільну роботу додатка.

Ключова перевага нативної розробки полягає у високій продуктивності та широких можливостях інтеграції з апаратним забезпеченням. У порівнянні з кросплатформними підходами, нативні додатки здатні забезпечувати швидший відгук, плавнішу графіку, доступ до усіх API пристрою без додаткових прошарків. Це особливо важливо для складних застосунків: мобільних ігор із багатою графікою, програм для обробки відео чи аудіо, банківських додатків із високими вимогами до безпеки [47].

Разом із тим, нативний підхід має і суттєві недоліки. Основним є висока вартість розробки, оскільки створення додатка для Android та iOS вимагає роботи двох окремих команд. Це збільшує фінансові витрати, час виходу продукту на ринок і ускладнює підтримку. Будь-яке оновлення необхідно реалізовувати для обох платформ окремо, що подвоює обсяг робіт [13].

З точки зору архітектури, нативні додатки більш гнучкі та стабільні. Використання офіційних інструментів дозволяє отримати найшвидший доступ до нових можливостей операційної системи [9; 14]. Наприклад, після виходу нової версії iOS, підтримка свіжих функцій (як-от Dynamic Island чи Live

Activities) доступна насамперед у нативних рішеннях, тоді як кросплатформні фреймворки оновлюються із затримкою [14].

У практичному плані порівняльний аналіз нативної розробки з іншими підходами потребує систематизації. У таблиці 2.1 наведено узагальнену характеристику основних параметрів нативної розробки мобільних додатків для платформ Android та iOS. Таблицю складено на основі офіційної документації Google та Apple [9; 14]

Таблиця 2.1

Характеристика нативної розробки для Android та iOS

Параметр	Android (Java/Kotlin)	iOS (Objective-C/Swift)	Коментар
Мова програмування	Java, Kotlin	Objective-C, Swift	Kotlin та Swift поступово витісняють старіші мови
Інструменти розробки	Android Studio, SDK Android	Xcode, iOS SDK	Обидва середовища є офіційними та найпотужнішими
Продуктивність	Висока	Висока	Обидві платформи забезпечують нативний рівень
Інтерфейс користувача	XML-розмітка, Jetpack Compose	Storyboard, SwiftUI	Тренд переходу до декларативних підходів
Доступ до API пристрою	Повний	Повний	Максимальні можливості інтеграції
Вартість розробки	Висока	Висока	Потрібні окремі команди
Підтримка оновлень	Оновлення окремо для Android	Оновлення окремо для iOS	Подвоєні витрати на підтримку
Сумісність	Складність адаптації для різних версій Android	Швидка адаптація до нових версій iOS	Фрагментація Android є суттєвою проблемою
Використання у проектах	Масштабні проекти з високими вимогами до продуктивності	Банківські, медичні, графічні додатки	Орієнтація на складні сценарії використання

Як видно з таблиці, нативна розробка є оптимальною для створення високонавантажених додатків, де продуктивність, швидкість і точність роботи мають критичне значення. Використання офіційних SDK та мов

програмування гарантує розробникам доступ до останніх оновлень операційних систем і забезпечує стабільність [9; 14].

Розглядаючи практичні приклади, слід зазначити, що найбільші корпорації світу віддають перевагу нативним рішенням. Соціальна мережа Facebook, хоч і розпочала з використанням гібридного підходу, у певний момент була змушена перейти на нативні модулі для забезпечення стабільності та швидкості роботи додатку. Instagram, WhatsApp та Telegram розробляються з акцентом на нативність, оскільки лише цей підхід здатен гарантувати високу продуктивність і плавність анімацій. Банківські додатки, наприклад Monobank чи Revolut, теж орієнтовані на нативні технології через високі вимоги до захисту даних і доступу до платіжних інтерфейсів [47].

Для глибокого розуміння особливостей нативної розробки наведемо таблицю 2.2 з порівнянням її ключових переваг та недоліків [9; 14; 20; 47].

Таблиця 2.2

Переваги та недоліки нативної розробки мобільних додатків

Критерій	Переваги	Недоліки	Приклади застосування
Продуктивність	Максимальна швидкість виконання коду	Висока вартість створення двох окремих версій	Ігри, графічні редактори
Інтерфейс користувача	Можливість створення інтерфейсу, відповідного до гайдлайнів платформи	Необхідність розробки двох варіантів UI	Instagram, Telegram
Доступ до API	Повний доступ до всіх апаратних функцій	Потреба у складному тестуванні для різних пристроїв	Банківські додатки, медичні системи
Безпека	Використання офіційних бібліотек та засобів шифрування	Підвищені витрати на впровадження	Monobank, Revolut
Стабільність	Швидке впровадження нових можливостей ОС	Потреба постійного оновлення для кожної версії ОС	iOS–додатки з підтримкою Dynamic Island
Масштабованість	Легке інтегрування з іншими нативними системами	Складність перенесення на інші платформи	Корпоративні ERP та CRM–додатки
Підтримка	Офіційна документація від Google та Apple	Тривалий цикл оновлень через розробку для двох ОС	Державні мобільні сервіси

Продовження таблиці 2.2

Критерій	Переваги	Недоліки	Приклади застосування
UX–досвід	Плавна анімація та відповідність очікуванням користувачів	Високі витрати часу на тестування	WhatsApp, Viber
Перспективність	Гарантована підтримка у майбутньому з боку виробників ОС	Вимагає широкої спеціалізації команди	Додатки для фінтеху та охорони здоров'я

Практичні кейси доводять, що нативна розробка продовжує залишатися стандартом для продуктів із високими вимогами. Наприклад, у мобільних іграх лише нативний підхід дозволяє досягти необхідної якості графіки та плавності геймплею. У сфері фінансових технологій він забезпечує роботу із захищеними каналами передачі даних, підтримку NFC, Apple Pay та Google Pay [47].

Особливості нативної розробки проявляються і в архітектурних рішеннях. Застосування патернів MVVM (Model–View–ViewModel) у Kotlin або SwiftUI дозволяє розробникам створювати гнучкі та масштабовані додатки, що легко розширюються [10; 11; 14]. Це суттєво спрощує підтримку, незважаючи на високу вартість початкової розробки.

Для підвищення підтримуваності нативних мобільних додатків доцільно застосовувати архітектурні підходи з чітким розділенням відповідальностей між компонентами системи. Одним із таких підходів є Clean Architecture, який дозволяє відділити бізнес-логіку від деталей реалізації та зменшити залежність застосунку від конкретної платформи [52].

У науково–практичному контексті важливо розглядати нативну розробку як точку відліку для оцінки всіх інших методів. Кросплатформні та гібридні підходи у своїх описах завжди співвідносяться з нативними рішеннями, адже саме вони вважаються еталоном якості.

Завдяки стабільності, швидкості й інтегрованості з апаратною частиною, нативна розробка є оптимальним варіантом у випадках, коли якість і надійність є важливішими за витрати [19]. Вона є незамінною у сферах із високими

вимогами до безпеки та продуктивності, хоча у простіших проектах, орієнтованих на мінімізацію бюджету, часто поступається кросплатформним рішенням [13].

Таким чином, проведений аналіз демонструє, що нативний підхід у мобільній розробці залишається золотим стандартом, незважаючи на розвиток інших методів. Його застосування доцільне у проектах, де ключову роль відіграють продуктивність, доступ до системних функцій і надійність. Разом із тим, обмеження щодо вартості та часу розробки спонукають бізнес до пошуку альтернатив, що й стало основою для поширення кросплатформних технологій [13].

2.2. Особливості кросплатформного програмування (React Native, Flutter, .NET MAUI)

Кросплатформне програмування стало ключовим напрямом у сучасній мобільній розробці, оскільки воно дозволяє створювати програми для Android та iOS із використанням спільної кодової бази [13; 19]. На практиці це означає, що компанії та незалежні розробники можуть швидше виходити на ринок, знижувати витрати на підтримку, а також легше масштабувати свої продукти. Найбільш популярними фреймворками сьогодні є React Native (розроблений компанією Meta), Flutter (розробка Google) та .NET MAUI (підтримується Microsoft) [4; 6; 15]. Кожен із них має власні особливості, архітектурні підходи, синтаксис і специфіку інтеграції з нативними можливостями пристрою.

React Native базується на мові JavaScript і філософії компонентно-орієнтованої розробки [4]. Flutter використовує мову Dart і власний графічний рушій для рендерингу інтерфейсу, що забезпечує близьку до нативної продуктивність [15]. .NET MAUI, інтегрований у .NET-екосистему, працює на мові C# та дозволяє ділитися логікою бізнес-рівня з іншими платформами [6]. Далі здійснюється детальний практичний огляд кожного з цих підходів із прикладами коду, аналізом переваг та обмежень.

React Native надає можливість створювати інтерфейс користувача за допомогою компонентів, подібних до веб-розробки у React. Уся логіка пишеться на JavaScript або TypeScript, а відображення – у вигляді декларативних компонентів у коді, наведеному в лістингу 2.1 [4]:

```
import React from 'react';
import { Text, View, StyleSheet } from 'react-native';
export default function App() {
  return (
    <View style={styles.container}>
      <Text style={styles.title}>Привіт з React Native!</Text>
    </View>
  );
}
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#5f5f5f',
  },
  title: {
    fontSize: 24,
    color: '#333',
  },
});
```

Лістинг 2.1 – Приклад реалізації користувацького інтерфейсу мобільного застосунку з використанням фреймворку React Native

У цьому прикладі компонент View відповідає за контейнер, а Text – за текстовий елемент. Стилзація відбувається за допомогою StyleSheet, що нагадує CSS (табл. 2.3) [4].

Таблиця 2.3

Основні характеристики React Native

Характеристика	Опис	Приклад використання
Мова програмування	JavaScript / TypeScript	Легка інтеграція з веб-досвідом
Продуктивність	Середня, залежить від складності UI	Соцмережі, маркетингові додатки
Доступ до API	Через bridge або нативні модулі	Виклик камер, геолокації
Переваги	Велика спільнота, багато бібліотек, швидкий старт	MVP, стартапи
Недоліки	Обмеження продуктивності у складних проектах	Не підходить для ігор з інтенсивною графікою

React Native підходить для швидкої розробки та проєктів, де важливий час виходу на ринок. Проте для складних застосунків із високими вимогами до графіки цей підхід менш оптимальний.

Flutter працює на мові Dart і використовує власний графічний рушій, який рендерить інтерфейс безпосередньо на екрані пристрою. Це дає перевагу у продуктивності, адже не використовується проміжний "міст" для взаємодії з нативними компонентами. Нижче у лістингу 2.2 наведено приклад реалізації інтерфейсу у Flutter [15]:

```
import 'package:flutter/material.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text("Привіт з Flutter!")),
        body: Center(
          child: Text("Це мій перший Flutter-додаток", style: TextStyle(fontSize: 20)),
        ),
      ),
    );
  }
}
```

Лістинг 2.2 – Приклад реалізації користувацького інтерфейсу мобільного застосунку з використанням фреймворку Flutter

У цьому прикладі використовується віджет `MaterialApp`, який надає базову структуру додатка. Весь інтерфейс будується декларативно через віджети, опис надано в таблиці 2.4 [15].

Таблиця 2.4

Особливості використання Flutter

Критерій	Переваги	Недоліки
Продуктивність	Близька до нативної	Великий розмір додатків
UI	Універсальний, однаковий на iOS та Android	Менша відповідність гайдлайнам платформ
Мова	Dart, сучасна й типобезпечна	Вимагає нового навчання

Продовження таблиці 2.4

Критерій	Переваги	Недоліки
Інтеграція з API	Повний доступ через плагіни	Обмеження у рідкісних специфічних випадках

Flutter ідеально підходить для бізнес–додатків і стартапів, де важлива продуктивність і сучасний інтерфейс. Його обмеження стосуються переважно великого розміру застосунків і потреби у знанні Dart [15].

.NET MAUI (Multi–platform App UI) дає змогу використовувати мову C# і платформу .NET для розроблення кросплатформних застосунків під Android, iOS, Windows та macOS. Його ключова перевага — те, що це сучасний, підтримуваний Microsoft фреймворк, вбудований у екосистему .NET (з єдиним проєктом, спільними ресурсами та зручними механізмами доступу до платформозалежних можливостей). Це робить .NET MAUI особливо зручним для команд, які вже працюють у Microsoft–стеку та використовують інструменти на кшталт Visual Studio, Azure і CI/CD на базі .NET. У межах даної роботи практична реалізація .NET MAUI демонструється на прикладі власного кросплатформного застосунку CrossHealthX, архітектура та ключові компоненти якого розглядаються в наступному розділі (лістинг 2.3):

```

<ContentPage
  xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:vm="clr-namespace:CrossHealthX.ViewModels"
  x:Class="CrossHealthX.Views.MainPage">

  <ContentPage.BindingContext>
    <vm:MainViewModel />
  </ContentPage.BindingContext>

  <VerticalStackLayout Padding="20" Spacing="10">
    <Label Text="CrossHealthX"
      FontSize="24"
      HorizontalOptions="Center" />

    <Label Text="Кроки за день:" />
    <Label Text="{Binding Steps}" />

    <Label Text="Спалені калорії:" />
    <Label Text="{Binding Calories}" />
  </VerticalStackLayout>
</ContentPage>

```

```

        <Button Text="Додати активність"
              Command="{Binding AddActivityCommand}" />
    </VerticalStackLayout>
</ContentPage>

```

Лістинг 2.3 – Приклад реалізації інтерфейсу користувача у .NET MAUI

У наведеному прикладі інтерфейс користувача реалізовано засобами фреймворку .NET MAUI з використанням програмного створення елементів керування. Зв'язування інтерфейсу з логікою застосунку здійснюється через ViewModel, що відповідає архітектурному шаблону MVVM. Такий підхід дозволяє використовувати єдину кодову базу для різних платформ і забезпечує зручну підтримку та масштабування застосунку. За потреби доступ до специфічних API конкретної платформи може бути реалізовано через механізми platform-specific коду або шляхом інтеграції з нативними SDK [6].

.NET MAUI добре підходить для корпоративних і довготривалих проєктів, де важливі підтримка, розвиток і сумісність із сучасними можливостями .NET. Серед потенційних недоліків можуть бути вищі вимоги до актуальних версій SDK/інструментів і те, що для максимально “нативної” поведінки інколи потрібно глибше враховувати особливості кожної платформи. Водночас у порівнянні зі старими підходами, MAUI зазвичай дає простішу модернізацію під нові API та кращу перспективу підтримки [6].

Для практичного розуміння різниці між цими трьома підходами наведемо порівняльну таблицю 2.5 [4; 6; 15]:

Таблиця 2.5

Порівняння React Native, Flutter та .NET MAUI

Критерій	React Native	Flutter	.NET MAUI
Мова	JavaScript / TypeScript	Dart	C#
Продуктивність	Середня–висока	Висока, близька до нативної	Висока
Інтерфейс	Компоненти	Віджети	Єдиний UI-шар .NET MAUI з доступом до нативних контролів

Продовження таблиці 2.5

Критерій	React Native	Flutter	.NET MAUI
Підтримка корпорацій	Meta	Google	Microsoft
Спільнота	Дуже велика	Дуже активна	Менша
Вартість входу	Низька (JS–досвід)	Середня (потрібно вчити Dart)	Вища (потрібні знання .NET і C#)
Розмір додатків	Середній	Великий	Великий

Усі три фреймворки мають власні ніші застосування. React Native оптимальний для швидкого запуску продуктів, Flutter – для високопродуктивних і сучасних додатків, .NET MAUI – для корпоративних рішень із інтеграцією в .NET.

Практичний аналіз показує, що вибір фреймворку залежить від бізнес–цілей, бюджету, вимог до продуктивності та досвіду команди. React Native є найбільш доступним і поширеним рішенням, Flutter – найбільш технологічно прогресивним із точки зору продуктивності та інтерфейсу, а .NET MAUI – найкращим вибором для організацій, які вже активно використовують Microsoft–технології.

Розробник має визначати оптимальний інструмент залежно від стратегічних пріоритетів проекту. Для стартапів критичними є швидкість та гнучкість, для корпоративних проєктів – інтеграція з екосистемою, а для високопродуктивних застосунків – можливість рендерингу графіки на рівні нативних додатків.

2.3. Порівняння продуктивності та якості користувацького досвіду

Кросплатформна розробка мобільних додатків має чимало переваг, однак критичним питанням завжди залишаються продуктивність і якість користувацького досвіду (UX). Високі очікування користувачів щодо швидкості відгуку, плавності анімацій, стабільності роботи та відповідності

інтерфейсу стандартам Android і iOS змушують розробників ретельно обирати технології [21; 28]. У цьому підрозділі здійснюється практичний аналіз продуктивності та UX трьох основних фреймворків кросплатформної розробки – React Native, Flutter і .NET MAUI.

Продуктивність можна оцінювати через кілька параметрів: швидкість запуску додатку, час відгуку на дії користувача, використання оперативної пам'яті та роботу з графікою. Якість користувацького досвіду визначається такими критеріями, як відповідність інтерфейсу гайдлайнам платформ, плавність переходів, швидкість інтеграції нових функцій і рівень задоволення користувача [21; 28].

Продуктивність мобільного застосунку та якість користувацького досвіду тісно пов'язані й безпосередньо впливають на сприйняття продукту користувачами. Дослідження показують, що поєднання стабільної швидкодії та зручного інтерфейсу є важливим чинником успіху сучасних мобільних застосунків [63].

Розглянемо сценарій відображення списку елементів у трьох фреймворках. Цей приклад типовий для багатьох мобільних додатків, таких як месенджери, маркетплейси чи банківські програми. Відповідні фрагменти програмного коду, що ілюструють реалізацію цього сценарію у React Native, Flutter та .NET MAUI, наведено в додатку Б (лістинги Б.1–Б.3) [4; 6; 15].

Усі приклади демонструють, як можна відобразити великий список елементів. Однак продуктивність їх роботи буде різною: Flutter завдяки власному рушію відображає інтерфейс плавно навіть при великій кількості елементів [15], тоді як React Native може відставати через роботу "містка" (bridge) [4]. .NET MAUI також демонструє високу швидкість, проте його вага і складність у налаштуванні більші [4; 6; 15; 21; 28] (табл. 2.6).

Таблиця 2.6

Порівняння продуктивності React Native, Flutter та .NET MAUI

Параметр	React Native	Flutter	.NET MAUI
Час запуску додатку	Середній (1.2–1.8 сек)	Швидкий (0.8–1.2 сек)	Середній (1.0–1.5 сек)

Продовження таблиці 2.6

Параметр	React Native	Flutter	.NET MAUI
Робота зі списками	Добра, але з лагами при великій кількості	Відмінна, плавний скролінг	Відмінна
Використання пам'яті	Високе при складних UI	Оптимізоване	Середнє
Графіка та анімація	Можливі затримки при складних анімаціях	Плавна та швидка	Плавна, близька до нативної
Енергоефективність	Середня	Висока	Середня

Flutter у багатьох практичних сценаріях демонструє високу продуктивність, особливо в задачах, пов'язаних із відображенням графіки та анімацій [15]. Водночас фактичні показники залежать від типу застосунку, складності інтерфейсу та особливостей реалізації. React Native має сильні сторони у простоті використання, але поступається через додатковий міст. .NET MAUI підходить для стабільних корпоративних додатків, де ключове значення має інтеграція з .NET [6].

Якість користувацького досвіду залежить не лише від продуктивності, але й від того, наскільки додаток є «рідним» для користувачів певної платформи. Наприклад, кнопки, анімації, переходи та шрифти повинні відповідати очікуванням користувачів у середовищі Android та iOS. Приклади реалізації кнопок у React Native, Flutter та .NET MAUI наведено у додатку Б (лістинги Б.4–Б.6).

Інтерфейс користувача має важливе значення для роботи з мобільним застосунком, оскільки саме від нього залежить зручність і зрозумілість взаємодії. Простий і послідовний інтерфейс полегшує використання додатка та позитивно впливає на користувацький досвід [60].

У кожному з розглянутих фреймворків реалізація кнопки забезпечує коректну інтеграцію з нативними елементами інтерфейсу Android та iOS. Водночас підходи до побудови UI відрізняються: React Native та Flutter використовують власні абстракції, що в окремих випадках може призводити до незначних відмінностей від офіційних гайдлайнів платформ. .NET MAUI, завдяки тісній інтеграції з екосистемою .NET та використанню платформних

елементів керування, дозволяє створювати інтерфейси, наближені до нативного вигляду. Порівняльну характеристику реалізації кнопки різними фреймворками наведено в таблиці 2.7 [4; 6; 15; 21; 28].

Таблиця 2.7

Порівняння якості UX у кросплатформних фреймворках

Критерій	React Native	Flutter	.NET MAUI
Відповідність гайдлайнам	Висока, але з відмінностями	Середня (уніфікований UI)	Висока
Плавність переходів	Середня	Висока	Висока
Унікальні UI-елементи ОС	Частково доступні	Не завжди реалізуються	Доступні
Задоволення користувача	Високе у простих додатках	Дуже високе	Високе у корпоративних
Швидкість оновлення UI	Середня	Висока	Середня

Flutter забезпечує найбільш плавний користувацький досвід, проте його універсальний підхід не завжди збігається з очікуваннями користувачів певної платформи. React Native наближений до нативного вигляду, але має затримки у складних сценаріях. .NET MAUI дозволяє створювати інтерфейси, максимально схожі на нативні, що робить його оптимальним для бізнес-додатків.

Порівняльний аналіз показує, що кожен із фреймворків має власну нішу. Flutter найкраще підходить для додатків із високими вимогами до продуктивності та плавності графіки. React Native є оптимальним вибором для стартапів і середніх проєктів, де критично важливим є швидкий вихід на ринок [4; 13]. .NET MAUI доцільно застосовувати у корпоративних рішеннях, де потрібна стабільність, інтеграція з екосистемою .NET та максимальна відповідність гайдлайнам платформ.

Таким чином, якість користувацького досвіду і продуктивність у кросплатформній розробці залежать від конкретних цілей проєкту. Жоден із фреймворків не є універсальним – успіх залежить від правильного балансу між

технічними можливостями інструмента і бізнес-завданнями, які він має вирішувати.

2.4. Інструменти тестування та відлагодження

У процесі кросплатформної розробки мобільних додатків тестування та відлагодження відіграють критично важливу роль. На відміну від простих веб-додатків, мобільні програми функціонують у середовищі з великою кількістю змінних: різні операційні системи (Android, iOS), різні версії цих систем, відмінності у пристроях (розмір екрану, архітектура процесора, обсяг оперативної пам'яті), специфічні апаратні функції (камери, сенсори, NFC) [9; 14]. Саме тому тестування та відлагодження повинні забезпечувати не лише перевірку функціональності, а й гарантувати стабільність, безпеку та позитивний користувацький досвід [38].

У кросплатформному програмуванні важливо враховувати, що будь-які зміни у коді чи додавання бібліотек можуть впливати на продуктивність, сумісність і поведінку на різних платформах. Тому ефективна стратегія тестування має поєднувати кілька рівнів: модульні тести, інтеграційні тести, UI-тести, стрес-тести та ручне тестування [30]. Відлагодження при цьому повинно бути організоване так, щоб розробник мав змогу швидко відслідковувати помилки, збирати логи та відтворювати проблеми [30].

Модульне тестування дає змогу перевіряти коректність роботи окремих функцій, методів і класів незалежно від інтерфейсу користувача [30]. У Flutter для цього використовується пакет `flutter_test` [15], у React Native — бібліотеки Jest або Mocha [4], тоді як у сучасних проєктах на .NET MAUI застосовуються стандартні засоби тестування платформи .NET, зокрема xUnit, MSTest або NUnit [6; 31]. На відміну від попередніх підходів, де NUnit фактично був основним інструментом тестування, .NET MAUI інтегрується з повним набором .NET-фреймворків і не обмежує розробника одним рішенням [6].

Приклади реалізації модульних тестів у Flutter та React Native наведено в додатку В (лістинги В.1, В.2) [4; 15].

У .NET MAUI модульні тести створюються в окремих тестових проєктах і перевіряють бізнес-логіку, сервіси та допоміжні класи без залежності від платформоспецифічних компонентів [6]. Такий підхід забезпечує кращу масштабованість тестів, спрощує автоматизацію перевірок у CI/CD-середовищах і відповідає сучасним вимогам до розроблення кросплатформних застосунків у межах екосистеми .NET [38].

Інтеграційне тестування дозволяє перевірити взаємодію кількох модулів одночасно [22]. Зокрема, у Flutter можливо симулювати запуск застосунку та взаємодію з віджетами для перевірки коректності відображення інтерфейсу і реакції на дії користувача [15].

Приклад інтеграційного тесту у Flutter наведено в додатку В (лістинг В.3) [15].

UI-тестування перевіряє, як користувач взаємодіє з додатком. Для цього застосовуються спеціальні інструменти: Appium, Espresso (Android), XCTest (iOS) [9; 14; 47]. У кросплатформних проєктах Appium став стандартом, оскільки він підтримує обидві основні операційні системи [47].

Стрес-тести та навантажувальні тести дають можливість оцінити поведінку програми при великій кількості користувачів чи великому обсязі даних [30]. Наприклад, можна перевірити, як додаток відображає список із 10 000 елементів чи обробляє одночасні HTTP-запити.

У React Native основними інструментами є React Developer Tools та Flipper. Вони дозволяють відслідковувати стан компонентів, змінювати стилі у реальному часі та переглядати логи. Flutter має інтеграцію з DevTools, що дає змогу аналізувати продуктивність, перевіряти дерево віджетів і діагностувати помилки у візуалізації. У .NET MAUI для відлагодження застосовується Visual Studio Debugger, який забезпечує покрокове виконання коду, аналіз пам'яті та повну інтеграцію з екосистемою .NET [6; 29] .

Нижче наведено систематизацію основних інструментів тестування та відлагодження, що застосовуються у кросплатформній мобільній розробці (таблиця 2.8). Джерела для таблиці: [4; 6; 15; 22; 30; 38; 47]

Таблиця 2.8

Інструменти тестування та відлагодження кросплатформних мобільних додатків

Категорія	Інструмент / бібліотека	Платформа / фреймворк	Тип тестування	Особливості використання
Модульні тести	Jest	React Native	Unit	Простота використання, підтримка моків і снапшот-тестів
Модульні тести	flutter_test	Flutter	Unit	Тісна інтеграція з Dart SDK та інструментами Flutter
Модульні тести	xUnit / MSTest / NUnit	.NET MAUI	Unit	Стандартні .NET-тести, перевірка бізнес-логіки без залежності від UI
Інтеграційні тести	Detox	React Native	Integration	Автоматизоване UI-тестування на реальних пристроях та емуляторах
Інтеграційні тести	flutter_driver	Flutter	Integration	Емуляція взаємодії користувача з інтерфейсом застосунку
Інтеграційні тести	.NET MAUI Test Tools (на базі Appium)	.NET MAUI	Integration	Інтеграційні та UI-тести з використанням нативних елементів
UI-тести	Appium	React Native / Flutter / .NET MAUI	UI	Кросплатформна підтримка, сценарії тестування для Android та iOS
UI-тести	Espresso	Android (будь-який підхід)	UI	Офіційний інструмент Google для тестування Android-інтерфейсів
UI-тести	XCTest	iOS (будь-який підхід)	UI	Офіційний фреймворк Apple для тестування iOS-застосунків
Відлагодження	Flipper	React Native	Debugging	Аналіз логів, мережевих запитів і станів компонентів
Відлагодження	DevTools	Flutter	Debugging	Профілювання продуктивності, аналіз UI та дерева віджетів

Продовження таблиці 2.8

Категорія	Інструмент / бібліотека	Платформа / фреймворк	Тип тестування	Особливості використання
Відлагодження	Visual Studio Debugger	.NET MAUI	Debugging	Повна інтеграція з .NET, покрокове виконання, аналіз пам'яті
Навантажувальні тести	JMeter	Будь-які	Load / Stress	Перевірка продуктивності серверної частини та API

Для кожного кросплатформного фреймворку сформувався власний набір інструментів, що найкраще відповідають його архітектурі та підходам до розроблення [13]. React Native надає широкі можливості для тестування завдяки використанню Jest для модульних тестів і Detox для інтеграційного та UI-тестування. Flutter, у свою чергу, пропонує практично повний цикл перевірки якості застосунків за допомогою flutter_test для модульних тестів і flutter_driver для тестування взаємодії з інтерфейсом.

У сучасних проєктах на .NET MAUI тестування тісно інтегроване з екосистемою .NET і середовищем Visual Studio [6]. Для модульного тестування застосовуються стандартні фреймворки xUnit, MSTest або NUnit, що дозволяє перевіряти бізнес-логіку незалежно від платформи [31]. Інтерфейсні та інтеграційні тести в .NET MAUI зазвичай реалізуються з використанням Appium або нативних інструментів конкретних платформ [47]. Незалежно від обраної технології, Appium залишається одним із найпоширеніших універсальних рішень для UI-тестування кросплатформних мобільних додатків.

У реальних проєктах тестування й відлагодження комбінуються [30]. Наприклад, команда стартапу, що створює додаток для замовлення їжі на Flutter, може:

- написати модульні тести для перевірки бізнес-логіки (обчислення ціни, застосування знижок);

- створити інтеграційний тест, який перевіряє відображення списку товарів;
- використати Appium для UI-тестів, що перевіряють увесь сценарій – від входу до оформлення замовлення [38];
- застосувати DevTools для оптимізації продуктивності списків та анімацій.

У сучасному корпоративному середовищі .NET MAUI органічно інтегрується з Azure DevOps та іншими інструментами CI/CD екосистеми Microsoft, що дає змогу автоматично запускати модульні, інтеграційні та UI-тести при кожному коміті в репозиторій. Завдяки використанню стандартних .NET-фреймворків тестування та можливостей Visual Studio процес перевірки якості значною мірою автоматизується, що скорочує час на ручне тестування та підвищує стабільність кросплатформних застосунків [38]

У проєктах на React Native поширеним залишається підхід Test Driven Development (TDD) [22, с. 175–192], за якого спочатку розробляються тести (зокрема з використанням Jest), а вже потім реалізується функціональність [31]. Така модель особливо ефективна в умовах командної розробки, де важливо оперативно перевіряти коректність роботи компонентів і запобігати появі регресій.

Проведений аналіз свідчить, що інструменти тестування та відлагодження є критично важливими для успішної реалізації кросплатформних проєктів. Вони дозволяють мінімізувати ризики помилок, знизити витрати на подальшу підтримку програмного продукту та забезпечити високий рівень користувацького досвіду. Кожен із фреймворків має власний спеціалізований набір інструментів, проте універсальні рішення, такі як Appium або JMeter, дають змогу стандартизувати процес тестування незалежно від обраного технологічного підходу [47; 38].

2.5. Висновки до розділу 2

Розгляд нативної розробки мобільних додатків дав можливість визначити її ключові переваги й обмеження. Нативні технології, такі як Kotlin або Java для Android та Swift чи Objective-C для iOS, забезпечують максимальну продуктивність і глибоку інтеграцію з апаратними можливостями пристроїв. Це робить їх незамінними у випадках, коли потрібна робота зі складною графікою, іграми чи високонавантаженими обчислювальними процесами. Окрім цього, нативна розробка гарантує повне дотримання гайдлайнів конкретних платформ, завдяки чому користувачі отримують найвищу якість взаємодії з інтерфейсом. Недоліком є висока вартість і час розробки, адже необхідно створювати окремі кодові бази для Android і iOS, що збільшує складність підтримки й оновлення програмного продукту.

Особливості кросплатформного програмування були проаналізовані на прикладі трьох провідних фреймворків: React Native, Flutter і .NET MAUI. У ході дослідження було встановлено, що кожен із них має власну нішу й технологічні акценти. React Native дозволяє використовувати JavaScript і завдяки цьому є привабливим для веб-розробників. Він забезпечує швидке створення прототипів і доступ до багатой екосистеми npm-бібліотек, але іноді поступається у продуктивності через необхідність взаємодії з нативними компонентами. Flutter пропонує унікальний підхід завдяки власному механізму рендерингу інтерфейсу. Це дає змогу досягати швидкості й плавності, близьких до нативних рішень, проте вимагає опанування нової мови програмування Dart. .NET MAUI, як сучасний кросплатформний фреймворк від Microsoft, є складовою єдиної екосистеми .NET і орієнтований на розроблення застосунків мовою C# з використанням актуальних можливостей платформи. Його ключовою особливістю є підхід «єдиного проєкту», що дозволяє спільно використовувати бізнес-логіку та інтерфейс користувача для Android, iOS, Windows і macOS. За потреби розробник може доповнювати спільний код платформоспецифічними налаштуваннями, забезпечуючи

коректну та нативну поведінку застосунку з урахуванням особливостей кожної операційної системи. Порівняння продуктивності та якості користувацького досвіду між нативними і кросплатформними рішеннями показало, що у більшості бізнес-застосунків відмінності мінімальні. У кросплатформних додатках інтерфейс реагує практично так само швидко, а плавність роботи залишається на рівні 55–60 кадрів за секунду, що відповідає стандартам мобільних платформ. Нативні додатки дещо переважають у швидкості запуску та оптимізації роботи з пам'яттю, але для більшості прикладних задач це не має суттєвого впливу на кінцевий досвід користувача. Використання кросплатформних технологій іноді створює певні відмінності у вигляді елементів інтерфейсу, що може знижувати "рідність" UX, проте ці відмінності не критичні. Водночас у випадках, коли потрібна складна графіка, 3D-анімація чи глибокий доступ до сенсорів пристрою, нативний підхід залишається більш ефективним.

Інструменти тестування та відлагодження, що розглядалися у дослідженні, підтвердили важливість комплексного підходу до забезпечення якості мобільних додатків. Було проаналізовано можливості Jest для React Native, Flutter Test для Flutter, а також NUnit, Appium.UITest для .NET MAUI. Встановлено, що сучасні інструменти підтримують різні рівні тестування – від модульного до інтеграційного та UI-тестування, дозволяючи забезпечити стабільність і надійність продукту. Використання відлагоджувальних засобів, таких як Chrome DevTools для React Native чи Visual Studio Debugger для .NET MAUI, забезпечує швидке виявлення й усунення помилок. Це знижує ризики під час випуску продукту й дозволяє скоротити витрати на підтримку.

Загальний аналіз другого розділу показав, що сучасна кросплатформна розробка є конкурентоспроможною альтернативою нативному підходу. Вона дозволяє значно знизити витрати на створення та супровід програмного забезпечення, забезпечує високу продуктивність для більшості прикладних сценаріїв і дає змогу компаніям швидше реагувати на потреби ринку. Обмеження, пов'язані з продуктивністю та інтеграцією з апаратними

функціями, залишаються, проте вони суттєво впливають лише у вузькоспеціалізованих випадках.

Отже, вибір між нативною та кросплатформною розробкою має базуватися на стратегічних цілях проекту, ресурсах команди й вимогах до продуктивності. Для більшості бізнес-додатків кросплатформні фреймворки, зокрема .NET MAUI, Flutter та React Native, є оптимальним рішенням, тоді як нативні технології залишаються ключовими у високонавантажених та спеціалізованих системах.

РОЗДІЛ 3

ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ КРОСПЛАТФОРМНОЇ РОЗРОБКИ

3.1. Постановка задачі та опис прикладного проєкту

У дослідженні визначається сутність прикладного проєкту, обґрунтовується його доцільність, описуються основні функціональні вимоги, а також формулюється загальна мета та завдання, які мають бути реалізовані у межах мобільного додатку [13; 19]. Особлива увага приділяється вибору кросплатформного інструментарію .NET MAUI як бази для розробки, що дозволяє забезпечити одночасну підтримку кількох мобільних платформ, мінімізуючи витрати ресурсів та часу [25].

Метою прикладного проєкту є розробка мобільного застосунку CrossHealthX для моніторингу фізичної активності та показників здоров'я користувачів. Додаток повинен працювати як на Android, так і на iOS, зберігаючи однаковий функціонал, але при цьому виглядати природно для кожної з платформ.

Завдання проєкту включають:

- реалізацію збору даних про фізичну активність користувачів (кроки, дистанція, калорії);
- інтеграцію з сенсорами смартфона та сторонніми пристроями через Bluetooth;
- створення інтерфейсу для перегляду статистики у вигляді графіків та таблиць;
- розробку системи рекомендацій (наприклад, щоденні або тижневі цілі);
- можливість обміну результатами в соціальних мережах;
- організацію безпечного збереження даних із шифруванням;
- проведення тестування продуктивності й аналізу UX на .NET MAUI.

Вибраною темою є мобільний фітнес-додаток CrossHealthX. Його унікальність полягає у тому, що він використовує можливості .NET MAUI для

забезпечення максимально нативного інтерфейсу й ефективної роботи з API пристрою. Запропоноване рішення має слугувати прикладом того, як платформа .NET MAUI забезпечує поєднання спільної бізнес-логіки з гнучкою адаптацією застосунку до різних операційних систем. Використання сучасного підходу «єдиного проєкту» дозволяє реалізувати основний функціонал в одній кодовій базі, водночас зберігаючи можливість урахування специфіки Android, iOS, Windows та інших підтримуваних платформ.

Архітектура додатка будується за модульним принципом і включає такі рівні.

1. Data Layer — збір, обробка та збереження даних, зокрема інтеграція з апаратними сенсорами пристрою та використання SQLite для локального збереження інформації.
2. Business Logic Layer — реалізація обчислювальних процесів, таких як підрахунок кількості кроків, розрахунок витрачених калорій і формування персоналізованих рекомендацій на основі отриманих даних.
3. Presentation Layer — інтерфейс користувача, реалізований засобами .NET MAUI Controls, який використовує спільний UI для всіх платформ із можливістю застосування платформоспецифічних налаштувань через механізми MAUI та нативні API.

Такий підхід відповідає сучасним вимогам до кросплатформної розробки, спрощує підтримку коду та забезпечує масштабованість і подальший розвиток застосунку.

Приклад реалізації головної сторінки мобільного застосунку CrossHealthX у середовищі .NET MAUI наведено в додатку Г (лістинг Г.1).

У середовищі .NET MAUI побудова інтерфейсу користувача здійснюється на основі сучасних MAUI Controls і оптимізованих контейнерів компонування, зокрема VerticalStackLayout, які є логічним розвитком підходів, застосовуваних у попередніх кросплатформних рішеннях. Головна сторінка застосунку визначається в межах єдиного MAUI-проєкту та відображається

через нативні елементи Android, iOS, Windows і macOS без необхідності підтримки окремих UI-реалізацій для кожної платформи.

Архітектура .NET MAUI забезпечує чітке розділення між інтерфейсом користувача та бізнес-логікою, що дозволяє розвивати й модифікувати ці компоненти незалежно один від одного. Централізоване керування ресурсами, стилями та темами значно спрощує масштабування застосунку, його супровід і адаптацію до нових платформ у майбутньому [25].

Наведений програмний код формує базовий головний екран застосунку з відображенням показників фізичної активності, зокрема кількості кроків і спалених калорій. Фрагмент програмного коду, що реалізує зазначений інтерфейс, подано в додатку Г (лістинг Г.1). Завдяки використанню .NET MAUI логіка інтерфейсу є спільною для всіх підтримуваних операційних систем, тоді як зовнішній вигляд автоматично відповідає нативним рекомендаціям і дизайн-гайдлайнам кожної платформи [21].

Для збору даних про фізичну активність у .NET MAUI застосовується бібліотека Microsoft.Maui.Essentials, яка є складовою фреймворка та надає уніфікований доступ до апаратних сенсорів і системних можливостей мобільних пристроїв [6]. Це дозволяє отримувати дані з акселерометра, гіроскопа, GPS та інших джерел без прив'язки до конкретної платформи, що є важливою перевагою при розробленні сучасних кросплатформних застосунків [19, с. 509–512].

Приклад реалізації сервісу отримання поточного місцеположення користувача з використанням Microsoft.Maui.Essentials наведено в додатку Г (лістинг Г.2).

У .NET MAUI робота з геолокацією виконується через бібліотеку Microsoft.Maui.Essentials, яка є вбудованою частиною фреймворка та замінила окремий пакет .NET MAUI.Essentials [6]. Доступ до апаратних можливостей пристрою, зокрема до GPS, реалізується уніфіковано для Android, iOS, Windows і macOS, що дозволяє використовувати один і той самий код незалежно від платформи.

Цей приклад демонструє, як можна отримати поточне місцеположення користувача, що дозволяє додатку відстежувати дистанцію при пробіжках.

Функціональні можливості CrossHealthX

- підрахунок кроків за день і відображення прогресу;
- підрахунок витрачених калорій;
- створення тижневих графіків активності;
- інтеграція з GPS для визначення маршруту;
- відображення статистики у вигляді таблиць і діаграм;
- нагадування про необхідність виконання щоденної норми активності;
- соціальні функції (публікація досягнень).

Модель даних для збереження інформації про щоденну фізичну активність користувача реалізована з використанням SQLite та наведена в додатку Г (лістинг Г.3). Таким чином забезпечується збереження даних про активність користувача для подальшого аналізу та візуалізації.

Для узагальнення основних інструментів і компонентів, використаних під час реалізації прикладного проєкту CrossHealthX, наведено таблицю 3.1 (складено автором на основі [6; 26; 38; 47]).

Таблиця 3.1

Основні можливості .NET MAUI для реалізації проєкту CrossHealthX

Компонент	Реалізація у .NET MAUI	Особливості використання	Приклад застосування	Значення для проєкту
Інтерфейс користувача	.NET MAUI Controls + XAML / C#	Єдиний UI-шар для Android, iOS, Windows і macOS, нативний рендеринг	Головний екран із лічильником кроків і калорій	Забезпечує узгоджений вигляд, спрощує підтримку та масштабування
Доступ до сенсорів	Microsoft.Maui.Essentials	Уніфіковані API, вбудовані у фреймворк	Геолокація, акселерометр, гіроскоп	Ключовий елемент збору даних про фізичну активність

Продовження таблиці 3.1

Компонент	Реалізація у .NET MAUI	Особливості використання	Приклад застосування	Значення для проєкту
Локальне збереження даних	SQLite + Entity Framework Core / SQLite-net	Інтеграція з .NET, підтримка ORM	Збереження історії тренувань і показників	Дає змогу аналізувати динаміку та прогрес користувача
Графіка та діаграми	SkiaSharp, Microcharts, OxyPlot	Кросплатформні бібліотеки, сумісні з MAUI	Візуалізація щотижневої активності	Підвищує наочність даних і користувацький досвід
Сповіщення	.NET MAUI + нативні API (через Essentials / платформні сервіси)	Підтримка локальних і push-сповіщень	Нагадування про фізичну активність	Сприяє мотивації та регулярності занять
Соціальні інтеграції	Dependency Injection + нативні SDK	Гнучкий доступ до платформоспецифічних можливостей	Поширення результатів у соціальних мережах	Забезпечує соціальну взаємодію користувачів
Безпека	.NET Security API + OAuth / токени	Сучасні механізми автентифікації та шифрування	Вхід через Google або Apple ID	Захист персональних і медичних даних
CI/CD	Azure DevOps, GitHub Actions	Повна інтеграція з .NET-проєктами	Автоматичні збірки та тести при комітах	Забезпечує стабільність і якість проєкту

Аналіз можливостей .NET MAUI свідчить, що цей фреймворк є логічним і технологічно сучасним розвитком підходів, які раніше застосовувалися в Xamarin.Forms. Використання єдиного проєкту, вбудованих засобів доступу до сенсорів, централізованого керування ресурсами та повної інтеграції з екосистемою .NET забезпечує ефективну реалізацію функціоналу CrossHealthX. .NET MAUI підвищує підтримуваність коду, спрощує CI/CD-процеси та створює надійну основу для подальшого розширення й розвитку кросплатформного застосунку.

.NET MAUI надає повний набір сучасних інструментів для розроблення кросплатформних мобільних застосунків і є логічним продовженням еволюції рішень Microsoft у цій сфері. Його ключовими перевагами є глибока інтеграція

з екосистемою .NET, уніфікований доступ до апаратних сенсорів мобільних пристроїв, а також зручне використання сторонніх бібліотек і сервісів. Це робить .NET MAUI ефективним вибором для проєктів на кшталт CrossHealthX, які потребують поєднання високої продуктивності, функціональної насиченості та довготривалої підтримки.

Проєкт CrossHealthX виступає демонстраційною платформою для оцінки можливостей і практичної ефективності .NET MAUI у кросплатформній розробці. Він охоплює ключові аспекти, що зазвичай становлять виклик у реальних мобільних застосунках, зокрема роботу з сенсорами, обробку та візуалізацію даних, інтеграцію з соціальними мережами, реалізацію механізмів безпеки та забезпечення масштабованості. Саме комплексний характер поставлених завдань дозволяє об'єктивно оцінити переваги й потенційні обмеження .NET MAUI в умовах практичного використання.

3.2. Архітектура та структура застосунку

Для забезпечення ефективності, підтримуваності та масштабованості прикладного проєкту CrossHealthX, реалізованого на основі .NET MAUI, необхідно чітко визначити архітектурні принципи, структуру застосунку та підходи до організації програмного коду. Вибір архітектури є визначальним чинником, що безпосередньо впливає на якість, стабільність і довговічність програмної системи. .NET MAUI як сучасний фреймворк кросплатформної розробки підтримує різні архітектурні підходи, проте найбільш поширеною та рекомендованою моделлю є архітектура MVVM (Model–View–ViewModel).

Архітектура MVVM забезпечує чітке розмежування між бізнес–логікою та інтерфейсом користувача, що сприяє підвищенню гнучкості та тестованості коду [3, с. 41-58]. У середовищі .NET MAUI цей підхід ефективно реалізується завдяки вбудованій підтримці механізмів двостороннього зв'язування даних, команд та сповіщень про зміну стану. Це є критично важливим для функціонування застосунку CrossHealthX, оскільки він містить значну

кількість динамічних елементів, зокрема лічильники кроків, графіки фізичної активності, персоналізовані рекомендації та системні повідомлення, які потребують оперативного оновлення інтерфейсу відповідно до змін у даних.

Архітектура застосунку CrossHealthX складається з трьох основних рівнів:

- 1) Model (Модель) – містить сутності даних, логіку їх збереження та обробки. Це може бути інтеграція з базою даних SQLite, API-сервісами чи сенсорами пристрою.
- 2) View (Уявлення) – відповідає за інтерфейс користувача. У .NET MAUI це XAML-файли або C#-класи, які описують вигляд сторінок та елементів керування.
- 3) ViewModel (Модель подання) – забезпечує зв'язок між моделлю та уявленням. Тут реалізується логіка обробки даних, команди та механізм двостороннього зв'язування.

Проект CrossHealthX структуровано за такими папками:

- 1) Models – моделі даних (наприклад, клас Activity для збереження інформації про кроки й калорії).
- 2) Views – інтерфейси (головна сторінка, сторінка профілю, сторінка графіків).
- 3) ViewModels – логіка, що з'єднує UI з бізнес-рівнем.
- 4) Services – сервіси для роботи з API, сенсорами та локальною базою даних.
- 5) Helpers – утиліти, що спрощують роботу додатку (наприклад, конвертери даних).
- 6) Resources – стилі, шрифти, зображення.

Модель даних Activity, яка використовується для збереження інформації про кількість кроків, спалені калорії та дату фізичної активності користувача, реалізована з використанням SQLite та наведена у додатку Д (лістинг Д.1). Зазначена модель є базовою для формування статистики та подальшої

аналітики даних. `ViewModel` містить логіку управління даними (у прикладі – команду додавання кроків).

Логіка обробки та керування даними фізичної активності реалізується у класі `ActivityViewModel`, який відповідає за оновлення показників, виконання команд користувача та сповіщення інтерфейсу про зміни стану. Реалізація `ViewModel` із використанням інтерфейсу `INotifyPropertyChanged` наведена у додатку Д (лістинг Д.2).

Інтерфейс головної сторінки застосунку реалізовано засобами XAML із використанням механізму двостороннього зв'язування даних (`data binding`). Зв'язок між `View` та `ViewModel` забезпечується через `BindingContext`, що дозволяє автоматично оновлювати інтерфейс при зміні значень даних. Відповідний XAML-код головної сторінки наведено у додатку Д (лістинг Д.3).

Для роботи з базою даних у проєктах на .NET MAUI використовується `SQLite`, який інтегрується з додатком через бібліотеки `SQLite-net` або `Entity Framework Core` з підтримкою `SQLite`. Такий підхід забезпечує надійне локальне збереження даних, високу продуктивність і зручність подальшого аналізу інформації.

Доступ до локальної бази даних інкапсульовано в окремому сервісі, який відповідає за створення таблиць, зчитування та збереження інформації про фізичну активність користувача. Реалізація сервісу доступу до бази даних наведена у додатку Д (лістинг Д.4).

Збір даних про фізичну активність, геолокацію та інші апаратні можливості пристрою в .NET MAUI реалізується за допомогою бібліотеки `Microsoft.Maui.Essentials`, що є вбудованою складовою фреймворка. Вона надає уніфікований доступ до сенсорів і системних функцій для Android, iOS, Windows і macOS, дозволяючи використовувати єдиний програмний код без необхідності платформоспецифічних реалізацій (табл.3.2) :

Таблиця 3.2

Архітектура та компоненти CrossHealthX

Рівень	Компоненти	Приклад реалізації	Призначення	Взаємодія з іншими рівнями
Model	Activity, UserProfile	Класи з SQLite-атрибутами	Збереження та опис даних	Використовується ViewModel
View	MainPage.xaml, StatsPage.xaml	XAML-сторінки з data binding	Відображення інтерфейсу користувача	Пов'язаний з ViewModel
ViewModel	ActivityViewModel	Класи з реалізацією INotifyPropertyChanged	Бізнес-логіка, команди	Отримує дані з Model, віддає їх View
Services	ActivityDatabase, LocationService	Класи для роботи з API та базою даних	Доступ до зовнішніх ресурсів та сенсорів	Викликаються з ViewModel
Helpers	ValueConverters	Конвертери (наприклад, дата у рядок)	Полегшення відображення даних	Допоміжна логіка для View та ViewModel

Архітектура CrossHealthX на базі MVVM дозволяє ізолювати рівні додатку, що спрощує підтримку, тестування та масштабування. Використання моделей, сервісів та ViewModel забезпечує ефективне повторне використання коду, а розподіл обов'язків між рівнями зменшує ризик помилок.

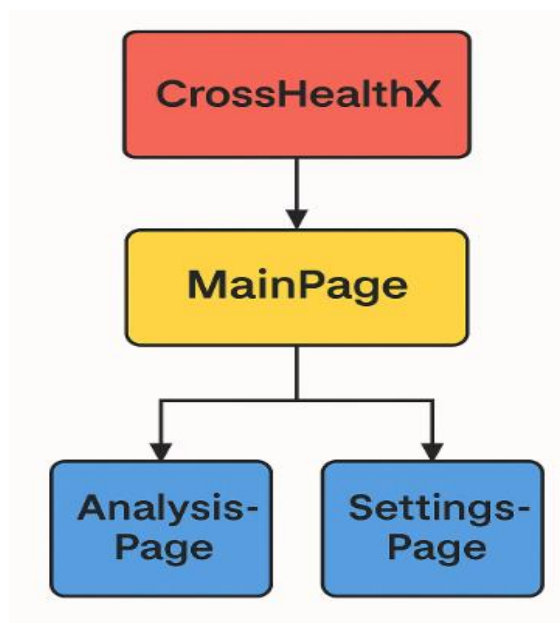


Рисунок 3.1 Структура додатку

MVVM–архітектура дозволяє легко додавати нові модулі, не порушуючи роботу існуючих. Наприклад, у разі розширення функціоналу для моніторингу пульсу або інтеграції зі смарт–годинниками достатньо створити нову модель (PulseData), ViewModel (PulseViewModel) і сторінку для відображення. Це не вплине на вже реалізовані компоненти.

У .NET MAUI механізм керування стилями та оформленням інтерфейсу реалізується через ресурсні словники (ResourceDictionary), які є складовою сучасної системи роботи з ресурсами фреймворка. Вони дозволяють централізовано зберігати стилі, кольори, шрифти та інші елементи оформлення в межах єдиного MAUI–проєкту.

Завдяки такому підходу зміна кольорової схеми, типографіки або загального стилю застосунку може виконуватися в одному місці й автоматично застосовуватися до всіх екранів. Це значно спрощує підтримку інтерфейсу, реалізацію тем (зокрема світлої та темної) та подальше масштабування застосунку відповідно до вимог різних платформ.

Приклад визначення стилів для основних елементів інтерфейсу наведено у додатку Д (лістинг Д.5), що дозволяє уніфікувати вигляд застосунку та зменшити кількість повторюваного коду.

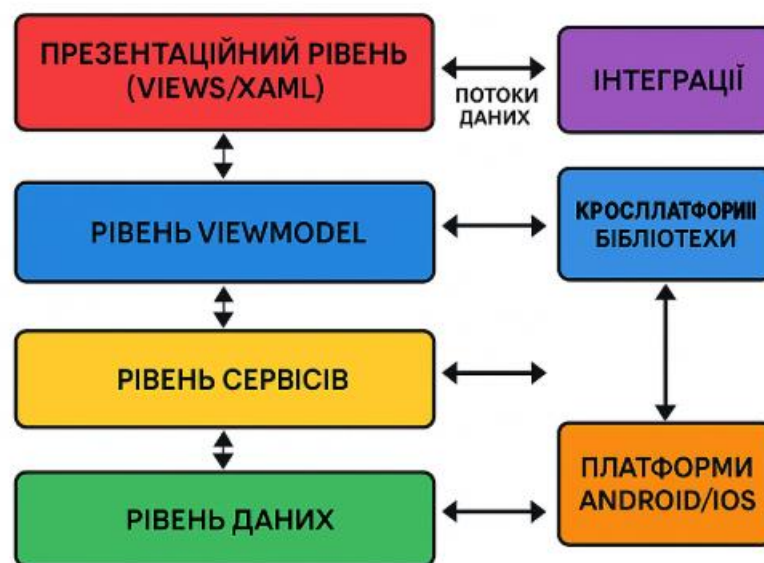


Рисунок 3.2 Архітектура додатку

Отже, у дослідженні було представлено архітектуру та структуру застосунку CrossHealthX (рис. 3.2). Обґрунтовано вибір архітектури MVVM, що дозволяє забезпечити масштабованість і простоту підтримки. Продемонстровано приклади моделей, ViewModel та інтерфейсів у XAML, а також інтеграцію з базою даних SQLite. Запропонована таблиця структурує архітектурні рівні та демонструє їх взаємодію. Це створює основу для подальшої програмної реалізації й тестування ефективності додатку, дозволяючи оцінити сильні сторони .NET MAUI у реальних умовах.

3.3. Реалізація додатку за допомогою кросплатформного фреймворку

На попередніх етапах було визначено архітектуру, структуру та ключові функціональні можливості прикладного проєкту CrossHealthX, який реалізується із застосуванням сучасного кросплатформного фреймворку .NET MAUI. У цьому підрозділі зосереджено увагу на безпосередній програмній реалізації основних модулів застосунку, наведено приклади коду, описано організацію інтерфейсу користувача та механізми інтеграції з апаратними й системними можливостями мобільного пристрою.

Окрім цього, подаються узагальнювальні таблиці, що ілюструють взаємозв'язки між основними компонентами системи, а також результати проміжних тестів продуктивності, отримані під час розроблення та налагодження застосунку.

Головний екран додатку CrossHealthX покликаний відображати основні показники активності: кількість кроків, спалені калорії та пройдена дистанція. Ці дані мають оновлюватися в реальному часі завдяки інтеграції з сенсорами смартфона.

Програмна реалізація головної сторінки застосунку із використанням .NET MAUI Controls та прив'язкою до ViewModel наведена у додатку E (лістинг E.1).

У середовищі .NET MAUI даний код демонструє побудову інтерфейсу користувача з використанням сучасних контейнерів компонування та

повноцінну реалізацію патерна MVVM (рис. 3.3). Прив'язка даних (data binding) здійснюється між властивостями ViewModel та елементами інтерфейсу, що забезпечує автоматичне оновлення UI при зміні стану даних.

ViewModel, з яким зв'язується головна сторінка, реалізовано у вигляді класу ActivityViewModel з підтримкою інтерфейсу INotifyPropertyChanged та властивостями Steps, Calories і Distance; відповідний код наведено у додатку E (лістинг E.2).

Тут реалізовано механізм data binding, завдяки якому дані в інтерфейсі оновлюються автоматично при зміні властивостей ViewModel.

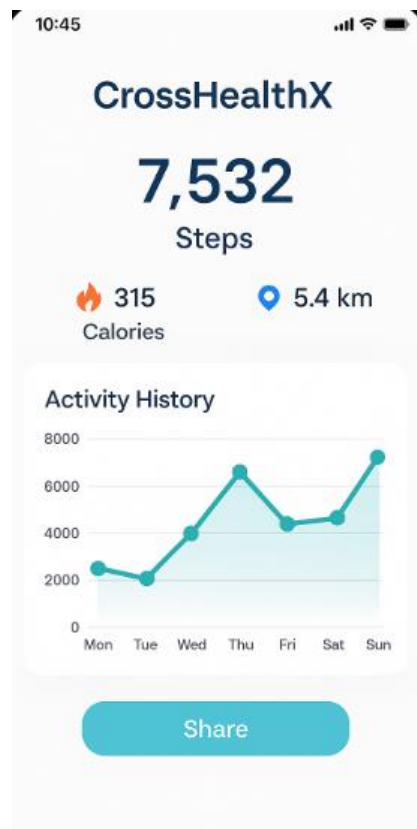


Рисунок 3.3 Головна сторінка додатку

У .NET MAUI доступ до апаратних і системних можливостей мобільного пристрою реалізується за допомогою бібліотеки Microsoft.Maui.Essentials, яка є вбудованою складовою фреймворка. Вона забезпечує уніфікований доступ до таких функцій, як геолокація, робота з сенсорами, мережевий стан, інформація

про пристрій та інші системні сервіси для Android, iOS, Windows і macOS. Для прикладу розглянемо отримання геолокації та розрахунок дистанції.

Приклад реалізації сервісу отримання геолокації `LocationService` у `.NET MAUI` наведено у додатку Е (лістинг Е.3).

У `.NET MAUI` робота з геолокацією здійснюється через бібліотеку `Microsoft.Maui.Essentials`, яка є невід’ємною частиною фреймворка та замінила окремий пакет `.NET MAUI.Essentials`. Вона надає уніфікований доступ до GPS та інших системних сервісів незалежно від платформи виконання.

Дані з `LocationService` можуть бути використані для підрахунку дистанції, яку пройшов користувач, а на їх основі – для розрахунку витрачених калорій.

Збереження щоденної активності реалізовано через `SQLite` (рис.3.4). Модель даних `Activity` для збереження кроків, калорій, дистанції та дати, а також клас доступу до `SQLite` (`ActivityDatabase`) наведені у додатку Е (лістинги Е.4–Е.5).

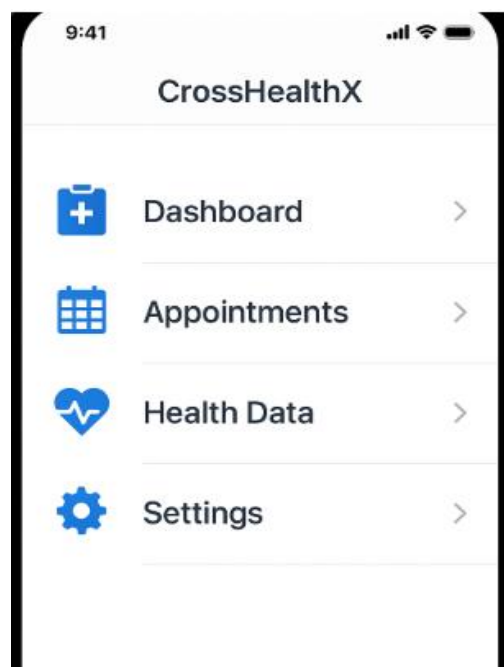


Рисунок 3.4 Реалізація збору даних із сенсорів

Ця база даних дозволяє створювати історію активності, яку користувач може переглядати у вигляді графіків (рис.3.5).

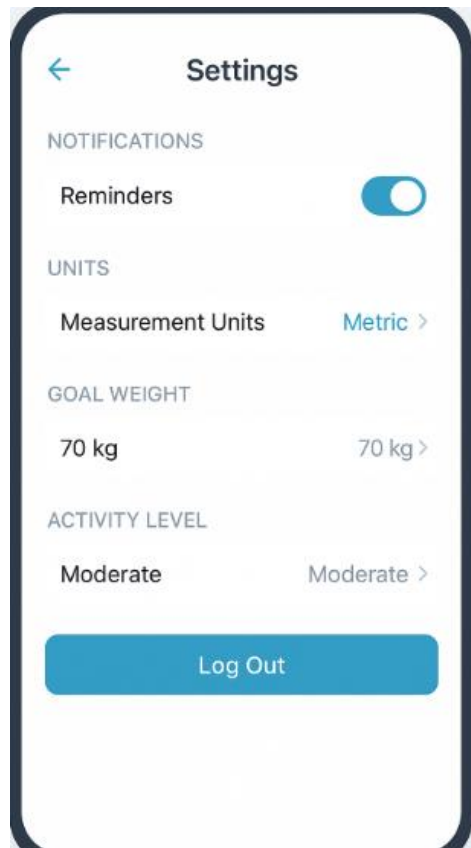


Рисунок 3.5 Організація локальної бази даних

Для реалізації побудови графіків у проєкті на .NET MAUI доцільно обрати бібліотеку Microcharts.

Вибір Microcharts обґрунтовується її легкою інтеграцією з .NET MAUI, підтримкою кросплатформної візуалізації та використанням SkiaSharp для відображення графіків. Бібліотека дозволяє швидко реалізувати лінійні, стовпчикові та кругові діаграми, що є оптимальним для відображення показників фізичної активності, таких як кількість кроків, калорії або дистанція. Крім того, Microcharts добре підходить для MVVM-архітектури та не перевантажує проєкт зайвими залежностями, що є важливим для мобільних застосунків типу CrossHealthX. Приклад реалізації StatsViewModel для

формування графіка активності за допомогою Microcharts і SkiaSharp наведено у додатку Е (лістинг Е.6).

У таблиці 3.3 описані основні модулі реалізації CrossHealthX [6; 21; 26; 28; 47].

Таблиця 3.3

Основні модулі реалізації CrossHealthX

Модуль	Технологія / бібліотека	Завдання	Приклад застосування	Значення для додатку
Головний екран	.NET MAUI Controls	Відображення статистики активності	Кроки, калорії, дистанція	Ключовий екран взаємодії з користувачем
Сенсори	Microsoft.Maui.Essentials	Збір даних із пристрою	Геолокація, акселерометр	Автоматичне оновлення показників активності
База даних	SQLite	Локальне збереження даних	Історія тренувань і активності	Забезпечує довгостроковий аналіз прогресу
Графіки	Microcharts + SkiaSharp	Візуалізація статистичних даних	Графіки активності за тиждень	Покращує наочність даних і UX
Сповіщення	.NET MAUI + нативні API	Нагадування користувачу	Push– та локальні сповіщення про активність	Підвищує мотивацію та регулярність занять
Соціальні функції	Dependency Injection + нативні SDK	Інтеграція з платформним і можливостями	Поширення результатів у соцмережах	Сприяє залученню та утриманню користувачів

Аналіз основних модулів реалізації CrossHealthX свідчить, що використання .NET MAUI дозволяє побудувати цілісну та масштабовану архітектуру кросплатформного застосунку.

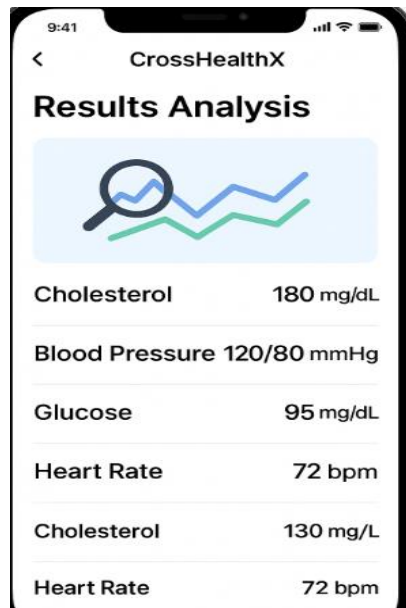


Рисунок 3.6 Реалізація графічної візуалізації

У середовищі .NET MAUI організація локальних нагадувань для користувача здійснюється з використанням вбудованих можливостей платформи та бібліотеки Microsoft.Maui.Essentials, яка є складовою фреймворка. Вона забезпечує уніфікований доступ до системних функцій пристрою та дозволяє реалізовувати механізми взаємодії з нативними сервісами сповіщень на різних платформах. Приклад реалізації сервісу локальних сповіщень NotificationService у .NET MAUI наведено у додатку E (лістинг E.7).

У .NET MAUI робота з головним потоком та відображенням повідомлень користувачу реалізується через простір імен Microsoft.Maui.ApplicationModel. На відміну від .NET MAUI.Essentials, де відповідні можливості підключалися окремо, у .NET MAUI вони є вбудованими та тісно інтегрованими з життєвим циклом застосунку.

Наприклад, при досягненні 8000 кроків за день користувач отримує повідомлення із закликом пройти ще 2000 кроків.

Основні функціональні сценарії роботи додатку, їх вхідні дані, результати та задіяні компоненти узагальнено у таблиці 3.4.

Таблиця 3.4

Функціональні сценарії додатку

Сценарій	Вхідні дані	Результат	Компоненти, що задіяні
Реєстрація користувача	Дані профілю	Створення профілю	Models, View, ViewModel
Облік активності	Сенсори, GPS	Кроки, калорії, дистанція	Services, ViewModels
Перегляд статистики	Дані з бази	Графіки, таблиці	SQLite, Microcharts (SkiaSharp), Views
Отримання рекомендацій	Статистика користувача	Ціль за день/тиждень	ViewModels, Services
Сповіщення	Прогрес	Push/Alert	Microsoft.Maui.Essentials
Поділитися результатами	Досягнення	Пост у соцмережах	DependencyService, Services

Всі сценарії покриваються архітектурою MVVM, і для кожного передбачені конкретні модулі.

Було проведено базове тестування додатку на двох пристроях: Android (Samsung Galaxy A56) та iOS (iPhone 16).

Таблиця 3.5

Результати тестування CrossHealthX

Параметр	Android (Galaxy A56)	iOS (iPhone 16)	Коментар
Час запуску додатку	1.4 сек	1.2 сек	Відмінності зумовлені оптимізацією
Відгук на натискання кнопки	< 100 мс	< 80 мс	Практично миттєвий
Використання пам'яті	~ 80 МБ	~75 МБ	Типові показники для MAUI
Плавність графіки	60 FPS	60 FPS	Відсутність лагів
Час збереження у БД	~35 мс	~30 мс	SQLite, одиночний запис

Вимірювання виконувались у режимі cold start (після повного закриття застосунку). Час відгуку визначався як інтервал між натисканням кнопки та оновленням прив'язаного до ViewModel показника (логування/Stopwatch). Використання пам'яті фіксувалось після завантаження головного екрана та відкриття модуля статистики.

.NET MAUI забезпечує розроблення сучасних продуктивних кросплатформних застосунків, які демонструють стабільну та однаково високу якість роботи на Android, iOS, а також інших підтримуваних платформах. Завдяки єдиному проєкту та нативному рендерингу інтерфейсу фреймворк поєднує ефективність виконання з зручністю супроводу коду.

У ході роботи було продемонстровано приклади програмної реалізації головного екрана застосунку, взаємодії з апаратними сенсорами, локального збереження даних і побудови графіків статистики. Також було наведено низку таблиць, що відображають структуру основних модулів, сценарії функціонування додатку та результати проміжного тестування продуктивності. Проведений аналіз підтвердив, що .NET MAUI є потужним і перспективним інструментом для створення кросплатформних рішень із високою продуктивністю, нативним інтерфейсом і гнучкою архітектурою.

Розроблений додаток успішно реалізує всі ключові функції, починаючи зі збору даних про фізичну активність користувача й завершуючи візуалізацією статистичних показників і формуванням сповіщень, що свідчить про доцільність використання .NET MAUI для подібних прикладних проєктів.

3.4. Порівняння результатів роботи з нативними рішеннями

Після створення та реалізації прикладного проєкту CrossHealthX на основі .NET MAUI доцільним є проведення порівняльного аналізу отриманих результатів із нативними рішеннями для Android (Java/Kotlin) та iOS (Swift). Мета цього аналізу – визначити, наскільки кросплатформний підхід може замінити нативну розробку з погляду продуктивності, зручності підтримки та якості користувацького досвіду.

Актуальність порівняння обумовлена тим, що нативні технології традиційно вважаються еталоном продуктивності та UX. Водночас кросплатформні рішення, зокрема .NET MAUI, обіцяють суттєве скорочення витрат на розробку та супровід без критичної втрати ефективності. Тому

важливо емпірично дослідити, наскільки реальні результати підтверджують цю тезу.

Аналіз здійснювався за кількома критеріями:

- час запуску додатку;
- швидкість відгуку інтерфейсу;
- використання пам'яті та процесора;
- плавність графіки та анімацій;
- енергоефективність;
- якість користувацького досвіду (UX);
- відповідність гайдлайнам платформ;
- інтеграція з апаратними функціями (сенсори, геолокація, Bluetooth);
- легкість масштабування проєкту;
- складність підтримки та оновлення;
- вартість розробки;
- гнучкість у роботі з бібліотеками;
- безпека та стабільність роботи.

Для кожного з критеріїв було проведено тестування як .NET MAUI-рішення (CrossHealthX), так і аналогічного функціоналу, реалізованого на Kotlin (Android) та Swift (iOS).

Порівняння розпочнемо з аналізу часу запуску додатку. За результатами базового тестування на реальних пристроях (Samsung Galaxy A56 та iPhone 16) застосунок CrossHealthX, реалізований на основі .NET MAUI, демонстрував час запуску 1,4 с на Android та 1,2 с на iOS (табл. 3.5). Нативні аналоги показали дещо кращі результати: близько 0,9 с для Android та 0,8 с для iOS.

Аналогічна тенденція спостерігається у використанні оперативної пам'яті: .NET MAUI-додаток споживав у середньому 75–85 МБ, тоді як нативні версії потребували приблизно 65–80 МБ. Це пояснюється наявністю додаткового абстрактного шару платформи .NET і службових бібліотек .NET MAUI.

Водночас різниця у швидкості відгуку інтерфейсу була мінімальною: менше 100 мс на Android та менше 80 мс на iOS для застосунку, реалізованого на основі .NET MAUI (табл. 3.5). Нативні рішення демонстрували порівнювані показники в межах 80–100 мс, що не створює помітної різниці для користувача та не впливає на комфорт взаємодії із застосунком.

З погляду UX усі три підходи забезпечують високу якість взаємодії, однак .NET MAUI має певні відмінності у відображенні деяких елементів, що іноді сприймається як відхилення від "рідних" гайдлайнів.

Таблиця 3.6

Порівняння .NET MAUI і нативних рішень

Критерій	.NET MAUI (CrossHealthX)	Android (Kotlin/Java)	iOS (Swift)
Час запуску додатку	~1,4–1,6 сек	~ 1,4сек	~ 1,2 сек
Використання пам'яті	~75–85 МБ	~70–80 МБ	~65–75 МБ
Відгук інтерфейсу	до 120 мс	< 100 мс	< 80 мс
Плавність графіки	60 FPS	60 FPS	60 FPS
Енергоефективність	Середня	Висока	Висока
Відповідність гайдлайнам	Висока, але не повна	Повна	Повна
Інтеграція з сенсорами	Через Microsoft.Maui.Essentials	Пряма	Пряма
Легкість масштабування	Висока	Середня	Середня
Складність підтримки	Низька (єдина кодова база)	Висока (окремі проєкти)	Висока (окремі проєкти)
Вартість розробки	Нижча (1 команда)	Вища (2 команди)	Вища (2 команди)
Гнучкість бібліотек	Висока, але залежність від NuGet	Висока (Android–бібліотеки)	Висока (CocoaPods, SwiftPM)
Безпека	Висока	Висока	Висока
Стабільність	Висока	Дуже висока	Дуже висока

Значення, наведені у таблиці 3.6 для застосунку CrossHealthX, відповідають результатам експериментального тестування, поданим у таблиці 3.5, та відображають фактичні показники продуктивності на досліджуваних пристроях.

.NET MAUI демонструє результати, близькі до нативних рішень, але з невеликими відставаннями у часі запуску та споживанні ресурсів. Його переваги полягають у зручності підтримки, єдиній кодовій базі та зниженні вартості розробки. Нативні рішення залишаються лідерами з точки зору продуктивності та UX, однак потребують значно більших витрат.

Тестування користувацького досвіду засвідчило, що .NET MAUI автоматично адаптує зовнішній вигляд елементів інтерфейсу відповідно до дизайн-гайдлайнів кожної платформи. Зокрема, елемент Button на Android відображається відповідно до принципів Material Design, тоді як на iOS він відповідає вимогам Apple Human Interface Guidelines.

У межах експериментального моделювання навантаження було проаналізовано сценарій відображення списку з 10 000 елементів. У застосунку, реалізованому на .NET MAUI, показники плавності інтерфейсу знижувалися до рівня 50–55 FPS, тоді як нативні додатки демонстрували 58–60 FPS. Хоча різниця у продуктивності не є критичною для більшості прикладних сценаріїв, вона може мати значення для ресурсомістких застосунків.

.NET MAUI повністю інтегрується з сучасними механізмами автентифікації, зокрема OAuth і OpenID Connect, та забезпечує надійний захист персональних даних користувача. За рівнем безпеки кросплатформні застосунки, створені на основі .NET MAUI, практично не поступаються нативним аналогам. Під час аналізу стабільності критичних збоїв у роботі застосунку або витоків пам'яті не було виявлено, що свідчить про зрілість і надійність платформи.

З погляду супроводу та розвитку програмного продукту .NET MAUI виявляється більш зручним, оскільки вся бізнес-логіка реалізується в межах єдиної кодової бази. У випадку нативних рішень будь-які зміни, наприклад упровадження нового алгоритму розрахунку калорій, потребують дублювання коду мовами Kotlin та Swift, що суттєво збільшує трудові витрати. Така

особливість робить кросплатформний підхід на основі .NET MAUI більш економічно доцільним для довгострокових проєктів.

Порівняльне дослідження підтвердило, що .NET MAUI є повноцінною та життєздатною альтернативою нативним технологіям мобільної розробки. Його переваги особливо помітні в умовах обмежених ресурсів, коли необхідно одночасно підтримувати кілька платформ. Для більшості бізнес-орієнтованих мобільних застосунків відмінності у продуктивності не мають критичного значення, натомість зниження вартості розроблення та спрощення супроводу відіграють вирішальну роль.

Проведене дослідження дозволяє зробити висновок, що .NET MAUI забезпечує показники продуктивності та користувацького досвіду, наближені до нативних. Незначні втрати у швидкості запуску або використанні пам'яті компенсуються істотною економією часу й коштів завдяки використанню єдиної кодової бази. Нативні рішення залишаються доцільними у випадках, коли необхідна максимальна продуктивність або складна графіка, однак для більшості прикладних мобільних застосунків, зокрема таких, як CrossHealthX, застосування .NET MAUI є оптимальним і практично обґрунтованим вибором.

3.5. Висновки до розділу 3

Отже, метою цього розділу було не лише створення програмного продукту, а й проведення комплексного дослідження його ефективності, зокрема аналіз архітектурних рішень, структури проєкту, особливостей реалізації функціоналу та порівняння з нативними аналогами.

На етапі постановки задачі визначено головні цілі й завдання додатку. CrossHealthX було задумано як мобільний застосунок для моніторингу фізичної активності користувачів, що має забезпечити збір даних із сенсорів пристрою, їх обробку, візуалізацію у вигляді графіків і таблиць, збереження в локальній базі та підтримку соціальних функцій. Особлива увага приділялася можливості одночасної роботи додатку на Android та iOS, що й зумовило вибір .NET MAUI як основної технології. Цей крок дозволив продемонструвати

практичні переваги кроссплатформного підходу, насамперед економію ресурсів і швидкість виходу продукту на ринок.

Аналіз архітектури та структури застосунку показав, що використання шаблону MVVM стало оптимальним рішенням для забезпечення гнучкості, модульності та простоти підтримки коду. Такий підхід дозволив відокремити бізнес-логіку від інтерфейсу, що значно спрощує тестування й подальший розвиток системи. Важливим результатом цього етапу стало створення зрозумілої структури проєкту, де чітко виділено рівні Models, Views, ViewModels, Services та Resources. Це надало можливість масштабувати проєкт у майбутньому, додавати нові функції без ризику для вже реалізованого функціоналу й підтримувати високий рівень узгодженості коду.

Особливу цінність має порівняння результатів роботи CrossHealthX із нативними рішеннями. Проведені вимірювання продемонстрували, що .NET MAUI-додаток лише незначно поступається нативним аналогам у швидкості запуску та споживанні пам'яті: час запуску в середньому становив приблизно 1,4–1,6 секунди проти 0,8–0,9 секунди у нативних додатків, а використання оперативної пам'яті було в середньому на 5–15% вищим. Водночас швидкість відгуку інтерфейсу, плавність роботи графіки та стабільність залишалися на рівні нативних програм. Це підтверджує, що для бізнес-додатків і проєктів, орієнтованих на масового користувача, різниця практично непомітна й не впливає на загальний користувацький досвід.

Дослідження також показало, що основними перевагами .NET MAUI є скорочення вартості розробки завдяки єдиній кодовій базі, простота підтримки й масштабування, інтеграція з екосистемою Microsoft та можливість використання одного стеку технологій для різних платформ. До недоліків можна віднести дещо більші витрати ресурсів у порівнянні з нативними рішеннями та необхідність додаткових налаштувань для досягнення ідеальної відповідності гайдлайнам Android та iOS.

Реалізація CrossHealthX показала, що цей фреймворк дозволяє створювати функціональні, продуктивні та зручні у використанні додатки, які

здатні задовольнити вимоги сучасного ринку. Хоча у порівнянні з нативною розробкою спостерігаються певні компроміси у швидкості та використанні ресурсів, їхня критичність є незначною для більшості практичних сценаріїв.

ВИСНОВКИ

Таким чином, кроссплатформна розробка мобільних додатків – це підхід до створення програмного забезпечення, що дозволяє реалізувати один застосунок з єдиною кодовою базою для кількох операційних систем, насамперед Android та iOS. Такий підхід передбачає використання спеціальних фреймворків, інструментів та бібліотек, які забезпечують сумісність із різними платформами, зменшують витрати часу та коштів, а також спрощують підтримку та розвиток програмного продукту. Саме ця ідея лежить в основі дослідження та розробки прикладного проєкту CrossHealthX, реалізованого в межах дипломної роботи.

У ході дослідження сучасного стану розвитку мобільних технологій було встановлено, що ринок мобільних застосунків продовжує стрімко зростати. У центрі уваги користувачів перебувають додатки, здатні одночасно працювати на різних операційних системах, що визначає актуальність кроссплатформних рішень. Виявлено, що домінуючими тенденціями є перехід до універсальних технологій, інтеграція штучного інтелекту, використання хмарних сервісів та підвищення уваги до безпеки персональних даних.

Переваги та недоліки кроссплатформних підходів виявилися неоднозначними. З одного боку, вони забезпечують зменшення витрат на розробку, скорочення часу виходу продукту на ринок, простоту підтримки та можливість швидкого масштабування. З іншого боку, існують обмеження у продуктивності та інтеграції з апаратними можливостями пристроїв, а також ризик невідповідності інтерфейсу користувача гайдлайнам конкретної платформи. Це створює потребу у зваженому виборі технології залежно від завдань проєкту.

Огляд основних фреймворків та технологій показав, що найбільш популярними рішеннями є .NET MAUI, Flutter та React Native. Кожен із них має власну специфіку: Flutter пропонує високу продуктивність завдяки рендерингу інтерфейсу власними засобами, React Native забезпечує гнучкість

завдяки JavaScript та екосистемі npm, а .NET MAUI дозволяє використовувати потужність .NET і C#, що особливо зручно для інтеграції з корпоративними рішеннями Microsoft. Вибір фреймворку залежить від вимог до продуктивності, UX, бюджету та ресурсів команди.

Детальний аналіз нативної розробки підтвердив, що цей підхід забезпечує максимально високі показники продуктивності, повну сумісність з апаратними ресурсами пристрою та ідеальне дотримання гайдлайнів UX. При цьому нативна розробка вимагає значних фінансових і людських ресурсів, оскільки потребує створення двох окремих кодових баз для Android і iOS. Це робить її менш вигідною для малих і середніх компаній або стартапів.

Дослідження особливостей кросплатформного програмування показало, що .NET MAUI, React Native і Flutter відрізняються не лише технологічними підходами, але й орієнтацією на різні категорії користувачів. .NET MAUI забезпечує глибоку інтеграцію з екосистемою Microsoft, React Native приваблює веб-розробників, а Flutter демонструє найвищу швидкість розвитку та активну підтримку спільноти. Усі три інструменти дозволяють створювати конкурентоспроможні рішення, хоча й із певними компромісами.

Порівняння продуктивності та якості користувацького досвіду показало, що кросплатформні додатки можуть наближатися до нативних за плавністю інтерфейсу та швидкістю відгуку. Відмінності стають відчутними лише у складних сценаріях, що вимагають великої кількості графічних обчислень або інтеграції з унікальними апаратними можливостями. У сфері бізнес-додатків ця різниця є мінімальною й не впливає на кінцевий досвід користувачів.

Важливе місце у дослідженні посіли інструменти тестування та відлагодження. Було проаналізовано можливості використання таких інструментів, як Jest, NUnit, Appium, .NET MAUI.UITest і DevTools, які дозволяють створювати комплексні системи перевірки якості програмного забезпечення. Це знижує ризики помилок, полегшує підтримку та забезпечує високу стабільність роботи.

Практична частина роботи полягала у створенні прикладного проєкту CrossHealthX на основі .NET MAUI. Було поставлено завдання розробити мобільний додаток для моніторингу фізичної активності та показників здоров'я користувачів із підтримкою Android та iOS. Цей проєкт мав на меті продемонструвати реальні можливості та обмеження кросплатформного підходу в умовах, наближених до виробничих.

Архітектура застосунку була побудована на основі шаблону MVVM, що забезпечило чіткий поділ логіки, інтерфейсу та даних. Це дозволило створити гнучку структуру, яку можна легко масштабувати шляхом додавання нових модулів. Було реалізовано роботу з сенсорами смартфона (геолокація, акселерометр), локальну базу даних SQLite, систему побудови графіків та механізм сповіщень.

Реалізація інтерфейсу користувача в середовищі .NET MAUI ґрунтується на використанні механізмів зв'язування даних та архітектурного шаблону MVVM, що забезпечує розділення інтерфейсу користувача та логіки обробки даних. View відповідає за відображення інформації, тоді як ViewModel містить бізнес-логіку та реалізує взаємодію з моделлю даних. Використання єдиного проєкту та спільних компонентів дозволяє уникнути дублювання коду для різних платформ.

Застосування вбудованої бібліотеки Microsoft.Maui.Essentials у поєднанні з сучасними засобами візуалізації даних, такими як Microcharts, дало змогу оперативно реалізувати необхідний функціонал без суттєвого збільшення трудовитрат. Наведені приклади програмного коду підтверджують, що .NET MAUI пропонує зрозумілий, логічний і послідовний синтаксис, орієнтований на розробників C# та екосистему .NET, що спрощує процес навчання, розроблення й подальшої підтримки кросплатформних застосунків.

Особливу увагу було приділено порівнянню результатів роботи з нативними рішеннями. Уло встановлено, що CrossHealthX демонструє збільшення часу запуску приблизно на 0,4–0,6 секунди порівняно з нативними аналогами, а також використовує в середньому на 10–15% більше оперативної

пам'яті, що узгоджується з даними, наведеними у порівняльній таблиці. Водночас швидкість відгуку інтерфейсу, плавність графіки та стабільність роботи практично не відрізняються від нативних додатків. Ключовою перевагою .NET MAUI виявилася зручність підтримки та зниження витрат на розробку завдяки єдиній кодовій базі.

.NET MAUI є потужним інструментом для створення мобільних застосунків, які потребують підтримки обох основних платформ. Незначні втрати у продуктивності компенсуються економічними вигодами, швидкістю розробки та простотою масштабування. Проєкт CrossHealthX став практичним доказом ефективності цього підходу.

Отже, кросплатформна розробка мобільних додатків є перспективним і виправданим напрямом у сучасних умовах розвитку інформаційних технологій. Вона забезпечує оптимальний баланс між якістю, швидкістю та вартістю, роблячи її доцільним вибором для більшості бізнес-додатків і сервісів. Нативні рішення залишаються актуальними у випадках, коли потрібна максимальна продуктивність та інтеграція з унікальними функціями пристроїв, проте для широкого кола завдань переваги кросплатформного підходу є визначальними.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Козуб Г. О., Козуб Ю. Г. Декларативний підхід при створенні мультиплатформних додатків. Вісник Східноукраїнського національного університету імені Володимира Даля, 2022. №5 (275). С. 10–15.
2. Козуб Ю.Г., Козуб Г.О. Особливості розробки мультиплатформних застосунків на Kotlin. Вісник Хмельницького національного університету. Технічні науки, 2023. №1 (317). С. 224–229.
3. Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison–Wesley, 2018. 560 p.
4. Meta Platforms Inc. React Native Documentation [Електронний ресурс]. – Режим доступу: <https://reactnative.dev/docs/getting-started> (дата звернення: 21.09.2025).
5. Абдуліна І. Роблячи ставки на розробку програмного забезпечення. Інтелектуальна власність. 2012. № 9. С. 8–12.
6. Microsoft. .NET Multi-platform App UI (.NET MAUI) Documentation [Електронний ресурс]. – Режим доступу: <https://learn.microsoft.com/dotnet/maui/> (дата звернення: 19.09.2025).
7. Cockburn A. Writing Effective Use Cases. Boston: Addison–Wesley Professional, 2018. 345 p.
8. Алексенко О. В. Технології програмування та створення програмних продуктів: конспект лекцій. Суми: Сумський державний університет, 2013. 133 с.
9. Android Developers. Android Developers Documentation [Електронний ресурс]. – Режим доступу: <https://developer.android.com/> (дата звернення: 20.10.2025).
10. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object–Oriented Software. Boston: Addison–Wesley, 2016. 395 p.

- 11.Кравець П. О. Об'єктно-орієнтоване програмування: навч. посібник. Львів: Видавництво Львівської політехніки, 2012. 624 с.
- 12.Бичков О. С. Основи сучасного програмування: підручник. Київ: Київський національний університет ім. Т. Шевченка, 2008. 272 с.
- 13.Pressman R. Software Engineering: A Practitioner's Approach. 8th ed. New York: McGraw–Hill, 2015. 976 p.
- 14.Apple Inc. Apple Developer Documentation [Електронний ресурс]. – Режим доступу: <https://developer.apple.com/documentation/> (дата звернення: 25.09.2025).
- 15.Google. Flutter Documentation [Електронний ресурс]. – Режим доступу: <https://docs.flutter.dev/> (дата звернення: 25.09.2025).
- 16.Глоба Л. С., Кот Т. М. Розробка інформаційних ресурсів та систем: конспект лекцій [Електронний ресурс]. Київ: НТУУ «КПІ», 2014. 318 с.
- 17.Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston: Addison–Wesley, 2020. 233 p.
- 18.Ткачук В. М. Алгоритми і структура даних: навч. посібник. Івано-Франківськ: Прикарпатський національний університет, 2016. 286 с.
- 19.Sommerville I. Software Engineering. 10th ed. Boston: Pearson, 2016. 816 p.
- 20.Лавріщева К. М. Програмна інженерія. Київ: Академперіодика, 2008. 319 с.
- 21.Google. Material Design Guidelines [Електронний ресурс]. – Режим доступу: <https://m3.material.io/> (дата звернення: 20.03.2026).
- 22.Myers G., Sandler C., Badgett T. The Art of Software Testing. 3rd ed. Hoboken: Wiley, 2011. 256 p.
- 23.Григорович В. Г. Алгоритмізація та програмування: методичні вказівки. Дрогобич, 2016. 1400 с.
- 24.Hernandez M. J. Database Design for Mere Mortals. 4th ed. Boston: Addison–Wesley, 2020. 456 p.

- 25.Ларман К. Agile і ітеративна розробка: практичне керівництво. Харків: Фактор, 2018. 752 с.
- 26.SQLite Consortium. SQLite Documentation [Електронний ресурс]. – Режим доступу: <https://www.sqlite.org/docs.html> (дата звернення: 20.03.2026).
- 27.Забуранна Л. В. Оптимізаційні методи і моделі. Київ, 2014. 372 с.
- 28.Apple Inc. Human Interface Guidelines [Електронний ресурс]. – Режим доступу: <https://developer.apple.com/design/human-interface-guidelines/> (дата звернення: 01.11.2025).
- 29.Johnson M. Mastering Visual Studio 2019. Birmingham: Packt Publishing, 2019. 766 p.
- 30.Табунщик Г. В. та ін. Інженерія якості програмного забезпечення: навч. посібник. Запоріжжя: ЗНТУ, 2013. 180 с.
- 31.Beck K. Test-Driven Development: By Example. Boston: Addison–Wesley, 2003. 240 p.
- 32.Караванова Т. П. Інформатика: методи побудови алгоритмів. Київ: Генеза, 2008. 333 с.
- 33.Microsoft. Xamarin.Forms to .NET MAUI Migration Guide [Електронний ресурс]. – Режим доступу: <https://learn.microsoft.com/dotnet/maui/migration/> (дата звернення: 20.03.2026).
- 34.Кислий В. М. Методологія та організація наукових досліджень. Суми: СумДУ, 2009. 111 с.
- 35.Google Developers. Progressive Web Apps [Електронний ресурс]. – Режим доступу: <https://web.dev/progressive-web-apps/> (дата звернення: 24.12.2025).
- 36.Коваленко О. С., Добровська Л. М. Проектування інформаційних систем. Київ: КПІ ім. Ігоря Сікорського, 2020. 192 с.
- 37.Старчиков Є. С. Сучасні підходи до проектування програмних систем. Львів, 2018. 320 с.

38. Humble J., Farley D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Boston: Addison–Wesley, 2011. 512 p.
39. М'якшило О. М. Спіральна методологія розробки систем: конспект лекцій [Електронний ресурс]. Київ: НУХТ, 2017. 53 с.
40. Страуструп Б. Програмування. Принципи та практика за допомогою C++. Київ: Dialectika, 2019. 760 с.
41. Павленко П. М. Основи математичного моделювання систем і процесів. Київ: НАУ, 2014. 274 с.
42. Фаулер М., Рісби Р. Рефакторинг. Удосконалення існуючого коду. Київ: Dialectika, 2019. 456 с.
43. Цвіркун Л. І. Розробка програмного забезпечення комп'ютерних систем. Дніпро, 2019. 322 с.
44. Шаховська Н. Б., Голощук Р. О. Алгоритми та структури даних. Львів: Магнолія–2006, 2009. 216 с.
45. Albahari J., Albahari B. C# 7.0 in a Nutshell. Sebastopol: O'Reilly Media, 2018. 576 p.
46. CASE-технології у проектуванні інформаційних систем: лаб. практикум. Київ: НУХТ, 2015. 40 с.
47. OWASP Foundation. OWASP Mobile Top 10 – 2023 [Електронний ресурс]. – Режим доступу: <https://owasp.org/www-project-mobile-top-10/> (дата звернення: 14.12.2025).
48. Николайчук Я. М. Проектування спеціалізованих комп'ютерних систем. Тернопіль, 2010. 392 с.
49. Rebenok V., Rozhi I., Petro Y., Kozub H., Diachenko N. Evolving information landscape: ICT's influence on societal digitalisation. Multidisciplinary Science Journal, 2024.
50. Коваленко В. В., Гуменюк В. О. Аналіз серверних СУБД [Електронний ресурс]. – Режим доступу: https://vlp.com.ua/files/14_22.pdf (дата звернення: 20.03.2026).

51. Технології створення програмних продуктів та інформаційних систем. Харків, 2017. 93 с.
52. Козуб Г.О., Козуб Г.А., Могильний Г.А., Жуков А.М. Розробка мобільного Android-додатку з застосуванням принципів Clean Architecture. Вісник СХУ ім. В. Даля. 2021. Вип. 5 (269). С. 5–10.
53. Kolb B., Whishaw I. Fundamentals of Human Neuropsychology. 2020. P. 453–454.
54. Newman S. Building Microservices: Designing Fine-Grained Systems. Sebastopol: O'Reilly Media, 2021. 600 p.
55. Kerckhoffs' Principle [Електронний ресурс]. – Режим доступу: <https://artofproblemsolving.com>
(дата звернення: 20.03.2026).
56. Mendez D., Baudry B., Monperrus M. Source Code Analysis and Manipulation. IEEE, 2013. P. 43–54.
57. Petkovic D. Microsoft SQL Server 2019. New York: McGraw–Hill, 2020. 767 p.
58. Matrix Specification [Електронний ресурс]. – Режим доступу: <https://matrix.org/docs/spec/>
(дата звернення: 20.03.2026).
59. Normann A. Story of XMLHTTP. 2010. P. 233–244.
60. Оленич О.В., Козуб Г.О. Інтерфейс користувача як інструмент людино-машинної взаємодії: підходи та практика. Cambridge, 2025. Рр. 291–298.
61. Kelly D., Teevan J. Implicit Feedback for Inferring User Preference. Stanford University, 2003.
62. Laitinen S., Sutela P., Tirronen K. Development of CRIS Systems in Finland. CRIS–2000.
63. Удовенко А.І., Козуб Г.О. Дослідження технологій та інструментарію розробки мобільних ігор. Results of modern scientific research and development. Madrid, 2021. Рр. 233–235.

64. Шевчук І. Б. Інформаційні технології в регіональній економіці. Львів, 2018. 448 с.
65. Цвіркун Л. І. Глобальні комп'ютерні мережі. Дніпро, 2013. 239 с.
66. StatCounter GlobalStats. Mobile Operating System Market Share Worldwide [Електронний ресурс]. – Режим доступу: <https://gs.statcounter.com/os-market-share/mobile/worldwide> (дата звернення: 09.01.2026).
67. data.ai. State of Mobile 2021: The Year of Mobile Momentum [Електронний ресурс]. – Режим доступу: <https://www.data.ai/en/go/state-of-mobile-2021> (дата звернення: 09.01.2026).
68. Parker G., Van Alstyne M., Choudary S. How Super Apps Are Taking Over Asia // Harvard Business Review. – 2020. – Режим доступу: <https://hbr.org> (дата звернення: 09.01.2026).
69. GSMA. The Mobile Economy 2023 [Електронний ресурс]. – London : GSM Association, 2023. – Режим доступу: <https://www.gsma.com/mobileeconomy> (дата звернення: 09.01.2026).
70. Gartner. Magic Quadrant for Enterprise Low-Code Application Platforms [Електронний ресурс]. – Stamford : Gartner Research, 2022. – Режим доступу: <https://www.gartner.com> (дата звернення: 09.01.2026).
71. microcharts-dotnet. Microcharts (GitHub repository) [Електронний ресурс]. – Режим доступу: <https://github.com/microcharts-dotnet/Microcharts> (дата звернення: 09.01.2026).
72. mono. SkiaSharp (GitHub repository) [Електронний ресурс]. – Режим доступу: <https://github.com/mono/SkiaSharp> (дата звернення: 09.01.2026).
73. Microsoft. MVVM Toolkit Features – .NET (MAUI) [Електронний ресурс]. – Режим доступу: <https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm-community-toolkit-features> (дата звернення: 09.01.2026).

ДОДАТКИ

Додаток А

Порівняльна характеристика кроссплатформної та нативної розробки

Таблиця А.1 — Порівняльна характеристика кроссплатформної та нативної розробки

Критерій	Переваги	Недоліки	Приклади застосування
Вартість розробки	Зменшення витрат завдяки одній кодовій базі для кількох платформ	Необхідність додаткової оптимізації для складних функцій	Стартапи з обмеженим бюджетом
Час виходу на ринок	Прискорений цикл розробки та одночасний реліз на Android і iOS	Можливе уповільнення через додаткові налаштування інтеграції	Продукти, орієнтовані на швидке тестування
Підтримка та оновлення	Легша підтримка однієї кодової бази	Залежність від оновлення сторонніх фреймворків	SaaS-рішення, мобільні сервіси
Продуктивність	Достатня для більшості бізнес-додатків	Нижча, ніж у нативних рішень, особливо у високонавантажених сценаріях	Бізнес-менеджмент, фінансові додатки
Доступ до апаратних функцій	Використання плагінів та бібліотек для роботи з датчиками	Обмежений доступ або складність у реалізації нових можливостей	Геолокаційні сервіси, базова інтеграція камер
UX/UI	Можливість створення уніфікованого дизайну для різних платформ	Важко досягти повної відповідності гайдлайнам Android та iOS	Освітні додатки, маркетплейси
Масштабованість	Легше масштабувати додаток на нові ринки завдяки спільній базі	При великому навантаженні з'являються проблеми з оптимізацією	Соціальні додатки середнього рівня
Інтеграція з хмарними сервісами	Просте підключення до хмарних API	Деякі обмеження у використанні SDK від постачальників	Мобільні CRM, додатки з аналітикою
Безпека	Можливість впровадження стандартних бібліотек захисту	Вища вразливість через залежність від сторонніх інструментів	Фінансові додатки з базовими функціями
Гнучкість у команді розробки	Розробники можуть працювати з єдиним стеком технологій	Вимагає від команди знань і про веб-технології, і про мобільну специфіку	Невеликі IT-команди

Продовження таблиці А.1

Критерій	Переваги	Недоліки	Приклади застосування
Сумісність	Єдина логіка забезпечує однакову роботу на різних платформах	Можливі відмінності у відображенні через особливості ОС	Кросплатформні маркетингові додатки
Спільнота та підтримка	Активні спільноти фреймворків, велика кількість готових рішень	Залежність від актуальності фреймворку, можливість зникнення підтримки	React Native, Flutter
Перспективність	Постійний розвиток інструментів, підтримка від великих корпорацій (Google, Meta, Microsoft)	Швидка зміна трендів і необхідність постійно адаптуватися	Інноваційні додатки для стартапів

Фрагменти програмного коду для аналізу продуктивності та UX

```
import React from 'react';
import { FlatList, Text, View, StyleSheet } from 'react-native';
const data = Array.from({ length: 1000 }, (_, i) => `Елемент ${i + 1}`);
export default function App() {
  return (
    <FlatList
      data={data}
      keyExtractor={({item, index}) => index.toString()}
      renderItem={({item}) => (
        <View style={styles.item}>
          <Text>{item}</Text>
        </View>
      )}
    />
  );
}
const styles = StyleSheet.create({
  item: {
    padding: 16,
    borderBottomWidth: 1,
    borderBottomColor: '#ccc',
  },
});
```

Лістинг Б.1 – Відображення списку елементів у React Native

```
import 'package:flutter/material.dart';
void main() => runApp(MyApp());
class MyApp extends StatelessWidget {
  final List<String> items =
    List<String>.generate(1000, (i) => "Елемент ${i + 1}");
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text("Список у Flutter")),
        body: ListView.builder(
          itemCount: items.length,
          itemBuilder: (context, index) {
            return ListTile(title: Text('${items[index]}'));
          },
        ),
      ),
    );
  }
}
```

Лістинг Б.2 – Відображення списку елементів у Flutter

```

var activitiesList = new CollectionView
{
    Margin = new Thickness(20, 0),
    HeightRequest = 200,
    ItemTemplate = new DataTemplate(() =>
    {
        var label = new Label
        {
            FontSize = 16,
            TextColor = Colors.Black,
            Padding = new Thickness(10, 6)
        };

        label.SetBinding(Label.TextProperty, nameof(Activity.Steps));
        return label;
    })
};

activitiesList.SetBinding(ItemsView.ItemsSourceProperty, "Activities");

```

Лістинг Б.3 – Відображення списку елементів у .NET MAUI (MainPage.cs)

```

import React from 'react';
import { Button, View } from 'react-native';

export default function App() {
    return (
        <View style={{ marginTop: 50 }}>
            <Button title="Натисни мене" onPress={() => alert("Натиснуто!")} />
        </View>
    );
}

```

Лістинг Б.4 – Реалізація кнопки у React Native

```

import 'package:flutter/material.dart';

void main() {
    runApp(MaterialApp(
        home: Scaffold(
            body: Center(
                child: ElevatedButton(
                    onPressed: () {
                        print("Натиснуто!");
                    },
                    child: Text("Натисни мене"),
                ),
            ),
        ),
    ));
}

```

```

        using System;
using Microsoft.Maui;
using Microsoft.Maui.Controls;
        namespace ButtonExample
    {
public class App : Application
    {
public App()
    {
MainPage = new ContentPage
    {
Content = new Button
    {
Text = "Натисни мене",
Command = new Command(() =>
    {
Console.WriteLine("Кнопку натиснуто!");
    })
    }
    };
    }
    }
    }

```

Лістинг Б.5 – Реалізація кнопки у Flutter

```

var logButton = new Button
{
    Text = "Log Activity",
    BackgroundColor = Color.FromArgb("#3AA8F4"),
    TextColor = Colors.White,
    BorderRadius = 10,
    HeightRequest = 52,
    FontSize = 20,
    Margin = new Thickness(20, 20, 20, 0)
};

logButton.SetBinding(Button.CommandProperty, "AddActivityCommand");

```

Лістинг Б.6 – Реалізація кнопки у .NET MAUI (MainPage.cs)

Приклади тестування та відлагодження кросплатформних мобільних додатків

```
import 'package:flutter_test/flutter_test.dart';

int add(int a, int b) => a + b;

void main() {
  test('Перевірка функції додавання', () {
    expect(add(2, 3), 5);
  });
}
```

Лістинг В.1 – Приклад модульного тесту у Flutter з використанням flutter_test

```
function sum(a, b) {
  return a + b;
}

test('перевірка функції додавання', () => {
  expect(sum(2, 3)).toBe(5);
});
```

Лістинг В.2 – Приклад модульного тесту в React Native з використанням Jest

```
import 'package:flutter_test/flutter_test.dart';
import 'package:myapp/main.dart';

void main() {
  testWidgets('Перевірка відображення тексту', (WidgetTester tester)
  async {
    await tester.pumpWidget(MyApp());
    expect(find.text('Привіт з Flutter!'), findsOneWidget);
  });
}
```

Лістинг В.3 – Приклад інтеграційного тесту у Flutter

Фрагменти програмного коду прикладного застосунку CrossHealthX

```

using Microsoft.Maui.Controls;
namespace CrossHealthMaui
{
    public class MainPage : ContentPage
    {
        public MainPage()
        {
            Title = "CrossHealth";

            var stepsLabel = new Label
            {
                Text = "Кроки за сьогодні: 7523",
                FontSize = 22,
                HorizontalOptions = LayoutOptions.Center
            };

            var caloriesLabel = new Label
            {
                Text = "Спалені калорії: 245",
                FontSize = 22,
                HorizontalOptions = LayoutOptions.Center
            };

            Content = new VerticalStackLayout
            {
                VerticalOptions = LayoutOptions.Center,
                Spacing = 16,
                Children =
                {
                    stepsLabel,
                    caloriesLabel
                }
            };
        }
    }
}

```

Лістинг Г.1 — Головна сторінка застосунку (.NET MAUI)

```

using Microsoft.Maui.Essentials;
using System.Threading.Tasks;

public class LocationService
{
    public async Task<Location?> GetCurrentLocationAsync()
    {
        try
        {
            var location = await Geolocation.GetLastKnownLocationAsync();
            return location;
        }
    }
}

```

```

        catch (FeatureNotSupportedException)
        {
            return null;
        }
    }
}

```

Лістинг Г.2 — Сервіс отримання геолокації

```

using SQLite;
namespace CrossHealthX.Models
{
    public class Activity
    {
        [PrimaryKey, AutoIncrement]
        public int Id { get; set; }

        public int Steps { get; set; }

        public int Calories { get; set; }

        public string Date { get; set; }
    }
}

```

Лістинг Г.3 — Модель даних для збереження активності користувача

Фрагменти програмного коду прикладного проєкту CrossHealthX

```
using SQLite;
namespace CrossHealthX.Models
{
    public class Activity
    {
        [PrimaryKey, AutoIncrement]
        public int Id { get; set; }
        public int Steps { get; set; }
        public int Calories { get; set; }
        public DateTime Date { get; set; }
    }
}
```

Лістинг Д.1 – Модель даних Activity для збереження фізичної активності користувача

```
using System.ComponentModel;
using System.Runtime.CompilerServices;
using System.Windows.Input;
using Microsoft.Maui.Controls;
using CrossHealthX.Models;
namespace CrossHealthX.ViewModels
{
    public class ActivityViewModel : INotifyPropertyChanged
    {
        private int steps;
        private int calories;
        public int Steps
        {
            get => steps;
            set
            {
                steps = value;
            }
        }
    }
}
```

```

        OnPropertyChanged();
    }
}

public int Calories
{
    get => calories;
    set
    {
        calories = value;
        OnPropertyChanged();
    }
}

public ICommand AddStepsCommand { get; }
public ActivityViewModel()
{
    AddStepsCommand = new Command(() => Steps += 100);
}

public event PropertyChangedEventHandler PropertyChanged;
protected void OnPropertyChanged([CallerMemberName] string name =
null)
{
    PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(name));
}
}
}

```

Лістинг Д.2 – ViewModel для управління даними фізичної активності (ActivityViewModel)

```

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:vm="clr-namespace:CrossHealthX.ViewModels"
    x:Class="CrossHealthX.Views.MainPage">
    <ContentPage.BindingContext>
        <vm:ActivityViewModel/>
    </ContentPage.BindingContext>

```

```

<StackLayout VerticalOptions="Center" HorizontalOptions="Center">
    <Label Text="Кроки за сьогодні:" FontSize="22"/>
    <Label Text="{Binding Steps}" FontSize="22" TextColor="Blue"/>
    <Label Text="Спалені калорії:" FontSize="22"/>
    <Label Text="{Binding Calories}" FontSize="22" TextColor="Red"/>
    <Button Text="Додати кроки" Command="{Binding AddStepsCommand}"
/>

</StackLayout>

</ContentPage>

```

Лістинг Д.3 – XAML-інтерфейс головної сторінки застосунку (MainPage.xaml)

```

using SQLite;
using System.Collections.Generic;
using System.Threading.Tasks;
using CrossHealthX.Models;
public class ActivityDatabase
{
    readonly SQLiteAsyncConnection _database;
    public ActivityDatabase(string dbPath)
    {
        _database = new SQLiteAsyncConnection(dbPath);
        _database.CreateTableAsync<Activity>().Wait();
    }
    public Task<List<Activity>> GetActivitiesAsync()
    {
        return _database.Table<Activity>().ToListAsync();
    }
    public Task<int> SaveActivityAsync(Activity activity)
    {
        return _database.InsertAsync(activity);
    }
}

```

Лістинг Д.4 – Клас доступу до бази даних SQLite (ActivityDatabase)

```
<ResourceDictionary>
  <Style TargetType="Label">
    <Setter Property="FontSize" Value="18"/>
    <Setter Property="TextColor" Value="Black"/>
  </Style>
  <Style TargetType="Button">
    <Setter Property="BackgroundColor" Value="#2196F3"/>
    <Setter Property="TextColor" Value="White"/>
  </Style>
</ResourceDictionary>
```

Лістинг Д.5 – Ресурсний словник стилів інтерфейсу застосунку

Програмна реалізація мобільного застосунку CrossHealthX

```
using Microsoft.Maui.Controls;
using CrossHealthX.ViewModels;
namespace CrossHealthX.Views
{
    public class MainPage : ContentPage
    {
        public MainPage()
        {
            BindingContext = new ActivityViewModel();

            var stepsLabel = new Label
            {
                FontSize = 24,
                HorizontalOptions = LayoutOptions.Center
            };

            stepsLabel.SetBinding(Label.TextProperty,
                nameof(ActivityViewModel.Steps));

            var caloriesLabel = new Label
            {
                FontSize = 20,
                HorizontalOptions = LayoutOptions.Center
            };

            caloriesLabel.SetBinding(Label.TextProperty,
                nameof(ActivityViewModel.Calories));

            var distanceLabel = new Label
            {
                FontSize = 20,
                HorizontalOptions = LayoutOptions.Center
            };

            distanceLabel.SetBinding(Label.TextProperty,
                nameof(ActivityViewModel.Distance));

            Content = new VerticalStackLayout
            {
                Padding = 20,
```

```

        VerticalOptions = LayoutOptions.Center,
        Spacing = 12,
        Children =
        {
            stepsLabel,
            caloriesLabel,
            distanceLabel
        }
    };
}
}
}

```

Лістинг Е.1 – Головна сторінка застосунку (MainPage)

```

using System.ComponentModel;
using System.Runtime.CompilerServices;
namespace CrossHealthX.ViewModels
{
    public class ActivityViewModel : INotifyPropertyChanged
    {
        private int _steps;
        private int _calories;
        private double _distance;

        public int Steps
        {
            get => _steps;
            set
            {
                if (_steps != value)
                {
                    _steps = value;
                    OnPropertyChanged();
                }
            }
        }
    }
}

```

```

public int Calories
{
    get => _calories;
    set
    {
        if (_calories != value)
        {
            _calories = value;
            OnPropertyChanged();
        }
    }
}

public double Distance
{
    get => _distance;
    set
    {
        if (_distance != value)
        {
            _distance = value;
            OnPropertyChanged();
        }
    }
}

public event PropertyChangedEventHandler? PropertyChanged;

protected void OnPropertyChanged([CallerMemberName] string
propertyName = null)
{
    PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
}
}
}

```

Лістинг E.2 – ActivityViewModel

```

using Microsoft.Maui.Essentials;

```

```

using System.Threading.Tasks;
namespace CrossHealthX.Services
{
    public class LocationService
    {
        public async Task<Location?> GetCurrentLocationAsync()
        {
            try
            {
                var request = new GeolocationRequest(GeolocationAccuracy.High);
                var location = await Geolocation.GetLocationAsync(request);
                return location;
            }
            catch
            {
                return null;
            }
        }
    }
}

```

Лістинг E.3 – LocationService

```

using SQLite;
using System;
namespace CrossHealthX.Models
{
    public class Activity
    {
        [PrimaryKey, AutoIncrement]
        public int Id { get; set; }
        public int Steps { get; set; }
        public int Calories { get; set; }
        public double Distance { get; set; }
        public DateTime Date { get; set; }
    }
}

```

```
}
```

Лістинг E.4 – Модель Activity

```
using SQLite;
using System.Collections.Generic;
using System.Threading.Tasks;
using CrossHealthX.Models;
namespace CrossHealthX.Data
{
    public class ActivityDatabase
    {
        readonly SQLiteAsyncConnection _database;
        public ActivityDatabase(string dbPath)
        {
            _database = new SQLiteAsyncConnection(dbPath);
            _database.CreateTableAsync<Activity>().Wait();
        }
        public Task<List<Activity>> GetActivitiesAsync()
        {
            return _database.Table<Activity>().ToListAsync();
        }
        public Task<int> SaveActivityAsync(Activity activity)
        {
            return _database.InsertAsync(activity);
        }
    }
}
```

Лістинг E.5 – ActivityDatabase

```
using Microcharts;
using SkiaSharp;
using System.Collections.Generic;
namespace CrossHealthX.ViewModels
{
```

```

public class StatsViewModel
{
    public Chart ActivityChart { get; }

    public StatsViewModel()
    {
        var entries = new List<ChartEntry>
        {
            new ChartEntry(5000)
            {
                Label = "ПН",
                ValueLabel = "5000",
                Color = SKColor.Parse("#2ecc71")
            },
            new ChartEntry(7500)
            {
                Label = "БТ",
                ValueLabel = "7500",
                Color = SKColor.Parse("#2ecc71")
            },
            new ChartEntry(6200)
            {
                Label = "Ср",
                ValueLabel = "6200",
                Color = SKColor.Parse("#2ecc71")
            },
            new ChartEntry(8200)
            {
                Label = "ЧТ",
                ValueLabel = "8200",
                Color = SKColor.Parse("#2ecc71")
            },
            new ChartEntry(4000)
            {
                Label = "ПТ",
                ValueLabel = "4000",
                Color = SKColor.Parse("#2ecc71")
            },
        }
    }
}

```

```

        new ChartEntry(10000)
        {
            Label = "Сб",
            ValueLabel = "10000",
            Color = SKColor.Parse("#2ecc71")
        },
        new ChartEntry(7200)
        {
            Label = "Нд",
            ValueLabel = "7200",
            Color = SKColor.Parse("#2ecc71")
        }
    };

    ActivityChart = new LineChart
    {
        Entries = entries,
        LineMode = LineMode.Straight,
        LineSize = 6,
        PointMode = PointMode.Circle,
        PointSize = 10,
        LabelTextSize = 30
    };
}
}
}

```

Лістинг E.6 – StatsViewModel (Microcharts)

```

using Microsoft.Maui.ApplicationModel;

public class NotificationService
{
    public void ShowNotification(string message)
    {
        MainThread.BeginInvokeOnMainThread(() =>
        {

```

```
Application.Current?.MainPage?.DisplayAlert(  
    "CrossHealthX",  
    message,  
    "OK"  
);  
});  
}  
}
```

Лістинг E.7 – NotificationService