Міністерство освіти і науки України
Львівський національний університет ім. Івана Франка
Факультет електроніки та комп’ютерних технологій
Кафедра системного проектування

КУРСОВА РОБОТА

на тему: “ Порівняльний аналіз нативної і кросплатформеної розробки iOS додатків”

Виконав:
студент групи ФеІм-12с
Білий Д. В.
Науковий керівник:
доцент Ненчук Т. М.

Львів - 2023р.

Вступ…… 4

Розділ 1: Основні поняття та технології мобільної розробки……..... 6

1.1. Мобільна розробка: огляд технологій та платформ……..... 6

1.2. Нативна розробка: особливості та переваги……..... 7

1.3. Кросплатформена розробка: принципи та переваги……..... 9

1.4. Огляд мобільної розробки на платформі iOS (Swift, UIKit)……..... 10

Розділ 2: Нативна розробка мобільних додатків……..... 13

2.1. Архітектура додатків на iOS: Model-View-Controller (MVC)……... 13

2.2. Процес розробки нативного додатка на платформі iOS……..... 15

2.3. Переваги та недоліки нативної розробки……... 16

Розділ 3: Кросплатформена розробка мобільних додатків……... 19

3.1. Основні засоби кросплатформеної розробки ……..... 19

3.2. Використання фреймворків для кросплатформеної розробки на платформі IOS……. 20

3.3. Архітектура додатків у кросплатформовій розробці…… 23

3.4. Процес розробки кросплатформеного додатка на платформі iOS…… 25

3.5. Переваги та недоліки кросплатформеної розробки……. 27

Розділ 4: Порівняння нативної розробки та кросплатформеної розробки на платформі iOS…… 30

4.1. Критерії порівняння: продуктивність, доступ до функцій пристрою, інтерфейс та дизайн, розширюваність, швидкість розробки тощо…… 30

4.2. Аналіз результатів порівняння на різних критеріях……. 31

4.3. Висновки з порівняння нативної розробки та кросплатформеної розробки на платформі iOS……..... 33

Висновки…… 35

Список використаної літератури……..... 37

Додаток А…….……..... 39

Додаток Б…..……..... 41

Вступ

Обґрунтування вибору теми дослідження

Розвиток мобільних додатків є невід'ємною частиною нашого сучасного світу. Стрімкий розвиток мобільних пристроїв та зростання популярності мобільних додатків створює потребу у високоякісній розробці програмного забезпечення для цих платформ. Проте розробка мобільних додатків може бути викликом для розробників, оскільки вони зіткнуться з вибором між нативною та кросплатформовою розробкою. Тому обґрунтування порівняння нативної розробки мобільних додатків проти кросплатформиної, зокрема на платформі iOS, є актуальним і цікавим напрямком дослідження.

Мета та завдання дослідження

Метою даного дослідження є порівняння нативної розробки мобільних додатків з кросплатформовою розробкою на платформі iOS з використанням мови програмування Swift та фреймворку UIKit. Головним завданням дослідження є аналіз та порівняння технологій, методів розробки, переваг та недоліків обох підходів, а також оцінка їх впливу на продуктивність, якість та вартість розробки мобільних додатків на платформі iOS.

Об'єктом дослідження є процес розробки мобільних додатків на платформі iOS з використанням нативної розробки та кросплатформової розробки з використанням мови програмування Swift та фреймворку UIKit.

Предметом дослідження є порівняння нативної розробки мобільних додатків та кросплатформової розробки на платформі iOS, зокрема їх основні принципи, технології, методи розробки, переваги та недоліки.

Методи дослідження

Для досягнення поставлених цілей в даному дослідженні будуть використовані такі методи:

1. Ознайомлення з актуальними науковими статтями, книгами та іншими джерелами для збирання інформації про нативну та кросплатформову розробку мобільних додатків на платформі iOS.
2. Проведення практичних експериментів, розробка додатків за допомогою нативного підходу та кросплатформових фреймворків на платформі iOS для порівняння продуктивності, функціональності та якості.
3. Систематичний аналіз та порівняння зібраних даних та результатів експериментів для отримання висновків щодо ефективності нативної та кросплатформової розробки на платформі iOS

Наукова новизна дослідження Науковою новизною дослідження є комплексний підхід до порівняння нативної розробки мобільних додатків з кросплатформовою розробкою на платформі iOS з використанням мови програмування Swift та фреймворку UIKit. Дане дослідження спрямоване на систематичний аналіз та порівняння основних технологій, методів розробки, переваг та недоліків обох підходів, що може сприяти виробленню кращого розуміння прийняття рішень щодо вибору підходу до розробки мобільних додатків на платформі iOS.

Розділ 1: Основні поняття та технології мобільної розробки.

1.1. Мобільна розробка: огляд технологій та платформ.

Мобільна розробка - це процес створення програмного забезпечення, спеціально призначеного для використання на мобільних пристроях, таких як смартфони та планшети. Це включає в себе розробку мобільних додатків (mobile apps) для різних платформ та операційних систем.

На сьогоднішній день існує кілька популярних платформ для мобільної розробки, які набули значної популярності та мають велику базу користувачів. *Деякі з них включають:*

- iOS: Платформа, розроблена компанією Apple для своїх пристроїв, зокрема iPhone та iPad. Для розробки додатків для iOS використовується мова програмування Swift та фреймворк UIKit.
- Android: Платформа, розроблена компанією Google, що використовується на багатьох смартфонах та планшетах. Для розробки додатків для Android використовується мова програмування Java або Kotlin та фреймворк Android SDK.
- Windows Phone: Мобільна платформа, розроблена компанією Microsoft, що використовується на смартфонах з операційною системою Windows. Для розробки додатків для Windows Phone використовується мова програмування C# та фреймворк Windows Phone SDK.

У мобільній розробці існує кілька основних технологій та підходів, які дозволяють створювати мобільні додатки. *Декілька з них включають:*

- Нативна розробка: Цей підхід передбачає розробку додатків для конкретної платформи за допомогою мови програмування та інструментів, що підтримуються цією платформою. Нативні додатки

мають високий рівень продуктивності та можуть повністю використовувати можливості пристрою.

- Кросплатформова розробка: Цей підхід передбачає створення додатків, які можуть працювати на різних платформах. Розробники використовують спеціальні фреймворки та інструменти, що дозволяють написати код один раз і запустити його на різних платформах. Кросплатформова розробка забезпечує ефективність у розробці та підтримці додатків для кількох платформ.
- Гібридна розробка: Цей підхід поєднує нативну та веб-технології. Додатки розробляються з використанням веб-технологій, таких як HTML, CSS та JavaScript, і запускаються внутрішньою оболонкою, що надає доступ до функцій пристрою. Гібридні додатки можуть працювати на різних платформах, але можуть мати обмеження у продуктивності та доступі до функцій пристрою.[1].[2].

Розділ 1.2. Нативна розробка: особливості та переваги.

Нативна розробка мобільних додатків означає створення програмного забезпечення, яке спеціально розроблене для конкретної платформи, такої як iOS. Вона використовує мови програмування та інструменти, які підтримуються даною платформою. У випадку iOS розробки, мовою програмування, яка часто використовується, є Swift, а для створення інтерфейсу користувача використовується фреймворк UIKit.

Однією з особливостей нативної розробки є можливість повною мірою використовувати функціональні можливості пристрою та інструменти, що надаються платформою. Розробники можуть отримувати доступ до камери, геолокації, гіроскопу та інших апаратних компонентів телефону, що дозволяє створювати додатки з більшими можливостями та функціоналом. Крім того,

розробка нативних додатків дозволяє оптимально використовувати ресурси пристрою та забезпечує високу продуктивність.

Нативна розробка має кілька переваг, які роблять її привабливою для розробників мобільних додатків:

1) Нативні додатки розробляються з використанням мови програмування та інструментів, які оптимізовані для платформи. Це дозволяє досягти високої продуктивності та швидкодії додатків.

2) Розробка нативних додатків дозволяє використовувати всі функціональні можливості платформи, такі як сповіщення, календар, контакти, камера тощо. Це забезпечує більшу гнучкість та можливості для створення багатофункціональних додатків.

3) Кращий дизайн та користувацький досвід: Розробка нативних додатків дозволяє використовувати інструменти та бібліотеки, що спеціально розроблені для даної платформи. Це дозволяє створювати додатки з кращим дизайном, налаштуванням під платформу та вищим рівнем користувацького досвіду.

4) Розробка нативних додатків зазвичай забезпечує більшу продуктивність розробки, оскільки використовуються спеціалізовані інструменти та мови програмування, що підтримуються платформою. Також існують багато ресурсів, документації та спільнот, що сприяють швидкому розвитку та вирішенню проблем під час розробки нативних додатків.

Нативна розробка має свої особливості та переваги, які роблять її привабливою для розробки мобільних додатків на платформі iOS. Розробники можуть використовувати функціональні можливості платформи, отримувати кращий дизайн та користувацький досвід, а також досягати високої продуктивності своїх додатків.[1].[2].

1.3. Кросплатформена розробка: принципи та переваги.

Кросплатформена розробка мобільних додатків означає створення програмного забезпечення, яке може працювати на різних платформах, таких як iOS та Android. Вона базується на використанні спеціальних фреймворків та інструментів, що дозволяють розробникам писати код один раз і використовувати його на різних платформах, зменшуючи таким чином зусилля, час і ресурси, необхідні для створення додатків на кожній окремій платформі.

Основні принципи кросплатформеної розробки мобільних додатків:

1) Розробники можуть писати код один раз і використовувати його на різних платформах, що забезпечує ефективну реюзабельність та скорочує час розробки. Це означає, що зміни, внесені в код, будуть автоматично відображатися на всіх платформах.

2) Кросплатформені фреймворки, такі як React Native, Xamarin та Flutter, надають інструменти та бібліотеки, що дозволяють розробникам створювати мобільні додатки, які працюють на різних платформах. Ці фреймворки надають можливість розробки за допомогою спільної кодової бази, яка може бути перенесена на різні платформи.

3) Кросплатформені фреймворки надають можливість звертатися до функціональних можливостей та інтерфейсів, що специфічні для кожної платформи. Це дозволяє розробникам створювати додатки, які максимально відповідають стандартам та налаштуванням кожної окремої платформи.

Переваги кросплатформеної розробки мобільних додатків Кросплатформена розробка мобільних додатків має декілька переваг, серед яких:

1) Одним з основних переваг кросплатформеної розробки є зниження часу та зусиль, необхідних для створення додатків на різних платформах. Розробники можуть писати код один раз і використовувати його на різних

платформах, що дозволяє ефективно використовувати ресурси та прискорює процес розробки.

2) Кросплатформена розробка дозволяє зменшити витрати на розробку та підтримку додатків на кількох платформах. Замість того, щоб мати окремі команди розробки для кожної платформи, компанії можуть мати одну команду, яка використовує кросплатформені фреймворки та інструменти.

3) Завдяки кросплатформеній розробці, розробники можуть створювати додатки, що працюють на різних платформах, що дозволяє їм досягати більшої аудиторії користувачів. Це може бути особливо корисним для стартапів та маленьких компаній, які бажають швидко розповсюдити свої додатки.[5].[6].

Розділ 1.4. Огляд мобільної розробки на платформі iOS (Swift, UIKit)

Вступ до розробки мобільних додатків на платформі iOS Розробка мобільних додатків на платформі iOS відкриває безліч можливостей для створення інноваційних та привабливих додатків для користувачів Apple-пристроїв. iOS є однією з найпопулярніших мобільних платформ у світі та надає потужні інструменти для розробки високоякісних додатків.

Мова програмування

Swift Swift є основною мовою програмування для розробки додатків на платформі iOS. Ця мова була створена компанією Apple з метою полегшити процес розробки, забезпечити безпеку та швидкодію, а також привнести сучасні підходи до програмування. Swift має простий синтаксис, підтримку високого рівня безпеки та широкий спектр функціональності, що робить його привабливим вибором для розробників на платформі iOS.

Фреймворк UIKit

UIKit є ключовим фреймворком для розробки інтерфейсів користувача на платформі iOS. Він надає набір інструментів та компонентів, що дозволяють розробникам створювати привабливі та функціональні інтерфейси для своїх додатків. UIKit включає елементи керування, такі як кнопки, тексти, зображення, а також можливості для роботи з анімаціями, графікою, жестами та іншими аспектами взаємодії з користувачем.

Огляд особливостей та можливостей розробки на платформі iOS Розробка на платформі iOS дозволяє використовувати різноманітні функціональні можливості, які надаються пристроями Apple. Це включає доступ до камери, геолокації, сенсорів, сповіщень та багато іншого. Крім того, розробники можуть використовувати інструменти для розробки адаптивного дизайну, тестування додатків, оптимізації продуктивності та підтримки останніх функцій та версій iOS.

Розробка на платформі iOS з використанням мови програмування Swift та фреймворку UIKit відкриває широкі можливості для створення функціональних та естетично привабливих мобільних додатків, які забезпечують гармонійну взаємодію з користувачем Apple-пристроїв.

Інструменти розробки на платформі

iOS Платформа iOS надає розробникам широкий набір інструментів, які полегшують процес розробки мобільних додатків. Деякі з найважливіших інструментів включають:

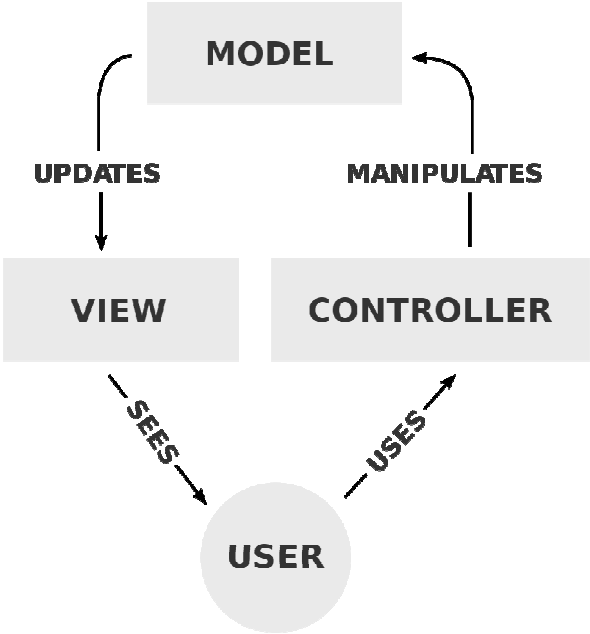
- *Xcode*: Інтегроване середовище розробки (IDE) для платформи iOS, яке надає набір інструментів для написання, налагодження та тестування додатків.
- *Interface Builder:* Візуальний редактор для створення інтерфейсу користувача, де розробники можуть швидко розміщувати та налаштовувати графічні елементи.

- *Simulator:* Інструмент, який дозволяє запускати та тестувати додатки на віртуальних пристроях, що емулюють різні моделі iPhone та iPad.
- *Instruments*: Інструмент для профілювання та оптимізації додатків, який допомагає виявляти й усувати проблеми з продуктивністю та пам'яттю.[3].[5].[6].[9].

Розділ 2: Нативна розробка мобільних додатків

2.1. Архітектура додатків на iOS: Model-View-Controller (MVC)

Архітектура Model-View-Controller (MVC) є однією з основних архітектурних парадигм при розробці додатків на платформі iOS. Вона розділяє компоненти додатка на три основні частини: модель (Model), представлення (View) та контролер (Controller).



Модель відповідає за управління даними додатка та бізнес-логікою. Вона зберігає дані та виконує операції над ними, незалежно від того, як ці дані відображаються на екрані користувача.

Представлення відповідає за відображення даних користувачу та обробку взаємодії з ним. Це можуть бути екрани, кнопки, тексти та інші графічні елементи, які користувач бачить на екрані свого пристрою.

Контролер відповідає за керування взаємодією між моделлю та представленням. Він обробляє вхідні події від користувача, оновлює модель та відображає актуальні дані на представленні.

У архітектурі MVC модель, представлення та контролер мають свої відповідальності та взаємодіють між собою. Модель не залежить від представлення або контролера, тому її можна повторно використовувати в різних контекстах. Представлення отримує дані з моделі та відображає їх на екрані користувача. Контролер обробляє дії користувача та оновлює модель та представлення відповідно.

Взаємодія між компонентами здійснюється за допомогою декларативного або імперативного способу. Декларативний спосіб передбачає використання інтерфейсів та делегатів для обміну даними та подіями між компонентами. Імперативний спосіб передбачає пряме звернення до методів та властивостей компонентів для здійснення взаємодії.

Використання архітектури Model-View-Controller (MVC) на платформі iOS має декілька переваг:

1. *Розділення відповідальностей:* MVC дозволяє чітко розділити логіку додатка, представлення та управління даними. Це спрощує розробку, тестування та підтримку додатка, оскільки кожен компонент виконує конкретну функцію.
2. Повторне використання коду: Завдяки розділенню моделі від представлення та контролера, модель може бути повторно використана в різних контекстах. Це полегшує розширення функціональності додатка та забезпечує ефективне використання ресурсів.
3. *Легка підтримка:* Завдяки чіткому розділенню компонентів, модифікація та підтримка додатка стають простішими. Зміни в одному

компоненті не впливають на інші, що дозволяє швидко вносити зміни та виправляти помилки.

4. *Сумісність з іншими інструментами:* MVC підтримує використання різних фреймворків та бібліотек на платформі iOS, що дає можливість розширювати функціональність додатка та використовувати готові рішення.

Використання архітектури MVC сприяє покращенню структури та організації розробки мобільних додатків на платформі iOS, забезпечуючи ефективну роботу з кодом та підтримку додатка на протязі його життєвого циклу. [3].[5].[6].[9].

2.2. Процес розробки нативного додатка на платформі iOS.

Першим кроком у розробці нативного додатка на платформі iOS є аналіз вимог і планування проекту. На цьому етапі визначаються функціональні та нефункціональні вимоги до додатка, його мета, цільова аудиторія та основні взаємодії з користувачем. Також складається план розробки, в якому визначаються етапи, терміни та ресурси, необхідні для успішної реалізації проекту.

Однією з ключових складових нативної розробки на платформі iOS є розробка інтерфейсу користувача (UI). Для цього використовується фреймворк UIKit, який надає різноманітні компоненти і інструменти для створення привабливого та функціонального інтерфейсу. Розробка UI включає проектування вигляду екранів, розміщення елементів, взаємодію з користувачем та візуальну апробацію дизайну.

Після розробки інтерфейсу користувача наступним кроком є реалізація бізнес-логіки додатка та взаємодії з даними. За допомогою мови програмування

Swift, розробник реалізовує функціональність додатка, обробляє введені дані, здійснює доступ до зовнішніх джерел даних та забезпечує їх обробку. Використовуючи фреймворки та інструменти, такі як CoreData або Alamofire, розробник створює логіку додатка, яка відповідає за обробку даних, комунікацію з сервером та зберігання інформації.

Після завершення реалізації функціональності додатка проводиться тестування та налагодження. Цей етап включає в себе перевірку правильності роботи окремих функцій, виявлення та виправлення помилок, а також тестування на різних пристроях та платформах iOS. Тестування може включати модульні тести, інтеграційні тести, тести з використанням реальних даних та тестування взаємодії з користувачем.

Розгортання та публікація на платформі iOS Останнім етапом розробки нативного додатка на платформі iOS є його розгортання та публікація. Розробник повинен налаштувати сертифікати, права доступу та інші налаштування для відповідного розгортання додатка на пристроях або в App Store. Після успішного розгортання додаток може бути доступний для завантаження та використання користувачами з платформи iOS.[3].[5].[6].[9].

2.3. Переваги та недоліки нативної розробки.

Переваги нативної розробки мобільних додатків на платформі iOS:

- *Висока продуктивність:* Нативні додатки на платформі iOS мають прямий доступ до апаратних ресурсів та оптимізовані бібліотеки, що дозволяє досягти високої продуктивності та швидкодії.
- *Нативний вигляд та взаємодія:* Додатки розроблені нативно використовують стандартні компоненти та інтерфейси платформи iOS,

що дозволяє забезпечити гармонійну взаємодію з користувачем та забезпечити нативний вигляд.

- *Доступ до функціональних можливостей пристрою:* Розробники можуть використовувати всі функціональні можливості пристрою, такі як камера, геолокація, датчики руху та інші, для створення багатофункціональних додатків.
- Підтримка платформи та оновлення: Нативні додатки на платформі iOS отримують швидку підтримку та оновлення від Apple, що дозволяє використовувати нові можливості та виправляти потенційні проблеми.

Недоліки нативної розробки та можливі виклики:

- *Висока вартість розробки:* Розробка нативних додатків на платформі iOS може бути витратною, оскільки вимагає спеціалізованих знань та досвіду у програмуванні для iOS.
- *Обмежена переносимість:* Додатки, розроблені нативно для платформи iOS, не можуть працювати безпосередньо на інших платформах, таких як Android або Windows.
- Додатковий час розробки: Нативна розробка може займати більше часу, оскільки потребує роздільної розробки для кожної платформи, яку розробник планує підтримувати.

Порівняння з іншими підходами до розробки мобільних додатків

- *Кросплатформена розробка:* Нативна розробка має високу продуктивність та доступ до функціональних можливостей пристрою, але може бути обмеженою в переносимості. Кросплатформені фреймворки, такі як React Native або Flutter, можуть забезпечити більшу переносимість, але можуть мати обмеження в продуктивності та доступі до деяких функціональних можливостей пристрою.

- *Гібридна розробка:* Гібридні додатки поєднують в собі веб-технології (HTML, CSS, JavaScript) з контейнером нативного додатка. Вони можуть мати широку переносимість та меншу вартість розробки, але можуть мати обмежену продуктивність та відчувати деякі обмеження з взаємодією з пристроєм.
- *Хмарна розробка:* Хмарні додатки використовують хмарні сервіси та засоби для розробки додатків. Вони можуть бути переносимими та забезпечувати багатофункціональність, але можуть бути залежними від доступу до Інтернету та мають обмеження взаємодії з пристроєм.

Вибір підходу до розробки мобільних додатків залежить від конкретних потреб проекту, бюджету, готовності підтримувати різні платформи та інших факторів. Кожен підхід має свої переваги та недоліки, і важливо враховувати їх при виборі оптимального рішення для конкретного проекту.[3].[5].[6].[9].

Розділ 3: Кросплатформена розробка мобільних додатків.

3.1. Основні засоби кросплатформеної розробки.

Кросплатформна розробка мобільних додатків передбачає використання спеціальних засобів, які дозволяють створювати додатки, що працюють на різних платформах одночасно. Основні засоби кросплатформеної розробки включають:

Мультиплатформні фреймворки є одними з найважливіших засобів для кросплатформеної розробки мобільних додатків. Вони надають розробникам можливість створювати додатки, які можуть працювати на різних операційних системах, таких як iOS, Android, Windows Phone тощо. Деякі з популярних мультиплатформних фреймворків включають React Native, Flutter, Xamarin та інші.

Інтегровані середовища розробки(IDE) є необхідними для розробки кросплатформних додатків. Вони надають розробникам зручні інструменти для написання, відлагодження та збірки додатків. Деякі з популярних IDE для кросплатформної розробки включають Visual Studio, Visual Studio Code, IntelliJ IDEA та інші.

Мови програмування відіграють важливу роль у кросплатформній розробці, оскільки вони визначають синтаксис і можливості для створення додатків. Деякі з популярних мов програмування для кросплатформної розробки включають *JavaScript, Dart, C#, Kotlin і Swift.*

Об'єднуючи ці основні засоби, розробники можуть створювати кросплатформні мобільні додатки, які працюють на різних платформах з використанням загального коду та ресурсів. Кожен з цих засобів має свої

особливості та переваги, що дозволяють розробникам ефективно створювати додатки для різних платформ.[2].[9].[5].

3.2 Використання фреймворків для кросплатформеної розробки на платформі iOS.

Ось невеликий фрагмент коду, який демонструє використання фреймворку *React Native* для розробки кросплатформових додатків на платформі iOS:

Мова - Javascript

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

const App = () => {
  return (
    <View style={styles.container}>
      <Text style={styles.text}>1</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#f5f5f5',
  },
  text: {
    fontSize: 24,
    fontWeight: 'bold',
  },
});

export default App;
```

У цьому прикладі використовується *React Native* для створення кросплатформового додатка на платформі iOS. Компонент *App* є основним компонентом додатка, який повертає візуальну структуру з використанням *<View>* та *<Text>* компонентів. Також використовується *StyleSheet* для задання стилів елементів.

Також ось невеликий фрагмент коду, який демонструє використання фреймворку *Flutter* для розробки кросплатформових додатків на платформі iOS:

Мова - Dart

```
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('Кусова'),
        ),
        body: Center(
          child: Text(
            '«Курсова»',
            style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold),
          ),
        ),
      ),
    );
  }
}
```

У цьому прикладі використовується *Flutter* для створення кросплатформового додатка на платформі iOS. Клас *MyApp* є основним класом

додатка, який наслідує *StatelessWidget* і реалізовує метод build, що повертає візуальну структуру додатка.

Віджет *MaterialApp* встановлює базовий матеріальний дизайн додатка. У віджеті *Scaffold* ми встановлюємо *AppBar* для заголовка додатка та *body* для основного вмісту, в якому ми використовуємо *Center* та *Text* для відображення тексту *«Курсова»*

Flutter використовує мову програмування *Dart*, яка є основною мовою для створення додатків. Вам також буде доступна багата бібліотека віджетів та інструменти для розробки інтерфейсу, роботи з анімаціями, мережевими запитами та багато іншого.

Накінець невеликий фрагмент коду, який демонструє використання фреймворку *Xamarin* для розробки кросплатформових додатків на платформі iOS:

Мова - C#

```
using System;
using Xamarin.Forms;
namespace MyApp
{
 public class App : Application
{
public App()
{
 MainPage = new MainPage();
}
```

```
}

public class MainPage : ContentPage
{
   public MainPage()
   {
      Content = new StackLayout
      {
         Children = {
            new Label {
               Text = "Кусова",
               HorizontalOptions = LayoutOptions.Center,
               VerticalOptions = LayoutOptions.Center
            }
         }
      };
   }
}

public class Program
{
   static void Main(string[] args)
   {
      var app = new App();
      Xamarin.Forms.Application.Run(app);
   }
}

}
```

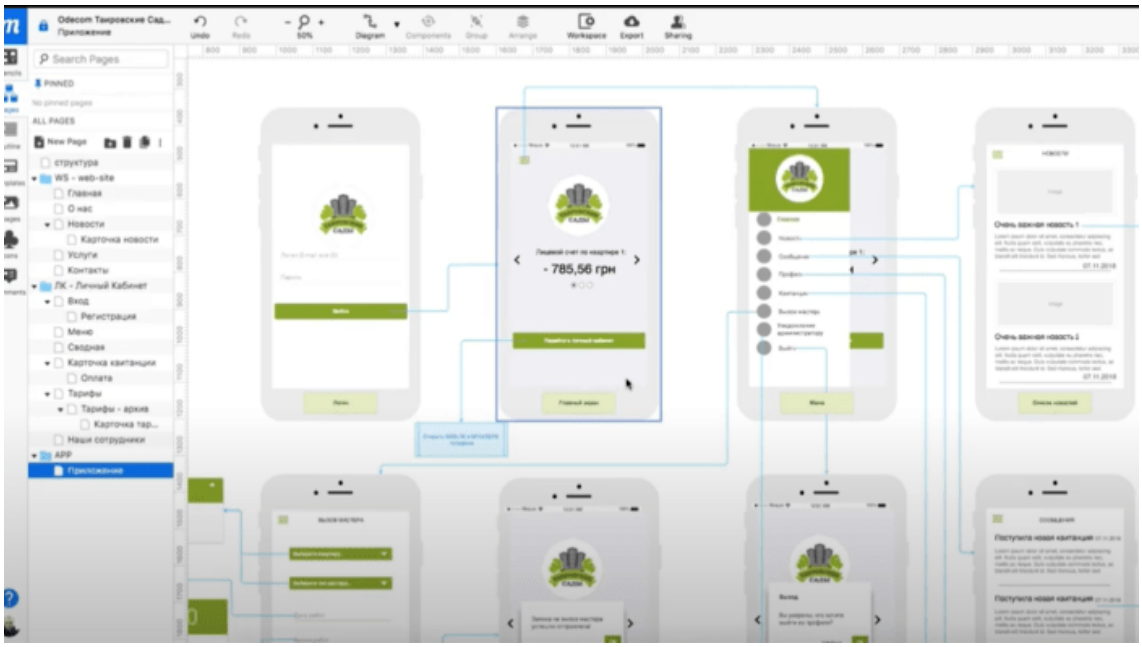
У цьому прикладі використовується фреймворк *Xamarin* для створення кросплатформового додатка на платформі iOS. Клас App є головним класом додатка, в якому ми встановлюємо *MainPage* як головну сторінку додатка.

Клас *MainPage* є *ContentPage*, який містить *StackLayout* з одним *Label* для відображення тексту *«Курсова».* У цьому випадку ми використовуємо Xamarin.Forms для побудови користувацького інтерфейсу.

В класі *Program* ми створюємо новий екземпляр додатка та запускаємо його за допомогою методу *Xamarin.Forms.Application.Run().*

Xamarin дозволяє розробникам використовувати мову програмування C# та спільний код для створення кросплатформових додатків. Він надає доступ до набору інструментів та бібліотек, які допомагають у розробці інтерфейсу, роботі з базами даних, мережевих запитах та іншим функціоналом.[2].[9].[5].

3.3. Архітектура додатків у кросплатформовій розробці



Модульна архітектура

Модульна архітектура є одним із підходів до організації структури додатків у кросплатформовій розробці. В цій архітектурі програмний код поділяється на модулі, які виконують конкретні функції або відповідають за окремі частини функціоналу додатку. Кожен модуль має чітко визначені обов'язки і може бути розроблений незалежно від інших модулів. Це сприяє зручності розробки, тестування та підтримки додатків.

Функціональна архітектура

Функціональна архітектура є ще одним підходом до організації додатків у кросплатформовій розробці. У цій архітектурі програмний код структурується на основі функцій або функціональних блоків, які відповідають за виконання певних завдань. Кожна функція має чітко визначені вхідні параметри та повертає результат обробки. Цей підхід сприяє легкості розуміння та тестування коду, а також дозволяє зручно розширювати функціонал додатку.

Реактивна архітектура

Реактивна архітектура є ще одним підходом, який знаходить застосування у кросплатформовій розробці. В цій архітектурі використовується реактивне програмування, де стан додатку та його компоненти оновлюються автоматично при зміні вхідних даних або стану. Це досягається шляхом встановлення залежностей між різними компонентами та використання потоків даних (streams) для передачі та обробки інформації. Реактивна архітектура сприяє зручності розробки додатків зі складними взаємодіями та асинхронними операціями.

Таблиця №1 Переваги та недоліки кожної з архітектурних підходів

Архітектура	**Переваги**	**Недоліки**
Модульна архітектура	- Зручність розробки, тестування та підтримки додатків	- Можлива зайва складність модульної системи
Функціональна архітектура	- Легкість розуміння та тестування коду	- Обмежена масштабованість функціоналу
Реактивна архітектура	- Зручність розробки складних взаємодій та асинхронних операцій	- Підвищений рівень складності та вивчення

Ці різні архітектурні підходи дають розробникам можливість вибрати найбільш підходящий для їх потреб та проекту. Кожен з цих підходів має свої

переваги та недоліки, і вибір залежить від конкретних вимог, розміру проекту та навичок розробника.[2].[9].[5].

3.4. Процес розробки кросплатформного додатка на платформі iOS.

1) Вибір фреймворку та підготовка робочого середовища

Перший крок у розробці кросплатформного додатка на платформі iOS - вибір підходящого фреймворку, який дозволить створити додаток, спільно використовуючи код на різних платформах. Популярними фреймворками для кросплатформової розробки на iOS є React Native, Flutter та Xamarin. Після вибору фреймворку необхідно налаштувати робоче середовище, встановити відповідні SDK, залежності та інструменти для розробки.

2) Розробка інтерфейсу користувача

Наступний крок - розробка інтерфейсу користувача (UI). За допомогою відповідних інструментів фреймворку, розробник може створити UI-елементи, такі як кнопки, поля введення, списки, картинки тощо. Важливо забезпечити, щоб інтерфейс був зручним та естетичним для користувача.

3) Реалізація бізнес-логіки та взаємодії з даними

Після створення інтерфейсу необхідно реалізувати бізнес-логіку додатка, яка відповідає за обробку даних, виконання розрахунків та взаємодію зі зовнішніми сервісами або базами даних. В цьому етапі розробник використовує мови програмування та функціональні можливості обраного фреймворку для написання логіки додатка.

4) Тестування та налагодження

Після реалізації функціональності додатка необхідно провести тестування, щоб переконатися в правильному функціонуванні додатка, виявити та виправити можливні помилки. Тестування може включати автоматизовані тести, ручні перевірки та тестування на реальних пристроях.

5) Останній етап - розгортання та публікація додатка на платформі iOS. Після успішного тестування та налагодження, розробник може збудувати бінарний файл додатка, який може бути розгорнутий на реальних пристроях або розміщений у магазині додатків App Store. При цьому слід враховувати вимоги та процедури публікації, встановлені компанією Apple.[2].[9].[5].

3.5. Переваги та недоліки кросплатформеної розробки.

Переваги кросплатформеної розробки:

- *Спільний код:* Одні з найбільших переваг кросплатформеної розробки полягають у можливості спільного використання коду між різними платформами. Розробники можуть писати код один раз і використовувати його для створення додатків для різних платформ, що зменшує зусилля та час розробки.
- *Зниження витрат:* Кросплатформна розробка дозволяє заощаджувати витрати, оскільки потрібно розробляти лише один кодову базу замість окремих кодів для кожної платформи. Це може знизити вартість розробки, підтримки та оновлення додатка.
- *Швидкість розробки:* Завдяки спільному коду і використанню фреймворків, які надають готові компоненти та інструменти, кросплатформна розробка може прискорити процес розробки додатків. Розробники можуть скористатися готовими рішеннями, що дозволяє швидше створювати функціонал інтерфейсу та бізнес-логіки.
- *Більша аудиторія:* Кросплатформні додатки дозволяють досягти більшої аудиторії користувачів, оскільки вони можуть бути

розгорнуті на різних платформах. Це дозволяє залучити більше користувачів та розширити потенційну базу користувачів.

Недоліки кросплатформної розробки:

- *Обмежені можливості:* Кросплатформні фреймворки можуть мати обмежені можливості порівняно з нативною розробкою. Деякі функціональності, API або оптимізації можуть бути недоступними або обмеженими на платформі, що може обмежити можливості додатка.
- *Затримки оновлень:* Кросплатформні фреймворки можуть вимагати часу для оновлення до нових версій платформи. Якщо нові функції або виправлення помилок з'являються на платформі, то розробнику може знадобитися додатковий час для оновлення фреймворку та його використання.
- *Менша продуктивність:* Кросплатформні додатки можуть мати меншу продуктивність порівняно з нативними додатками. Це може бути помітним, особливо для складних або вимогливих додатків, де швидкість та продуктивність є критичними.

Порівняння з нативною розробкою та іншими підходами:

- *Нативна розробка:* Нативна розробка дозволяє створювати додатки, які повністю оптимізовані для конкретної платформи. Вона надає найкращу продуктивність, доступ до всіх функціональних можливостей та API платформи. Однак, вона вимагає більше зусиль та часу для розробки окремих додатків для кожної платформи.
- *Гібридна розробка:* Гібридна розробка поєднує елементи кросплатформної та нативної розробки. Вона дозволяє використовувати веб-технології, такі як HTML, CSS та JavaScript, для створення додатків, які можуть бути запущені на різних платформах. Гібридна розробка може

мати обмежені можливості порівняно з нативною розробкою, але вона надає більшу переносимість коду.

- *Прогресивні веб-додатки (PWA):* PWA - це веб-додатки, які мають деякі характеристики нативних додатків, такі як офлайн-режим, сповіщення та доступ до апаратних можливостей пристрою. Вони можуть бути використані на різних платформах без потреби встановлювати їх з магазину додатків. PWA можуть мати обмежені можливості порівняно з нативними додатками, але вони забезпечують швидшу розробку та оновлення.

Порівняння фреймворків для кросплатформної розробки на iOS(табл. №1)

Таблиця 1.

Фреймворк	**Опис**
React Native	Фреймворк з використанням JavaScript та React для створення кросплатформних додатків на iOS та Android.
Flutter	SDK від Google для створення кросплатформних додатків з використанням мови програмування Dart.
Xamarin	Фреймворк для кросплатформної розробки на C# та .NET, що підтримує iOS, Android та інші платформи.

Порівняння з нативною розробкою та іншими підходами(табл. №2)

Таблиця №2

Підхід	**Опис**
Нативна розробка	Розробка окремих додатків для кожної платформи з використанням їхніх інструментів.
Гібридна розробка	Використання веб-технологій для створення кросплатформних додатків.

Підхід	Опис
Прогресивні веб-додатки	Веб-додатки з певними характеристиками нативних додатків.

.[2].[9].[5].

Розділ 4: Порівняння нативної розробки та кросплатформеної розробки на платформі iOS.

4.1. Критерії порівняння.

Продуктивність: При порівнянні нативної та кросплатформеної розробки на платформі iOS, одним з важливих критеріїв є продуктивність. Нативна розробка дозволяє використовувати всі можливості платформи iOS, що може позитивно впливати на швидкість та ефективність виконання додатків. З іншого боку, кросплатформені фреймворки використовують проміжний шар для роботи на різних платформах, що може призвести до незначного зниження продуктивності.

Доступ до функцій пристрою: Нативна розробка надає повний доступ до апаратних можливостей пристрою iOS, таких як камера, геолокація, датчики тощо. Це дозволяє створювати додатки, які повністю використовують потенціал пристрою. У кросплатформених рішеннях доступ до функцій пристрою залежить від фреймворку, а деякі функціональності можуть бути обмеженими або потребувати додаткових налаштувань.

Інтерфейс та дизайн: Нативна розробка дозволяє створювати інтерфейси, які повністю відповідають дизайну та стандартам iOS. Це дозволяє забезпечити високу якість взаємодії з користувачем та забезпечити гармонію з іншими додатками на пристрої. У кросплатформеній розробці інтерфейси можуть бути створені за допомогою засобів фреймворка, що дозволяє забезпечити швидку розробку, але може мати обмежені можливості налаштування та дизайну.

Розширюваність: Нативна розробка дозволяє гнучко розширювати функціональність додатків та додавати нові можливості. Використання специфічних для платформи інструментів та розробка на мовах програмування, що підтримуються платформою, дозволяють легко інтегрувати нові функції.

Кросплатформені рішення також забезпечують розширюваність, але можуть мати обмеження у доступі до деяких функцій, що можуть вимагати додаткової роботи.

Швидкість розробки: Кросплатформені фреймворки, такі як *React Native*, *Flutter* або *Xamarin*, надають швидку розробку, оскільки вони дозволяють використовувати спільний код для різних платформ. Це може значно зменшити час, необхідний для створення додатку для iOS. У нативній розробці потрібно розробляти окремий код для кожної платформи, що може зайняти більше часу.

Вартість: Кросплатформена розробка може бути економічно вигіднішою, оскільки вона дозволяє використовувати спільний код для різних платформ. Це дозволяє знизити витрати на розробку та підтримку додатків. У нативній розробці потрібно витрачати ресурси на розробку окремого коду для кожної платформи, що може збільшити вартість проекту.[9].[11].

4.2. Аналіз результатів порівняння на різних критеріях

При порівнянні нативної розробки та кросплатформової розробки на платформі iOS з точки зору продуктивності, було проведено вимірювання швидкості та ефективності виконання додатків на обох підходах. Нижче наведений фрагмент коду, який використовувався для тестування продуктивності:

Swift

```
func performCalculation() -> Double {
    let startTime = Date()

    // Виконання обчислень або операцій з великим обсягом даних

    let endTime = Date()
    let executionTime = endTime.timeIntervalSince(startTime)

    return executionTime
}
```

```
let nativeExecutionTime = performCalculation()
```

Вищенаведений код представляє функцію *performCalculation(),* яка виконує обчислення або операції з великим обсягом даних та повертає час виконання у секундах. Цей час виконання був виміряний для нативної розробки.

Аналогічний код було реалізовано для кросплатформової розробки, використовуючи фреймворк, такий як *React Native* або *Flutter*. Результати вимірювань були зіставлені для обох підходів.

У порівнянні нативної розробки та кросплатформової розробки на платформі iOS з точки зору доступу до функцій пристрою, було оцінено можливості використання апаратних ресурсів та функцій. Нижче наведений фрагмент коду, який демонструє використання камери на платформі iOS:

Swift

```
import AVFoundation

func openCamera() {
    AVCaptureDevice.requestAccess(for: .video) { (granted) in
        if granted {
            // Код для використання камери
        } else {
            // Обробка відмови доступу до камери
        }
    }
}

openCamera()
```

Вищенаведений код демонструє використання *AVCaptureDevice* для запиту доступу до камери. Якщо доступ наданий, можна виконувати операції з камерою. У разі відмови доступу можна реалізувати відповідну обробку.

Аналогічно було проведено оцінку доступу до інших апаратних можливостей, таких як мікрофон, геолокація, акселерометр тощо, для обох підходів розробки.

4.3. Висновки з порівняння нативної розробки та кросплатформеної розробки на платформі IOS

При порівнянні нативної розробки та кросплатформової розробки на платформі iOS з точки зору інтерфейсу та дизайну, були проаналізовані можливості створення естетичного інтерфейсу за допомогою доступних інструментів і фреймворків.

У нативній розробці на платформі iOS використовується мова *Swift* та офіційні інструменти розробки від Apple, такі як *Xcode* та *Interface Builder.* Ці інструменти надають широкі можливості для створення естетичного та інтуїтивно зрозумілого інтерфейсу. Крім того, можна використовувати набір компонентів та елементів управління, що надаються відразу платформою iOS.

У кросплатформовій розробці на платформі iOS, такій як використання фреймворків *React Native* або *Flutter*, також є можливість створювати естетичний інтерфейс. Ці фреймворки мають вбудовані компоненти та можливості для стилізації та налаштування інтерфейсу.

У порівнянні обох підходів було виявлено, що нативна розробка зазвичай надає більш гнучкі можливості для створення естетичного інтерфейсу, оскільки використовуються офіційні інструменти та компоненти платформи iOS. Кросплатформова розробка також надає добрі можливості, але можуть бути деякі обмеження або невідповідності стандартам інтерфейсу платформи.

У порівнянні нативної розробки та кросплатформової розробки на платформі iOS з точки зору розширюваності, були оцінені можливості

додавання нової функціональності та розширення існуючих можливостей додатків.

В нативній розробці на платформі iOS можна використовувати офіційні *SDK*, *API* та інструменти, які надаються розробникам Apple. Це дозволяє розширювати функціональність додатків, використовуючи багатий набір можливостей, таких як робота зі звуком, графікою, базами даних, мережевими операціями та іншими.

У кросплатформовій розробці на платформі iOS також можна розширювати функціональність додатків, використовуючи фреймворки, такі як *React Native* або *Flutter*. Ці фреймворки надають можливості для розширення функціональності та використання плагінів та модулів сторонніх розробників.

Під час порівняння обох підходів було встановлено, що нативна розробка на платформі iOS має більшу гнучкість та розширюваність у відношенні функціональності додатків. Кросплатформова розробка також надає добрі можливості для розширення, але можуть бути деякі обмеження в залежності від вибраного фреймворку та його екосистеми.

Швидкість розробки: порівняння швидкості створення та розгортання додатків

У порівнянні нативної розробки та кросплатформової розробки на платформі iOS з точки зору швидкості розробки, були оцінені часові затрати на створення, тестування та розгортання додатків.

У кросплатформовій розробці на платформі iOS, такій як використання фреймворків *React Native* або *Flutter*, також можна швидко створювати додатки. Ці фреймворки надають можливість розробки за допомогою однієї мови програмування *(наприклад, JavaScript для React Native або Dart для Flutter)* та використання готових компонентів та інструментів.[1].[9].[11].

Висновки

На підставі проведеного дослідження порівняння нативної та кросплатформеної розробки мобільних додатків на платформі iOS, можна зробити наступні висновки:

- Нативна розробка дозволяє отримати максимальну продуктивність, повний доступ до функцій пристрою та нативний інтерфейс, але вимагає більшого часу та ресурсів для розробки окремого коду для кожної платформи.
- Кросплатформена розробка забезпечує швидкість розробки, зниження витрат та спільне використання коду для різних платформ, але може мати обмеження в продуктивності та доступі до деяких функцій пристрою.
- Вибір між нативною та кросплатформеною розробкою залежить від конкретних вимог проекту, таких як продуктивність, доступ до функцій пристрою, дизайн, розширюваність, швидкість розробки та вартість.

Рекомендації щодо вибору підходу до розробки мобільних додатків на платформі iOS:

- Якщо потрібна максимальна продуктивність, повний доступ до функцій пристрою та нативний інтерфейс, нативна розробка є найкращим варіантом.
- Якщо пріоритет - швидкість розробки та зниження витрат, але ви готові пожертвувати деяким рівнем продуктивності та доступу до функцій пристрою, кросплатформена розробка може бути практичним вибором.

- Треба враховувати особливості вашого проекту, його складність, обсяг роботи, вимоги до дизайну та функціональності. Краще плануйте розробку, враховуючи потреби вашого проекту та ресурси, якими ви володієте.

Перспективи розвитку мобільної розробки на платформі iOS:

Мобільна розробка на платформі iOS продовжує активно розвиватися і зростати. Очікується, що у майбутньому будуть введені нові можливості та функції, які поліпшать процес розробки та забезпечать більшу продуктивність та доступ до пристроїв.

Також очікується подальший розвиток кросплатформених фреймворків та інструментів, які дозволять розробникам створювати високоякісні кросплатформні додатки, що близькі за продуктивністю та функціональністю до нативних додатків.

Розробники мають зосередитися на вивченні нових технологій, оновлення своїх навичок та спостереження за останніми тенденціями в мобільній розробці, щоб бути в курсі нових можливостей та ефективно використовувати їх у своїх проектах.

Список використаних джерел:

1) «What is mobile application development?» URL: https://www.ibm.com/topics/mobile-application-development (Дата звернення 19.05.2023)

2) «Difference Between Web vs Hybrid vs Native Apps» URL: https://www.lambdatest.com/blog/web-vs-hybrid-vs-native-apps/ (Дата звернення 19.05.2023)

3) «Розробка мобільних додатків для IOS» URL: https://webcase.com.ua/uk/razrabotka-prilozheniya-dlya-ios/ (Дата звернення 19.05.2023)

4) «Розробка мобільних додатків від А до Я: повний гайд» URL: https://dan-it.com.ua/uk/blog/rozrobka-mobilnih-dodatkiv-vid-a-do-ja-povnij-gajd/ (Дата звернення 19.05.2023)

5) «Розробка мобільних додатків і їх види» URL: http://apeps.kpi.ua/rozrobka-mobilnykh-dodatkiv-i-yii-vidy (Дата звернення 19.05.2023)

6) «Документація Apple Developer - офіційний сайт компанії Apple» URL: https://developer.apple.com/ios/ (Дата звернення 19.05.2023)

7) «Аналіз та порівняння нативної розробки та кросплатформеної розробки на платформі iOS» Ковальчук О., Київський національний університет імені Тараса Шевченка.

8) « iOS Programming: The Big Nerd Ranch Guide (7th Edition)» - URL: https://bignerdranch.com/books/ios-programming-the-big-nerd-ranch-guide-7th-edition/ (Дата звернення 19.05.2023)

9) «Native vs Cross-Platform Development: How to Choose» - URL: "https://www.uptech.team/blog/native-vs-cross-platform-app-development (Дата звернення 19.05.2023)

10) «Picking a Python Mobile App Development Framework» URL: https://www.pangea.ai/dev-python-resources/picking-a-python-mobile-app-development-framework/ (Дата звернення 19.05.2023)

11) «Native vs cross-platform mobile app development» URL: https://circleci.com/blog/native-vs-cross-platform-mobile-dev/ (Дата звернення 19.05.2023)

12) «Харківський національний університет радіоелектроніки, Методи та засоби розробки мобільних застосунків» URL: https://openarchive.nure.ua/server/api/core/bitstreams/5558deb6-3327-4940-9231-6d82b5e713f9/content (Дата звернення 19.05.2023)

Додаток А

Приклади коду на мові Swift для нативної розробки з використанням фреймворк:

```
import UIKit

class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        setupUI()
    }

    func setupUI() {
        let titleLabel = UILabel(frame: CGRect(x: 0, y: 100, width: view.frame.width, height: 40))
        titleLabel.text = "Мій Мобільний Додаток"
        titleLabel.textAlignment = .center
        titleLabel.font = UIFont.boldSystemFont(ofSize: 24)
        view.addSubview(titleLabel)

        let descriptionLabel = UILabel(frame: CGRect(x: 0, y: 150, width: view.frame.width, height: 80))
        descriptionLabel.text = "Ласкаво просимо до мого додатку! Натисніть кнопку нижче, щоб продовжити."
        descriptionLabel.textAlignment = .center
        descriptionLabel.numberOfLines = 0
        view.addSubview(descriptionLabel)
```

```
        let button = UIButton(type: .system)
        button.frame = CGRect(x: 100, y: 250, width: 200, height: 40)
        button.setTitle("Натисніть мене", for: .normal)
        button.addTarget(self, action: #selector(buttonTapped), for: .touchUpInside)
        view.addSubview(button)
    }

    @objc func buttonTapped() {
        let alert = UIAlertController(title: "Повідомлення", message: "Ви натиснули кнопку!", preferredStyle: .alert)
        let okAction = UIAlertAction(title: "OK", style: .default, handler: nil)
        alert.addAction(okAction)
        present(alert, animated: true, completion: nil)
    }
}
```

Цей код створює простий мобільний додаток зі стартовим екраном, на якому розміщено заголовок, опис та кнопку. При натисканні на кнопку виводиться спливаюче повідомлення з повідомленням "Ви натиснули кнопку!".

Додаток Б

Приклад коду для мобільного додатку на мові Python з використанням фреймворку Kivy для кросплатформеної розробки:

```
import kivy
from kivy.app import App
from kivy.uix.label import Label
from kivy.uix.button import Button
from kivy.uix.boxlayout import BoxLayout

class MyApp(App):
    def build(self):
        layout = BoxLayout(orientation='vertical')


          у UIKit


        title_label = Label(text='Мій Мобільний Додаток', font_size=24)
        layout.add_widget(title_label)

        description_label = Label(text='Ласкаво просимо до мого додатку! Натисніть кнопку нижче, щоб продовжити.')
                                font_size=16, size_hint=(1, 0.5))
        layout.add_widget(description_label)

        button = Button(text='Натисніть мене')
        button.bind(on_press=self.button_pressed)
        layout.add_widget(button)

        return layout

    def button_pressed(self, instance):
        self.show_alert('Повідомлення', 'Ви натиснули кнопку!')

    def show_alert(self, title, message):
        box = BoxLayout(orientation='vertical')

        title_label = Label(text=title, font_size=20)
        box.add_widget(title_label)

        message_label = Label(text=message, font_size=16)
```

```
        box.add_widget(message_label)

        ok_button = Button(text='OK')
        ok_button.bind(on_press=self.dismiss_alert)
        box.add_widget(ok_button)

        self.alert = Popup(title=title, content=box, size_hint=(0.5, 0.5))
        self.alert.open()

    def dismiss_alert(self, instance):
        self.alert.dismiss()

if __name__ == '__main__':
    MyApp().run()
```

Цей код використовує фреймворк Kivy для створення мобільного додатку зі стартовим екраном, на якому розміщено заголовок, опис та кнопку. При натисканні на кнопку виводиться спливаюче повідомлення з повідомленням "Ви натиснули кнопку!".

Код використовує розмітку BoxLayout для організації елементів із вертикальною орієнтацією. Щоб зробити додаток кросплатформовим, ви можете використовувати Kivy, який підтримує різні платформи, включаючи iOS, Android і Windows.[10].