Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Метод проєктування мобільних застосунків

(тема)

Виконав:

студент II курсу, групи СПм-23-2

Нижник В.В.
(прізвище, ініціали)

Спеціальність

123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: ст. викл. Єрьоміна Н. С.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Нижнику Віталію Валерійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Метод проєктування мобільних застосунків

затверджена наказом по університету від " 22 " листопада 2024 р. № 1236 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 20 січня 2025 р.

3. Вхідні дані до роботи 1) типи мобільних застосунків: нативні, гібридні, веб-застосунки; 2) платформи для розробки: Android, iOS, HarmonyOS; 3) фреймворки та технології: Flutter, React Native, PWA, Swift, Kotlin, Java; 4) ключові параметри розробки: продуктивність, масштабованість, кросплатформність, UX/UI; 5) обмеження і вимоги: бюджет на розробку, час на створення MVP, доступ до функцій пристроїв; 6) бізнес-контекст: цільова аудиторія, галузь використання; 7) інтеграційні вимоги: використання API та зовнішніх сервісів.

4. Перелік питань, що потрібно опрацювати у роботі

1) огляд основних підходів до розробки мобільних застосунків;
2) аналіз платформ розробки та їхніх особливостей;
3) вибір та обґрунтування фреймворків і технологій для реалізації застосунків;
4) порівняння ключових параметрів розробки;
5) формування рекомендацій щодо вибору оптимальної технології залежно від цілей;
6) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) _____

Слайд-презентація – 16 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Огляд типів мобільних застосунків	26.11.24-30.11.24	
2	Вибір та обгрунтування методики дослідження	02.12.24-05.12.24	
3	Аналіз методів розробки	06.12.24-28.12.24	
4	Створення рекомендацій з вибору методу роз-	29.12.24-03.01.25	
	робки		
5	Оформлення матеріалів кваліфікаційної роботи	04.01.25-07.01.25	
6	Подання кваліфікаційної роботи керівникові	08.01.25-11.01.25	
	та її попередній захист		
7	Подання кваліфікаційної роботи на рецензу-	13.01.25-17.01.25	
	вання		

Дата видачі завдання 25 листопада 2024 р.

Студент _____
(підпис)

Керівник роботи _____ ст. викл. Єрьоміна Н. С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 82 с., 16 рис., 1 табл., 1 дод., 29 джерел.

МОБІЛЬНІ ЗАСТОСУНКИ, НАТИВНІ ЗАСТОСУНКИ, ГІБРИДНІ ЗАСТОСУНКИ, ВЕБ-ЗАСТОСУНКИ, ANDROID, IOS, HARMONY OS, КРОСПЛАТФОРМНІСТЬ, UX/UI, FLUTTER, REACT NATIVE, PWA, API, ІНТЕГРАЦІЯ, ФРЕЙМВОРК, ПРОДУКТИВНІСТЬ, МАСШТАБОВАНІСТЬ.

Метою кваліфікаційної роботи є дослідження сучасних підходів до розробки мобільних застосунків, аналіз їхніх переваг і недоліків, а також формування рекомендацій щодо вибору оптимальної технології розробки з урахуванням бізнес-цілей, ресурсів і вимог до функціональності продукту.

У ході виконання кваліфікаційної роботи проведено аналіз трьох основних підходів до створення мобільних застосунків: нативної, гібридної та веб-розробки. Розглянуто особливості роботи з платформами Android, iOS і HarmonyOS, їхні архітектурні рішення, можливості інтеграції апаратних функцій і забезпечення продуктивності. Виявлено переваги нативної розробки у створенні високопродуктивних застосунків із глибокою інтеграцією функцій пристрою. Гібридні застосунки відзначено як економічне та швидке рішення для кросплатформних проектів, тоді як веб-застосунки визнані оптимальними для швидкої розробки та масштабування в межах обмеженого бюджету.

У результаті сформовано рекомендації для вибору оптимального методу розробки залежно від цільової аудиторії, бюджетів та вимог проекту. Отримані результати мають практичне значення для створення ефективних і конкурентоспроможних мобільних застосунків у різних галузях.

ABSTRACT

Master’s thesis: 82 pages, 16 figures, 1 table, 1 appendices, 29 sources.

MOBILE APPLICATIONS, NATIVE APPLICATIONS, HYBRID APPLICATIONS, WEB APPLICATIONS, ANDROID, IOS, HARMONY OS, CROSS-PLATFORM, UX/UI, FLUTTER, REACT NATIVE, PWA, API, INTEGRATION, FRAMEWORK, PRODUCTIVITY, SCALABILITY.

The major goal of this thesis is to explore modern approaches to mobile application development, analyze their advantages and disadvantages, and provide recommendations for selecting the optimal development technology, considering business goals, resources, and functional requirements of the product.

In order to achieve the goal, an analysis was conducted on the three main approaches to mobile application development: native, hybrid, and web-based. The specifics of working with platforms such as Android, iOS, and HarmonyOS were examined, along with their architectural solutions, the possibilities for integrating hardware functions, and ensuring performance. The advantages of native development were identified, particularly for creating high-performance applications with deep integration of device functions. Hybrid applications were recognized as an economical and rapid solution for cross-platform projects, while web applications were deemed optimal for fast development and scalability within limited budgets.

As a result, recommendations were formulated for selecting the most appropriate development method depending on the target audience, budgets, and project requirements. The findings have practical significance for creating efficient and competitive mobile applications across various industries.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Бекенд – серверна частина програми, що обробляє дані

Кросплатформність – здатність програмного забезпечення працювати на різних платформах

Фреймворк – структура або каркас для розробки програмного забезпечення

API – Application Programming Interface (програмний інтерфейс додатка)

CI/CD – Continuous Integration / Continuous Delivery (безперервна інтеграція та доставка)

CSS – Cascading Style Sheets (каскадні таблиці стилів)

Flutter – кросплатформний фреймворк для розробки мобільних застосунків

HarmonyOS – операційна система компанії Huawei для різних пристроїв

HTML – HyperText Markup Language (мова розмітки гіпертексту)

JS – JavaScript (мова програмування для веб-застосунків)

MVP – Minimum Viable Product (мінімально життєздатний продукт)

PWA – Progressive Web Applications (прогресивні веб-застосунки)

React Native – Фреймворк для створення кросплатформних застосунків із використанням JavaScript

SDK – Software Development Kit (набір засобів для розробки програмного забезпечення)

UI – User Interface (користувацький інтерфейс)

UX – User Experience (досвід користувача)

ВСТУП

Збільшення використання мобільних пристроїв у повсякденному житті впливає на різні сфери, стимулюючи компанії до розробки цифрових рішень, що відповідають вимогам користувачів. Мобільні застосунки стали ключовим інструментом для забезпечення зручного доступу до послуг, а також важливим каналом для комунікації та залучення клієнтів. Їх значення помітне не лише в сфері розваг, але й у таких критичних галузях, як банківські послуги, охорона здоров'я, освіта та державні сервіси.

Конкуренція на ринку мобільних застосунків стає дедалі інтенсивнішою, тому ефективний процес розробки є основою для створення продуктів, які виділяються своєю якістю, функціональністю та стабільністю. Це вимагає ретельного підходу до вибору архітектури застосунка, середовища розробки, а також методів тестування, що забезпечують стабільну роботу продукту навіть за умов високого навантаження.

Крім того, важливим є не лише сам процес розробки, але й етапи підтримки та оновлення мобільних застосунків. Регулярне оновлення допомагає зберігати актуальність застосунка, підтримувати відповідність новим стандартам безпеки та забезпечувати нові функції, які підвищують зацікавленість користувачів.

Водночас зростає попит на кросплатформні рішення, які дозволяють охопити більшу аудиторію за рахунок доступності на різних операційних системах, як-от Android та iOS.

Метою цього дослідження є аналіз і систематизація сучасних методів розробки мобільних застосунків. Окрім цього, передбачається створити рекомендацію, що дозволить вибрати найбільш ефективну технологію для створення застосунків з урахуванням таких факторів, як доступ до функцій пристрою, швидкість роботи, інтеграція з іншими системами та зручність масштабування в майбутньому.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Використання мобільних пристроїв

Ресурс досліджень Statista повідомляє, що станом на початок 2024 року кількість користувачів смартфонів у світі досягла близько 6,9 мільярда, що становить 87% від загального населення.

Прогнозується, що до 2025 року понад 72% глобального інтернет-трафіку надходитиме з мобільних пристроїв. Це зростання зумовлене розвитком 5G-мереж та зниженням вартості мобільного інтернету [13].

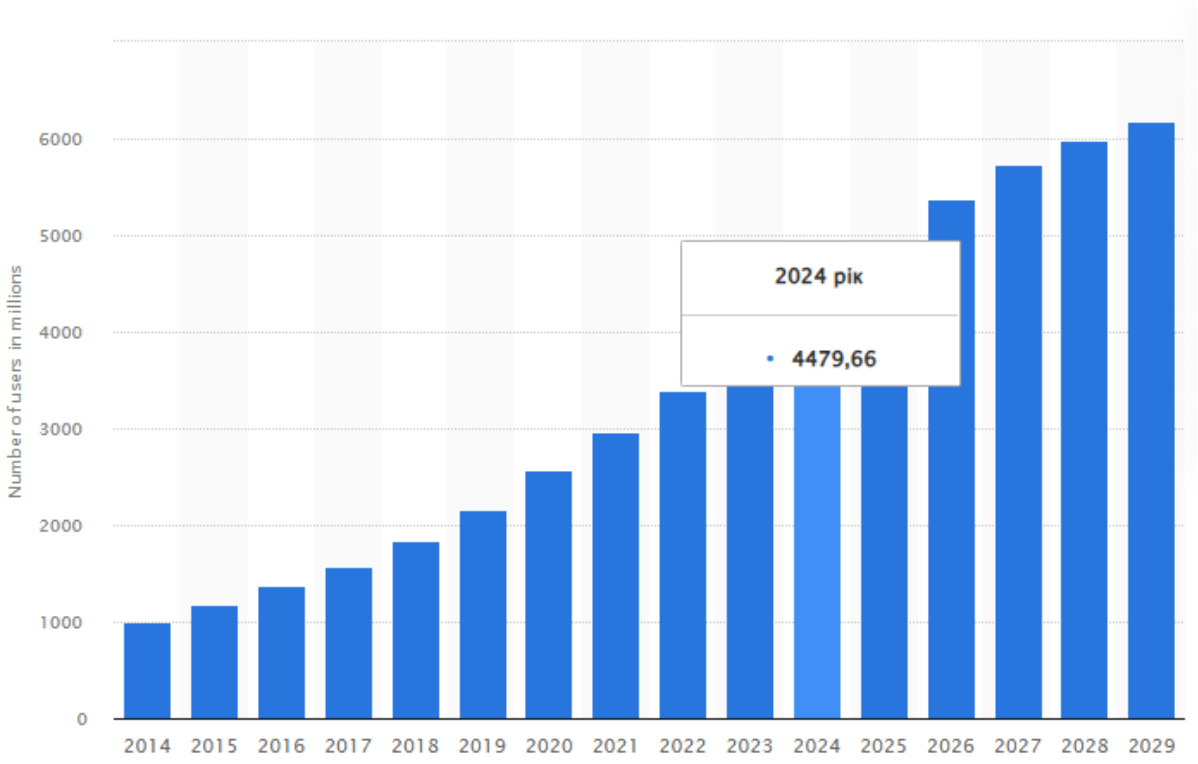

Рисунок 1.1 – Тенденція та прогнози зростання кількості смартфонів у світі

Звіт Global Digital за 2024 рік вказує, що середній час, проведений користувачем у мобільному інтернеті, зріс до 5,5 годин на день.

У звіті також зазначено, що 55% інтернет-користувачів віддають перевагу мобільним пристроям для доступу до мережі [7].

Згідно з даними Ericsson Mobility Report (2024), до кінця року близько 40% глобальних мобільних підключень припадатиме на 5G. Це дозволило покращити швидкість передачі даних та стабільність інтернет-з’єднання, що стимулює попит на мобільний контент та сервіси, особливо потокове відео, доповнену реальність та онлайн-ігри [8].

Смартфони як основний інструмент для інтернет-покупок: За статистикою, понад 73% транзакцій в електронній комерції здійснюються на мобільних пристроях, причому очікується подальше зростання цього показника. Близько 55% покупців зазначають, що здійснили покупку після перегляду продукту в соціальних мережах [25].

1.2 Мобільні операційні системи

З моменту появи смартфонів на ринку функціонувала значна кількість мобільних операційних систем, серед яких Samsung Bada, Symbian, Firefox OS і Windows Mobile/Phone. Проте жодна з цих платформ не змогла скласти серйозну конкуренцію Android і iOS, які залишаються лідерами завдяки компаніям Google і Apple відповідно.

Наразі на ринку також з'явилася нова операційна система – HarmonyOS від компанії Huawei, що має на меті об'єднати різні пристрої в єдину екосистему, включаючи смартфони, телевізори та інші "розумні" пристрої.

Також слід відзначити такі платформи, як Fuchsia від Google, яка розробляється для інтеграції з IoT-пристроями, а також Tizen від Samsung, який продовжує використовуватися в розумних телевізорах та смарт-годинниках.

Станом на сьогодні, згідно з даними ірландської компанії StatCounter, що займається аналізом інтернет-трафіку, розподіл популярності ОС у світі виглядає, як показано на рисунку 1.2. Можна побачити, що Android є значно

популярнішою системою за iOS та HarmonyOS [11].

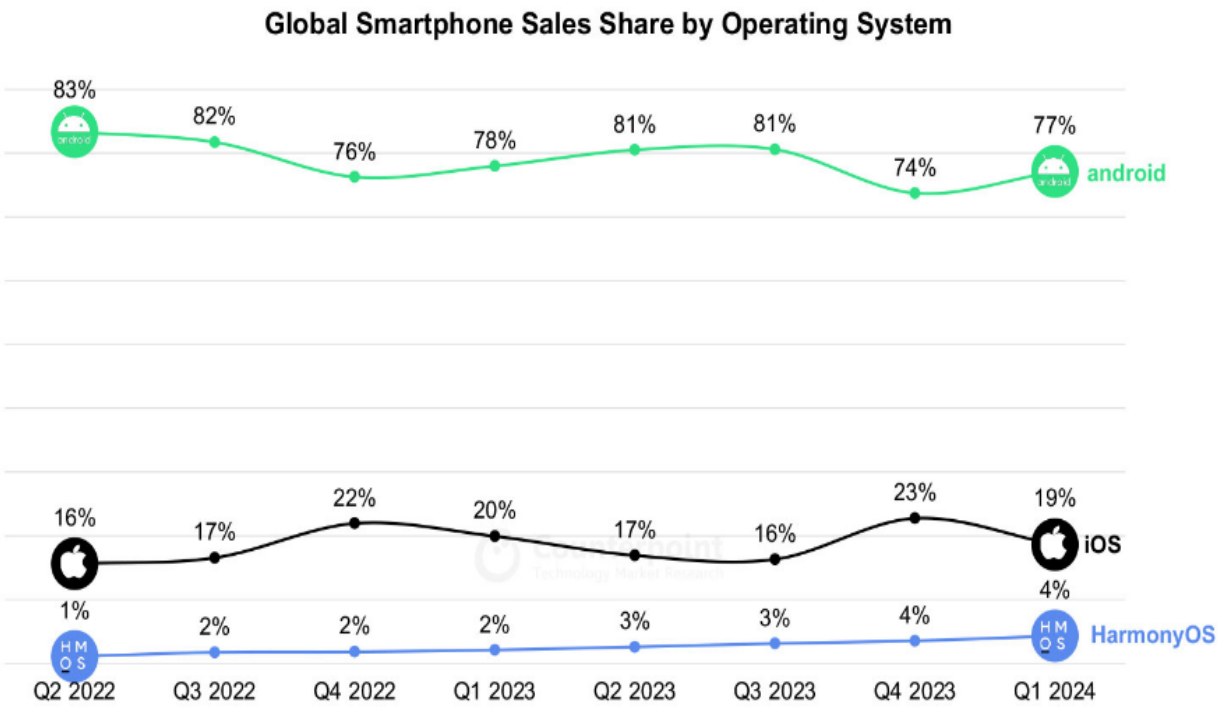

Рисунок 1.2 – Розподіл мобільних операційних систем у світі

Операційна система Android приваблює численних нових виробників бюджетних смартфонів, що сприяє її популярності. Проте при створенні застосунків, орієнтованих на конкретні країни та їх ринки, потрібно також аналізувати локальне розповсюдження ОС на мобільних пристроях [11]. До прикладу, у США останнім часом iOS отримала більшу популярність, ніж Android, а на китайському ринку оберти набирає HarmonyOS. Дана статистика наведена на рисунках 1.3 та 1.4.

Відтак, при створенні мобільного застосунка важливо орієнтуватися на підтримку основних мобільних ОС – Android, iOS, а також нових учасників, таких як HarmonyOS, адже вони можуть стати конкурентоспроможними на ринку та залучити нову аудиторію користувачів.

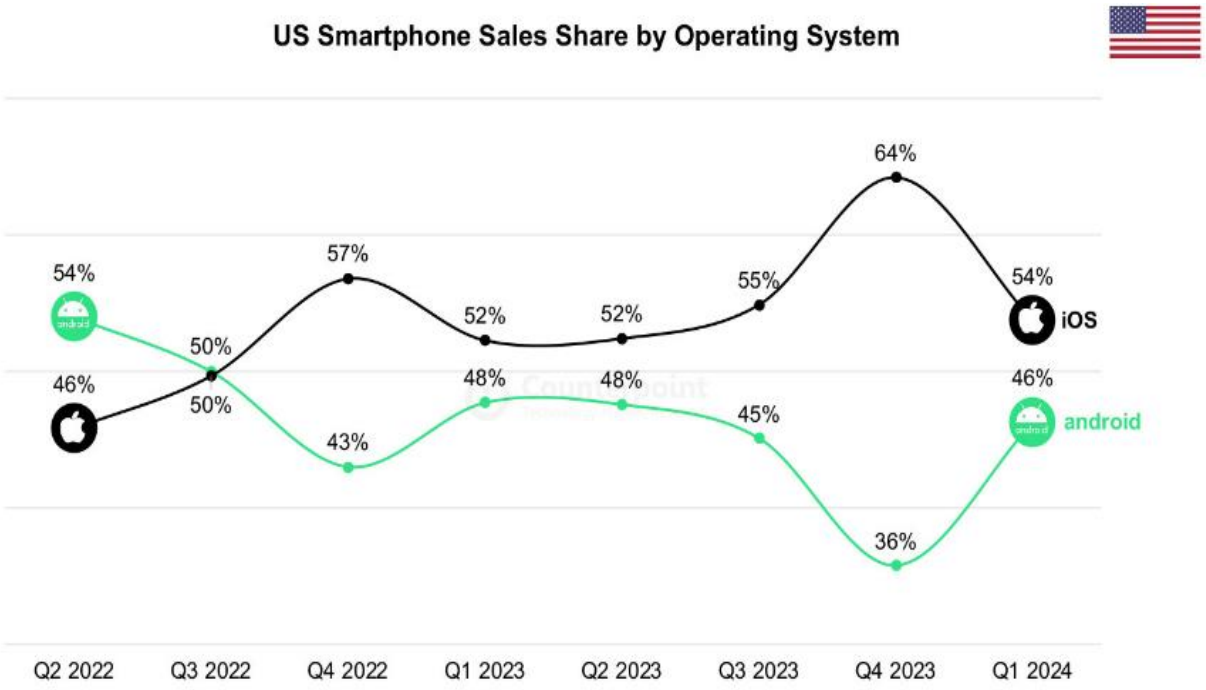

Рисунок 1.3 – Розподіл мобільних операційних систем у США

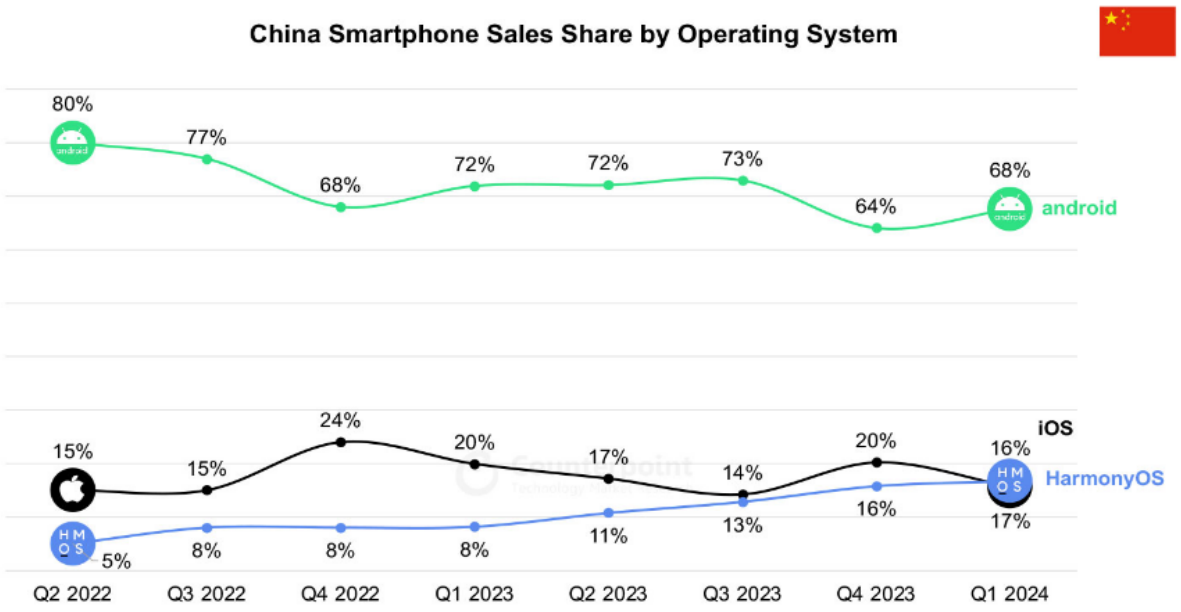

Рисунок 1.4 – Розподіл мобільних операційних систем у Китаї

1.3 Види мобільних застосунків

На сьогоднішній день існують три основні категорії мобільних

застосунків:

- нативні застосунки;
- веб-застосунки;
- гібридні застосунки.

Нативні застосунки – це програми, які розробляються спеціально для певної мобільної платформи, використовуючи мови програмування, що підтримуються цією платформою. Для Android основними мовами є Java і Kotlin, тоді як для iOS це Objective-C та Swift [15].

Ці застосунки забезпечують високу продуктивність, оскільки оптимізовані для конкретної операційної системи. Вони можуть безпосередньо взаємодіяти з різними функціями пристрою, такими як камера, мікрофон, геолокація, сенсори руху, а також системи сповіщень [20]. Завдяки цьому, нативні застосунки часто використовуються для створення ігор, графічно інтенсивних застосунків та застосунків, що вимагають складної обробки даних.

Користувачі можуть завантажувати такі застосунки з магазинів застосунків, і, завдяки вбудованим базам даних, вони можуть працювати навіть в офлайн-режимі. Основною перевагою є можливість створення інтерфейсу, що виглядає та поводиться органічно в межах платформи, забезпечуючи позитивний досвід використання.

Серед прикладів нативних застосунків можна виділити:

- Microsoft Office (Word, Excel, PowerPoint) – нативні програми для роботи з документами, які повністю використовують можливості мобільних платформ для продуктивності;

- Fitbit – застосунок для моніторингу фізичної активності, який використовує нативні функції пристроїв для збору даних про активність та здоров'я;

- Adobe Photoshop Express – мобільна версія популярного редактора зображень, яка використовує нативні елементи для забезпечення високої продуктивності і доступу до функцій редагування;

Веб-застосунки – це програми, які виконуються у веб-браузері, не вимагаючи установки на мобільний пристрій. Вони створюються за допомогою стандартних веб-технологій, таких як HTML5, CSS та JavaScript. Основна особливість веб-застосунків полягає в тому, що вони можуть працювати на будь-якому пристрої, підключеному до Інтернету, що робить їх універсальними [15].

Ці застосунки зазвичай легші за нативні та гібридні програми, оскільки не займають пам’ять на пристрої користувача, а всі дані зберігаються на серверах. Це також забезпечує користувачеві доступ до даних з будь-якого пристрою. Проте веб-застосунки потребують стабільного з'єднання з Інтернетом, і їх функціональність може бути обмежена в порівнянні з нативними застосунками, оскільки вони не можуть отримати доступ до всіх можливостей пристрою [2].

Прогресивні веб-застосунки (PWA) є новим підходом до веб-застосунків, дозволяючи їм мати можливості, подібні до нативних, такі як робота в офлайн-режимі, доступ до апаратного забезпечення пристрою та можливість установки на домашній екран [26]. Це надає користувачеві можливість отримувати переваги як веб-застосунків, так і нативних, зберігаючи при цьому простоту використання веб-інтерфейсу.

Прикладами веб-застосунків є:

- Slack – платформа для командного спілкування, доступна через веб-браузер, що дозволяє користувачам обмінюватися повідомленнями і файлами в режимі реального часу;
- Asana – інструмент для управління проектами, що працює через веб-інтерфейс і дозволяє командам організовувати та стежити за завданнями;
- Canva – онлайн-редактор графіки, який дозволяє користувачам створювати дизайни за допомогою веб-браузера без потреби в установці програмного забезпечення.

Гібридні застосунки об’єднують елементи як нативних, так і веб-застосунків. Вони створюються за допомогою веб-технологій (HTML, CSS,

JavaScript) і потім упаковуються у нативну оболонку, що дозволяє їм бути доступними через магазини застосунків [21].

Основна перевага гібридних застосунків полягає в їхній кросплатформній природі. Розробники можуть створювати один код, який працює на різних платформах, що значно знижує витрати часу і ресурсів. Однак, незважаючи на цю зручність, гібридні застосунки можуть страждати від проблем з продуктивністю, оскільки їхній код часто працює через інтерфейс, що додає додатковий шар абстракції. Це може призводити до затримок у відгуках і зниження швидкості, особливо в застосунках, що потребують інтенсивного використання графіки.

Гібридні програми також можуть мати різний вигляд на різних платформах, залежно від технології розробки, що може ускладнити забезпечення єдиного брендингу. Однак технології, такі як Apache Cordova, Xamarin, React Native, та Flutter постійно вдосконалюються, надаючи розробникам нові можливості для доступу до API пристроїв, зменшуючи таким чином обмеження, що існували раніше.

Гібридними застосунками є:

- Uber – застосунок для замовлення таксі, який поєднує нативний код із веб-технологіями для швидкого доступу до карт та інших функцій;
- Twitter – мобільний застосунок для мікроблогінгу, який поєднує елементи нативної і веб-розробки, забезпечуючи швидкий доступ до інформації та інтерактивність;
- Discord – платформа для спілкування, яка використовує гібридний підхід, дозволяючи користувачам взаємодіяти через голосові та текстові канали, доступні як через веб, так і через мобільний застосунок.

1.4 Вибір методу розробки

Вибір оптимального методу розробки мобільних застосунків є важливим етапом, що потребує ретельного аналізу. Рішення залежить від

цілей проекту, ресурсів команди, досвіду розробників, бюджету та термінів реалізації. Розглянемо основні фактори, які впливають на вибір між нативною, гібридною та веб-розробкою.

1.4.1 Нативна розробка

Нативні застосунки створюються для конкретних платформ, використовуючи специфічні мови програмування, такі як Swift для iOS та Kotlin або Java для Android. Цей підхід забезпечує максимальну продуктивність і можливості для використання апаратних функцій пристрою, таких як камери, GPS та сенсори.

Переваги:

- висока продуктивність: нативні застосунки працюють швидше, оскільки оптимізовані для конкретної платформи;
- кращий доступ до функцій пристрою: можливість інтеграції з апаратними компонентами забезпечує гнучкість у реалізації функціоналу;
- оптимізований UX/UI: інтерфейс виглядає та працює природно на цільовій платформі, що покращує досвід користувача.

Недоліки:

- високі витрати: розробка окремого коду для кожної платформи потребує значних ресурсів та часу;
- необхідність у великій команді: для реалізації проектів на кількох платформах потрібно більше розробників з різними навичками.

Нативна розробка є ідеальним вибором для великих підприємств з обмеженнями на бюджет, де критично важливими є продуктивність та інтеграція з апаратним забезпеченням.

1.4.2 Гібридна розробка

Гібридні застосунки поєднують елементи нативних та веб-застосунків,

використовують HTML, CSS і JavaScript, але упаковуються у нативний контейнер. Це дозволяє запускати одну й ту ж програму на різних платформах.

Переваги:

- кросплатформність: можливість запуску програми на кількох платформах з використанням одного коду, що скорочує час розробки;
- економія ресурсів: зниження витрат на розробку та обслуговування завдяки повторному використанню коду.

Недоліки:

- продуктивність: гібридні застосунки можуть бути менш швидкими у порівнянні з нативними, що може вплинути на користувацький досвід;
- обмежений доступ до функцій: деякі нативні функції можуть бути недоступні або потребувати додаткових плагінів.

Гібридний підхід підходить для проектів, що потребують швидкого виходу на ринок з обмеженим бюджетом, але не вимагають максимального використання апаратних можливостей.

1.4.3 Веб-розробка

Веб-застосунки працюють у браузері, не вимагаючи встановлення. Вони зазвичай реалізуються за допомогою HTML, CSS та JavaScript. Це економічно вигідний спосіб створення мобільних застосунків, особливо якщо у бізнесу вже є веб-сайт.

Переваги:

- швидка розробка: можливість швидкого розгортання без витрат на створення нативного коду;
- доступність: користувачі можуть отримувати доступ до програми через браузер, що не потребує її завантаження;
- легкість у масштабуванні: зміни можуть бути внесені на сервері без необхідності оновлення програми для кожного користувача.

Недоліки:

- обмежена продуктивність: може бути повільнішою і менш інтерактивною порівняно з нативними або гібридними застосунками;
- залежність від Інтернету: для повноцінної роботи потрібне стабільне з'єднання.

Веб-застосунки ідеально підходять для e-commerce або бізнесу, який не планує розміщувати застосунок у магазинах застосунків і не потребує розширеного функціоналу.

Таблиця 1.1 – Зведена таблиця порівняння типів мобільних застосунків

Характеристика	Нативні застосунки	Гібридні застосунки	Веб-застосунки
1	2	3	4
Вартість розробки	Висока	Середня	Низька
Продуктивність	Висока	Висока, наближена до нативного	Залежить від складності застосунка
Швидкість розробки	Низька	Середня	Висока
Вартість підтримки	Висока	Середня	Низька
Канал розподілу	Магазини застосунків	Магазини застосунків	Браузер та магазини застосунків
Доступ до інструментів девайсу	Повний	Повний / Наближений до повного	Частковий
Масштабованість	Обмежена	Висока	Висока
Підходить для складних застосунків	Так	Так, але з обмеженнями	Ні
Гнучкість оновлень	Низька	Висока	Висока

Продовження таблиці 1.1

1	2	3	4
Взаємодія з користувачем	Висока	Висока, але можливі обмеження	Залежить від інтернет-з'єднання
Безпека	Висока	Залежить від фреймворка	Залежить від безпеки веб-сервера
Наявність офлайн-режиму	Так	Може бути реалізований	Зазвичай немає
Досвід користувача (UX)	Оптимізований для платформи	Залежить від дизайну сайту	Залежить від дизайну сайту
Складність у тестуванні	Висока	Середня	Низька

2 НАТИВНІ ЗАСТОСУНКИ

Нативні застосунки (від англ. native – рідний) створюються спеціально для певної апаратно-програмної платформи, а їх розробка ведеться на мовах, які призначені для цих платформ. Кожна з них, такі як iOS, Android або HarmonyOS, має свої власні SDK (Software Development Kit – набір засобів для розробки), що містять специфічні інструменти і технології, пов'язані з певними мовами програмування. Це дозволяє нативним застосункам працювати швидко і коректно, оскільки вони оптимізовані для конкретних операційних систем.

Розробка нативних застосунків має ряд суттєвих переваг:

- висока продуктивність (нативні застосунки зазвичай працюють швидше, оскільки їх код оптимізується для конкретного пристрою, що дозволяє максимально використовувати його ресурси [2], це особливо важливо для застосунків, які обробляють великі обсяги даних або використовують графіку);
- максимальне використання можливостей платформи (завдяки тісній інтеграції з апаратним забезпеченням та операційною системою, нативні застосунки мають доступ до всіх функцій пристрою, таких як Bluetooth, NFC, камера, GPS тощо. Це робить їх ідеальними для програм, які потребують специфічних функцій пристрою);
- поліпшений користувацький інтерфейс (оскільки нативні застосунки використовують елементи інтерфейсу, що відповідають стилю платформи, користувачі отримують знайомий і зручний інтерфейс, що покращує загальний досвід взаємодії);
- краще позиціонування в магазинах застосунків (застосунки з високим рейтингом користувацького досвіду частіше займають кращі позиції в магазинах застосунків. Завдяки високій продуктивності та оптимізації, нативні застосунки отримують більше позитивних відгуків, що веде до

зростання популярності).

Нативні застосунки розробляються з метою вирішення конкретних завдань на конкретній платформі, що дозволяє досягти кращої відповідності між функціями застосунка і апаратними можливостями пристроїв. Це важливо для програм, які потребують точних даних про фізичне або географічне розташування, оскільки нативні застосунки можуть отримувати доступ до таких даних без затримок.

Крім того, оскільки нативні застосунки інтегруються з мобільною операційною системою, користувачі можуть насолоджуватися безперешкодним користувацьким досвідом. Це особливо важливо з огляду на різноманітність екранів та їх роздільну здатність. Сьогодні, коли ринок мобільних пристроїв пропонує безліч варіантів екранів, адаптація застосунка під різні формати стає критично важливою.

Проте, розробка нативних застосунків також має свої недоліки:

- високі витрати та тривалість розробки (процес створення окремих застосунків для різних платформ може бути дорогим і тривалим. Це пов'язано з необхідністю наймати різних спеціалістів, оскільки програмістам потрібно володіти знаннями специфічних мов, таких як Objective-C або Swift для iOS та Kotlin або Java для Android);
- несумісність з іншими операційними системами (нативні застосунки, створені для однієї платформи, не можуть бути безпосередньо запущені на іншій. Це означає, що для кожної платформи потрібно розробляти окремий застосунок, що підвищує витрати на підтримку);
- втрачені можливості для інших платформ (розробка лише для однієї операційної системи може призвести до втрати потенційної аудиторії. Обмеження цільового ринку знижує можливість отримання доходів, оскільки користувачі, які користуються іншими системами, залишаються поза увагою).

Нативні застосунки забезпечують високий рівень продуктивності та інтеграції з апаратними функціями, що робить їх привабливими для

розробників. Однак висока вартість розробки, тривалість процесу та обмеження у доступності на інших платформах можуть стати суттєвими недоліками. Перед тим як ухвалити рішення про розробку нативного застосунка, важливо ретельно проаналізувати потреби цільової аудиторії та ресурси компанії.

Незважаючи на виклики, нативні застосунки залишаються ключовим інструментом для забезпечення високої продуктивності та позитивного досвіду користувачів у світі мобільних технологій. Успішна реалізація нативних проектів може суттєво вплинути на бізнес та його здатність конкурувати на ринку.

2.1 Особливості розробки для Android

Android був випущений Google 23 вересня 2008 року як операційна система з відкритим кодом, заснована на ядрі Linux. Його розробка розпочалася ще у 2003 році в стартапі Android Inc., що спеціалізувався на програмному забезпеченні для камер, але згодом змінив свій фокус на мобільні пристрої. У 2005 році Google придбала компанію, після чого Android став основним проєктом у сфері мобільних технологій. З того часу система пройшла величезний шлях розвитку, стаючи з кожною новою версією більш потужною, гнучкою та функціональною. Наразі Android 15 є останньою версією, яка впроваджує нові стандарти безпеки, продуктивності та інтеграції штучного інтелекту [3].

Android вирізняється великою кількістю моделей пристроїв із різними характеристиками та вартістю. Станом на 2024 рік понад 70% мобільних пристроїв у світі працюють на Android, що робить його найпоширенішою мобільною платформою. Користувачі мають можливість обирати смартфони, планшети, телевізори, годинники та інші гаджети під будь-які вимоги. Така масштабна фрагментація, хоча й корисна для споживачів, створює значні виклики для розробників, яким доводиться адаптувати застосунки до різних

екранів, роздільних здатностей, об’ємів оперативної пам’яті та версій операційної системи [2]. На відміну від iOS, де кількість моделей пристроїв обмежена, Android-платформа охоплює величезний спектр конфігурацій.

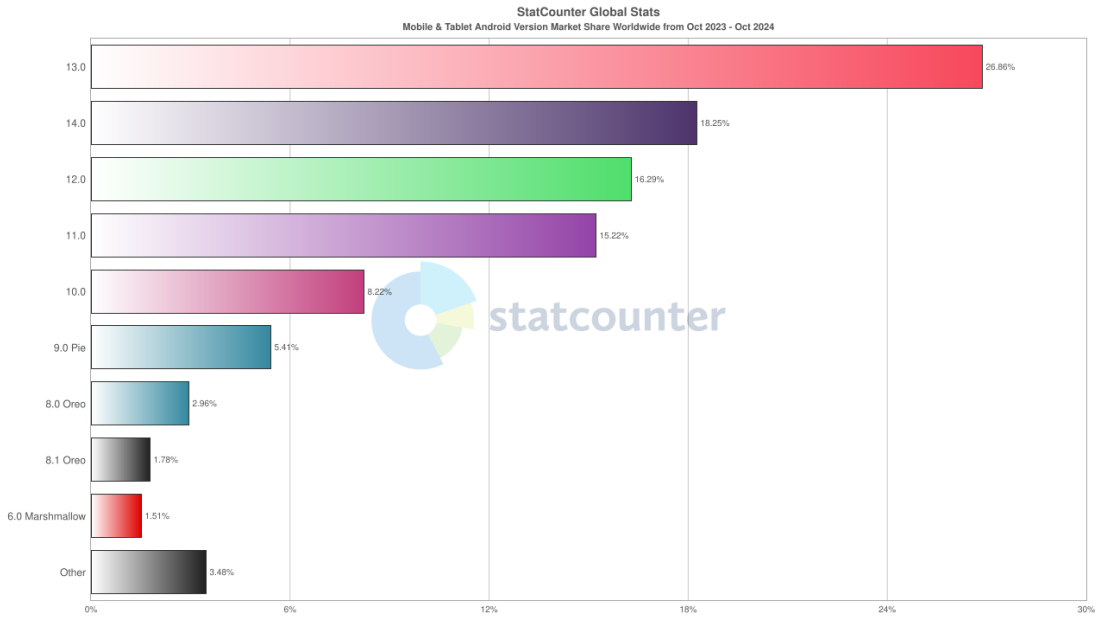

Рисунок 2.1 – Розподіл версій ОС Android на жовтень 2024 року

Рисунок 2.2 – Google Pixel Fold – приклад смартфону з нестандартним екраном

Однією з важливих характеристик Android-застосунків є їхня модульна структура, яка складається з окремих компонентів – активностей та фрагментів. Такий підхід дозволяє забезпечити роботу застосунків навіть на пристроях із низькою потужністю або малим об’ємом оперативної пам’яті. На відміну від iOS, де застосунок часто виступає як єдиний блок, Android дозволяє використовувати фрагментацію компонентів, що надає гнучкості в управлінні ресурсами. Зокрема, при нестачі пам’яті певні частини застосунка можуть автоматично вивантажуватися, зберігаючи загальну функціональність. Така архітектура є не лише зручною, але й критично важливою для застосунків, які розробляються для бюджетних або застарілих пристроїв.

Для розробників Android важливими викликами залишаються адаптація інтерфейсу користувача (UI) та користувацького досвіду (UX) під різноманітні екрани, особливо з огляду на багатовіконний режим, який став популярним у останніх версіях ОС. Android-пристрої мають широкий діапазон розмірів, починаючи від компактних смартфонів і закінчуючи планшетами та пристроями з подвійними екранами. Відповідно, UI/UX-дизайнери змушені враховувати різні щільності пікселів, можливість відображення в горизонтальній і вертикальній орієнтаціях, а також варіації оболонок від таких виробників, як Samsung, Xiaomi та One Plus. З кожною новою версією Android з’являються нові можливості, такі як адаптивні іконки, динамічні теми, персоналізовані віджети, що підвищують інтерактивність та спрощують налаштування пристроїв.

У порівнянні з iOS, Android пропонує вищу ступінь кастомізації та можливість для користувачів налаштовувати пристрій під свої потреби. Водночас, iOS відома своєю стабільністю та плавністю роботи застосунків, завдяки тому, що має менше пристроїв і більший контроль над програмним забезпеченням. Android підтримує більше типів застосунків, включаючи різноманітні віджети, системи багатовіконності, що значно збільшує

можливості для багатозадачності. Проте, багатовіконний режим часто вимагає додаткових зусиль з боку розробників, особливо для застосунків, які повинні адаптуватися до складних сценаріїв взаємодії [6].

Програмісти на Android використовують здебільшого Java та Kotlin. Починаючи з 2019 року, Google рекомендує Kotlin як основну мову завдяки її зручності, зменшеній кількості помилок і підвищеній ефективності написання коду. Kotlin інтегрує корисні функції, що відсутні в Java, такі як підтримка більшої кількості функцій, обробка помилок на рівні компіляції, що зменшує ризик виникнення збоїв у роботі застосунків [2]. Крім того, Android Studio, випущена Google у 2013 році, стала основним середовищем розробки для Android, надаючи вбудовані інструменти для тестування, налагодження та оптимізації продуктивності [18].

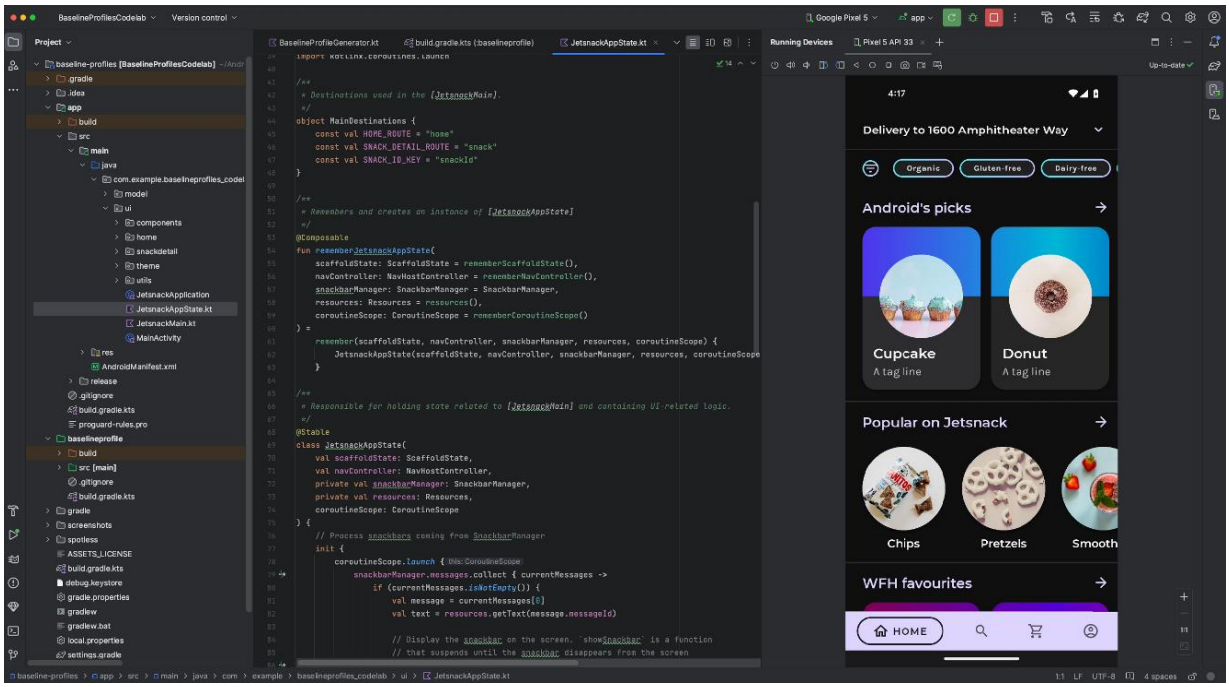

Рисунок 2.3 – Інтерфейс Android Studio

У 2024 році одним із ключових трендів для Android є інтеграція технологій штучного інтелекту та машинного навчання. Нова версія Android 15 надає розширені інструменти для розробки застосунків з елементами AI,

включаючи персоналізовані рекомендації, розпізнавання обличчя та голосу, а також оптимізацію продуктивності за допомогою машинного навчання. Крім того, зростає популярність технології доповненої реальності (AR), яка активно інтегрується в застосунки для освіти, ігор та електронної комерції [3].

Інший важливий тренд – це адаптація застосунків до складних екранів, таких як згинальні та подвійні екрани. Це відкриває нові можливості для UX-дизайну, проте ставить перед розробниками нові виклики, пов'язані з оптимізацією інтерфейсу та адаптацією контенту під такі унікальні дисплеї.

Розробка нативних застосунків для Android базується на створенні застосунків, що оптимально використовують можливості операційної системи Android та апаратного забезпечення пристроїв. Це забезпечує швидкість, ефективність і доступ до всіх функцій пристрою. Ось основні особливості:

- Android SDK (Software Development Kit). Android SDK містить бібліотеки, інструменти та документацію для розробки застосунків. Основні компоненти включають Android API для доступу до функцій ОС, а також емулятори, що дозволяють тестувати застосунок на різних версіях Android;

- архітектурні компоненти (Android надає готові компоненти архітектури (LiveData, ViewModel, Room, Navigation) для полегшення управління життєвим циклом застосунка, збереження даних і роботи з базами даних. Ці компоненти сприяють створенню стабільних і легко підтримуваних застосунків);

- доступ до апаратного забезпечення (нативні застосунки можуть використовувати всі апаратні функції пристрою, такі як камера, GPS, датчики, біометрія, що дає розробникам широкі можливості для створення застосунків із розширеними функціями);

- UI/UX особливості. Розробка для Android вимагає адаптації інтерфейсу під різні розміри екранів, орієнтацію, щільність пікселів та унікальні можливості оболонок різних виробників, що потребує особливої

уваги до зручності використання (UX);

- безпека (Android забезпечує шари безпеки, такі як контроль доступу до дозволів, шифрування даних, а також вимоги до сертифікації, що дозволяє розробникам захищати особисті дані користувачів і гарантувати безпеку застосунків).

Таким чином, Android продовжує розвиватися як гнучка, багатофункціональна операційна система, що дозволяє розробникам створювати застосунки для широкого спектра пристроїв і потреб користувачів.

2.2 Особливості розробки для iOS

Операційна система iOS, створена компанією Apple, була вперше випущена у 2007 році разом з першим iPhone. З того часу iOS пройшла значний шлях розвитку та регулярно оновлюється, додаючи нові функції та покращення безпеки. Остання версія, iOS 18 (випущена у 2024 році), включає багато нових можливостей, таких як поліпшені інструменти для налаштування інтерфейсу, підтримку машинного навчання та покращення конфіденційності користувачів. iOS також відома своєю оптимізацією для власного апаратного забезпечення Apple, що дозволяє досягти високої продуктивності та стабільності [2].

На відміну від Android, Apple підтримує обмежену кількість пристроїв (iPhone, iPad, iPod Touch), що дозволяє мінімізувати фрагментацію. Це полегшує тестування застосунків, оскільки розробникам потрібно адаптувати застосунок лише для кількох варіантів розмірів екранів та версій iOS. Проте iOS підтримує старіші моделі пристроїв досить довго, що забезпечує широке охоплення користувачів навіть на застарілих моделях, які продовжують отримувати оновлення безпеки та виправлення помилок [16].

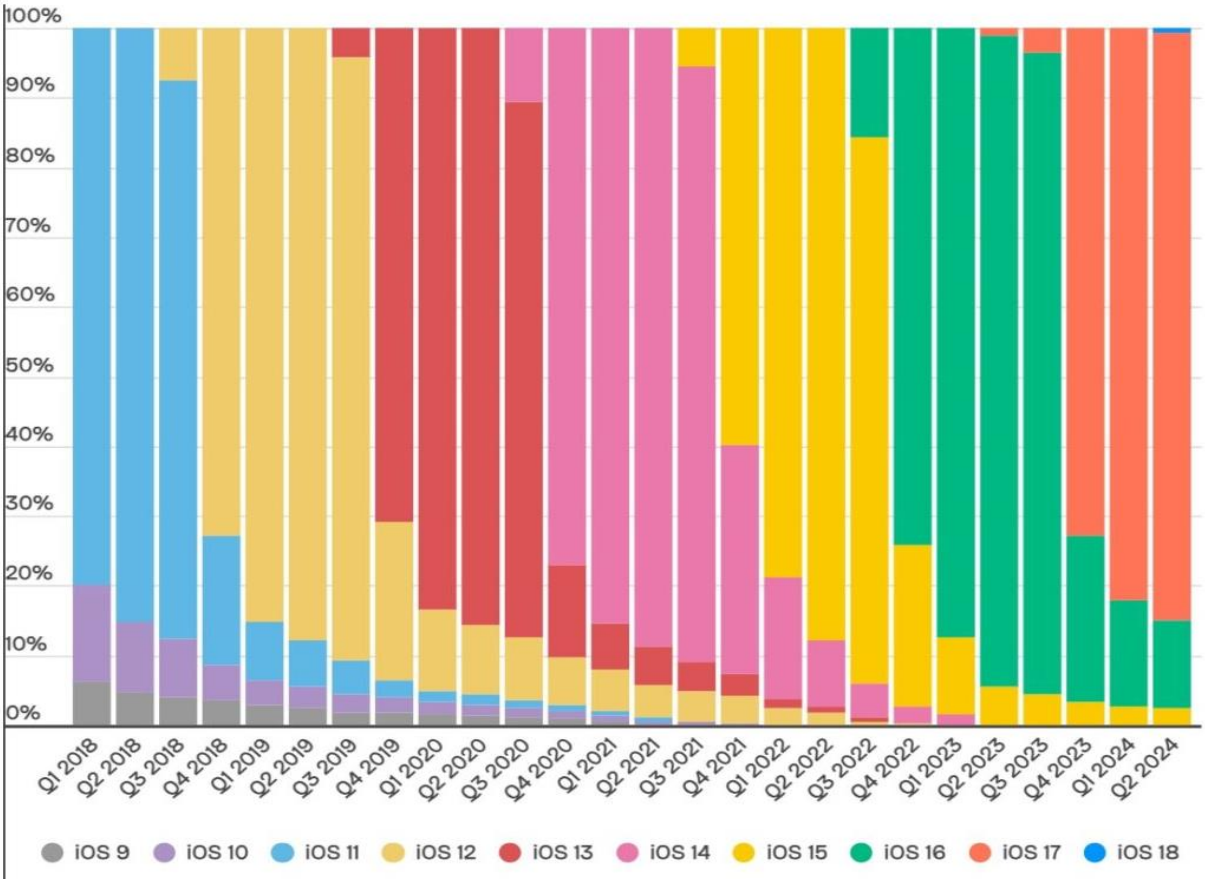

Рисунок 2.4 – Розподіл версій iOS на період 2018-2024 років

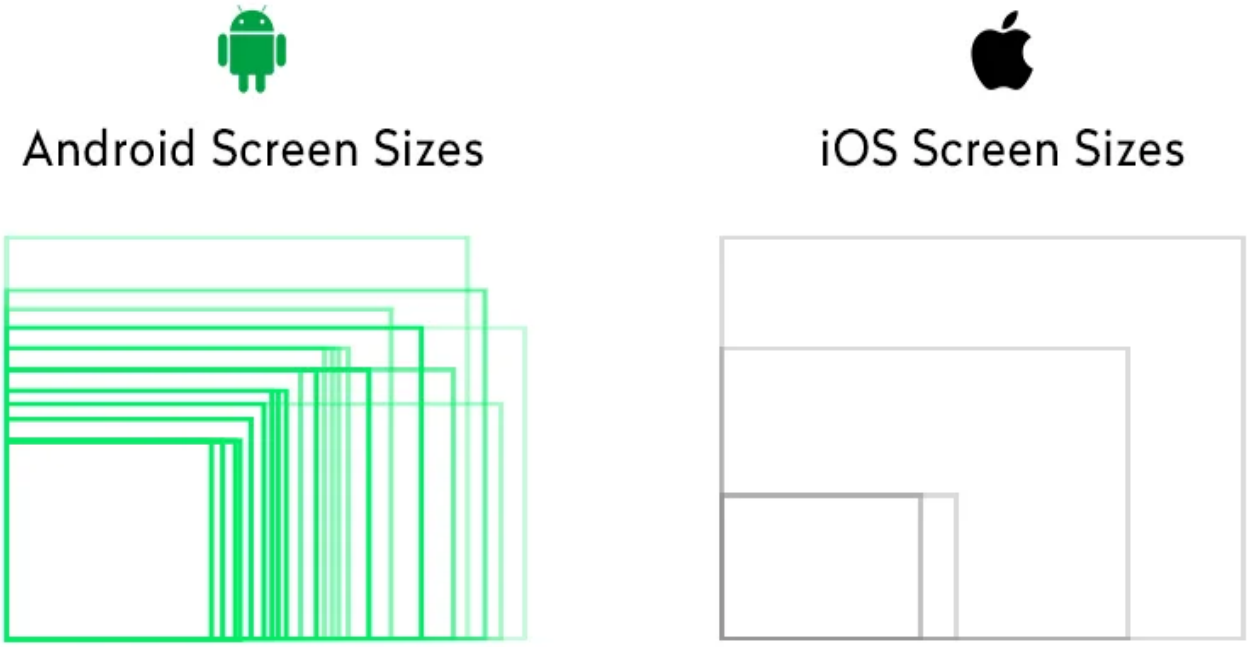

Рисунок 2.5 – Порівняння фрагментацій екранів пристроїв з Android та iOS

Розробка застосунків для iOS здійснюється за допомогою офіційних інструментів Apple, які забезпечують глибоку інтеграцію з екосистемою Apple і спрощують процес розробки.

Xcode – офіційне середовище розробки для iOS, яке включає симулятори, редактори коду, інструменти для налагодження, а також інтеграцію з App Store Connect. Xcode підтримує Swift та Objective-C, надає інструменти автозаповнення та налагодження інтерфейсу за допомогою Interface Builder. У 2024 році Xcode підтримує iOS 18 та включає нові можливості для роботи з оптимізованими бібліотеками [29].

Swift – основна мова програмування для iOS, що підтримує сучасні підходи до кодування, є безпечною та високопродуктивною. Swift щороку вдосконалюється, пропонуючи нові фреймворки та можливості для оптимізації застосунків [9].

UIKit та SwiftUI – UIKit є класичним фреймворком для створення користувацького інтерфейсу, тоді як SwiftUI, представлений у 2019 році, надає декларативний підхід до розробки UI та дозволяє швидко створювати інтерфейси, що адаптуються під різні пристрої. У iOS 18 SwiftUI отримав нові можливості для покращення анімацій та управління даними.

Core Data та CloudKit – фреймворки для роботи з локальними та хмарними базами даних. Core Data забезпечує локальне зберігання даних, тоді як CloudKit дозволяє зберігати дані в iCloud, що забезпечує синхронізацію між пристроями.

Instruments – набір інструментів для аналізу продуктивності та оптимізації коду, які дозволяють відслідковувати використання пам'яті, процесора та інших ресурсів. Це особливо важливо для підтримки високої швидкодії застосунка, зокрема на старіших пристроях.

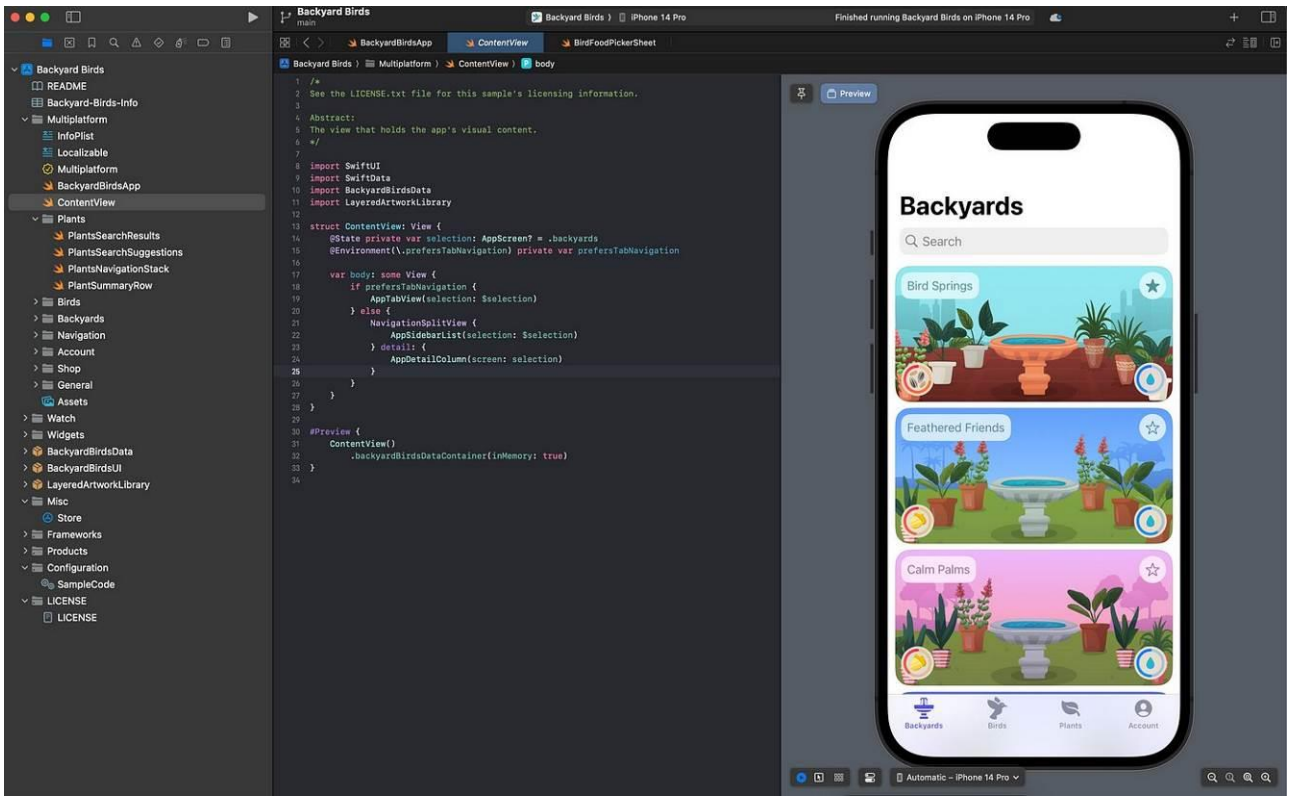

Рисунок 2.6 – Інтерфейс Xcode

Розробка інтерфейсу для iOS вимагає дотримання специфічних стандартів Apple.

Адаптивний дизайн – інтерфейс повинен коректно виглядати на різних пристроях, включаючи iPhone та iPad з різними розмірами екранів. Використання Auto Layout та Size Classes у Xcode дозволяє оптимізувати розташування елементів [14].

Жестове керування – iOS інтегрує велику кількість жестів, які додають зручності у використанні застосунків. Користувачі звикли до стандартів Apple, тому важливо слідувати рекомендаціям з використання жестів, таких як свайпи, тапи, подвійний тап та утримання.

Темна тема – з виходом iOS 13 Apple додала підтримку темної теми. Розробники мають адаптувати свої застосунки до цього режиму, що забезпечує кращу читабельність і комфорт для очей користувачів.

Сумісність з голосовим асистентом Siri – застосунки можуть

інтегруватися з Siri через Shortcuts API, дозволяючи користувачам запускати певні функції застосунка голосом.

Apple серйозно ставиться до захисту даних користувачів, і розробникам потрібно враховувати високі стандарти безпеки.

Серед головних вимог:

- політика доступу до даних – доступ до особистої інформації, такої як контакти або місцезнаходження, вимагає явного дозволу користувача;
- робота з конфіденційною інформацією – обробка даних повинна бути здійснена за шифруванням, Apple забезпечує API для забезпечення шифрування даних як у зберіганні, так і при передачі по мережі;
- App Store Review Guidelines – застосунки, що публікуються у App Store, проходять ретельну перевірку відповідності правилам Apple, це включає заборону на зловмисний код, рекламу та неналежний контент;
- App Tracking Transparency – політика, введена в iOS 14.5, що вимагає явного запиту на дозвіл відстеження даних користувача в рекламних цілях.

App Store є основною платформою для розповсюдження застосунків для iOS. Публікація застосунка включає кілька етапів: підготовку метаданих (скріншоти, опис), заповнення категорій, встановлення ціни або типу розповсюдження. App Store Connect – це платформа, де розробники керують застосунками, відстежують статистику завантажень, доходи та рейтинг.

Розробка нативних застосунків для iOS має низку унікальних аспектів, які роблять процес специфічним для платформи Apple та допомагають оптимізувати взаємодію користувача з застосунками на пристроях під управлінням iOS 18.

Сучасні архітектурні підходи – популярні архітектури MVC (Model-View-Controller), MVVM (Model-View-ViewModel) та VIPER дозволяють структурувати код і спрощують підтримку великих застосунків. SwiftUI, як декларативний фреймворк, у 2024 році спонукає до використання MVVM, що надає більше гнучкості та робить код більш зрозумілим.

Адаптивний інтерфейс – з підтримкою Auto Layout та Size Classes у

Xcode, розробники можуть створювати адаптивні інтерфейси, що автоматично підлаштовуються під різні пристрої (iPhone, iPad) і розміри екранів. iOS 18 додала розширені інструменти для кращого відображення інтерфейсу на нових пристроях, таких як iPad Pro з різною орієнтацією екрану [14].

Жести та інтерактивність – новітня підтримка Apple для жестів надає користувачам можливість виконувати дії, такі як свайпи, довге натискання, подвійний тап, що стало стандартом для iOS. Розробники можуть легко інтегрувати ці функції через UIKit та SwiftUI [14].

Інтеграція з екосистемою Apple – можливості iOS дозволяють застосункам взаємодіяти з іншими продуктами Apple, такими як iCloud для синхронізації даних, Apple Pay для платежів, HealthKit для збереження медичних даних і навіть з HomeKit для управління пристроями розумного дому.

App Tracking Transparency (ATT) – політика, введена у 2021 році та покращена у iOS 18, яка забезпечує прозорість і контроль користувача над трекінгом його даних, особливо для рекламних цілей. Це є важливою особливістю, що сприяє підвищенню конфіденційності користувачів.

Безпека та конфіденційність – Apple надає розробникам API для реалізації шифрування даних, двофакторної автентифікації та захисту користувацьких даних. Доступ до чутливих даних, таких як місце розташування та контакти, можливий лише після отримання дозволу користувача.

Рекомендації щодо інтерфейсу – Apple надає чіткі керівництва щодо дизайну, які допомагають створювати привабливі та зручні застосунки. Застосунки повинні підтримувати темну тему (Dark Mode), бути адаптованими для всіх розмірів екранів та забезпечувати високу якість UI/UX, що користувачі iOS очікують від застосунків в App Store.

Такі особливості роблять iOS розробку ще більш орієнтованою на потреби користувачів, дозволяючи створювати застосунки високої якості з

максимальним рівнем безпеки та зручності для користувача.

2.3 Особливості розробки для HarmonyOS

HarmonyOS була випущенна компанією Huawei в 2019 році як відповідь на обмеження з боку уряду США, які обмежили доступ Huawei до Android [12]. Метою було створити незалежну, універсальну операційну систему для різних типів пристроїв, від смартфонів і носимих пристроїв до пристроїв IoT [2]. З часом HarmonyOS розвивався і в 2024 році досяг версії HarmonyOS 4, поширившись на понад 700 мільйонів пристроїв по всьому світу. Ця система побудована на основі мікроядра, що забезпечує високий рівень безпеки і ефективності, що дозволяє адаптувати її до різноманітних пристроїв і сценаріїв використання.

За результатами участі у програмі тестування SDK для платформи HarmonyOS 2.0 у 2021 році спостерігачі стверджують, що код версії HarmonyOS для IoT-пристроїв, опублікований у репозиторії OpenHarmony, є кодом, наданим в емуляторі HarmonyOS 2.0 і стверджує, що він жодним чином не дублюється. У першому випадку система базується на власному мікроядрі LiteOS, тоді як HarmonyOS 2.0 надає системне середовище Android 10 на базі ядра Linux і набір типових Android-застосунків.

Розробка застосунків для HarmonyOS здійснюється за допомогою офіційних інструментів: Huawei DevEco Studio, HarmonyOS SDK, Ark Compiler, Huawei Mobile Services,

Huawei DevEco Studio – офіційне середовище розробки для HarmonyOS, яке базується на Android Studio і підтримує Java, C++, а також власну мову Huawei – ArkTS, засновану на TypeScript. DevEco Studio дозволяє працювати з симуляторами, профілювати застосунки, тестувати продуктивність і забезпечує зручні інструменти для дизайну інтерфейсу користувача.

Рисунок 2.7 – Інтерфейс HarmonyOS на різних пристроях

HarmonyOS SDK. Пакет розробки HarmonyOS (SDK) включає інструменти та ресурси, які дозволяють створювати застосунки, оптимізовані для екосистеми Huawei. SDK надає API для таких функцій, як розподілені фреймворки застосунків, що дозволяють застосункам працювати на різних пристроях без додаткової адаптації. У новіших версіях HarmonyOS значна увага приділяється мові ArkTS, що базується на TypeScript і забезпечує високу продуктивність та оптимізоване управління застосунками.

Ark Compiler – це рішення від Huawei, яке дозволяє компілювати код безпосередньо в машинний рівень інструкцій, що сприяє підвищенню продуктивності застосунків. Компілятор підтримує різні мови, що робить його зручним для розробників з різних середовищ програмування.

Huawei Mobile Services (HMS) Core є важливим елементом для розробки застосунків HarmonyOS. Він включає інструменти для аналітики, геолокації, платіжних систем, картографії тощо, подібні до Google Services. HMS Core швидко розвивається, додаючи конкурентоспроможні функції, як-от Petal Maps та інші сервіси на основі ШІ.

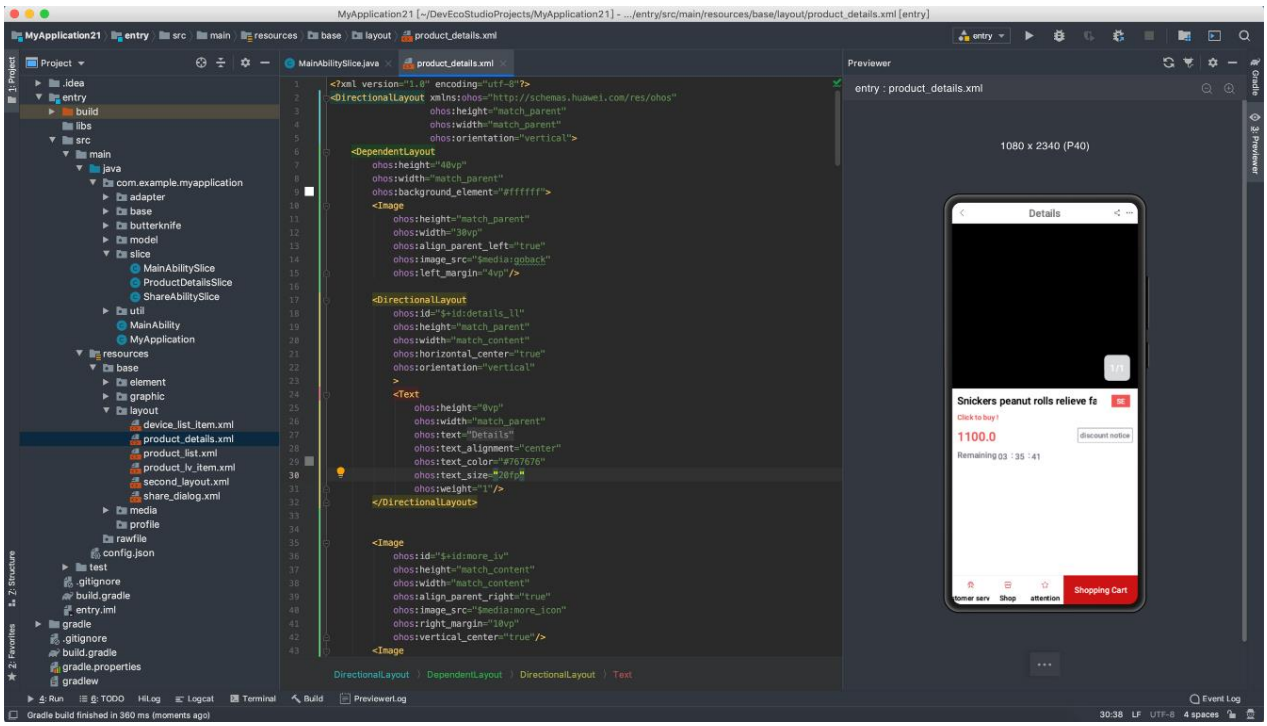

Рисунок 2.8– Інтерфейс Huawei DevEco Studio

Розробка застосунків для HarmonyOS має низку унікальних аспектів, що відрізняють її від традиційних платформ, таких як Android або iOS:

Підтримка Java і ArkTS. ArkTS, як новий стандарт для HarmonyOS, дозволяє створювати швидші застосунки завдяки оптимізації компіляції. Для розробників, які працювали з Java на Android, процес адаптації до HarmonyOS буде відносно простим, оскільки основні концепції залишаються схожими.

Розподілені можливості. HarmonyOS відрізняється розподіленими можливостями, що дозволяють застосункам безперешкодно функціонувати на різних пристроях. Розробники можуть створювати застосунки, що використовують ресурси та інтерфейси підключених пристроїв, що дозволяє реалізовувати нові сценарії використання, як-от управління домашніми пристроями за допомогою смартфона [1].

ArkTS та ArkUI Framework. Введення ArkTS в HarmonyOS підвищує ефективність розробки, пропонуючи високопродуктивне, реактивне середовище для створення інтерфейсів, оптимізоване для HarmonyOS. ArkUI

забезпечує декларативний підхід до розробки інтерфейсів, що спрощує створення адаптивних інтерфейсів для різних пристроїв Huawei.

Розширені можливості ШІ. HarmonyOS інтегрує розширені можливості ШІ на основі фреймворку MindSpore. Нативні застосунки можуть використовувати ШІ для таких завдань, як розпізнавання голосу, обличчя та управління жестами. Віртуальний асистент Celia також підтримує інтеграцію ШІ, що дозволяє реалізувати складні завдання.

Покращена безпека. HarmonyOS забезпечує високий рівень безпеки завдяки мікроядру та багаторівневим методам захисту користувацьких даних. Huawei використовує фреймворк Star Shield, що забезпечує захист даних, цілісності пристрою та дозволів застосунків, що є корисним для розробників, які працюють із конфіденційними даними або корпоративними застосунками.

Зручна робота на кількох пристроях. У HarmonyOS реалізовані інструменти для управління даними та завданнями на різних пристроях, що дозволяє користувачам взаємодіяти із застосунком на смартфоні, планшеті чи телевізорі одночасно, забезпечуючи плавний обмін інформацією між пристроями.

HarmonyOS представляє унікальний підхід до розробки мобільних застосунків і застосунків для різних пристроїв, що особливо підходить для використання апаратної екосистеми Huawei. Цей підхід, разом із розширеними можливостями ШІ та високим рівнем безпеки, дозволяє створювати високопродуктивні інтегровані застосунки для широкого спектру пристроїв.

2.4 Аспекти вибору системи

Для розробників, які обирають платформу для створення нативного застосунка, iOS, Android та HarmonyOS представляють різні стратегії з унікальними перевагами та викликами.

iOS від Apple є контрольованою та однорідною екосистемою,

орієнтованою на стабільність та ефективність. Завдяки суворим стандартам розробки Apple, програми легко адаптуються до різних пристроїв, оскільки на ринку обмежений набір моделей і роздільних здатностей екрану. Інструменти, такі як Xcode і Swift, роблять розробку зручною. Однак, поріг для запуску застосунка в App Store є високим, і слід дотримуватися вимог до дизайну.

Android залишається найбільш популярною операційною системою, з потужною підтримкою Google та відомими інструментами розробки, такими як Android Studio, а також мовами Kotlin і Java. Android 15 продовжує підтримувати розподілені пристрої, проте його фрагментованість створює певні труднощі для розробників, особливо в аспектах UI/UX, адаптації до різних моделей і версій ОС. Завдяки цьому розробникам потрібно докладати більше зусиль для оптимізації програм, що може впливати на час та вартість проєкту.

Huawei розвиває HarmonyOS з метою незалежності від Android і створення нової універсальної екосистеми, здатної функціонувати на всіх типах пристроїв. В HarmonyOS особливу увагу приділено розподіленим можливостям, тобто програмам, що працюють одночасно на різних пристроях. Ark Compiler та підтримка мови ArkTS значно полегшують розробку. HarmonyOS орієнтована на інтеграцію з екосистемою Huawei, але її обмежена частка ринку і концентрація на азійському регіоні може обмежити глобальну рентабельність.

Вибір платформи для нативного застосунка залежить від цільової аудиторії, доступу до ресурсів, бюджету та ринку. iOS вигідна для ринку преміум-класу, Android – для масового охоплення, а HarmonyOS стає вигідним рішенням для регіональних ринків і застосунків, орієнтованих на пристрої Huawei.

3 ГІБРИДНІ ЗАСТОСУНКИ

Гібридні мобільні застосунки, відомі також як кросплатформні, стали популярним вибором для багатьох компаній завдяки їхній універсальності та економічній ефективності. Вони поєднують у собі елементи нативних і веб-застосунків, дозволяючи використовувати один і той самий код для роботи на різних платформах. Це значно спрощує процес розробки, зменшує витрати й пришвидшує вихід продукту на ринок.

3.1 Переваги, недоліки та можливості гібридних застосунків

Гібридні застосунки створюються із застосуванням стандартних веб-технологій, таких як HTML5, CSS та JavaScript, і пізніше «загортаються» у нативну оболонку за допомогою спеціальних інструментів, таких як Apache Cordova або Capacitor. Це дозволяє цим застосункам працювати як у браузерах, так і як нативні програми на iOS та Android, використовуючи відповідні магазини застосунків для розповсюдження [1].

Основні переваги гібридних застосунків:

- економічна ефективність (Гібридна розробка дозволяє суттєво скоротити витрати, оскільки одна команда розробників створює єдиний код, який можна застосувати до різних платформ. Це знижує потребу наймати окремі команди для Android і iOS, що є стандартною практикою при нативній розробці);

- швидкий вихід на ринок (Завдяки повторному використанню коду, гібридні застосунки дозволяють значно пришвидшити процес розробки. Компанії можуть швидше представити свій продукт користувачам, що є критично важливим для конкурентних ринків);

- ширше охоплення аудиторії (Розробка гібридного застосунка забезпечує доступ до користувачів різних платформ без необхідності

створення окремих програм. Це особливо важливо для компаній, які прагнуть досягти глобальної аудиторії);

- уніфікований користувацький досвід (Використання спільного коду дозволяє створювати однаковий зовнішній вигляд та функціональність застосунка на всіх платформах. Це забезпечує стабільність UX і створює позитивне враження у користувачів);

- легкість оновлення та обслуговування (Зміни в коді автоматично впроваджуються на всіх платформах, що зменшує витрати на підтримку і спрощує оновлення застосунків у майбутньому).

Попри очевидні переваги, гібридні застосунки мають певні недоліки, які необхідно враховувати:

- оОбмежена продуктивність (У порівнянні з нативними програмами, гібридні застосунки часто демонструють нижчий рівень продуктивності, особливо при обробці складних графічних або обчислювальних задач. Це може бути критичним для ігор або програм із високими вимогами до швидкості);

- можливі проблеми з UI/UX (Врахування особливостей дизайну кожної платформи може бути складним завданням для розробників. Наприклад, елементи інтерфейсу, створені для Android, можуть не відповідати стандартам iOS, що впливає на загальну якість програми);

- обмежений доступ до апаратних функцій (Хоча сучасні фреймворки надають широкі можливості для інтеграції з апаратним забезпеченням пристроїв, вони все ще поступаються нативним рішенням у гнучкості використання функцій, таких як біометрія, AR/VR, або спеціалізовані датчики);

- складнощі з публікацією в магазинах застосунків (Процес перевірки та схвалення застосунків у магазинах, таких як Apple App Store та Google Play, може бути складнішим для гібридних застосунків через використання сторонніх фреймворків).

Розробка гібридних застосунків неможлива без використання

спеціалізованих інструментів, які забезпечують адаптацію веб-технологій для різних платформ. Сучасні фреймворки пропонують широкий спектр функціональних можливостей, допомагаючи розробникам створювати продуктивні, зручні та естетично привабливі застосунки.

Серед найбільш популярних і перспективних рішень можна виділити наступні:

- Flutter, розроблений компанією Google, є одним із найпопулярніших фреймворків для гібридної розробки. Його головною особливістю є використання мови програмування Dart, яка поєднує у собі продуктивність, зручність і сучасний синтаксис. Flutter дозволяє створювати швидкі, інтерактивні застосунки для iOS, Android, веб-платформ та навіть настільних операційних систем;

- React Native. Розроблений Facebook, React Native є одним із найбільш затребуваних фреймворків для створення кросплатформних застосунків [1]. Цей інструмент базується на JavaScript та React, популярній бібліотеці для розробки інтерфейсів користувача;

- Ionic – це потужний фреймворк, орієнтований на створення застосунків із використанням стандартних веб-технологій (HTML, CSS, JavaScript). Інтеграція з Capacitor дозволяє розробникам використовувати нативні API пристроїв, що розширює можливості гібридних застосунків;

- Xamarin, створений Microsoft, дозволяє створювати потужні гібридні застосунки з використанням мови програмування C#. Фреймворк забезпечує повний доступ до функцій нативного API, що робить його сильним інструментом для розробки бізнес-застосунків [4].

Окрім наведених вище, популярності також набирають Capacitor і NativeScript. Capacitor, який є спадкоємцем Apache Cordova, пропонує сучасніший підхід до роботи з нативними API. NativeScript забезпечує нативну продуктивність, дозволяючи розробникам створювати складні інтерфейси за допомогою JavaScript або TypeScript.

За даними сайту Statista представлено діаграму фреймворків, які

найбільше використовують розробники по всьому світу за 2019-2023 роки (рисунок 3.1) [5].

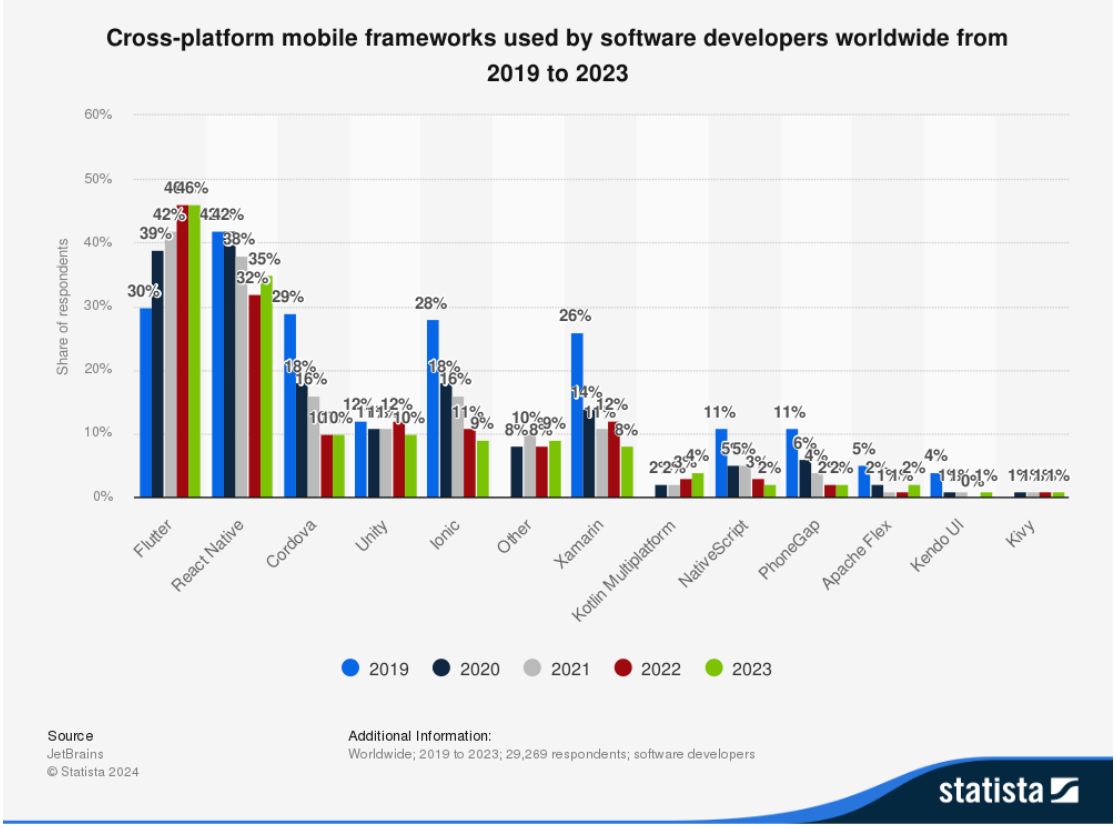

Рисунок 3.1 – Найпопулярніші кросплатформні мобільні фреймворки за 2019-2023 роки

Гібридні застосунки найчастіше використовуються для таких сфер:

- електронна комерція: платформи для замовлення товарів і послуг, що охоплюють глобальну аудиторію;
- соціальні мережі: застосунки, що потребують швидкого масштабування;
- освіта: інтерактивні навчальні програми, які доступні на різних пристроях;
- інструменти для бізнесу: CRM, ERP або інші рішення для оптимізації бізнес-процесів.

3.2 Особливості роботи з гібридними фреймворками

Гібридні застосунки продовжують розвиватися, а інновації у сфері фреймворків допомагають усунути їхні обмеження. Завдяки цьому вони стають все більш популярним вибором для компаній, які прагнуть поєднати якість, доступність і швидкість реалізації.

Розробка застосунків із використанням гібридних фреймворків, таких як React Native чи Flutter, вимагає врахування ряду аспектів, які можуть впливати на ефективність проекту та його кінцевий результат. Щоб мінімізувати ризики і покращити якість розробки, важливо приділити увагу наступним етапам:

- оцінка: аналіз ресурсів, знань команди та потенційних ризиків;
- дизайн: співпраця між дизайнерами та розробниками для досягнення оптимального результату;
- автоматизація процесів CI/CD: налаштування безперервної інтеграції та доставки;
- Splash screen: специфічні нюанси реалізації для різних фреймворків;
- верстка: забезпечення адаптації для всіх платформ одночасно;
- паралельна розробка web і mobile: використання спільних компонентів коду;
- налагодження (debug): вирішення багів, особливо у специфічних бібліотеках;
- робота з файлами системи: запити дозволів та забезпечення сумісності;
- доставка збірок клієнту: використання зручних сервісів, таких як TestFlight чи Crashlytics.

Слід розглянути детальніше кожен з етапів:

1 оцінка та планування (Перш ніж розпочати розробку, необхідно ретельно оцінити технічні можливості команди, доступність ресурсів і потенційні ризики. Це включає тестування програми на всіх задіяних

платформах (iOS, Android) та аналіз сумісності бібліотек. Важливо врахувати можливість виникнення багів у самих фреймворках, таких як React Native чи Flutter, і закласти буфер часу для їх виправлення);

2 дизайн (Робота над дизайном для гібридних застосунків вимагає тісної співпраці між дизайнерами та розробниками. Деякі елементи дизайну можуть бути складними для рендерингу у фреймворках, особливо у Flutter чи React Native. Щоб уникнути затримок, необхідно узгодити дизайн до його фінального затвердження клієнтом);

3 CI/CD (Автоматизація процесів безперервної інтеграції (CI) та доставки (CD) є критично важливою для успішної реалізації проекту. У React Native можуть виникати специфічні проблеми з автозбиранням через особливості інтеграції бібліотек для різних платформ. Рекомендується заздалегідь передбачити можливі складнощі та забезпечити додатковий час на налаштування);

4 реалізація splash screen (для реалізації стартового екрана застосунка (splash screen) важливо враховувати особливості обраного фреймворка. У Flutter цей елемент реалізується швидше завдяки готовим інструментам, тоді як у React Native його доводиться створювати нативно, що може викликати труднощі та потребує більше часу);

5 верстка та паралельна розробка (Рекомендується проводити верстку для iOS та Android паралельно, щоб уникнути майбутніх труднощів із адаптацією. Якщо веб-версія програми написана на React, це значно скорочує час розробки мобільного застосунка на React Native, оскільки логіка компонентів може бути повторно використана);

6 налагодження та тестування (React Native забезпечує зручні інструменти для тестування та юніт-тестів, тоді як у Flutter процес може бути більш складним через менш інформативні логи. Для великих проектів на Flutter необхідно закласти додатковий час для виявлення та виправлення помилок);

7 робота з файлами системи (При взаємодії з файловою системою

мобільного пристрою важливо заздалегідь планувати процеси запиту дозволів на доступ до SD-картки чи інших системних ресурсів. Для відправки файлів через програму можна використовувати ContentResolver. Незважаючи на схожість із нативною розробкою, ці аспекти можуть вимагати додаткового часу для налаштування);

8 доставка збірок (Для доставки збірок застосунків клієнтам і тестувальникам можна використовувати зручні сервіси, такі як Crashlytics, TestFairy або TestFlight. Вибір залежить від специфіки проекту та обраних платформ).

3.3 React Native

React Native – це популярний фреймворк для розробки мобільних застосунків із відкритим вихідним кодом, створений компанією Facebook у 2015 році [24]. Завдяки своїй гнучкості та потужності, він дозволяє створювати застосунки для таких платформ, як Android, iOS, Windows, macOS, Web, tvOS та UWP. React Native базується на React – бібліотеці для побудови інтерфейсів користувача – і використовує JavaScript для розробки кросплатформених застосунків із нативним виглядом і функціоналом.

React Native поєднує переваги нативних і гібридних підходів. Його архітектура базується на використанні потужних рендерингових і модульних механізмів, що забезпечують ефективну взаємодію між JavaScript і нативним кодом [24].

Фреймворк працює через такі основні компоненти:

- Fabric (механізм рендерингу). Fabric забезпечує ефективну синхронізацію між JavaScript і нативним інтерфейсом, дозволяючи динамічно відображати компоненти без затримок;

- TurboModules. Оптимізовані модулі, які забезпечують легкий доступ JavaScript до нативного функціоналу. Завдяки динамічному завантаженню, TurboModules зменшують споживання пам’яті та прискорюють завантаження

застосунка.

Завдяки цим компонентам застосунки працюють плавно та швидко, незалежно від складності функціоналу.

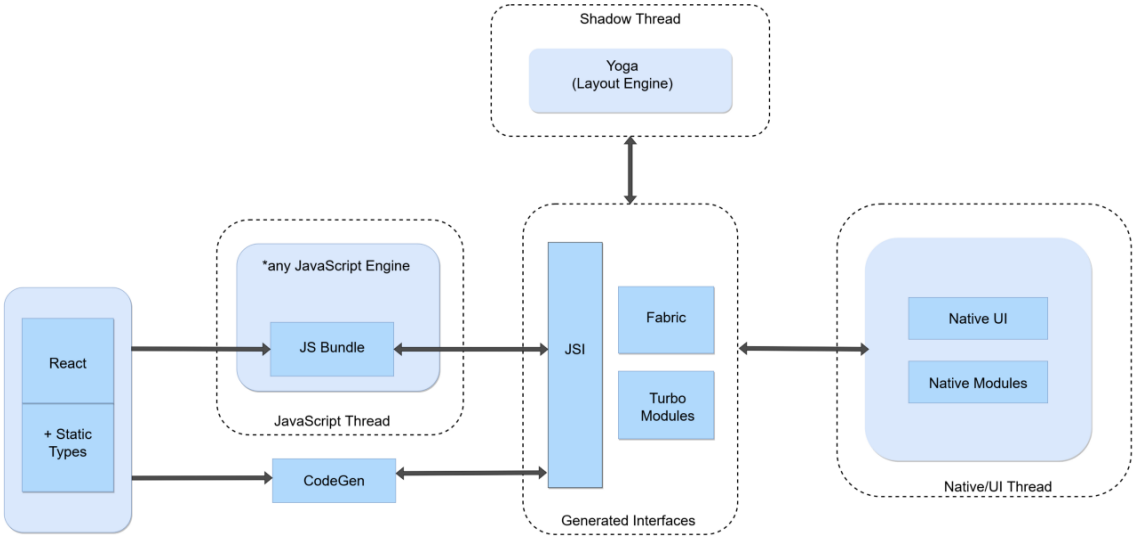

Рисунок 3.2 – Архітектура React Native

Переваги React Native:

- швидкий цикл розробки (React Native дозволяє використовувати функцію "Hot Reload", яка значно прискорює розробку, даючи змогу миттєво бачити зміни в інтерфейсі після редагування коду);

- кросплатформна підтримка (Єдиний код можна використовувати для розробки застосунків одразу для кількох платформ, що скорочує час і витрати);

- нативний вигляд і продуктивність (Fabric і TurboModules забезпечують високу продуктивність, плавність роботи інтерфейсу та органічний вигляд застосунків для користувачів кожної операційної системи);

- гнучкість і масштабованість (Фреймворк дозволяє легко інтегрувати

нативний код для специфічних функцій, що робить його ідеальним як для стартапів, так і для великих корпоративних рішень);

- відкритий вихідний код (Спільнота React Native активно розвивається, постійно додаючи нові бібліотеки, плагіни та інструменти для спрощення розробки).

Ключовим елементом React Native є компоненти. Кожен компонент представляє окрему частину інтерфейсу, наприклад текстовий блок, кнопку чи форму. Завдяки цьому підходу розробники можуть створювати багаторазово використовувані модулі.

Лістинг 3.1 – Простий приклад компонента у React Native

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

const CustomButton = () => {
    return (
        <View style={styles.container}>
            <Button title="Click Me" onPress={() =>
alert('Button Pressed')} />
        </View>
    );
};

const styles = StyleSheet.create({
    container: {
        marginTop: 50,
        padding: 20,
        alignItems: 'center',
    },
});

export default CustomButton;
```

У цьому прикладі компонент `Button` автоматично адаптується до платформи: на iOS це буде UIButton, а на Android – ButtonView. Завдяки Fabric цей процес забезпечує високу швидкість і узгодженість зовнішнього вигляду на різних платформах.

Виклики та обмеження:

- інтеграція нативного коду (Для складних функцій, таких як доступ до апаратного забезпечення, може знадобитися написання нативного коду мовами Swift, Objective-C (iOS) або Kotlin, Java (Android));

- особливості стилізації (Стилізація в JavaScript може мати певні обмеження для складних дизайнерських рішень, тому іноді потрібно комбінувати підходи);

- вирішення багів (При розробці можуть виникати проблеми із сумісністю бібліотек або специфічні помилки в самому фреймворку, які потребують часу для виправлення).

React Native використовується багатьма відомими компаніями для створення своїх застосунків, зокрема: Facebook (соціальна мережа, яка стала засновником цього фреймворку), Instagram (застосунок для обміну фотографіями), Tesla (мобільний застосунок для керування автомобілями).

React Native залишається одним із найперспективніших інструментів для створення кросплатформних мобільних застосунків, поєднуючи простоту, ефективність і потужність.

3.4 Flutter

Flutter – це фреймворк із відкритим вихідним кодом, розроблений компанією Google, що дозволяє створювати високопродуктивні мобільні застосунки для платформ Android та iOS, а також веб-застосунки. Вперше представлений у 2018 році, Flutter швидко завоював популярність завдяки своїй простоті, гнучкості та підтримці багатої бібліотеки віджетів. Він використовує мову програмування Dart, яка компілюється у нативний код, забезпечуючи високу швидкість виконання, порівнянну з такими мовами, як Swift чи Kotlin.

Особливості Flutter:

- висока продуктивність (Flutter не використовує нативні компоненти, як це роблять більшість інших фреймворків. Натомість фреймворк

самостійно створює весь інтерфейс, подібно до ігрових рушіїв. Це забезпечує однакову продуктивність і зовнішній вигляд застосунків на всіх платформах);

- декларативний підхід до створення інтерфейсу (Для побудови UI у Flutter використовується система віджетів, яка дозволяє створювати інтерфейси швидко та ефективно. Віджети можна об’єднувати для формування складніших структур. Для прискорення роботи інтерфейсу віджети оновлюються лише за необхідності – коли в них відбулися зміни);

- гнучкість у дизайні (Flutter підтримує два набори віджетів для реалізації дизайну: Material Design – для дотримання стандартів Google; Cupertino – для відтворення стилю iOS);

- гаряче перезавантаження (Hot Reload) (Ця функція дозволяє розробникам миттєво бачити внесені зміни в інтерфейсі без необхідності повного перезапуску застосунка).

Flutter складається з кількох рівнів, які взаємодіють між собою, забезпечуючи ефективну роботу застосунка [10]:

- рушій Flutter;
- базова бібліотека (Foundation library);
- віджети;
- шари фреймворку.

Рушій Flutter написаний переважно на C++ з використанням графічної бібліотеки Google Skia, рушій відповідає за рендеринг інтерфейсу, обробку тексту, графіки та інших низькорівневих операцій.

Базова бібліотека (Foundation library) – це набір класів і функцій, написаних на Dart, які забезпечують взаємодію між рушієм і віджетами.

Віджети є базовими будівельними блоками інтерфейсу Flutter. Вони описують кожен елемент UI – текст, кнопки, анімації. Складні компоненти створюються шляхом об’єднання простих віджетів.

Шари фреймворку:

- шар віджетів: дозволяє створювати інтерфейс із використанням композиції об'єктів;

- шар рендерингу: забезпечує макетування і динамічну зміну елементів інтерфейсу;

- API низького рівня: надає доступ до базових функцій, таких як обробка графіки чи введення.

Рисунок 3.3 – Архітектура Flutter

Лістинг 3.2 – Простий приклад компонента у Flutter

```
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("Flutter Example")),
        body: Center(
          child: ElevatedButton(
            onPressed: () {
              print("Button pressed!");
            },
            child: Text("Click Me"),
          ),
        ),
      ),
    );
  }
}
```

У цьому прикладі використовується ElevatedButton, який автоматично адаптується під платформи, забезпечуючи єдиний стиль у дизайні.

Переваги:

- можливість створення анімованих інтерфейсів із високою продуктивністю;
- швидкий цикл розробки завдяки Hot Reload;
- велика кількість готових компонентів для різних стилів дизайну.

Недоліки:

- застосунки Flutter мають більший розмір у порівнянні з нативними;
- складність при інтеграції з нативними API може вимагати досвіду у Swift/Objective-C або Kotlin/Java;
- використання Dart потребує додаткового навчання для новачків.

Flutter використовується багатьма великими компаніями, зокрема:

- Realtor.com: застосунок для пошуку нерухомості;
- Alibaba: для забезпечення швидкого доступу до онлайн-шопінгу;
- Tencent: створення різноманітних програм для мільйонів користувачів.

Flutter залишається потужним і сучасним інструментом для створення кросплатформних застосунків, що поєднує високу продуктивність із легкістю у використанні.

3.5 Xamarin

Xamarin – це платформа для розробки мобільних застосунків з використанням мови програмування C#. Вона дозволяє створювати застосунки для Android, iOS, macOS, Windows та інших платформ із нативним виглядом і продуктивністю [4]. Xamarin забезпечує доступ до нативних API кожної операційної системи, що дозволяє максимально використовувати можливості пристроїв.

Основні компоненти Xamarin:

- Xamarin.Platform. Дозволяє використовувати нативні API для створення високопродуктивних застосунків із нативним інтерфейсом;
- Xamarin.Forms. Це фреймворк для створення інтерфейсів, спільних для всіх платформ. Інтерфейс будується з використанням мови розмітки XAML і адаптується до стилів кожної платформи;
- Xamarin Test Cloud. Сервіс для тестування застосунків на великій кількості реальних пристроїв через хмару;
- Xamarin for Visual Studio. Інтеграція з Visual Studio надає розробникам єдине середовище для роботи з Android, iOS та Windows.

У Xamarin.Forms основними елементами побудови інтерфейсу є:

- представлення (Views): елементи UI, такі як кнопки, текстові поля тощо;
- макети (Layouts): визначають розташування елементів на екрані;
- сторінки (Pages): слугують контейнерами для представлень і макетів.

Лістинг 3.3 – Приклад компонента з використанням XAML

```
<!-- MainPage.xaml -->
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MyApp.MainPage">
  <StackLayout Padding="20">
    <Label Text="Welcome to Xamarin.Forms!"
```

```
            HorizontalOptions="Center"
            VerticalOptions="CenterAndExpand" />
    <Button Text="Click Me"
            Clicked="OnButtonClicked" />
  </StackLayout>
</ContentPage>
```

Лістинг 3.4 – Приклад компонента з використанням C#

```
// MainPage.xaml.cs
using Xamarin.Forms;

namespace MyApp
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }

        private void OnButtonClicked(object sender, EventArgs e)
        {
            DisplayAlert("Hello", "You clicked the button!",
"OK");
        }
    }
}
```

Переваги Xamarin:

- кросплатформність – єдиний код можна використовувати для розробки програм на кількох платформах;

- доступ до нативних API – розробники можуть взаємодіяти з платформоспецифічними можливостями пристрою, наприклад камерою, датчиками чи геолокацією;

- продуктивність – код на C# компілюється у нативний, що забезпечує швидкість виконання, порівнянну з нативними застосунками;

- інтеграція з Visual Studio – зручне середовище для розробки, що включає інструменти для налагодження, тестування та публікації застосунків.

Xamarin використовується багатьма компаніями для розробки власних кросплатформних застосунків:

- Alaska Airlines: мобільний застосунок для подорожей;
- The World Bank: фінансові застосунки для моніторингу проектів;
- APX: бізнес-рішення для управління командою.

Xamarin був потужним інструментом для розробки кросплатформних мобільних застосунків з використанням мови C#, забезпечуючи доступ до нативних API, високу продуктивність та інтеграцію з Visual Studio. Його основні компоненти, як-от Xamarin.Platform, Xamarin.Forms, та інші, дозволяли створювати сучасні програми з нативним виглядом.

Однак, з 1 травня 2024 року підтримка Xamarin і Xamarin.Forms офіційно завершена [28]. Тепер розробникам рекомендується:

- оновлювати Xamarin.Android, Xamarin.iOS і Xamarin.Mac до .NET для Android, iOS і Mac (з версії .NET 6);
- переносити проєкти з Xamarin.Forms до .NET Multi-platform App UI (MAUI), який є сучасним рішенням для створення кросплатформних інтерфейсів.

Останні доступні версії пакетів SDK для Xamarin – Android API 34 та Xcode 15. Подальша підтримка нових API та оновлень не передбачена, що робить перехід на .NET MAUI обов’язковим для продовження розробки та підтримки програм [28].

Таким чином, Xamarin поступився місцем сучасним технологіям у .NET-екосистемі, які пропонують розширені можливості для розробників.

3.6 .NET MAUI

.NET MAUI (Multi-platform App UI) – це сучасний кросплатформний фреймворк від Microsoft, що дозволяє створювати застосунки для Android, iOS, Windows та macOS із використанням єдиного кодового базису [22]. Це еволюція Xamarin.Forms, яка забезпечує тісну інтеграцію з .NET 6/7,

покращену продуктивність і спрощену архітектуру.

Основні можливості:

- один проект для всіх платформ (розробники можуть об’єднувати код, ресурси та платформо-залежні файли в одному проекті, зберігаючи їхню структурованість і організованість);

- гнучкий UI (побудова інтерфейсів користувача доступна через декларативний підхід (XAML) або імперативний підхід (C#), а також з використанням Blazor Hybrid, що дозволяє включати веб-компоненти у застосунок);

- Handlers (обробники) (замість рендерерів у Xamarin.Forms, .NET MAUI використовує більш ефективні обробники для взаємодії з нативними елементами платформи, що забезпечує кращу продуктивність і гнучкість).

Архітектура .NET MAUI побудована на багатошаровій структурі, що забезпечує модульність і розширюваність:

- UI-шар (візуальний рівень). Цей рівень включає XAML або C# компоненти, що описують інтерфейс користувача. Тут можна використовувати: Controls (елементи управління): кнопки, поля вводу, списки тощо; Layouts (макети): StackLayout, Grid, AbsoluteLayout та інші;

- Handlers (обробники) (Це основний механізм взаємодії UI з нативними компонентами платформи. Handlers замінюють рендерери з Xamarin.Forms, забезпечуючи швидшу обробку та кращу підтримку кастомізації);

- Core Services (основні служби): DependencyService: для доступу до специфічних функцій платформи; MessagingCenter: для реалізації механізмів підписки на події.

- Platform Layer (платформний рівень). Цей рівень забезпечує доступ до нативних API, таких як файлові системи, геолокація, датчики пристроїв тощо;

- Shared Code (спільний код). Основна частина логіки застосунка пишеться один раз і використовується на всіх платформах, з можливістю

додавати платформо-залежний код за потреби.

Додаткові можливості:

- Graphics API (це набір інструментів для малювання графіки та створення анімацій безпосередньо у .NET MAUI);

- Blazor Hybrid (інтеграція веб-технологій у кросплатформний застосунок. Це дає змогу використовувати компоненти Blazor для створення UI на основі HTML і CSS, працюючи разом із нативними елементами .NET MAUI);

- Resilient Deployment (гнучке розгортання) (.NET MAUI підтримує Hot Reload, що дозволяє розробникам швидко вносити зміни в застосунок і тестувати їх у реальному часі).

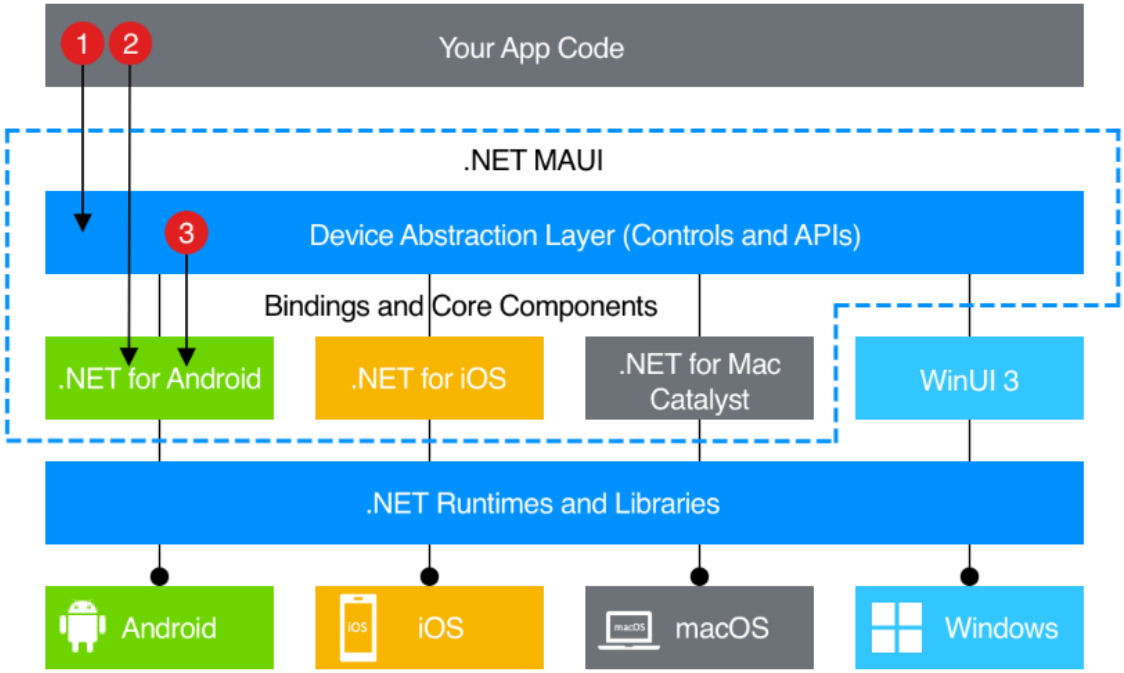

Рисунок 3.4 – Архітектура .NET MAUI

Лістинг 3.5 – Приклад компонента з використанням XAML

```
<!-- MainPage.xaml -->
<ContentPage
xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
```

```
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MyApp.MainPage">
  <VerticalStackLayout Padding="20">
    <Label Text="Hello, .NET MAUI!"
           FontSize="24"
           HorizontalOptions="Center"
           VerticalOptions="CenterAndExpand" />
    <Button Text="Press Me"
            Clicked="OnButtonClicked" />
  </VerticalStackLayout>
</ContentPage>
```

Лістинг 3.6 – Приклад компонента з використанням C#

```
// MainPage.xaml.cs
using Microsoft.Maui.Controls;

namespace MyApp
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }

        private void OnButtonClicked(object sender, EventArgs e)
        {
            DisplayAlert("Greetings!", "You clicked the
button.", "OK");
        }
    }
}
```

Переваги .NET MAUI:

- єдиний проект: об’єднання всіх платформ у межах одного проекту значно спрощує налаштування й підтримку;
- гнучкість розробки: можливість використання як XAML, так і C# для побудови інтерфейсу [27];
- покращена продуктивність: handlers замість рендерерів забезпечують кращу швидкодію та легкість у налаштуванні;
- інтеграція з екосистемою .NET: підтримка новітніх технологій .NET, таких як ASP.NET Core, Entity Framework Core та Blazor.

Оскільки .NET MAUI є відносно новою технологією, наразі немає широковідомих застосунків, які б були публічно визнані як створені з її використанням. Проте багато розробників та компаній активно досліджують та впроваджують .NET MAUI у своїх проектах.

.NET MAUI підходить для створення застосунків у різних галузях:

- фінансові сервіси: застосунки для моніторингу витрат і управління рахунками;
- системи електронної комерції: мобільні платформи для покупок;
- бізнес-інструменти: програми для управління командами або ресурсами;
- освітні застосунки: платформи для навчання та взаємодії з учнями.

.NET MAUI – це сучасна еволюція Xamarin.Forms, яка пропонує розробникам нові можливості для створення кросплатформних застосунків. Міграція з Xamarin.Forms до .NET MAUI є логічним кроком, оскільки платформа підтримує сучасні інструменти та стандарти розробки, залишаючись потужним рішенням для створення застосунків для Android, iOS, Windows та macOS із спільним кодом і спрощеним процесом розгортання.

3.7 Нативні модулі у гібридних застосунках

У гібридних застосунках основним завданням є забезпечення зручної та ефективної взаємодії з платформо-залежними можливостями пристроїв. Незважаючи на широку функціональність, яку надають гібридні платформи, іноді виникає потреба у використанні специфічних функцій, які неможливо реалізувати стандартними засобами [17]. Нативні модулі стають вирішенням цієї проблеми, дозволяючи розробникам інтегрувати нативний код безпосередньо у структуру гібридного застосунка [19].

Нативні модулі дозволяють отримати доступ до функцій операційної системи або апаратних можливостей пристрою, які не доступні через

стандартні API гібридних платформ. Це забезпечує:

- розширення функціональності: наприклад, доступ до нативних компонентів, таких як Bluetooth, датчики руху чи інтеграція з кастомними бібліотеками;

- оптимізацію продуктивності: окремі обчислювальні процеси, виконані за допомогою нативного коду, можуть працювати швидше, ніж через інтерпретовані мови, такі як JavaScript;

- адаптацію для різних платформ: нативні модулі дозволяють використовувати особливості кожної операційної системи, такі як специфічні системні API, що недоступні в стандартному наборі засобів гібридної платформи [17].

Нативні модулі працюють через взаємодію двох основних шарів:

- нативний шар: включає код, написаний мовами програмування, специфічними для платформи (наприклад, Swift для iOS, Java/Kotlin для Android). Цей код реалізує доступ до функціоналу пристрою чи платформи;

- гібридний шар: використовує міст (bridge) для асинхронного обміну даними між нативними модулями та основним кодом застосунка, написаним на JavaScript або іншій мові, що підтримується платформою.

Лістинг 3.7 – Приклад базового функціоналу нативного модуля для доступу до геолокації на платформі Android

```
import com.facebook.react.bridge.ReactApplicationContext;
import com.facebook.react.bridge.ReactContextBaseJavaModule;
import com.facebook.react.bridge.ReactMethod;
public class LocationModule extends ReactContextBaseJavaModule {
    LocationModule(ReactApplicationContext context) {
        super(context);
    }

    @Override
    public String getName() {
        return "LocationModule";
    }
    @ReactMethod
    public void getCurrentLocation(Callback successCallback) {
```

```
        String location = "50.4501, 30.5234";
        successCallback.invoke(location);
    }
}
```

У цьому прикладі модуль отримує дані про геолокацію та передає їх у JavaScript.

Нативні модулі широко застосовуються в таких сценаріях:

- мультимедійні функції: доступ до камер з підтримкою специфічних форматів зображень або відео;
- інтеграція з апаратним забезпеченням: наприклад, для обробки сигналів датчиків або роботи з периферійними пристроями через USB;
- розширена реальність: інтеграція з бібліотеками AR, які забезпечують високоякісну візуалізацію;
- OTT-застосунки: нативні модулі застосовуються для потокової передачі відео з використанням DRM-захисту або інтеграції голосових команд.

Використання нативних модулів у гібридних застосунках, хоча й забезпечує значну функціональність, супроводжується такими викликами:

- збільшення складності: розробка нативного коду вимагає глибоких знань платформозалежних мов програмування;
- необхідність окремого тестування: код потребує перевірки на кожній платформі, що підвищує витрати часу;
- складність у підтримці: додавання нативних модулів у гібридний застосунок збільшує кількість залежностей, які можуть ускладнити оновлення та масштабування.

Альтернативи, такі як створення повністю нативних застосунків або використання інших гібридних платформ (наприклад, Flutter), мають свої переваги. Однак для задач, де важливі час розробки та підтримка, нативні модулі у гібридних застосунках залишаються ефективним рішенням.

4 ВЕБ-ЗАСТОСУНКИ

4.1 Розвиток веб-застосунків

Веб-застосунки є важливим елементом сучасної цифрової екосистеми, забезпечуючи зручний доступ до онлайн-сервісів через браузери на різних пристроях, включаючи мобільні. Мобільні веб-застосунки мають кілька підходів до реалізації, кожен із яких має свої особливості, переваги та обмеження:

- мобільні версії сайтів (m.dot);
- респонсивний і адаптивний дизайн;
- прогресивні веб-застосунки.

Перші мобільні веб-рішення виникли у вигляді мобільних версій сайтів, які зазвичай мали префікс "m." у веб-адресі (наприклад, m.example.com). Ці сайти створювалися для вирішення проблем сумісності звичайних сайтів зі смартфонами.

Мобільні версії сайтів характеризувалися:

- спрощеним дизайном і навігацією;
- меншою кількістю контенту і сторінок;
- обмеженою функціональністю.

Хоча такі сайти були ефективним рішенням у той час, сьогодні вони вважаються застарілими через неможливість забезпечити повноцінний користувацький досвід і низьку ефективність у досягненні бізнес-цілей, таких як конверсія.

З розвитком технологій мобільного веб-дизайну з’явилися рішення, які дозволяють створювати сайти з чуйним (респонсивним) або адаптивним дизайном. Респонсивний дизайн забезпечує автоматичне підлаштування елементів сайту до розміру екрану користувача. Адаптивний дизайн передбачає створення декількох версій сайту для різних типів пристроїв.

Такі веб-застосунки мають наступні переваги:

- доступність на будь-якому пристрої з браузером, незалежно від операційної системи;
- простота розробки та обслуговування;
- можливість швидкого оновлення.

Однак ці рішення не дозволяють отримати доступ до функціоналу пристрою (наприклад, камери чи датчиків), що обмежує їхню інтерактивність.

Респонсивний та адаптивний веб-дизайни стали основою для створення прогресивних веб-застосунків (PWA), які поєднують у собі найкращі риси веб-застосунків і нативних програм [26]:

- PWA дозволяють зберігати застосунок на головному екрані смартфона, як звичайний мобільний застосунок;
- вони підтримують офлайн-режим, сповіщення та доступ до обмежених функцій пристрою.

Таким чином, сучасні веб-застосунки продовжують розвиватися, забезпечуючи зручність доступу та гнучкість для користувачів. Вони є ефективним рішенням для багатьох бізнесів, які прагнуть швидко виходити на ринок із мінімальними витратами.

4.2 Прогресивні веб-застосунки

Прогресивні веб-застосунки (Progressive Web Apps, PWA) – це сучасний підхід до веб-розробки, що дозволяє забезпечити досвід користувачів, подібний до нативних застосунків. Завдяки цій технології веб-сайт може виглядати й працювати, як мобільний застосунок, поєднуючи доступність веб-середовища та функціональність застосунків, створених для конкретних операційних систем [26].

PWA – це універсальне рішення, що працює через браузер, але дозволяє користувачам встановлювати застосунок на свої пристрої,

забезпечуючи доступ до функцій, які раніше були притаманні лише нативним застосункам. Завдяки цьому прогресивні веб-застосунки знаходять дедалі ширше застосування в різних галузях, зокрема в електронній комерції, сервісах доставки та соціальних мережах.

Прогресивні веб-застосунки були створені для вирішення кількох ключових проблем традиційних веб-сайтів та нативних застосунків. Сайти часто мали обмежений функціонал, а процес завантаження й встановлення нативних застосунків займав значний час і вимагав від користувача додаткових дій [23]. PWA забезпечують простий доступ до застосунка через браузер, уникаючи необхідності завантаження зі спеціалізованих магазинів застосунків.

Основною особливістю PWA є їхня здатність працювати незалежно від наявності Інтернет-з’єднання. Завдяки технології Service Worker, яка відповідає за кешування даних та обробку запитів у фоновому режимі, користувачі можуть отримувати доступ до основного функціоналу навіть в офлайн-режимі.

Іншою ключовою перевагою є підтримка push-сповіщень, які забезпечують зручну комунікацію з користувачами. Окрім цього, PWA дозволяють інтегруватися з апаратним забезпеченням пристроїв, зокрема камерою, геолокацією, мікрофоном тощо [23]. Це робить їх схожими на нативні застосунки з погляду функціональності.

Для створення прогресивного веб-застосунка необхідно реалізувати кілька основних компонентів. Service Worker є ключовим елементом технології, який забезпечує кешування контенту, офлайн-режим та роботу push-сповіщень. Він працює у фоновому режимі, незалежно від взаємодії користувача із самою сторінкою.

Іншим важливим компонентом є Web App Manifest – JSON-файл, що містить метаінформацію про застосунок. Цей файл визначає назву застосунка, іконки, кольори теми, стартову URL-адресу та інші параметри, які впливають на зовнішній вигляд застосунка після його встановлення.

Для підвищення продуктивності та швидкості завантаження використовується Application Shell – оболонка, яка швидко завантажується, навіть за повільного Інтернет-з'єднання. Вона містить основні елементи інтерфейсу, тоді як динамічний контент завантажується окремо.

Обов'язковою умовою для роботи PWA є передача даних через HTTPS-протокол, що забезпечує безпеку користувачів і захищений доступ до функціоналу застосунка.

Прогресивні веб-застосунки пропонують низку переваг, які роблять їх привабливим рішенням для багатьох бізнесів. По-перше, вони є кросплатформними й доступними для користувачів будь-якого пристрою з сучасним браузером, незалежно від операційної системи. По-друге, користувачам не потрібно завантажувати й встановлювати застосунок через магазини, що спрощує доступ до сервісу.

Завдяки автоматичному оновленню всі зміни в PWA впроваджуються розробниками у фоновому режимі, без необхідності дій з боку користувачів. Окрім того, PWA підтримуються пошуковими системами, що дозволяє індексувати їхній контент і сприяє підвищенню SEO-рейтингу.

Водночас PWA мають певні обмеження. Зокрема, на платформі iOS підтримка цієї технології залишається частковою – обсяг локальних даних обмежений до 50 МБ, немає доступу до вбудованих платежів та інтеграції з деякими сервісами Apple. Також робота офлайн залежить від попередньо закешованих даних, що може впливати на функціональність за складної логіки застосунка.

Успішні випадки використання PWA демонструють їхню ефективність у різних галузях. Наприклад, компанія Starbucks розробила PWA для збору замовлень, яка скоротила розмір застосунка на 99,84% порівняно з нативним рішенням. Це призвело до подвоєння кількості замовлень через мобільний Інтернет.

Інший приклад – AliExpress, де впровадження PWA дозволило збільшити конверсію для нових користувачів на 104%, а час утримання

користувачів зріс на 74%.

Сервіс Pinterest після переходу на PWA зафіксував зростання часу утримання користувачів на 40% і збільшення доходу від реклами на 44%.

Для створення прогресивних веб-застосунків розробники можуть використовувати готові фреймворки, наприклад, Vue Storefront, який сумісний із такими платформами e-Commerce, як Magento, PrestaShop та Shopware. Іншим універсальним рішенням є Quasar, що базується на Vue.js і дозволяє швидко створювати PWA.

Рішення Workbox пропонує зручні інструменти для інтеграції Service Worker, тоді як PWA Studio для Magento забезпечує легке розгортання прогресивних застосунків у середовищі електронної комерції.

Прогресивні веб-застосунки є ефективним рішенням для бізнесів, які прагнуть поєднати функціональність нативних застосунків із доступністю веб-технологій. Вони забезпечують зручний користувацький досвід, скорочують витрати на розробку та підтримку, а також сприяють зростанню конверсії та залученню аудиторії.

5 ВИБІР МЕТОДУ РОЗРОБКИ МОБІЛЬНОГО ЗАСТОСУНКА

5.1 Критерії вибору технології

Вибір технології залежить від потреб бізнесу та специфіки продукту. Основними критеріями є:

- продуктивність: швидкість роботи застосунка;
- кросплатформність: можливість запуску на Android, iOS або навіть у веб-браузері;
- доступ до функцій пристрою: інтеграція з апаратними можливостями (камера, GPS тощо);
- вартість розробки: економічність створення та підтримки продукту;
- гнучкість оновлень: швидкість внесення змін і масштабування.

Нативні застосунки – це оптимальний вибір для проектів, де важливі продуктивність і доступ до всіх функцій пристрою (наприклад, у банківських застосунках чи іграх). Використовуються Swift для iOS і Kotlin/Java для Android.

Переваги: найкраща продуктивність, інтеграція з апаратним забезпеченням, оптимальний UX.

Недоліки: високі витрати часу і бюджету, необхідність найму окремих команд.

Гібридні застосунки забезпечують можливість запуску одного коду на обох платформах. Найбільш популярні фреймворки – Flutter і React Native.

Переваги: економічність, швидкість розробки, спільний код для iOS та Android.

Недоліки: трохи нижча продуктивність, обмеження в доступі до апаратних функцій.

Веб-застосунки (PWA) використовують браузер для роботи, але імітують мобільні програми (наприклад, Uber чи Twitter як PWA).

Переваги: низька вартість, незалежність від магазинів застосунків, швидке оновлення.

Недоліки: обмежений доступ до функцій пристрою, залежність від інтернет-з'єднання.

5.2 Рекомендовані рішення

Для стартапів – гібридна розробка за допомогою Flutter або React Native. Цей підхід дозволяє швидко протестувати ідею та охопити одразу дві основні платформи з мінімальними витратами.

Для корпоративних застосунків і високонавантажених систем нативна розробка є найкращим вибором. Це забезпечить максимальну продуктивність, стабільність і доступ до всіх функцій пристрою. Наприклад, застосунки для банків, медичних установ або ігор потребують таких характеристик.

Для e-commerce і широкої аудиторії рекомендовано веб-застосунки (PWA). Вони дозволяють охопити максимальну аудиторію з низькими витратами. Це підходить для платформ замовлення товарів і послуг, які потребують лише базових функцій пристрою.

Для глобальних проектів із гнучкими вимогами ефективним вибором є гібридні застосунки, створені на Flutter. Фреймворк дозволяє розширювати застосунок за функціональністю, інтегрувати інновації (наприклад, AR/VR) і адаптувати продукт до нових ринків.

Для досягнення конкурентоспроможності мобільного застосунка необхідно враховувати цільову аудиторію, бізнес-цілі та бюджет. Гібридні застосунки є ідеальним рішенням для більшості комерційних проектів завдяки економічності та швидкості реалізації. Нативні застосунки залишаються лідером для високопродуктивних і складних рішень. PWA підходять для бюджетних проектів із широким охопленням.

ВИСНОВКИ

В результаті виконаної кваліфікаційної роботи проведено всебічний аналіз сучасних підходів і технологій розробки мобільних застосунків. Розглянуто нативні, гібридні та веб-застосунки, а також особливості сучасних платформ роботи застосунків. Встановлено переваги, недоліки та оптимальні сфери застосування кожного методу, що дозволило сформувати рекомендації для вибору технології залежно від бізнес-цілей, бюджету та цільової аудиторії.

У нативній розробці, яка використовується для конкретних платформ, відзначається висока продуктивність і можливості інтеграції з апаратними функціями, особливо у випадку Android і iOS. При цьому iOS забезпечує стабільність та ефективність завдяки контрольованій екосистемі Apple, тоді як Android від Google пропонує ширший спектр налаштувань та пристроїв, що стимулює адаптацію застосунків до різних конфігурацій. HarmonyOS, що є новою альтернативою від Huawei, інтегрує розподілені можливості та підтримує гнучку розробку, орієнтовану на азійський ринок.

Гібридні застосунки, що використовують спільний код для різних платформ, стають практичними для проектів з обмеженим бюджетом, проте поступаються у продуктивності та доступі до функцій пристроїв.

Веб-застосунки, що запускаються через браузер, оптимальні для швидкої розробки, однак потребують стабільного підключення до Інтернету й обмежені у доступі до апаратних можливостей.

Дослідження підкреслює важливість врахування особливостей апаратного середовища та стандартів платформ під час вибору методу розробки. Інтеграція новітніх рішень, таких як доповнена реальність (AR), машинне навчання та штучний інтелект, особливо на платформах Android та HarmonyOS, створює перспективи для розробки застосунків із розширеним функціоналом.

Використання сучасних фреймворків (наприклад, Flutter) забезпечує економію часу і ресурсів, а також дозволяє ефективно масштабувати застосунки для потреб різних платформ.

Отримані результати є цінними для бізнесу, стартапів і розробників, які стоять перед вибором технології для створення мобільного застосунка. Залежно від бюджету, вимог до функціональності та цільової аудиторії, рекомендації роботи можуть бути використані для створення ефективних і конкурентоспроможних цифрових рішень у сферах e-commerce, фінансових послуг, освіти, охорони здоров’я та інших.

Таким чином, робота визначила оптимальні підходи до розробки мобільних застосунків, сформувала чіткі рекомендації та надала перспективи для інтеграції сучасних технологій, орієнтованих на підвищення якості цифрових продуктів і задоволення потреб користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Кислий О. І. ПОРІВНЯЛЬНИЙ АНАЛІЗ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ДЛЯ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ. *Наукові записки молодих учених*. 2021. № 8. URL: https://phm.cuspu.edu.ua/ojs/index.php/SNYS/article/view/1863/pdf (дата звернення: 26.12.2024).

2. Нижник В. В., Єрьоміна Н. С. НАТИВНИЙ МЕТОД ПРОЕКТУВАННЯ МОБІЛЬНИХ ЗАСТОСУНКІВ. *VI Всеукраїнська студентська наукова конференція «ФОРМУВАННЯ СУЧАСНОЇ НАУКИ: МЕТОДИКА ТА ПРАКТИКА»*, м. Івано-Франківськ, 20 груд. 2024 р. С. 240–241.

3. Android 15 | Android Developers. *Android Developers*. URL: https://developer.android.com/about/versions/15 (дата звернення: 08.12.2024).

4. arvindpdmn, sivaraj. Xamarin. *Devopedia*. URL: https://devopedia.org/xamarin (дата звернення: 27.12.2024).

5. Cross-platform mobile frameworks used by global developers 2023 | Statista. *Statista*. URL: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/ (дата звернення: 21.12.2024).

6. Develop for Android | Android Developers. *Android Developers*. URL: https://developer.android.com/docs (дата звернення: 08.12.2024).

7. Digital 2024: Global Overview Report – DataReportal – Global Digital Insights. *DataReportal – Global Digital Insights*. URL: https://datareportal.com/reports/digital-2024-global-overview-report (дата звернення: 05.12.2024).

8. Ericsson Mobility Report June 2024. *Ericsson*. URL: https://www.ericsson.com/en/reports-and-papers/mobility-report/reports/june-2024 (дата звернення: 03.12.2024).

9. Featured | Apple Developer Documentation. *Apple Developer Documentation*. URL: https://developer.apple.com/documentation (дата звернення: 11.12.2024).

10. Flutter Architecture : Deep Dive into the Design & Structure - AK Coding. *AK Coding*. URL: https://akcoding.com/flutter/flutter-architecture/ (дата звернення: 26.12.2024).

11. Global Smartphone Sales Share by Operating System. *Counterpoint - Technology Market Research & Industry Analysis Firm*. URL: https://www.counterpointresearch.com/insights/global-smartphone-os-market-share/ (дата звернення: 01.12.2024).

12. HarmonyOS – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/HarmonyOS (дата звернення: 12.12.2024).

13. How many people have smartphones? Key smartphone statistics in 2024. *Increditools*. URL: https://increditools.com/how-many-people-have-smartphones/ (дата звернення: 27.11.2024).

14. Human Interface Guidelines | Apple Developer Documentation. *Apple Developer Documentation*. URL: https://developer.apple.com/design/human-interface-guidelines/ (дата звернення: 13.12.2024).

15. Hybrid vs. Native App Development: Pros and Cons of Each Option. *Cleveroad Inc. - Web and App development company*. URL: https://www.cleveroad.com/blog/native-vs-hybrid-app-development/ (дата звернення: 14.12.2024).

16. iOS Statistics By Market Share, Version and App Store Categories. *Coolest Gadgets*. URL: https://www.coolest-gadgets.com/ios-statistics (дата звернення: 11.12.2024).

17. Kamal A. Creating React Native Library: Bridging Native Modules for iOS and Android Integration. *Medium*. URL: https://akreview22.medium.com/creating-react-native-library-bridging-native-modules-for-ios-and-android-integration-c2f8d35c9765 (дата звернення: 04.01.2025).

18. Meet Android Studio | Android Developers. *Android Developers*. URL: https://developer.android.com/studio/intro (дата звернення: 10.12.2024).

19. Messias P. Implementing Native Code in React Native. *DEV Community*. URL: https://dev.to/paulocappa/implementing-native-code-in-react-native-2282 (дата звернення: 05.01.2025).

20. Native vs Hybrid App: Which One is Ideal for App Development? *MindInventory*. URL: https://www.mindinventory.com/blog/native-vs-hybrid-app/ (дата звернення: 14.12.2024).

21. Native vs. Hybrid vs. web app: What's the difference?. *UpWork*. URL: https://www.upwork.com/resources/native-hybrid-web-app-differences (дата звернення: 15.12.2024).

22. .NET Multi-platform App UI (.NET MAUI) | .NET. *Microsoft*. URL: https://dotnet.microsoft.com/en-us/apps/maui (дата звернення: 27.12.2024).

23. Overview of Progressive Web Apps (PWAs) - Microsoft Edge Developer documentation. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/microsoft-edge/progressive-web-apps-chromium/ (дата звернення: 03.01.2025).

24. React-Native-Advanced-Guide/New-Architecture/New-Architecture-in-depth.md at master · anisurrahman072/React-Native-Advanced-Guide. *GitHub*. URL: https://github.com/anisurrahman072/React-Native-Advanced-Guide/blob/master/New-Architecture/New-Architecture-in-depth.md (дата звернення: 30.12.2024).

25. Smartphone Usage Statistics 2025 (By Age & Country). *DemandSage*. URL: https://www.demandsage.com/smartphone-usage-statistics/ (дата звернення: 26.11.2024).

26. What are Progressive Web Apps? | Articles | web.dev. *web.dev*. URL: https://web.dev/articles/what-are-pwas (дата звернення: 24.12.2024).

27. What is .NET MAUI? - .NET MAUI. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-9.0 (дата звернення: 28.12.2024).

28. Xamarin official support policy | .NET. *Microsoft*. URL: https://dotnet.microsoft.com/en-us/platform/support/policy/xamarin (дата звернення: 27.12.2024).

29. Xcode - Apple Developer. *Apple Developer*. URL: https://developer.apple.com/xcode/ (дата звернення: 10.12.2024).