Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів розробки прогресивних та гібридних веб додатків порівняно з нативними додатками
(тема)

Виконав:
студент (ка) 2 курсу, групи ІПЗм-22-1

Авербах Д.М.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-наукова

Керівник доц. Русакова Н.Є.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Авербаху Данилу Михайловичу
(прізвище, ім’я, по батькові)

1. Тема роботи «Дослідження методів розробки прогресивних та гібридних веб додатків порівняно з нативними додатками»
Затверджена наказом по університету від 29.03.2024р. № 250 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 20.06.2024
3. Вихідні дані до роботи встановлений календарний план роботи, електронні ресурси за обраною тематикою, документація PWA, React Native та Kotlin (Android).
4. Перелік питань, що потрібно опрацювати в роботі мета роботи, аналіз предметної галузі, постановка задачі, теоретичне та практичне дослідження методів розробки додатків різного типу.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі та постановка задачі	23.01 – 14.02.24	*виконано*
2	Аналіз та вибір API для дослідження	15.02 – 24.02.24	*виконано*
3	Аналіз та моделювання предметної галузі	17.02 – 28.02.24	*виконано*
4	Планування експериментів	25.02 – 28.02.24	*виконано*
5	Програмна реалізація кожної з обраних для дослідження моделей	25.02 – 01.04.24	*виконано*
6	Експериментальні дослідження	02.04 – 20.04.24	*виконано*
7	Аналіз результатів експериментальних досліджень та розробка рекомендацій	20.04 – 23.04.24	*виконано*
8	Написання та оформлення тез доповіді	17.04 – 23.04.24	*виконано*
9	Підготовка пояснювальної записки	01.04 – 20.05.24	*виконано*
10	Підготовка презентації та доповіді	20.05 – 06.06.24	*виконано*
11	Перевірка на плагіат	10.06.24	*виконано*
12	Нормоконтроль	10.06.24	*виконано*
13	Рецензування	15.06.24	*виконано*
14	Занесення диплома в електронний архів	17.06.24	*виконано*
15	Попередній захист	17.06.24	*виконано*
16	Допуск до захисту у зав. кафедри	18.06.24	*виконано*

Дата видачі завдання «*29*» *березня* 2024 р.

Студент _____ Авербах Д.М.

Керівник роботи _____ доцент каф. ПІ Русакова Н.Є.

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 65 сторінок, 10 рисунків, 3 таблиці, 5 додатків, 26 джерел.

ГІБРИДНІ ДОДАТКИ, НАТИВНІ ДОДАТКИ, ПРОГРЕСИВНІ ДОДАТКИ, ANDROID, IOS, KOTLIN, PWA, REACT NATIVE.

Об’єктами дослідження є фреймворки та бібліотеки, які дозволяють будувати прогресивні, гібридні та нативні додатки.

Метою роботи є дослідження методів побудови прогресивних, гібридних та нативних додатки з точки зору продуктивності, енергоспоживання, часу відгуку, використання пам'яті та зручності користування.

Дослідження має на меті визначити найкращі підходи для розробки мобільних та веб-додатків в залежності від конкретних умов та вимог проекту, а також надання рекомендацій щодо вибору відповідної технології.

HYBRID APPS, NATIVE APPS, PROGRESSIVE APPS, ANDROID, IOS, NATIVE, PWA, REACT NATIVE.

The objects of research are frameworks and libraries that allow building progressive, hybrid and native applications.

The purpose of the work is to study the methods of building progressive, hybrid and native applications from the point of view of performance, energy consumption, response time, memory usage and ease of use.

The research aims to determine the best approaches for developing mobile and web applications depending on the specific conditions and requirements of the project, as well as providing recommendations for choosing the appropriate technology.

Заява щодо самостійного виконання кваліфікаційної роботи та можливості її публікації в електронному архіві відкритого доступу ElArKhNURE.

Я, Авербах Данило Михайлович, студент гр. ІПЗм-22-1, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів розробки прогресивних та гібридних веб додатків порівняно з нативними додатками», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

Вступ 8
1 Аналіз предметної галузі 9
1.1 Опис предметної галузі 9
1.2 Актуальність проблеми 10
1.3 Постановка задачі 11
2 Теоретичне дослідження 13
2.1 Нативні додатки 13
2.1.1 Загальна характеристика 13
2.1.2 Операційна система Android 15
2.1.3 Операційна система iOS 16
2.2 Гібридні додатки 17
2.3 Прогресивні додатки 22
3 Розробка додатків 26
3.1 Концепція додатків 26
3.2 Розробка дизайну 27
3.3 Розробка нативного додатку 29
3.4 Розробка гібридного додатку 32
4.5 Розробка прогресивного додатку 34
4 Порівняння додатків на практиці 37
4.1 Порівняння продуктивності 37
4.2 Порівняння користувацького досвіду 41
4.3 Порівняння доступності 42
4.4 Порівняння використання в автономному режимі 42
4.5 Порівняння складності підтримки 43
4.6 Порівняння безпечності 44
4.7 Підсумкове порівняння 45
4.8 Практична цінність отриманих результатів 45
Висновки 47
Перелік джерел посилання 48

Додаток А Перелік джерел посилання за науковими напрямами керівника та науковців кафедри програмної інженерії 51
Додаток Б Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ ... 52
Додаток В Слайди презентації 53
Додаток Г Апробація результатів роботи 63
Додаток Д Експертний висновок результатів перевірки кваліфікаційної роботи на відповідність оформлення вимогам ДСТУ 3008: 2015 65

ВСТУП

У сучасну цифрову епоху мобільні додатки стали невід'ємною частиною нашого повсякденного життя, пропонуючи широкий спектр функцій та послуг. Розробка програмного забезпечення для мобільних платформ вимагає ретельного дослідження та вибору найбільш підходящої методології, оскільки це безпосередньо впливає на якість продукту, швидкість розробки та взаємодію з різними пристроями та операційними системами.

Метою цієї роботи є поглиблений аналіз та порівняння методологій розробки прогресивних та гібридних мобільних додатків з нативними рішеннями. Ця робота є особливо актуальною, оскільки розробники стикаються з важливим вибором при виборі підходу, який найкраще відповідає конкретним вимогам та очікуванням користувачів проекту.

Ця робота буде приділена порівняльному аналізу основних переваг і недоліків прогресивних і гібридних додатків у порівнянні з нативними рішеннями. Будуть розглянуті такі аспекти, як продуктивність, вартість розробки, швидкість розгортання, адаптивність до різних платформ та споживання ресурсів пристрою.

Оскільки технології розробки додатків швидко змінюються, ця робота має на меті надати розробникам та бізнес-аналітикам інформацію, яка стане важливим ресурсом у визначенні найкращого способу розробки мобільних додатків.

Наукові результати цієї роботи були представлені на V Всеукраїнська студентська науковій конференції «НАУКОВИЙ ПРОСТІР: АНАЛІЗ, СУЧАСНИЙ СТАН, ТРЕНДИ ТА ПЕРСПЕКТИВИ». У доповіді «Дослідження методів розробки прогресивних та гібридних веб додатків порівняно з нативними додатками» було висвітлено результати дослідження [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Опис предметної галузі

З роками сучасне суспільство все більше віддає перевагу мобільним технологіям, що помітно в усіх сферах нашого життя. Мобільні додатки стають невід'ємною частиною роботи, освіти, розваг і навіть охорони здоров'я. Зручні та ефективні інструменти мобільних додатків визнані ключовими факторами, що сприяють комунікації та соціальному розвитку в цифрову епоху [2].

Зростання популярності мобільних додатків супроводжується появою різних операційних систем і пристроїв: хоча Android та iOS визнані провідними операційними системами, розробники також повинні враховувати такі платформи, як Windows і новітні пристрої Інтернету речей (IoT). та інші платформи. Забезпечення сумісності та оптимальної продуктивності в цій різноманітній сфері є ключовим завданням для розробників.

Мобільні додатки поділяються на три основні категорії: нативні, прогресивні та гібридні. Нативні додатки розробляються для конкретної платформи (Android або iOS), прогресивні додатки використовують веб-технології для створення крос-платформних додатків, а гібридні додатки поєднують переваги обох підходів. Вибір типу додатку залежить від вимог проекту, бюджету та стратегії розробки.

Очікування користувачів від мобільних додатків постійно зростають. Швидкість, простота використання, естетичність і безпека - основні характеристики успішного продукту. Підтримка оновлень, адаптація до різних екранів і функціональність для задоволення потреб різних груп користувачів стають стратегічно важливими аспектами розробки.

У цьому контексті дослідження та порівняння прогресивних, гібридних та нативних методів розробки додатків є ключовим викликом для забезпечення високоякісних, конкурентоспроможних мобільних додатків у сучасному цифровому середовищі.

Технологічне середовище для розробки мобільних додатків постійно змінюється, що вимагає від розробників постійної адаптації та використання передових технологій. Індустрія мобільних додатків є висококонкурентною, що

ускладнює диференціацію продуктів, оптимізацію продуктивності та ефективності продукту, а також надання інноваційного користувацького досвіду [2].

Враховуючи широке розповсюдження мобільних телефонів та планшетів у всьому світі, ринок мобільних додатків є перспективною бізнес-можливістю. Тому стратегічно важливо обрати правильний підхід до розробки, щоб максимізувати вплив продукту на цьому конкурентному ринку та досягти успіху в бізнес-середовищі.

Розглядаючи різні аспекти цільової області, зазначимо, що вибір методології розробки мобільного додатку має значний вплив на його успіх, користувацький досвід і вартість розробки. Вивчення прогресивних, гібридних та нативних додатків у контексті цих факторів може допомогти зрозуміти переваги та обмеження кожного методу і надати розробникам основу для усвідомленого вибору найбільш підходящого шляху в конкретних умовах проекту.

1.2 Актуальність проблеми

Актуальність обраної проблеми в контексті вивчення методів розробки мобільних додатків є дуже важливою у світлі стрімкого розвитку інформаційних технологій та поширеності мобільних пристроїв серед населення.

За останні кілька років спостерігається значне зростання кількості користувачів мобільних пристроїв у всьому світі. Статистика показує, що кількість смартфонів і планшетів стрімко зростає, створюючи серйозний виклик для розробників програмного забезпечення. Враховуючи цю тенденцію, розробникам необхідно визначити найкращий спосіб створення мобільних додатків, які будуть не лише ефективними, але й відповідатимуть високим стандартам користувацького досвіду та функціональності [3].

Прогресивні та гібридні додатки набувають все більшої популярності завдяки своїй кросплатформенній природі, що дозволяє ефективніше використовувати ресурси та скорочувати час розробки на різних операційних системах. Однак нативні додатки залишаються невід'ємною частиною мобільної

розробки, забезпечуючи максимальну продуктивність і доступ до функцій, унікальних для кожної платформи.

Оскільки конкуренція зростає, а толерантність користувачів до низької продуктивності та обмеженої функціональності знижується, вибір правильного підходу до розробки стає стратегічно важливим для успіху мобільних додатків на ринку.

Одне з ключових питань, яке необхідно вирішити цій роботі, полягає у визначенні переваг та обмежень кожного методу розробки в контексті зростаючих потреб користувачів та динамічного технологічного середовища. Актуальність цього питання визначається потребами розробників та бізнес-аналітиків, які шукають об'єктивну інформацію для ефективного вибору технічних підходів при створенні мобільних додатків.

1.3 Постановка задачі

Задачею даної роботи є детальне вивчення та порівняння методів розробки мобільних додатків, таких як прогресивний, гібридний та нативний, а також виявлення їхніх переваг та обмежень у різних контекстах.

Для досягнення загальної мети роботи були поставлені наступні завдання:

- провести огляд літератури для вивчення основних принципів та характеристик прогресивних, гібридних та нативних додатків;
- визначити основні методи, що використовуються в прогресивних, гібридних та нативних додатках;
- проаналізувати інструменти розробки, що використовуються для кожного методу;
- розробити додатки трьох типів, що розглядаються на прикладі додатку для прогнозу погоди з однаковим функціоналом;
- провести експерименти і тести для отримання даних про продуктивність і функціональність кожного методу;
- порівняти результати, щоб визначити сильні та слабкі сторони кожного методу розробки;

- визначте випадки використання, в яких певний метод, ймовірно, буде найбільш підходящим;
- зробити висновки на основі проведеного дослідження;
- надати рекомендації щодо вибору методології розробки мобільних додатків для конкретних умов та різних типів проектів.

Очікується, що будуть зроблені обґрунтовані висновки про переваги та недоліки передових, гібридних та нативних методів розробки мобільних додатків. Рекомендації нададуть розробникам та іншим зацікавленим особам практичні приклади для вибору найкращого методу в реальних умовах розробки мобільних додатків.

2 ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ

2.1 Нативні додатки

2.1.1 Загальна характеристика

Нативні додатки спеціально адаптовані до певної апаратної та програмної платформи і написані мовою, розробленою для цієї платформи.

iOS та Android, наприклад, кожна з цих операційних систем має свій власний інструментарій розробки (SDK) і технологічні стеки, пов'язані з певними мовами програмування. Такі додатки оптимізовані для операційної системи, на якій вони працюють, що робить їх ефективними та швидкими.

Порівняємо дві операційні платформи (див. табл. 2.1).

Таблиця 2.1 – Порівняння мобільних операційних систем (створена самостійно)

	Android	iOS
Programming languages	Java, Kotlin, C/C++	Swift, Objective-C, C/C++
Tools: IDEs, libraries	Android Studio (official), IntelliJ IDEA, Eclipse	XCode, AppCode, Swift Playgrounds
Development speed	Relatively slow	Relatively fast
App launch and associated fees	Google Play Store fee — $25, one-time	App Store Fee — $99, yearly

Розробка нативних додатків має свої унікальні особливості, і до переваг такого підходу можна віднести:

– висока ефективність;
– максимальне використання можливостей конкретної платформи;
– покращений користувацький інтерфейс;
– вищий рейтинг в магазині додатків.

Оскільки технологія, що використовується для створення програми, тісно пов'язана з конкретною операційною системою, нативний код забезпечує прямий доступ до всіх функцій цієї системи. Це дозволяє додаткам легко взаємодіяти з функціоналом мобільного пристрою і підвищує загальну продуктивність, особливо при відображенні графіки та мультимедійного контенту. Створення додатків з нативним кодом зменшує час відгуку, ризик збоїв і зависань, а також забезпечує безперебійну роботу додатків, зокрема ігор [4].

Нативні додатки розробляються з урахуванням конкретних завдань і вимог конкретної платформи, що дозволяє їм краще використовувати апаратні можливості пристрою, такі як Bluetooth, NFC, камера і GPS. Це особливо важливо, коли додаток використовує такі дані, як фізичне або географічне розташування. Нативні додатки інтегруються безпосередньо з операційною системою пристрою, що дозволяє користувачам легко взаємодіяти з додатком через знайомий інтерфейс, забезпечуючи позитивний користувацький досвід і постійне повторне використання.

Якість користувацького досвіду є важливим фактором у рейтингах додатків у магазині. Додатки з вищими оцінками користувацького досвіду отримують більше рекомендацій і збільшують доходи компанії. Здається, що нативні додатки мають вищий рейтинг у магазині завдяки своїй високій продуктивності та простоті використання.

З іншого боку, є й недоліки:

– витрати на розробку і багато часу на розробку;
– несумісність з іншими мобільними операційними системами;
– неможливість використовувати в інших операційних системах.

Створення окремих додатків для кожної платформи може подовжити процес розробки, оскільки один і той самий код неможливо перенести між різними системами. Тому може знадобитися залучення додаткових програмістів для роботи на різних платформах. Також важливо заздалегідь прийняти несумісність з іншими операційними системами.

Розробка програми, орієнтованої лише на одну платформу, може призвести до втрачених можливостей, особливо якщо не врахувати інші платформи. Орієнтація на конкретні ринки може призвести до втрати доходів.

2.1.2 Операційна система Android

Операційна система Android була випущена компанією Google 23 вересня 2008 року. До цього, 11 липня 2005 року, Google придбала Android, компанію-стартап, яка розробила операційну систему, зробивши Android одним із основних напрямків діяльності компанії. З тих пір Android стрімко розвивався і зараз встановлено на понад 70% усіх мобільних пристроїв у всьому світі [5], що робить його популярним вибором.

Однією з особливостей Android є високий ступінь фрагментації пристроїв. Це означає, що на ринку існує широкий спектр пристроїв з різними характеристиками, від функціональних можливостей до апаратної конфігурації. Хоча це вигідно для користувачів, оскільки вони можуть вибрати пристрій, який найкраще відповідає їхнім потребам, це створює проблеми для розробників під час розробки програм, які повинні адаптуватися до різних середовищ.

До апаратних особливостей пристроїв відносяться наявність або відсутність фронтальної камери, кількість SIM-карт, фізичних кнопок та інші параметри. Порівняно зі своїм конкурентом iOS, який дозволяє розробляти програми для обмеженої кількості пристроїв, Android пропонує ширший вибір, але також вимагає від розробників більше уваги до деталей.

Однією з труднощів для розробників є врахування різних версій операційної системи та різних оболонок, що може вплинути на представлення інтерфейсу користувача. Оскільки сьогодні використовується так багато версій Android, розробникам потрібно адаптувати свої програми до різних середовищ.

Крім того, розробники також повинні враховувати особливості архітектури самого додатка. На відміну від iOS, де програма є одним цілим, програми Android складаються з логічно незалежних і окремих частин, таких як дії та фрагменти. Це

забезпечує більшу гнучкість під час використання програм на пристроях з різними функціями та характеристиками.

Розробники можуть використовувати мови програмування Java і Kotlin для програмування додатків Android; Java є рідною мовою Android і дозволяє створювати програми, які взаємодіють з операційною системою та обладнанням. Визнана Google як мова вибору для розробки Android, Kotlin є вдосконаленою версією Java з меншою кількістю формальностей і правил; він підтримує сумісність з Java і надає більше можливостей для розробників.

Основними середовищами розробки для Android є Eclipse, IntelliJ IDEA та Android Studio. Кожна з них має власні функції та інструменти, які полегшують розробку додатків для цієї платформи.

2.1.3 Операційна система iOS

Компанія Apple випустила операційну систему iOS у 2007 році, яка успішно працювала як на iPhone, так і на iPad у 2019 році [6]. Проте вже відомо, що компанія розробила власну операційну систему для iPad, що свідчить про стратегічний підхід до розробки продуктів.

Додатки для iOS характеризуються обмеженим спектром типів пристроїв, що робить адаптацію до різних моделей iPhone менш складною, ніж різноманітність Android смартфонів. Користувачі iPhone активно оновлюють свою операційну систему, згідно зі статистикою App Store, і компанія також працює над низкою нових додатків для iPhone, від розробників вимагається адаптувати свої додатки до нових ситуацій, таких як темна тема, запроваджена у 2019 році [6].

Екрани сучасних пристроїв на iOS вражають своєю високою роздільною здатністю, що дозволяє використовувати дрібні шрифти без спотворень, які можуть виникати на дисплеях нижчої якості. Однотипна архітектура пристроїв спрощує розробку додатків, оскільки немає необхідності виконувати додаткові перевірки на наявність різних функцій, таких як камери, GPS-датчики, акселерометри тощо.

Apple пропонує можливість розробляти додатки для iOS на мовах програмування Objective-C та Swift. Objective-C є старішою мовою, але

залишається однією з основних мов для розробки додатків для платформи Apple. В 2014 році Swift була представлена в якості альтернативної мовою з більш простими та сучасними можливостями [7].

Успішна реалізація об'єктно-орієнтованого програмування (ООП) у Swift робить розробку більш гнучкою та приємною. Сувора типізація об'єктів та сучасні функції, такі як узагальнення та закриття, зменшують кількість помилок та роблять процес розробки більш продуктивним.

Основними середовищами розробки для iOS є Xcode від Apple та AppCode від JetBrains, які надають інтегровані інструменти для розробки та тестування додатків.

2.2 Гібридні додатки

Гібридні додатки – це додатки наступного покоління, оскільки вони забезпечують незалежність від платформи та одночасно доступ до вбудованих функцій пристрою. Крім того, вони ефективно працюють в автономному режимі та можуть бути завантажені в магазини додатків. Все це досягається за допомогою кращих гібридних фреймворків, включаючи Flutter та React Native.

Flutter – портативний інструментарій інтерфейсу для користувача створення спочатку скомпілованих додатків на мобільних, веб та настільні комп'ютери з однієї кодової бази. Офіційно представлений компанією Google у грудні 2018 року [8].

Він заснований на Dart – швидкій об'єктно-орієнтованій мові програмування. Він відносно новий і простий у освоєнні – особливо для досвідчених розробників, більш знайомих із Java та C#. Більше того, Dart підтримує sound null safety. Це означає, що змінні не можуть приймати значння null, якщо ви цього не скажете. Отже, за замовчуванням типи в мові Dart не підлягають обнулінню, що запобігає помилкам типізації [8].

React Native – фреймворк для побудови нативних додатків з використання React на мові JavaScript. Представлений раніше – 15 березня 2015 року на конференції Facebook [9].

Обидва ці інструменти є open source проектами з відкритим вихідним кодом. Програми Flutter виглядають так само добре в сучасних операційні системи, як і в старих версіях. Оскільки у них лише одна кодова база, додатки виглядають і поводяться однаково на iOS та Android, але завдяки Material Design та віджетам Cupertino вони також можуть імітувати дизайн платформи.

Flutter містить два набори віджетів, які відповідають певним мовам дизайну: віджети Material Design реалізують однойменна мова дизайну Google; віджети Cupertino імітують дизайн iOS від Apple. Це означає, що програма Flutter буде виглядати і поводитися природно кожної платформі, імітуючи їх рідні компоненти [8].

Компоненти програми, написаної за допомогою React Native, виглядають так само, як і в будь-якій спочатку вбудованій програмі на смартфоні (наприклад, кнопка на пристрої iOS виглядає так само, як рідна кнопка iOS, і те саме на Android).

Той факт, що React Native використовує власні компоненти, дозволяє з упевненістю стверджувати, що після будь-якого оновлення користувальницького інтерфейсу операційної системи компоненти вашого програми також будуть миттєво оновлені.

Тим не менш, це може зламати інтерфейс користувача програми, але це відбувається дуже рідко. Якщо потрібно, щоб програма виглядала майже ідентично на різних платформах, а також коректно відображалося на екрані пристроїв зі старою версією операційної системи (як це передбачено у Flutter), потрібно розглянути можливість використання сторонніх бібліотек. Вони дозволять використовувати віджети Material Design замість компонентів, встановлених по замовчуванням.

З Flutter 2 (анонсованим у березні 2021 року) можна використовувати одну і ту ж кодову базу для доставки власних додатків до п'яти операційних систем: iOS, Android, Windows, macOS та Linux; а також веб-інтерфейси, орієнтовані на такі браузери, як Firefox, Chrome, Safari або Edge [8].

React Native у свою чергу підтримує iOS та Android, але є окремі бібліотеки, які дозволяють використовувати один і той же код для створення програм iOS, Android, Web та Windows.

Час виходу на ринок програми з використанням Flutter зазвичай набагато швидше, ніж за нативної розробки. З використанням React Native зазвичай розробка також займає небагато часу, проте цей фреймворк використовує bridge та native elements, тому для кожної платформи може потрібна окрема оптимізація – проблема, з якою Flutter на основі віджетів не стикається. Це може зробити розробку програми з React Native довше.

Компоненти в React Native досить прості, тому якщо потрібно щось додаткове, то при стилізації знадобляться значні зусилля. Тільки кілька компонентів адаптовані до платформи, у той час як більшу частину часу доводиться використовувати інший компонент як для iOS, так і для Android; або стилізувати його інакше.

З іншого боку, з Flutter все є віджетом, що робить їх легко налаштовуваними. Більшість віджетів адаптивні, і можна використовувати один і той же віджет як на Android так і на iOS. Коли справа доходить до продуктивності, Flutter має перевагу, оскільки він скомпільований у власні бібліотеки ARM або x86, що робить його справді швидким. React Native не компілюється в власний код, і в нього все ще є шар JavaScript, що робить його меншим продуктивнішим, ніж Flutter [10].

Важко порівнювати їх загалом, тому що вони мають різну архітектуру і, отже, функціонують по-різному. Скористаємося деякими конкретними критеріями, щоби оцінити їх.

React Native використовує JavaScript як основу, в той час як Flutter використовує Dart. З цієї точки зору, React Native легше вивчати, тому що JavaScript є мовою № 1 серед програмістів, згідно опитування StackOverflow. Очевидно, що з урахуванням того, що майже 63% фахівців використовують його [11], перехід на React Native не складе труднощів. З Flutter ситуація інша. Цей фреймворк ґрунтується на Dart. Це новий і досі незріла мова. Незважаючи на те, що молоді розробники часто віддають перевагу вивчати його, все ще дуже мало програмістів,

які вже освоїли його. Однак, якщо розробник коли-небудь працював з об'єктно-орієнтованими мовами, він швидко освоїть Dart.

Продуктивність - це, ймовірно, найбільша різниця між Flutter та React Native. У цій категорії Flutter є безперечним переможцем. Завдяки движку C++, що використовується у фреймворку Flutter, і графічній бібліотеці Skia, це дозволяє створювати програми з більш високою продуктивністю порівняно з React Native. Крім того, процес кодування відбувається швидше. React Native використовує JavaScript міст до нативної мови. Це означає, що дві сторони – JavaScript та нативна мова обмінюються повідомленнями JSON для встановлення з'єднання. Ці повідомлення є асинхронними, і це призводить до досить плавної анімації. Однак, порівняно з Flutter, продуктивність інтерфейсу користувача все ще може мати деякі проблеми, такі як затримки у рендерингу. Фреймворк Flutter не має цієї проблеми, оскільки він не має цього мосту. Він з'єднується з власними компонентами з вбудованими бібліотеками та такими фреймворками, як Material Design чи Cupertino.

На даний момент у Flutter є багатший набір можливостей. React Native має використовувати сторонні бібліотеки, у той час як Flutter має вбудовані компоненти рендерингу, інструменти для тестування, навігації, доступу до API пристроїв і т.д.

Коли справа доходить до вибору середовища розробки, важливо вибрати ту, яка буде найзручнішою. У випадку Flutter цей вибір не дуже великий, тому що технологія все ще нова. Visual Studio Code, IntelliJ та Android Studio підтримують його. Зважаючи на те, що для React Native можна вибрати практично будь-яке середовище розробки.

Це ще одна різниця між Flutter та React Native, яке так часто обговорюють програмісти-початківці. Flutter має більше багату документацію порівняно з React Native. Це дозволяє молодим розробникам вивчати технологію, дотримуючись її. Але у випадку з React Native такий спосіб не спрацює, тому що його документація більше орієнтована на опис складних процесів і передбачає, що кожен, хто її читає, вже знайомий з JavaScript [10].

У розгортанні Flutter знову перемагає. Він має дуже простий робітник процес розгортання програми в App Store або Google Play. Можна, можливо зробити за допомогою командного рядка (якщо не потрібні додаткові параметри налаштування). Але навіть для них є безліч сторонніх інструментів. За допомогою React Native можна розгорнути програму тільки з допомогою сторонніх рішень. Більше того, знайти посібник про те, як це зробити, може виявитися непростим завданням.

Дедалі більше компаній залучають Flutter. В кінці кінців, спостерігаються щомісячні покращення в SDK Flutter, оскільки Google продовжує удосконалювати свій інструмент. Більше того, Flutter дозволяє створювати не тільки мобільні програми, але й програми для Інтернету та десктопів (підтримка десктопів Flutter доступна у вигляді бета-версії на стабільному каналі).

Зібравши все це разом та враховуючи, що провідні компанії, такі як Alibaba, вже використовують Flutter – майбутнє виглядає перспективним для інструментарію.

Що стосується React Native - Facebook в даний час зосереджений на масштабну перебудову архітектури технології.

Команда робить все можливе, щоб покращити підтримку як для користувачів React Native, так і для ширшої спільноти і завдяки цьому, спільнота тепер може легко пропонувати зміни в основних функціях фреймворку за допомогою процесу RFC, який використовує виділений репозиторій GitHub

Фактичним результатом таких покращень архітектури є Hermes – движок JavaScript з відкритим вихідним кодом, оптимізований для мобільних додатків, що покращує час взаємодії, а також знижує розмір програми та використання пам'яті [12].

Створення відкритого середовища для обговорення React Native – це важливий крок, який є як ознакою постійного вдосконалення, так і сигналом до світлого майбутнього технології.

Враховуючи, що React Native займає таке стабільне становище ринку і знаходиться на траєкторії безперервного розвитку, малоймовірно, що в найближчим часом він втратить свою актуальність.

2.3 Прогресивні додатки

Прогресивний веб-додаток (Progressive Web Application, PWA) - це технологія веб-розробки, яка візуально та функціонально трансформує сайт на додаток для різних пристроїв [13].

В даний час незважаючи на те, що технологія PWA дозволяє суттєво полегшити розробку у сфері веб-додатків, небагато розробників готові використати цю технологію у своїх проектах.

Технологія PWA створена в корпорації Apple у 2007 році та спочатку була доступна у браузері Safari для операційних систем macOS та iOS. Однак технологія не вразила розробників і незабаром була забута. Але в березні 2018 року відбулися суттєві зміни у сфері реалізації веб-стандартів. В оновленій операційній системі iOS корпорація Apple додала підтримку технології Service Worker у мобільну версію браузера Safar, і після цього всі інші компанії-розробники софту в області веб-технології почали активно впроваджувати технологію PWA у свої платформи [14].

PWA програми повторюють функціонал мобільного додатка. Веб-сторінки розміщуються на домені, а на пристрої відкриваються через браузери. Можна назвати прогресивні веб-застосунки просунутими версіями сайту, які адаптуються під пристрій користувача. При цьому спеціальна розробка під Android та IOS не потрібна.

Веб-застосунки при необхідності працюють автономно, надсилають Push повідомлення клієнтам, що важливо для постійної комунікації бренду з аудиторією. Встановлюються зазвичай через сайт компанії, але вже з'явилися і каталоги з Progressive Web Apps: appsco.pe, findpwa.com та інші.

Веб-програми прискорюють взаємодію з користувачами, створюють конкурентні переваги для бренду, просувають його товари та послуги через PWA

Push notifications. Тому технологію використовують різні бізнеси, особливо ті, де клієнти роблять покупки та замовляють послуги регулярно: магазини, салони краси, клінінг, автосалони, ресторани, доставка їжі, ЗМІ тощо.

Переваги PWA:

– прогресивний веб-додаток для користувача майже не відрізняється від мобільного, при цьому його розробка дешевша в рази;
– зробити PWA версію сайту набагато швидше. Існують навіть конструктори для створення. Наприклад, Microsoft заявляє, що прогресивну версію сайту через інструмент PWABuilder вдасться зібрати у перерві на обід;
– можливість встановлення з сайту, а не зі спеціального каталогу. Так PWA додаток не потрібно підганяти під вимоги стора, чекати на схвалення і боятися ризику видалення;
– підвищення лояльності та залученості користувачів до контенту бренду, які призводять до підвищення продажів та середнього чека. Нагадування клієнтам про акції та новини бренду відбувається через PWA notifications — звичні Push повідомлення, які повідомляють про акції бренду без запуску реклами;
– індексація програми пошуковими системами. Так як PWA - це просунута версія сайту, для неї працюють правила SEO-оптимізації. Ця особливість надає додатковий трафік;
– фвтоматичні оновлення та менша вага (не більше 3 мегабайт), ніж у мобільного додатка. Користувачеві не доведеться займати зайву пам'ять пристрою;
– швидка швидкість роботи в порівнянні з сайтом та можливість перейти в офлайн-режим (але з обмеженнями);

Недоліки PWA:

– не повний доступ до апаратних можливостей пристроїв, на які вони встановлюються, немає можливості використовувати всі технології. Наприклад, для IOS не вдасться скористатися Face ID та Bluetooth. Apple

дуже повільно адаптується під веб-програми, тому на їх пристроях є проблема з відправкою Push;

– залежність від браузера. Можливі обмеження через використання застарілих версій операційних систем, де не можна інсталювати оновлення. Також браузери не завжди забезпечують повний функціонал веб-додатків;
– велика витрата батареї. PWA кодують на JavaScript - мовою програмування, яка витрачає більше ресурсів системи, ніж нативні мови;
– найменша віральність. Користувачі звикли завантажувати програми з App Store (IOS) та Google Play (Android), а не шукати їх на сайті брендів. При створенні тільки PWA версії доведеться вкластися у проведення маркетингової кампанії з просування. Або все ж таки пройти через модерацію каталогів додатків, яка не завжди проста і швидка.

PWA приклади є у відомих брендів: Telegram, Youtube, TikTok, Aviasales, Tinder, Pinterest, Forbes, AliExpress, Uber та інших. Деякі з них значно покращили користувальницький досвід та збільшили дохід.

Tinder запустили веб-версію програми, яка займає лише 2,8 мегабайт пам'яті в порівнянні з додатком на Android (30 мегабайт). Швидкість завантаження вдалося зменшити з 11,91 до 4,69 секунди, що покращило user experience.

Pinterest також зміг скоротити час завантаження інтерфейсу з 23 до 5,6 секунди. На 40% збільшився час, який користувачі проводять у сервісі, а прибуток від реклами — на 44%. Значно (60%) підвищилося залучення до контенту [15].

Поліпшити досвід користувача з PWA вийшло у Forbes, додаток якого часто називали громіздким. Тепер люди проводять на 40% більше часу за читанням статей видання та переглядають на 15% більше контенту. Завантажувати сторінки також стали швидше. Якщо раніше очікування тривало в середньому 6,5 секунди, то в новій версії — всього 2,5 секунди [16].

Прогресивний веб-додаток збільшив коефіцієнт конверсії AliExpress на 104%. За один сеанс користувачі стали відвідувати вдвічі більше сторінок і витрачати 74% часу.

Компанія Uber використовує технологію PWA, щоб знизити швидкість завантаження програми на всіх пристроях навіть при 2G з'єднанні. Це важливо для виходу нових ринків. У результаті сервіс відкривається за 3 секунди і займатиме всього 50 кілобайт [17].

Щоб створити веб-програму, потрібні Web App Manifest та Service Worker.

Service Worker — це скрипт, який приймає інформацію (запити) від браузера через безпечне HTTPS з'єднання. Компонент відкриває браузер фоново, навіть якщо сторінка не активна. Завдяки Service Worker користувачеві надсилають Push повідомлення, а програма без участі користувача синхронізується з сайтом. Оновлення відбуваються автоматично, також доступний офлайн-режим [18].

Web App Manifest — маніфест у вигляді файлу manifest.json додається до коду сторінки сайту та передає інформацію браузеру про те, як має відображатися програма на пристрої. Визначає ім'я, ярлик, заставку, тему програми та інші елементи. Маніфест повідомляє, які дані залишаються незмінними, а які оновлюються [19].

Інші важливі елементи для розробки веб-застосунку:

Цифровий SSL сертифікат підтверджує справжність сайту. Потрібний для шифрованого з'єднання HTTPS з'єднання, яке забезпечує безпеку.

Application Shell — це оболонка програми або шаблон, до якого завантажуються дані з веб-сторінок сайту.

3 РОЗРОБКА ДОДАТКІВ

3.1 Концепція додатків

В роботі було розглянуто можливість порівняння трьох типів додатків на однаковому прикладі. Для цього запропоновано приклад додатку з прогнозом погоди. Додаток повинен надавати користувачам доступ до актуальної інформації про погоду разом із розширеними функціями, що забезпечують зручне та персоналізоване використання (див. рис. 3.1).

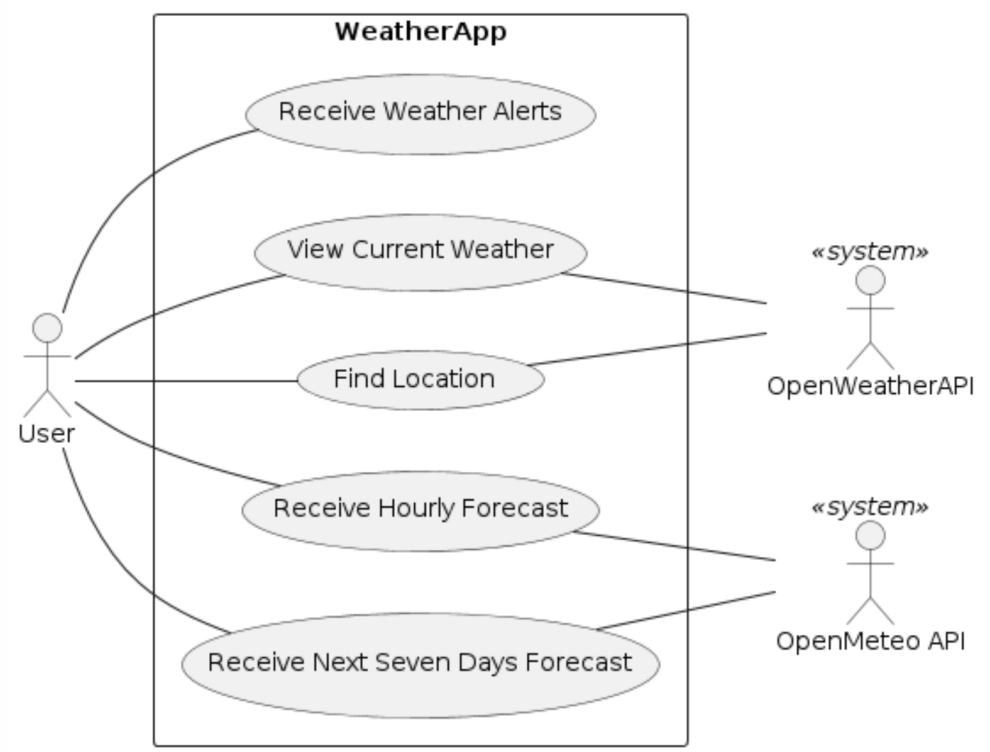

Рисунок 3.1 – Діаграма прецедентів (створений самостійно)

Опис функцій:

– зручний інтерфейс, чистий та інтуїтивно зрозумілий інтерфейс дозволить користувачам швидко отримувати необхідну інформацію про погоду без зайвих зусиль;
– актуальна інформація, додаток буде автоматично оновлювати інформацію про погоду, забезпечуючи користувачам актуальні дані;
– прогноз погоди, користувачі зможуть переглядати детальний прогноз погоди на найближчі дні та години, що дозволить їм краще планувати свої

активності;

– геолокація, додаток буде автоматично визначати місцезнаходження користувача та надавати інформацію про погоду в їхній області;
– сповіщення про погоду, користувачі отримуватимуть сповіщення про значні зміни в погоді, що дозволить їм своєчасно підготуватися до них.
– персоналізація, можливість налаштовувати відображення погодних параметрів та інтерфейсу дозволить кожному користувачеві створити комфортне для себе середовище.

3.2 Розробка дизайну

На основні описаної концепції додатку було створено дизайн, який буде використано для розробки нативного, гібридного та прогресивного мобільного додатку, для подальшого практичного порівняння цих додатків.

Спочатку було створено головку сторінку додатку (див. рис. 3.2), на якій користувач може побачити назву вибраної локацію, погоду на даний момент, атмосферний тиск, силу вітру, вологість. Також, якщо проскролити нижче, то можна буде побачити прогноз погоди по годинам на сьогодні та завтра.

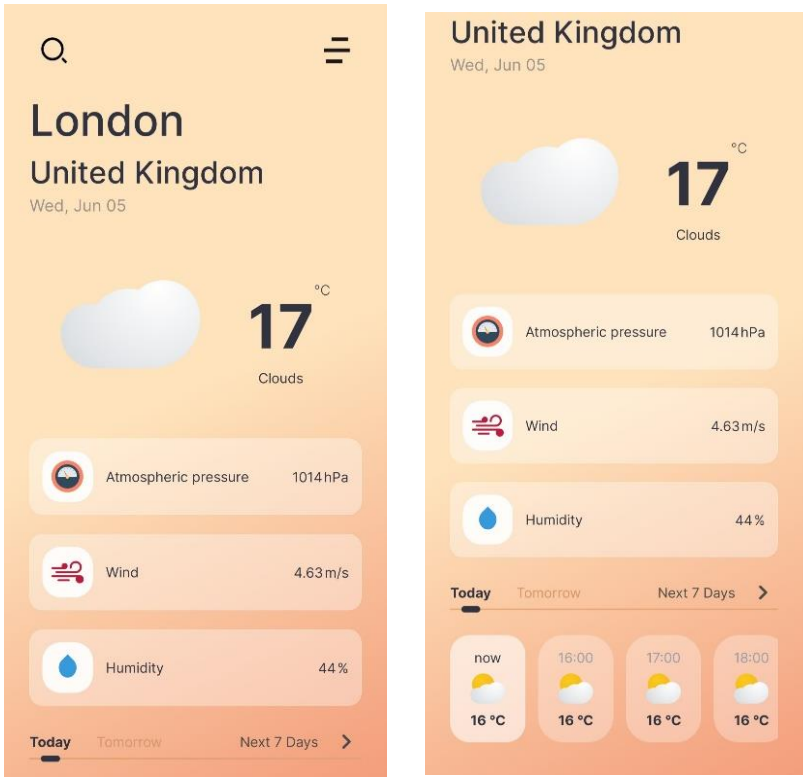

Рисунок 3.2 – Дизайн головної сторінки додатку (створений самостійно)

Також у додатку буде можливість пошуку локації (див. рис. 3.3), в якій користувачу потрібно подивитися прогноз погоди.

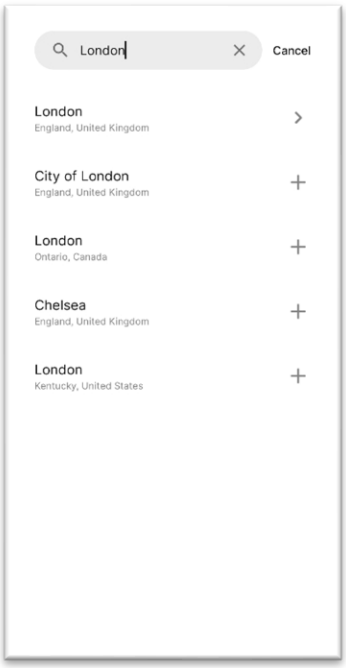

Рисунок 3.3 – Дизайн сторінки пошуку локації (створений самостійно)

Також в дизайні передбачено сторінку, на якій користувач може персоналізувати данні прогнозу погоди (див. рис. 3.4).

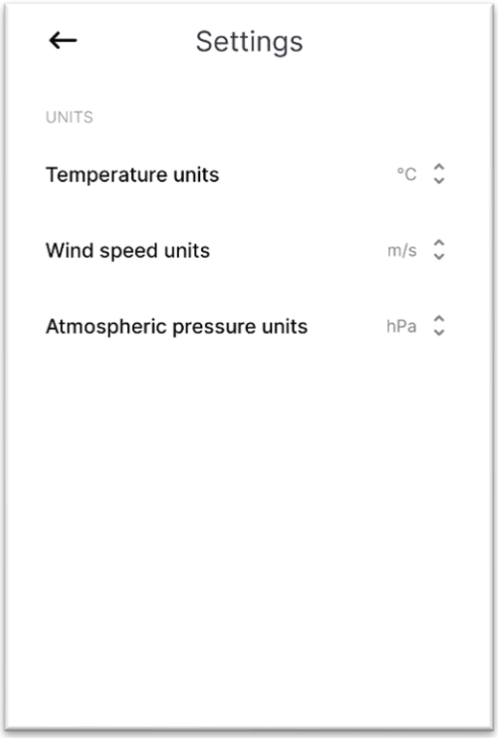

Рисунок 3.4 – Дизайн сторінки налаштувань додатку (створений самостійно)

Також буде створено сторінку з прогнозами погоди на наступній сім днів, в якій буде можливість подивитися мінімальну та максимальну температуру за день та час сходу та заходу сонця (див. рис. 3.5).

Рисунок 3.5 – Дизайн сторінки прогнозу на наступні сім днів (створений самостійно)

Отже, було створено дизайн, який вирізняється естетичністю та функціональністю, забезпечуючи зручний та приємний користувацький досвід

3.3 Розробка нативного додатку

Для розробки нативного додатку було обрано мову програвання Kotlin для операційної системи Android та середовище розробки Android Studio [20].

Основні причини, чому було обрано мову Kotlin:

– сумісність з Java. Kotlin повністю сумісний з Java, тому ви можете легко і поступово інтегрувати Kotlin-код у ваш існуючий проект Java. Це означає, що ви можете почати з Kotlin, не втрачаючи можливості використовувати існуючий Java-код;
– безпека та надійність. Kotlin пропонує безліч функцій, які допомагають уникнути типових помилок під час компіляції. Наприклад, він запобігає

NullPointerException за допомогою системи типізації Nullable та Non-nullable, що робить код більш безпечним та надійним.

– короткість та читабельність коду. Kotlin дозволяє писати менше коду, щоб досягти тих самих результатів, що й у Java. Він має коротшу синтаксичну конструкцію та багато вбудованих функцій, що робить код більш зрозумілим і легшим для обслуговування.
– розширена підтримка функцій Kotlin в Android Studio. Офіційний IDE для розробки Android, Android Studio, надає вбудовану підтримку Kotlin, що означає, що ви можете з легкістю створювати, тестувати та налагоджувати свої додатки Kotlin прямо з одного інтерфейсу.

Також для створення нативного додатку було використано принцип чистої архітектури. Чиста архітектура - це підхід до розробки програмного забезпечення, який має безліч переваг, що роблять його привабливим вибором для сучасних розробників. Чиста архітектура передбачає використання кількох шарів, які розділяють бізнес-логіку від деталей реалізації (рис. 3.6).

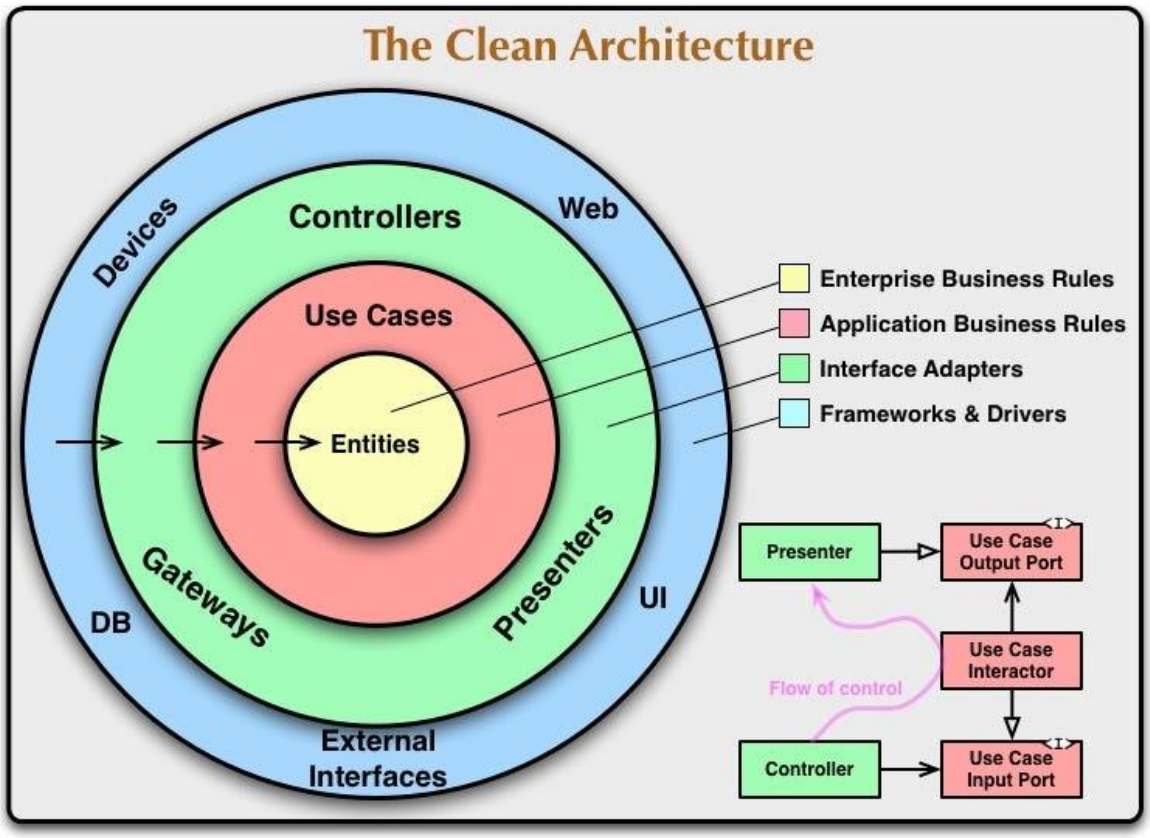

Рисунок 3.6 – Візуальне представлення чистої архітектури (за даними [21])

Ось з чого закладається чиста архітектура по шарам:

– сутності (Enterprise Business Rules). Цей шар містить бізнес-логіку та об'єкти доменної моделі, які є ядром системи. Він незалежний від будь-яких інших шарів. Містить загальні правила та бізнес-логіку. Незалежний від зовнішніх сервісів і фреймворків. Має бути максимально стабільним та змінюватися рідко.
– випидки використання (Application Business Rules). Цей шар відповідає за сценарії використання системи, що описують конкретні дії, які можна виконувати з сутності. Реалізує бізнес-логіку відповідно до вимог системи. Незалежний від зовнішніх фреймворків, баз даних та UI. Забезпечує реалізацію конкретних бізнес-правил і взаємодію з сутності.
– адаптери (Interfaces Adapters). Цей шар містить адаптери та інтерфейси, які переводять дані з формату, що використовується у внутрішніх шарах, у формат, що використовується у зовнішніх. Включає в себе інтерфейси для взаємодії з базами даних, веб-сервісами, користувацьким інтерфейсом тощо. Перетворює дані з одного формату в інший (DTO). Є "мостом" між використовуними випадками та зовнішніми системами.
– фреймворки та драйвери (Frameworks and Drivers). Цей шар містить конкретні реалізації для баз даних, фреймворків, бібліотек та інших деталей реалізації. Включає у себе фреймворки та бібліотеки (наприклад, веб-фреймворки, ORM, драйвери баз даних). Цей шар найменш стабільний і може змінюватися найбільше.

Ось кілька ключових причин, чому було обрано чисту архітектуру для розробки додатку:

– масштабованість. Чиста архітектура розділяє систему на кулі з чіткими обов'язками, що дозволяє легко додавати нові функції та компоненти без впливу на існуючу кодову базу. Це забезпечує можливість зростання додатку відповідно до нових вимог та технологій;
– підтримуваність. Завдяки чіткому розділенню обов'язків між кулями, чиста архітектура спрощує процес підтримки та оновлення коду. Коли

кодова база добре організована, нові розробники можуть швидше зрозуміти її структуру, що знижує витрати на навчання та помилки;

– тестованість. Чиста архітектура дозволяє тестувати бізнес-логіку незалежно від інтерфейсу користувача, бази даних та інших зовнішніх систем. Це полегшує написання автоматизованих тестів, що підвищує надійність та якість програмного забезпечення;
– незалежність від фреймворків. Один із принципів чистої архітектури – це незалежність від фреймворків. Це означає, що додаток можна легко адаптувати до нових технологій чи фреймворків, не змінюючи основної логіки. Така гнучкість особливо важлива у швидкоплинному світі технологій;
– чітке розділення обов'язків. Чиста архітектура сприяє чіткому розділенню обов'язків між різними частинами системи, що зменшує взаємозалежність між компонентами. Це знижує ризик виникнення помилок та конфліктів під час розробки та підтримки додатку;
– легкість у заміні компонентів. Завдяки незалежності шарів, компоненти системи легко замінювати чи оновлювати. Наприклад, ви можете змінити базу даних або інтерфейс користувача без значних змін у бізнес-логіці;
– покращена структура коду. Чиста архітектура забезпечує чітку та логічну структуру коду, що полегшує навігацію та розуміння проекту. Це особливо корисно у великих проектах із великою кількістю розробників.
– довготривала надійність. Розробка з використанням чистої архітектури забезпечує, що додаток буде легко підтримувати та розширювати протягом багатьох років. Це знижує ризики технічного боргу та забезпечує стабільність системи у довгостроковій перспективі.

3.4 Розробка гібридного додатку

Для розробки гібридного додатку було обрано мову програвання Javascript, фреймворк React Native з платформою Expo та середовище розробки WebStorm.

Основні причини, чому було обрано цей підхід:

– кросплатформенність. Одним з найбільших переваг React Native є можливість розробки мобільних додатків для обох платформ - iOS та Android - використовуючи один і той же код. Це зменшує час розробки та зусилля, оскільки розробникам не потрібно створювати окремі версії додатка для кожної платформи;
– швидкість розробки. React Native пропонує швидкий цикл розробки, оскільки він дозволяє гаряче перезавантаження (hot reloading), що дозволяє розробникам бачити зміни в реальному часі без перезапуску додатка [22];
– зменшення витрат. Одна команда розробників може створити мобільний додаток для обох платформ, що зменшує витрати на розробку і підтримку;
– багата екосистема. React Native має широкий вибір сторонніх бібліотек та компонентів, які допомагають розробникам швидше реалізувати певні функції та покращити користувацький інтерфейс.
– підтримка спільноти. React Native має велику та активну спільноту розробників, що означає, що завжди є допомога та ресурси для вирішення проблем та підтримки розробки.

В основі React Native лежать фундаментальні принципи, що забезпечують її гнучкість та масштабованість (див. рис. 3.7):

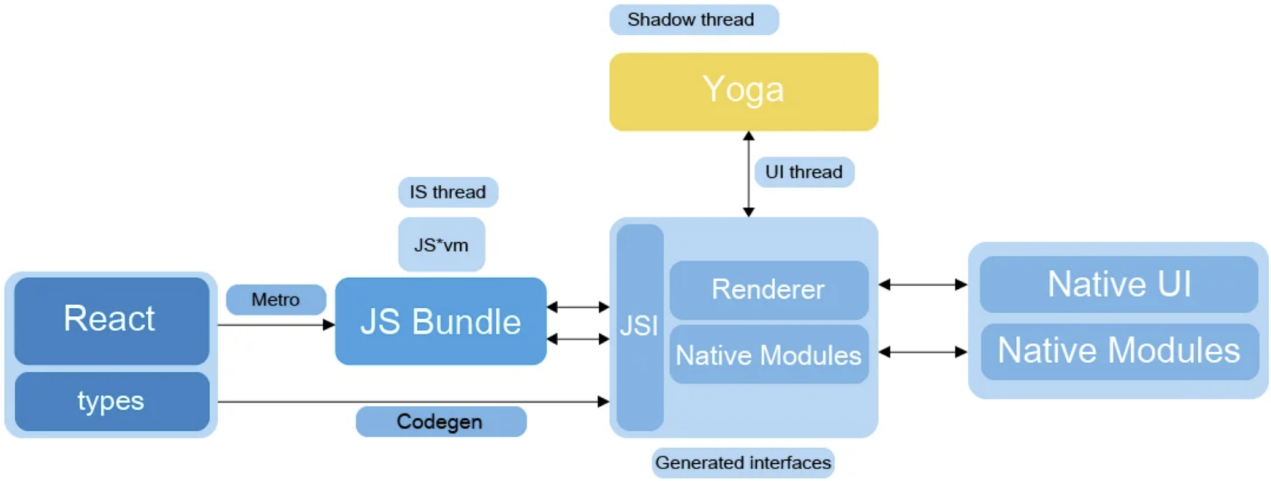

Рисунок 3.7 – Візуальне представлення React Native архітектури (за даними [23])

Архітектура React Native складається з декількох ключових компонентів:

а) JavaScript Layer:
   1) React: бібліотека для побудови користувацьких інтерфейсів, на якій базується React Native;
   2) JavaScript Code: логіка вашого додатку, написана на JavaScript/TypeScript;

б) Bridge:
   1) Bridge: комунікаційний міст між JavaScript-кодом та нативними компонентами. JavaScript-код працює в окремому потоці від нативного коду, і дані передаються через міст асинхронно. Це забезпечує взаємодію між двома різними середовищами виконання;

в) Native Layer:
   1) Native Modules: нативні модулі написані на Java/Kotlin для Android та Objective-C/Swift для iOS, які забезпечують доступ до нативних API.
   2) Native Components: нативні компоненти, які відображаються в мобільному додатку. Вони включають такі елементи, як текстові поля, кнопки, списки тощо;
   3) UI Thread: потік, у якому відображаються та оновлюються нативні компоненти;

г) Renderer**:**
   1) React Native Renderer: шар, який відповідає за відтворення елементів на екрані. Він перетворює React-компоненти в нативні компоненти, використовуючи міст для передачі команд на рендеринг.

4.5 Розробка прогресивного додатку

Для розробки прогресивного додатку було обрано мову програвання Javascript, фреймворк React та середовище розробки WebStorm. основна відмінність між звичайним веб-додатком і прогресивним полягає в наявності Service Worker (див. рис. 3.8). Service Worker - це скрипт, який працює в фоновому режимі веб-переглядача, незалежно від відкритих сторінок. Він дозволяє

реалізувати такі можливості, як офлайн робота, кешування ресурсів, підтримка сповіщень та інші.

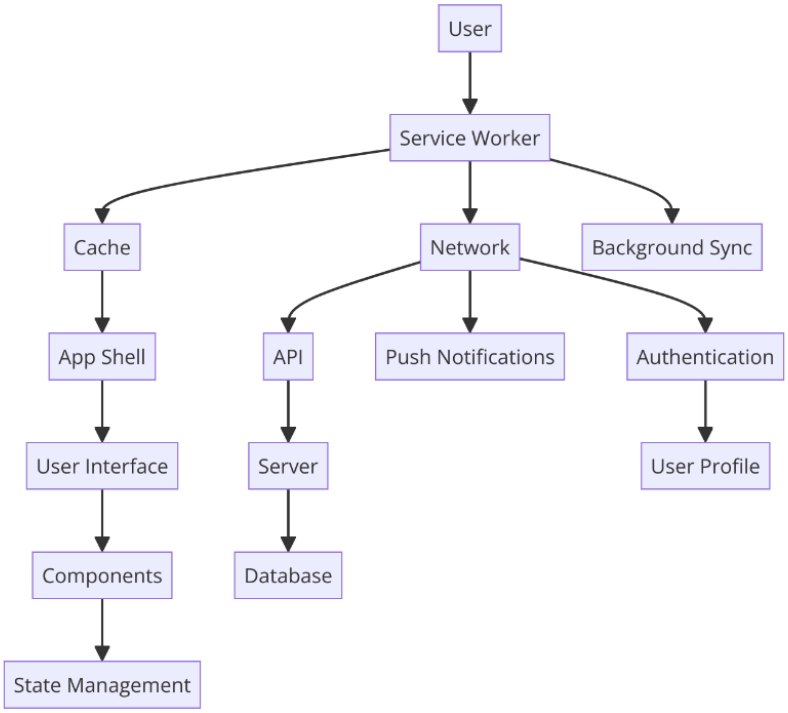

Рисунок 3.8 – Візуальне представлення архітектури Service Worker (створений самостійно)

Основні причини, чому було обрано фреймворк React порівняно з іншими:

– гнучкість. React надає більшу гнучкість у виборі інструментів і бібліотек для додаткової функціональності. Ви можете використовувати будь-які бібліотеки для управління станом, маршрутизації, стилізації тощо, це дозволяє розробникам створювати додатки, які відповідають їхнім потребам;
– простота вивчення. React має менший поріг входження для новачків. Він базується на JavaScript і використовує JSX, що дозволяє розробникам писати HTML-подібний код прямо в JavaScript файлі, що полегшує розуміння та модифікацію компонентів.
– висока продуктивність. React використовує віртуальний DOM, що дозволяє оптимізувати оновлення DOM-структури і зменшити витрати на ресурси. Це особливо корисно для створення великих додатків з багатою

динамічною взаємодією.

– широке співтовариство. У React широке та активне співтовариство, яке пропонує безліч інструментів, бібліотек і плагінів для розширення функціональності вашого додатку та спрощення розробки.

4 ПОРІВНЯННЯ ДОДАТКІВ НА ПРАКТИЦІ

Для порівняння додатків на практиці визначимо характеристики девайсу, на якому будуть тестуватися ці додатки. Це допоможе оцінити їх продуктивність і сумісність з конкретною апаратною конфігурацією. Основні характеристики, які було визначено для тестування, включають:

- версія Android – 14. Це найновіша версія операційної системи Android на момент створення цього пристрою. Вона включає останні функції та покращення в продуктивності та безпеці;
- архітектура процесора (CPU) – x86_64. Це 64-бітна архітектура, яка підтримує широкий спектр інструкцій і дозволяє ефективніше використовувати пам'ять порівняно з 32-бітною архітектурою;
- кількість ядер – 1. Тестування додатків на пристрої з обмеженими ресурсами допоможе впевнитися, що додатки працюватимуть навіть на менш потужних пристроях, які можуть використовувати кінцеві користувачі;
- оперативна пам'ять (RAM) - 3 GB. Цей обсяг оперативної пам'яті є мінімально достатнім для запуску сучасних додатків і операційної системи Android 14. Він дозволяє виконувати базові задачі, але може бути обмеженим для більш вимогливих додатків або багатозадачності.

4.1 Порівняння продуктивності

Продуктивність є вирішальним фактором при виборі між PWA, нативними та гібридними програмами. Це включає як швидкість, так і чуйність.

Спочатку розглянемо формули для вимірювання параметрів продуктивності. За допомогою формули 4.1 можна визначити час до першої взаємодії (Time to Interactive). Формула 4.2 дозволяє виміряти споживання пам’яті (Memory Usage) [24], а формула 4.3 — середній відсоток використання процесору (CPU Usage) [25].

$$\mathrm{TTI} = T_{\mathrm{interactive}} - T_{start\_load} \tag{4.1}$$

де TTI – час від початку завантаження до моменту, коли додаток стає повністю інтерактивним,

$T_{\mathrm{interactive}}$ – час до досягнення інтерактивного стану,

T_{start_load} – час початку завантаження.

$$M_{\mathrm{usage}} = M_{\mathrm{total}} - M_{free} \tag{4.2}$$

де M_{usage} – споживана пам’яті,

M_{total} – загальна доступна пам’ять,

M_{free} – вільна пам’ять.

$$\mathrm{CPU}_{\mathrm{usage}} = \frac{\mathrm{CPU}_{\mathrm{active}}}{\mathrm{CPU}_{\mathrm{total}}} \times 100\% \tag{4.3}$$

де $\mathrm{CPU}_{\mathrm{usage}}$ – відсоток використання процесора,

$\mathrm{CPU}_{\mathrm{active}}$ – час активної роботи процесора,

$\mathrm{CPU}_{\mathrm{total}}$ – загальний час роботи процесора.

Для тестування продуктивності PWA було обрано інструмент Lighthouse (див. рис. 4.1 та 4.2).

Lighthouse – це інструмент для автоматичного аудиту веб-сторінок, який допомагає покращити якість веб-додатків. Він розроблений Google і використовується для перевірки продуктивності, доступності, SEO та інших аспектів веб-сторінок.

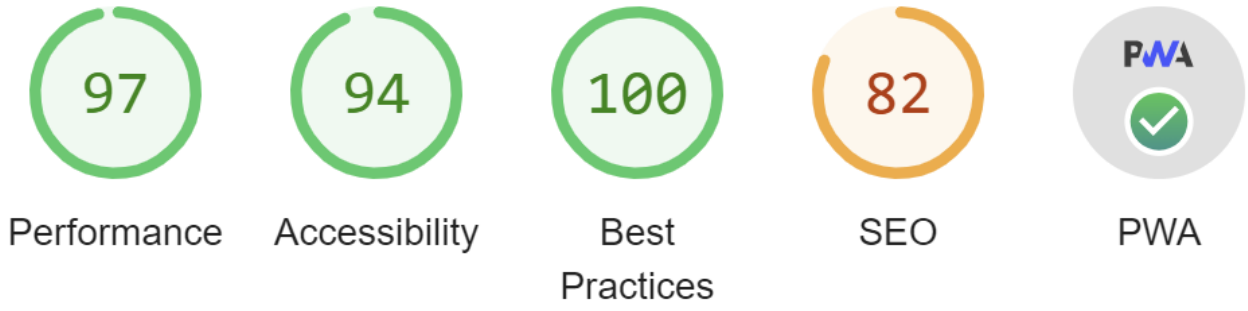

Рисунок 4.1 – Загальний результат тестування PWA за допомогою Lighthouse (створений самостійно)

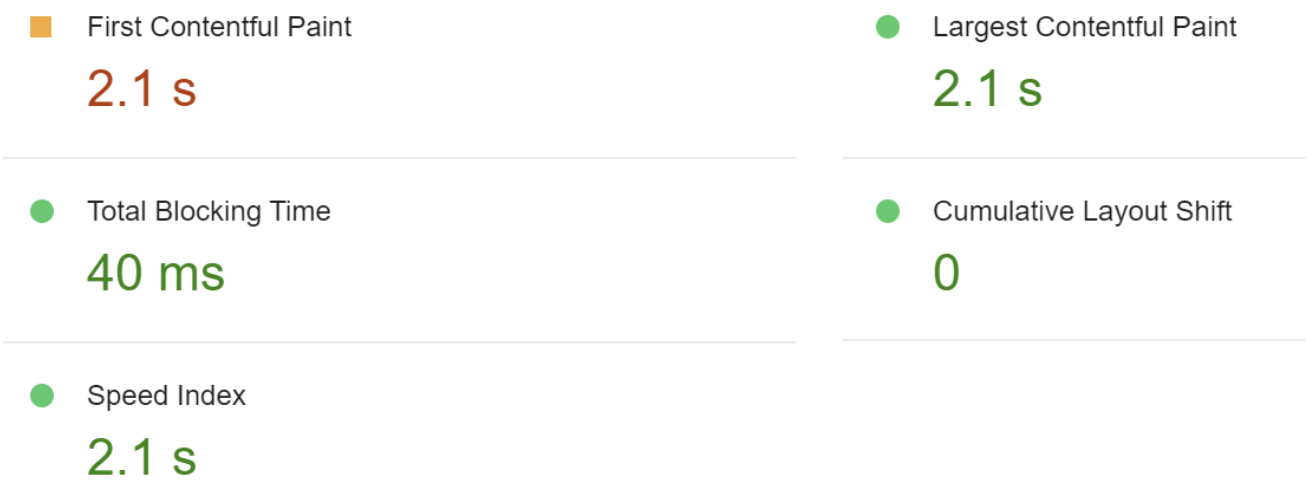

Рисунок 4.2 – Результат тестування продуктивності PWA за допомогою Lighthouse (створений самостійно)

Для тестування продуктивності для нативного та гібридного додатку було обрано інструмент Apptim. Apptim – це інструмент для тестування та оптимізації продуктивності мобільних додатків. Він допомагає розробникам і тестувальникам виявляти проблеми з продуктивністю, аналізувати їх і знаходити шляхи для покращення роботи додатків на різних платформах [26].

Для порівння продуктивності нативного, гібридного та прогресивного додатків створемо таблицю на основі даних отриманих від інструменту Apptim та Lighthouse (див. табл. 4.1).

Таблиця 4.1 – Порівняння продуктивності додатків (створена самостійно)

Параметр / Тип додатку	Розмір додатку	Середнє використання пам'яті	Час до першої взаємодії	Середній відсоток використання процесору
Нативний	6.5 MB	60 MB	2.5 s	36%
Гібридний	69.2 MB	115 MB	3.5 s	40%
Прогресивний	738 KB	7 MB	2.1 s	25%

PWA можуть запропонувати майже нативну продуктивність завдяки таким функціям, як сервіс-воркери для кешування активів і фонова синхронізація для мережевих запитів. Добре оптимізовані PWA завантажуються швидко навіть у повільних мережах. Вони також чуйні, оскільки покладаються на такі веб-технології, як JavaScript. Однак нативні програми все ще мають невелику перевагу, коли мова заходить про стабільно плавну анімацію та переходи.

Нативні додатки пропонують найкращу продуктивність, оскільки вони створені спеціально для цільової платформи. Вони мають прямий доступ до власних API і скомпільовані в машинний код, що дозволяє їм створювати анімацію зі швидкістю 60 кадрів в секунду та бездоганний UX. Однак продуктивність залежить від навичок розробника. Погано закодована нативна програма може бути повільною та незграбною.

Гібридні програми знаходяться десь посередині. Вони використовують веб-технології, але загорнуті в нативний контейнер, що дає їм доступ до деяких нативних API для підвищення продуктивності. Однак вони все ще не можуть повністю відповідати швидкості та плавності добре оптимізованих нативних програм, особливо для складних візуальних зображень та анімації. Продуктивність також може погіршуватися з часом у міру розвитку програми.

Підводячи підсумок, PWA можуть запропонувати чудову продуктивність, яка задовольняє потреби більшості користувачів, але нативні програми випереджають їх за стабільно бездоганну анімацію та переходи. Ефективність

гібридної програми залежить від складності, але, швидше за все, буде нижчою від добре закодованих нативних програм, особливо якщо програма з часом розростається.

4.2 Порівняння користувацького досвіду

Взаємодія з користувачем може суттєво відрізнятися між PWA, нативними та гібридними програмами.

PWA мають на меті надати досвід, подібний до нативних додатків, водночас використовуючи переваги веб-сайтів. Ключовою перевагою є можливість працювати між платформами, забезпечуючи узгоджений інтерфейс користувача на мобільному пристрої, настільному комп'ютері чи планшеті. PWA також завантажуються швидше, ніж веб-сайти, і можуть надавати такі функції, як push-повідомлення. Однак їм може бракувати деяких нативних можливостей і плавності порівняно з нативними програмами.

Нативні програми забезпечують найкращу взаємодію з користувачем, оскільки вони використовують специфічні для платформи компоненти інтерфейсу користувача та мають повний доступ до можливостей пристрою. Інтерфейс користувача швидкий, гнучкий і оптимізований для кожної платформи. Власні програми можуть повністю використовувати такі функції, як розширена анімація, жести, 3D-дотик тощо. Однак рідні програми потрібно створювати окремо для кожної платформи, що може зайняти більше роботи.

Гібридні програми використовують веб-технології, але загорнуті в нативний контейнер, який дозволяє отримати доступ до деяких можливостей пристрою, недоступних для веб-сайтів. Інтерфейс користувача відтворюється за допомогою веб-технологій, тому він може бути не таким гладким або швидким, як чистий нативний. Гібридні програми можуть працювати на різних платформах, але можуть мати обмеження щодо використання нативних функцій. Перевагою є швидша кросплатформна розробка, хоча потрібно враховувати компроміси UX.

Загалом нативні програми забезпечують найкращий і найбільш оптимізований UX. PWA націлені на UX, подібний до нативних додатків, але з

сильними веб нахилом, тоді як гібридні програми відносяться середньо. Вибір залежить від пріоритетів розвитку та цільових користувачів. Для споживчих додатків, які вимагають високоякісного власного UX, рекомендується нативна розробка.

4.3 Порівняння доступності

Доступність означає, наскільки легко люди з обмеженими можливостями можуть користуватися програмою. Це важливий момент для будь-якого підходу до розробки додатків.

PWA добре підтримують спеціальні можливості в сучасних браузерах. Вони використовують такі веб-стандарти, як WAI-ARIA, для доступних компонентів. Однак PWA можуть мати обмежений доступ на деяких старих платформах.

Нативні додатки мають найкращу підтримку доступності на цільовій платформі. iOS і Android забезпечують надійні API спеціальних можливостей для підтримки програм зчитування з екрана, синтезу мовлення з тексту, субтитрів тощо. Однак нативна програма націлена лише на одну платформу.

Гібридні додатки також мають хорошу підтримку доступності на мобільних пристроях, якщо створені належним чином за веб-стандартами. Вбудований веб-вміст може взаємодіяти з рідними API для забезпечення доступності. Але підтримка буде відрізнятися залежно від впровадження перевірки.

Загалом, рідні програми, як правило, забезпечують найнадійнішу підтримку доступності, за ними йдуть гібридні програми та PWA. Однак PWA мають перевагу веб-кодової бази, яка може бути доступна на кількох платформах. Тож явного переможця немає — це залежить від цільових платформ і потреб користувачів. Розробники повинні прагнути до високої доступності незалежно від підходу.

4.4 Порівняння використання в автономному режимі

Офлайн-можливості є важливим моментом, коли вибираєте між PWA, нативними та гібридні додатками.

PWA пропонують надійну офлайн-підтримку через сервіс-воркери та API

кешу. Коли користувач відвідує PWA, він кешується локально та завантажується як рідна програма в автономному режимі. PWA також можуть використовувати фонову синхронізацію для надсилання даних і отримання push-повідомлень, коли з’єднання буде відновлено.

Нативні додатки мають повний доступ до функцій пристрою та можуть створювати робочі процеси в автономному режимі, використовуючи локальне сховище. Дані зберігаються, і програма продовжує працювати без підключення до Інтернету.

Гібридні додатки також можуть використовувати кешування браузера та локальне сховище для підтримки в автономному режимі. Однак контейнер webview може обмежувати доступ до деяких нативних функцій, які повністю нативні програми можуть використовувати в автономному режимі. Продуктивність може постраждати порівняно з справді нативним.

Загалом PWA та нативні додатки загалом забезпечують кращі офлайнові можливості, ніж гібридні програми. PWA відповідають або перевершують гібридні програми для випадків використання в автономному режимі, тоді як нативні залишаються оптимальними для повного контролю. Вибір залежить від зважування офлайн-потреб з іншими факторами, такі як вартість розробки та час виходу на ринок.

4.5 Порівняння складності підтримки

Складність в обслуговуванні значно відрізняються між PWA, рідними та гібридними програмами.

PWA потребують дуже мало обслуговування, оскільки кодова база додатку знаходиться на сервері. Поки сервер підтримується, PWA оновлюватиметься автоматично для всіх користувачів, коли вони завантажуватимуть програму. Може знадобитися додаткове технічне обслуговування для Service Worker і кешу, але загальні потреби в техобслуговуванні мінімальні.

Нативні додатки вимагають випуску нових версій програми у відповідних магазинах програм, коли потрібно розгортати нові функції чи виправлення. Це

робить підтримку нативної програми складнішою, ніж PWA. Автоматизоване тестування та конвеєри CI/CD допомагають оптимізувати деякі з цих процесів.

Гібридні додатки вимагають розгортання магазину програм, як і нативні програми. Але вони також вимагають підтримки веб-кодової бази, яка відображається у власній обгортці. Це робить технічне обслуговування більш складним, ніж нативне або PWA. Будь-які зміни в інтерфейсі користувача, бізнес-логікі або даних потребують переупакування та повторного розгортання гібридної програми. Розробники також повинні забезпечити синхронізацію веб і рідних шарів шляхом постійного обслуговування.

Отже, підводячи підсумок, PWA мають найменші накладні витрати на технічне обслуговування, за ними йдуть власні додатки. Гібридні додатки вимагають найскладніших процесів обслуговування.

4.6 Порівняння безпечності

Нативні додатки написані спеціально для платформи, для якої вони створені, що дозволяє отримати доступ до вбудованих функцій безпеки, які надаються ОС і апаратним забезпеченням. Наприклад, додатки для iOS можуть скористатися функцією Apple App Transport Security, яка забезпечує безпечне підключення. Додатки Android можуть інтегруватися з системою Keystore для безпечного зберігання криптографічних ключів.

Гібридні додатки також мають доступ до власних функцій безпеки, оскільки вони, по суті, є нативними програмами, які запускають вбудовані веб-перегляди. Власний контейнер обробляє дозволи, сертифікати та шифрування, а веб-код отримує доступ до API через міст.

З іншого боку, PWA покладаються на модель безпеки веб-переглядача, який ізолює веб-ресурси, щоб запобігти доступу за його межами. Без вбудованої інтеграції PWA не можуть отримати доступ до нативного шифрування, сховищ ключів, довіреного обладнання тощо.

Однак PWA можуть застосовувати інші заходи, такі як HTTPS, CORS, політика безпеки вмісту та автентифікація за допомогою протоколів, таких як

OAuth, для посилення безпеки. При розміщенні на HTTPS PWA можуть використовувати TLS для безпечного зв’язку. Правильне впровадження є ключовим для того, щоб PWA забезпечували належну безпеку.

4.7 Підсумкове порівняння

На основі проведеного дослідження, наведемо порівняльний аналіз методів розробки нативних додатків, прогресивних веб додатків (PWA) та гібридних додатків (див. табл. 4.2).

Таблиця 4.2 – Порівняння додатків (створена самостійно)

Показник \ Тип додатку	Нативний	Гібридний	Прогресивний
Вартість розробки	Висока	Середня	Низька
Час розробки	Довгий	Середній	Швидкий
Продуктивність	Найвища	Середня	Залежить від браузера
Користувацький досвід	Найвищий	Середній	Низький
Доступності	Найвища	Середня	Середня
Доступ до апаратних ресурсів	Повний	Повний	Обмежений
Автономна робота	Повна	Часткова	Часткова
Можливість підтримки	Через магазини додатків	Через магазини додатків	Автоматичне оновлення
Безпека	Висока	Висока	Середня

Отже, створене порівняння додатків допоможе розробникам краще розуміти сильні та слабкі сторони технологій для розробки додатків. Це наддасть можливість здійснити необхідні покращення та вдосконалення, щоб збільшити конкурентоспроможність та задоволеність користувачів.

4.8 Практична цінність отриманих результатів

Отримані результати цього дослідження мають значну практичну цінність для різних сфер розробки та використання веб і мобільних додатків.

По-перше, результати дослідження можуть бути використані розробниками програмного забезпечення та IT-компаніями для прийняття більш обґрунтованих рішень щодо вибору технологій для розробки додатків. Моделі, що продемонстрували високу ефективність у створенні прогресивних та гібридних веб-додатків, дозволяють створювати продукти, які поєднують переваги як веб, так і нативних додатків. Це забезпечує кращий користувацький досвід та підвищує задоволеність клієнтів.

По-друге, результати дослідження можуть бути корисними для компаній, що займаються цифровою трансформацією. Впровадження прогресивних та гібридних веб-додатків дозволяє знизити витрати на розробку та підтримку додатків, оскільки одна платформа може використовуватись на різних пристроях і операційних системах. Це сприяє ефективнішому використанню ресурсів та зменшенню часу виходу на ринок.

Крім того, отримані результати можуть бути корисними для навчальних закладів та освітніх програм у сфері IT. Викладачі можуть використовувати ці дані для створення навчальних матеріалів та програм, що відображають сучасні тенденції в розробці програмного забезпечення. Це допоможе підготувати студентів до актуальних викликів та вимог ринку праці.

Також результати дослідження можуть сприяти розвитку нових методів інтеграції прогресивних та гібридних додатків у бізнес-процеси. Компанії можуть використовувати ці технології для покращення взаємодії з клієнтами, оптимізації внутрішніх процесів та підвищення загальної ефективності роботи.

Нарешті, отримані результати можуть бути використані для подальших наукових досліджень у галузі розробки програмного забезпечення. Вони можуть слугувати основою для вдосконалення існуючих методів та розробки нових підходів до створення веб і мобільних додатків, сприяючи розвитку науки та технологій у цій сфері.

ВИСНОВКИ

Дослідження методологій розробки додатків, включаючи порівняння нативного, прогресивного та гібридного підходів, показує, що кожна з цих методологій має свої переваги та недоліки.

Нативна розробка, забезпечує високий рівень продуктивності та оптимізації завдяки тому, що вона орієнтована на конкретну платформу. Використовуючи специфічні мови програмування та API, розробники можуть максимально розширити функціональність пристрою. Однак такий підхід вимагає багато зусиль і витрат, а також обмежує швидкість випуску оновлень.

Прогресивні та гібридні додатки, з іншого боку, намагаються знайти компроміс між ефективністю та універсальністю. Вони можуть використовувати спільну кодову базу на різних платформах, що зменшує зусилля та витрати. Однак це може вплинути на продуктивність і взаємодію з функціоналом пристрою.

Вибір підходу до розробки повинен бути обґрунтований відповідно до конкретних потреб та обставин проекту. Нативні підходи підходять для проектів, які вимагають високої ефективності та повної функціональності, тоді як прогресивні або гібридні рішення можуть бути кращими для більш універсальних та економічно ефективних проектів.

Загалом, вивчення та розуміння цих підходів допоможе розробникам обрати стратегію, яка найкраще відповідає конкретним вимогам та умовам проекту, забезпечуючи оптимальний баланс між ефективністю та універсальністю.

Також подальші дослідження можуть бути спрямовані на оптимізацію цих підходів для підвищення їх ефективності та зменшення часу розробки і впровадження. Можливим напрямком є розробка нових гібридних архітектур, які поєднують найкращі властивості прогресивних веб-додатків та нативних додатків. Впровадження таких рішень у різні галузі може значно покращити якість та швидкість створення програмного забезпечення, забезпечуючи високу продуктивність і зручність для кінцевих користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Авербах Д. Дослідження методів розробки прогресивних та гібридних веб-додатків порівняно з нативними додатками. URL: https://archive.liga.science/index.php/conference-proceedings/issue/view/ukr-17.05.2024/80 (дата звернення: 20.05.2024).

2. Exploring Mobile App Development for Social Causes. URL: https://moldstud.com/articles/p-exploring-mobile-app-development-for-social-causes (дата звернення: 05.04.2024).

3. The Impact of Mobile Technology In Our Lives. URL: https://blog.mobiversal.com /the-impact-of-mobile-technology-in-our-daily-life.html (дата звернення: 10.04.2024).

4. Кобзєв В., Зеленська Ю. Дослідження архітектурних рішень користувацького інтерфейсу для ефективного використання мобільних додатків. 27-й Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті», м. Харків. 2023. С. 125–126.

5. Android vs. Apple Market Share: Leading Mobile Operating Systems. URL: https://www.bankmycell.com/blog/android-vs-apple-market-share (дата звернення: 07.04.2024).

6. iOS version history. URL: https://en.wikipedia.org/wiki/IOS_version_history (дата звернення: 25.04.2024).

7. Objective-C and Swift: Their History and Where iOS Development is Going. URL: https://hangzone.com/objective-c-swift-history-apple-development-going (дата звернення: 15.05.2024).

8. Flutter. URL: https://flutter.dev/docs (дата звернення: 11.05.2024).

9. React Native. URL: https://reactnative.dev (дата звернення: 03.05.2024).

10. Flutter Vs. React Native: Let’s See Who the Winner. URL: https://www.mindinventory.com/blog/flutter-vs-react-native (дата звернення: 17.04.2024).

11. Stack Overflow Survey 2023 Revealed. URL: https://www.walturn.com/insights/stack-overflow-survey-2023-revealed (дата

звернення: 19.04.2024).

12. Use Hermes Engine. URL: https://docs.expo.dev/guides/using-hermes (дата звернення: 18.05.2024).

13. Progressive Web Apps. URL: https://web.dev/progressive-web-apps (дата звернення: 07.04.2024).

14. Груздо І., Россоха С. Современные проблемы кроссплатформенности в технологиях веб-разработки для мобильных приложений. Інформаційні та моделюючі технології : Всеукр. наук.-практ. конф., м. Черкаси, 29–31 трав. 2014 р. 2014. С. 10.

15. A Tinder Progressive Web App Performance Case Study. URL: https://medium.com/@addyosmani/a-tinder-progressive-web-app-performance-case-study-78919d98ece0 (дата звернення: 13.05.2024).

16. New Forbes PWA Doubles Engagement. URL: https://doccly.com/articles/new-forbes-pwa-redefines-site-and-doubles-engagement (дата звернення: 13.05.2024).

17. Uber PWA: The Perfect Case Study For Progressive Web App. URL: https://www.tigren.com/blog/uber-pwa (дата звернення: 13.05.2024).

18. Service worker concepts and usage. URL: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API (дата звернення: 17.05.2024).

19. The Web App Manifest. URL: https://web.dev/articles/add-manifest (дата звернення: 17.05.2024).

20. Kotlin vs Java – Which One Should I Choose For Android Development. URL: https://www.geeksforgeeks.org/kotlin-vs-java/ (дата звернення: 09.04.2024).

21. Kotlin Clean Architecture. URL: https://proandroiddev.com/kotlin-clean-architecture-1ad42fcd97fa (дата звернення: 14.04.2024).

22. Афанасьєва І., Корецький О. Використання реактивного програмування у розробці мобільних застосунків. SCIENCE, RESEARCH, DEVELOPMENT. Technics and technology.#4 (НАУКА, ИССЛЕДОВАНИЯ, РАЗВИТИЕ. Техника и технология. #4), 29–30 квіт. 2018 р. С. 79–83.

23. The Evolution of React Native: Exploring the Impact of Re-Architecture. URL:

https://litslink.com/blog/new-react-native-architecture (дата звернення: 03.05.2024).

24. Performance of Computer in Computer Organization. URL: https://www.geeksforgeeks.org/computer-organization-performance-of-computer/ (дата звернення: 19.05.2024).

25. CPU Scheduling Criteria. URL: https://www.geeksforgeeks.org/cpu-scheduling-criteria/ (дата звернення: 09.05.2024).

26. Mobile Apps Performance Testing using Apptim. URL: https://medium.com/@rakeshkarkare/mobile-apps-performance-testing-using-apptim-c78ed2bc97b8 (дата звернення: 11.05.2024).