Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Метод проектування мобільних додатків

(тема)

Виконав:
студент II курсу, групи СПм-19-1
Кобзар М.С.
(прізвище, ініціали)

Спеціальність
123 – Комп'ютерна інженерія
(код і повна назва спеціальності)

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

Освітня програма
Системне програмування
(повна назва освітньої програми)

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

Допускається до захисту

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

2020 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

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

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

Освітня програма Системне програмування
(повна назва)

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

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові Кобзар Марії Станіславівні
(прізвище, ім'я, по батькові)

1. Тема роботи Метод проектування мобільних додатків

затверджена наказом по університету від " 2 " листопада 2020 р. № 1486 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 14 грудня 2020 р.

3. Вхідні дані до роботи

1) Документація React Native
2) Документація Flutter
3) Документація Apache Cordova
4) Документація Ionic
5) Документація Xamarin
6) Документація PWA

4. Перелік питань, що потрібно опрацювати в роботі

1) Аналіз предметної області
2) Аналіз методів розробки нативних додатків
3) Аналіз методів розробки гібридних додатків
4) Аналіз методів розробки веб-додатків
5) Аналіз методу розробки гібридного додатку з нативними модулями
6) Вибір та обґрунтування методики розробки мобільних додатків
7) Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Слайди презентації - 10 шт

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

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання у керівника	02.11.20	
2	Аналіз проблеми та предметної області	03.11.20-10.11.20	
3	Аналіз методів розробки нативних додатків	11.11.20-13.11.20	
4	Аналіз методів розробки гібридних додатків	13.11.20-15.11.20	
5	Аналіз методів розробки веб-додатків	16.11.20-24.12.20	
6			
7	Оформлення матеріалів атестаційної роботи	24.11.20-29.12.20	
8	Подання атестаційної роботи керівникові та	30.11.20-01.12.20	
	її попередній захист		
9	Подання атестаційної роботи на рецензування	02.12.20-05.12.20	

Дата видачі завдання 2 листопада 2020 р.

Студент _____
(підпис)

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

РЕФЕРАТ

Пояснювальна записка атестаційної роботи: 66 с., 14 рис., 2 табл., 1 дод., 14 джерел.

НАТИВНІ ДОДАТКИ, ANDROID, iOS, ГІБРИДНІ ДОДАТКИ, PWA, API, ВЕБ-ДОДАТКИ, SDK.

Метою атестаційної роботи є аналіз сучасних методів розробки мобільних додатків.

У ході виконання атестаційної роботи були розглянуті принципи і методи побудови мобільних додатків та проведений аналіз існуючих рішень який дозволить враховувати особливості кожного методу та обрати технологію що відповідає вимогам бізнесу.

ABSTRACT

Master’s thesis: 66 pages, 14 figures, 2 tables, 1 appendices, 14 sources.

NATIVE APPLICATIONS, ANDROID, iOS, HYBRID APPLICATIONS, PWA, API, WEB APPLICATIONS, SDK.

The purpose of certification work is the analysis of modern methods of mobile application development.

During the attestation work the principles and methods of building mobile applications were considered and the analysis of existing solutions which will allow to consider features of each method and to choose the technology meeting the requirements of business was carried out.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – інтерфейс прикладного програмування (англ., Application Programming Interface)

OTT – метод надання відеопослуг через Інтернет (англ., Over the Top)

PWA – прогресивний веб-додаток (англ., Progressive Web Application)

SDK – набір засобів розробки (англ., Software Development Kit)

ОС – операційна система

ВСТУП

Популярність використання мобільних пристроїв у всьому світі продовжує зростати. Сьогодні користувачі витрачають більше часу на свої смартфони в різних цілях (соціальні мережі, електронна пошта, карти, новини, відео, комерційні додатки та інше). У сучасному світі важко уявити собі мобільний пристрій, на якому б не стояло жодного додатка.

Тож розробка мобільних додатків нині є однією з найпопулярніших завдань в сфері інформаційних технологій. Зростає потреба в розробці якісного і відповідного сучасним тенденціям продукту.

Успіх і відповідно і якість мобільного додатку залежить від вибору платформи та технологій для його розробки. Саме вони зумовлюють життєздатність і конкурентоспроможність додатку, його функціонал, масштабованість і складність обслуговування. Також це впливає і на вартість розробки.

Сьогодні існують різні методи і фреймворки, які допомагають створювати мобільні додатки. Таким чином, можливості сучасних технологій розробки дозволяють створювати мобільні додатки різної складності.

Тож вибір методу розробки мобільного додатка − це дуже важливий етап в його розробці, на який впливають кілька факторів, таких як технічна оцінка розробників, потреба в доступі до інформації на пристрої, вплив швидкості інтернету на додаток. Вибір тієї чи іншої платформи також залежить від бюджету і вимог, що пред'являються до майбутнього додатку.

Метою роботи є аналіз сучасних методів розробки мобільних додатків, та розробка методу вибору технології для їх створення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність мобільних пристроїв

Згідно результатів досліджень глобальної аналітичної компанія Counterpoint Research – користувачі проводять більше часу на смартфонах, ніж на будь-якому іншому пристрої.

Щодня половина користувачів смартфонами проводить за ними понад 5 годин, а 26% – більше 7 годин на день. Власник смартфона перевіряє свій мобільний телефон одразу після пробудження зранку та безпосередньо перед відходом до сну [1].

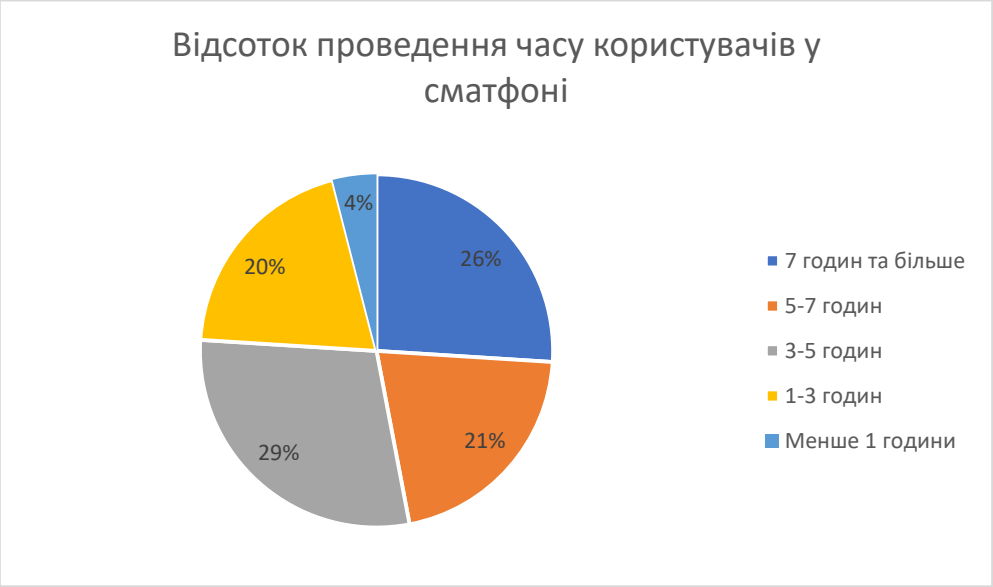


Рисунок 1.1 – Статистика проведення часу користувачів у смартфоні згідно даним Counterpoint Research

Стрімке зростання мобільних додатків пояснюється тим, що швидкісний мобільний інтернет стає доступнішим, як і недорогі смартфони.

За даними звітів Global Digital 2019, 68% жителів землі – власники мобільних телефонів, а 53% інтернет-трафіку припадає на мобільні телефони (рисунок 1.2).

Рисунок 1.2 – Статистика генерування трафіку згідно данним звітів Global Digital 2019

Згідно даним звітів Global Digital 2019, підготованих аналітичним агентством We Are Sotial та SMM-платформи Hootsuite відсоток генерування трафіку з мобільних пристроїв вже перевищив відсоток генерування з персональних комп'ютерів [2].

Статистика порталу Statista.com надає інформацію про глобальний трафік мобільних даних з 2009 по 2018 рік. У 2018 році глобальний рух мобільних даних становив 19,01 екзабайт на місяць. У 2022 році очікується, що рух мобільних даних у всьому світі досягне 77,5 екзабайт на місяць при загальному темпі зростання в 46% (рисунок 1.3).

Рисунок 1.3 – Процент відкриття сайтів згідно даним порталу Statista.com

1.2 Аналіз мобільних операційних систем

За весь час існування смартфонів з'явилася чимала кількість мобільних операційних систем, таких як Samsung Bada, Symbian, Firefox OS і Windows Mobile/Phone. Але жодна з них не змогла скласти гідну конкуренцію Android і iOS від компаній Google і Apple відповідно. На сьогодні картина поширеності ОС в світі згідно ірландському сервісу StatCounter займається аналізом трафіку в інтернеті та має вигляд як зображено на рисунку 1.4.

На операційній системі Android щорічно з'являється все більше і більше нових виробників дешевої мобільної техніки, що і пояснює її популярність, бо чим багатша країна, тим вище відсоток поширеності в ній iOS (що пов'язано, з більш високою вартістю техніки Apple). Так, в США і Сполученому Королівстві пристроїв на iOS належить 46% ринку, в Норвегії - 44%, в Бельгії - 38%.

Тож при розробці мобільного додатку слід орієнтується на підтримку лідерів мобільних ОС – Android та iOS.

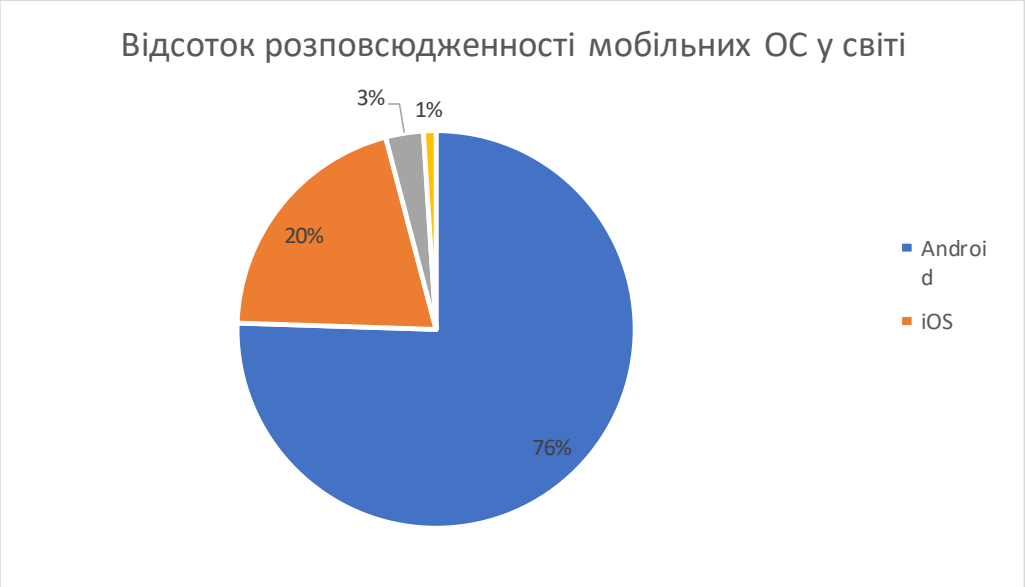


Рисунок 1.4 – Відсоток розповсюдженості мобільних ОС у світі згідно даним звітів сервісу StatCounter

1.3 Огляд існуючих рішень

На даний момент можна виділити наступні типи додатків для мобільних пристроїв:

- нативні додатки;
- гібридні додатки;
- веб-додатки.

Нативні додатки − це додатки, написані рідною (з англ. native - рідна) для певної платформи мовою програмування. Для Android цією мовою є Java або Kotlin тоді як для iOS − Objective-C або Swift. Нативні додатки мають переваги у взаємодії та взаємодії з нативними мобільними компонентами як камера, мікрофон, акселерометр, геолокація, адресна книга, плеєр і т.д. Крім того, користувачі можуть використовувати деякі програми без підключення до інтернету за допомогою вбудованої бази даних, передбаченої фреймворком [18].

В силу того, що нативні додатки оптимізовані під конкретну ОС, вони

органічно вписуються в будь-який смартфон, відрізняючись високою швидкістю роботи і продуктивністю.

Мобільні веб-додатки − це мобільна версія сайту, тільки з розширеним інтерактивом. Веб-додатки розроблюються за допомогою HTML5, CSS та JavaScript. Таким додаткам необхідний доступ до інтернету для належної поведінки та користувацького досвіду цієї групи програм. Ці програми можуть займати мінімальний простір пам’яті на пристрої користувача порівняно із нативними та гібридними програмами. Оскільки всі бази персональних даних зберігаються на інтернет-серверах, користувач може отримати потрібні дані з будь-якого пристрою через інтернет. По суті звичайний веб-сайт чи односторінковий додаток, який можна переглянути на маленьких екранах за допомогою гнучкого веб-дизайну. Таке рішення не прив'язане до конкретної платформы, що є його перевагою, але такий додаток не може використовувати програмне забезпечення смартфона, та у такого додатка відсутня можливість завантаження з магазинів додатків [6].

До веб-додатків відносять і PWA або прогресивні веб-додатки. Прогресивні веб-додатки − це веб-програми, які завантажуються як звичайні веб-сторінки або веб-сайти, але можуть запропонувати користувачеві такі функції, як робота в автономному режимі та доступ до апаратного забезпечення пристрою, традиційно доступний лише для мобільних додатків. Такі додакти встановлюються з сайту, а з лютого 2019 року PWA можна додавати в App Store і Google Play, даючи користувачеві можливість завантажити додаток зі звичного джерела.

Концепція гібридної програми − це поєднання нативних та веб додатків. До цієї категорії належать програми, розроблені за допомогою Apache Cordova, Xamarin, React Native, Flatter, Ionic та інших подібних технологій.

В першу чергу вони створені для підтримки додатків на багатьох платформах. Таким чином ці програми легше і швидше розробляти, так як це передбачає використання однієї кодової бази, яка працює в декількох

мобільних операційних системах, і вже немає необхідності створювати декілька нативних додатків наприклад для Android та iOS.

Та незважаючи на такі переваги, гібридні програми демонструють нижчу продуктивність. Часто програми не мають однакового зовнішнього вигляду в різних мобільних операційних системах. Звісно це все ще залежить від обраної технології розробки.

Гібридні додатки у багатьох джерах розподілять на гібридні та кросплатформні, основна різниця полягає у тому що гібридні додатки передбачають наявність вже готового HTML шаблону, до якого додається контейнер або інтерфейс, що дозволяє встановлювати та відкривати його як мобільний додаток. Так до гібридних технологій наприклад відносять PhoneGap, Cordova та Ionic. Але ці технології досить швидко дозвиваються та оновлються, якщо раніше у таких додатків не було доступу до API пристрою, то тепер це є можливо за допомогою сторонніх опенсорсних бібліотек. Тож такий розподіл на сьогодні не має необхыдності і тепер такі додатки розглядають у одній группі – гібридні або кросплатформні мобільні додатки.

1.4 Фактори для вибору методу розробки

Вибір виграшної основи серед усіх підходів до розробки додатків є складним завданням. Залежно від потреб вашого проекту, кількості розробників та їх навичок, бюджету та часових рамок, ви можете зробити вибір на користь будь-якої з трьох стратегій.

Визначитися з технологій можна ще на першому етапі створення мобільного застосування, а саме визначення, для чого потрібно додаток і які завдання воно виконуватиме. Наприклад якщо це додаток має на увазі роботу з безліччю інструментів, що входять в SDK тієї чи іншої платформи. Або передбачається їх гнучке налаштування то в такому випадку не обійтися без нативного підходу, саме у цьому підході є величезна різноманітність інструментів, що входять в SDK. Тобто все що потрібно − використовувати

ці інструменти в своєму нативному додатку.

У випадку з гібридом – на той чи інший нативний інструмент не завжди існує адаптація на базі фреймворка, обраного для гібридної розробки. Якщо такого інструменту немає − доведеться або чекати його появи, або розглядати альтернативні фреймворки.

Однак розробка власних додатків вимагає великих бюджетів, і ось чому. Нативний код, який використовується для розробки на одній платформі, не може бути використаний повторно для іншої. У той же час для власних додатків потрібні різні набори навичок для кожної платформи. Таким чином, якщо ви хочете орієнтуватися на користувачів двох або більше платформ, вам доведеться збільшити розмір своєї команди та знайти відповідних спеціалістів. У свою чергу, це призводить до збільшення витрат, які потрібно інвестувати в розробку, а отже, і в обслуговування програми. Більше того, при додаванні нових функцій до програми їх реалізація та функціональність можуть відрізнятися − деякі речі, легко досягнуті на iOS, можуть зайняти багато часу на Android, і навпаки. Час виходу на ринок може також відрізнятися.

Розробка нативних додатків − це завжди ідеальний вибір для великих підприємств, які вимагають чудової продуктивності та високої доступності своїх програм. Вони не турбуються щодо бюджетів, довгострокового планування та доступності розробників. Однак, якщо ви обмежені бюджетом, ви все одно можете розглянути нативну розробку додатків, обравши основну платформу для своїх користувачів.

Кроссплатформенний підхід − найкращий вибір, щоб один раз розробити програму та запускати її де завгодно. Такий підхід дозволяє заощадити час, скоротити витрати на розробку та обслуговування та отримати програму з майже ідеальною продуктивністю та чудовою масштабованістю. Кросплатформна розробка забезпечує пришвидшений випуск на ринок, оскільки ви можете повторно використовувати приблизно 75% коду [4].

Створення веб-програм − це дуже популярний та дешевий спосіб створення програм на основі HTML. Все, що вам потрібно − це налаштувати свій веб-сайт, щоб отримати мобільний вигляд та відчуття. Це швидкий спосіб створити мобільний додаток якщо у бізнесу вже є сайт, а якщо існує необхідність у встановленні додатку або доступу до функцій мобільного пристрою треба лише додати маніфест на сервіс-воркер та реалізувати прогресивний веб-додаток, також можна скористатися готовим рішенням та загорнути веб-додаток в контейнер гібридного фреймворку PhoneGap, Cordova або Ionic.

Тож веб-додатки це гарний вібір якщо немає необхідності у великій кількості функціоналу, продуктивніть таких додатків не досить висока, та звісно залежить від складності додатка. Це гарний вибір для e-Commerce та для бізнесу у якого немає необхідності розташовувати додаток у магазині додатків [19].

Таблиця 1.1 – Зведена таблиця порівнянння типів мобільних додатків

	Нативні додатки	Гібридні додатки	Веб-додатки
Вартість розробки	Висока	Середня	Низька
Продуктивність	Висока	Висока, наближена до нативного	Залежить від складності додатка
Швидкість розробки	Низька	Середня	Низька
Вартість підтримки	Висока, необхідність підтримувати декілька додатків	Середня	Низька
Канал розподілу	Стори	Стори	Стори та браузер

Продовження таблиці 1.1

Інтерфейс	Нативний	Нативний	Не нативний
Повторне використання коду	Немає	70 – 90%	100%
Доступ до інструментів девайсу	Повний	Повний / Наближений до повного	Часковий
Підтримка мультиплатформ	Немає	Присутня	Присутня

Для того, щоб полегшити процес вибору методології розробки мобільних додатків, можна звернутися то зведеної таблиці порівняння нативних, гібридних та веб-додатків.

Та така таблиця є досить узагальнена, у наступних розділах буде розглянута окремо кожна методологія та технології що їх реалізують.

2 НАТИВНІ ДОДАТКИ

Нативні додатки (від англ. native - рідна) розробляються під конкретну апаратно-програмну платформу і пишуться на мовах, створених для даної платформи.

І iOS, і Android мають свої SDK (від англ. Software Development Kit − набір засобів розробки) і свій стек технологій, зав'язані на певну мову програмування. Такі додатки оптимізовані під конкретні операційні системи, тому вони можуть працювати коректно і швидко.

Таблиця 2.1 – Зведена таблиця порівняння операційних систем

	Android	iOS
Programming Language	Java, C/C++, Kotlin	Objective-C, Swift
IDE	Android Studio, Eclipse	Xcode
SDK	Android SDK/NDK	iOS SDK

Розробка нативних додатків має свої особливості. Серед плюсів розробки нативних додатків можна виділити:

- висока продуктивність;
- максимальне використання можливостей платформи;
- кращий користувацький інтерфейс;
- краще позиціонування в магазинах додатків.

Оскільки технології, використовувані при розробці платформо залежних додатків, прямо пов'язані з цією платформою, власний нативний код має прямий доступ до всіх функцій операційної системи.

Це, простіше взаємодія програми з власними функціями мобільних пристроїв, підвищує загальну продуктивність програми, особливо при поданні графічного або мультимедійного контенту. Отже, створення

навантажених додатків з використанням нативного коду може знизити час відгуку, ймовірність збоїв і зависань. забезпечення безперебійної роботи ігор.

Нативні додатки розробляються, щоб вирішувати конкретні завдання на конкретній платформі. Це призводить до кращого відповідності можливостей додатків апаратних можливостей пристроїв, включаючи Bluetooth, NFC, камеру, GPS і т.д. Ця відповідність необхідна, коли програма має використовувати такі дані, як фізичне і географічне розташування та ін.

Оскільки нативні додатки безпосередньо інтегруються з мобільною операційною системою, сприймаючи і використовуючи всі доступні можливості пристрою, користувачі можуть рухатися по звичному інтерфейсу без особливого клопоту, що призводить до позитивного для користувача досвіду (UX) і стабільному повторного використання. Наприклад зараз, при великій кількості різноманітних варіантів дозволів екранів смартфонів дуже важливо мати додаток, оптимізоване під такий екран, щоб користувачеві було зручно цим додатком користуватися.

Якість користувацького досвіду є важливим рейтинговим показником в магазинах додатків. Якщо додаток має високу оцінку користувацького досвіду, воно буде більш високо оцінений магазином додатків, що веде до більшого числа рекомендацій для різної аудиторії і збільшення доходів від додатка, відповідно. Є припущення, що в магазинах додатків самі механізми ранжирування будуть краще представляти саме нативні додатки для платформи, через їх свідомо більш високу продуктивність і простоту використання.

Серед мінусів можна виділити:

- дороговизна і витрати часу на розробку;
- несумісність з іншою мобільною операційною системою;
- втрачені можливості застосування з іншими ОС.

Без сумніву, створення окремих додатків відразу під кожну з декількох операційних систем може значно продовжити процес розробки. Один і той же програмний код не може бути розгорнутий на різних платформах, і

програмістам буде потрібно більше часу для перетворення і перезапису коду, що збільшує витрати і час розробки. Якщо компанія хоче для кожної з платформ створювати окремі додатки, вона може виявитися змушена найняти додаткових програмістів-фахівців. Наприклад, один розробник буде зосереджений на розробці додатків для iOS, а інший − на розробці додатків для Android, що ще більше збільшує витрати.

Вам доведеться заздалегідь погодитися з несумісністю вашого застосування з іншими ОС. Коли розробляється додаток під конкретну ОС, його розробники використовують мову, специфічний тільки для цієї операційної системи: наприклад, Objective-C або Swift - для iOS, для різних мобільних пристроїв на базі Android - Kotlin і Java. У цьому контексті нативний додаток, який спочатку був написаний для iOS − не буде сумісний зі специфікацією на базі Android і навпаки.

Розробка додатків, орієнтованих тільки на одну платформу, може привести до втрачених можливостей. Особливо якщо інші платформи заздалегідь не беруться до уваги. Свідоме скорочення цільового ринку може привести до втрати доходу.

2.1 Android

Операційна система Android була випущена компанією Google 23 вересня 2008 року. 11 липня 2005 року корпорація купила стартап Android Inc, що займається цією розробкою, і перетворила цей напрям в одне з ключових. З тих пір Android швидко розвивається і тепер встановлений на більш ніж 83% всіх мобільних пристроїв в світі.

Android − це велика фрагментація пристроїв. Це плюс для користувачів адже є можливість вибрати телефон на будь-який смак і під будь-які технічні вимоги. Але дуже непросто для розробників, і це стосується як апаратної, так і програмної частини.

Апаратно пристрій може мати фронтальну камеру, а може і ні. Сімкарт

може бути будь-яка кількість. Фізичні кнопки можуть бути присутніми чи ні. Екрану може бути два: додатковий з тильного боку або на чохлі.

Наприклад, якщо вам потрібно розмістити зображення на весь екран iOS пристрою, ви використовуєте кілька зображень під типові розміри iPhone 6 і вище, iPhone 6 Plus і вище, iPhone X і iPhone X Max. У випадку ж з Android екрани мають і різний дозвіл, і різне співвідношення сторін, і різну щільність.

Також необхідно враховувати особливості відображення інтерфейсу на різних версіях ОС і різних оболонках. Кількість актуальних версій Android, які перебувають у використанні, багато. Остання версія, Pie (Android 9) встановлена тільки на 10% пристроїв. Версії 5.x (Lollipop) і молодше займають більше 20% ринку.

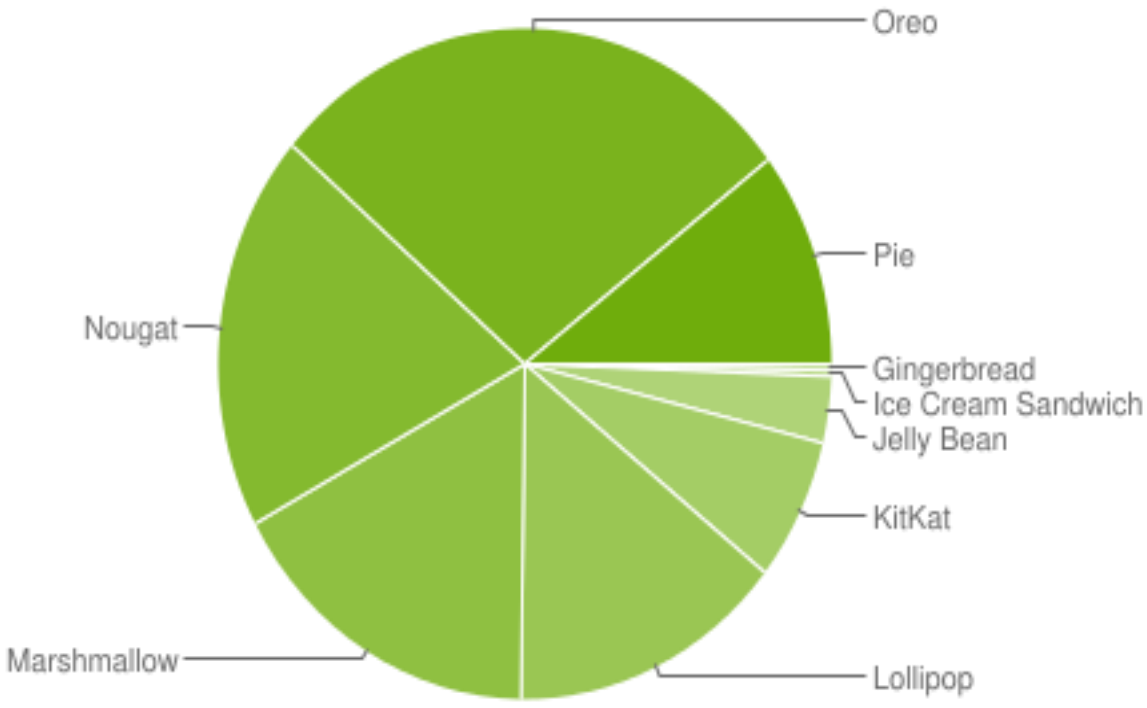


Рисунок 2.1 – Розподіл версій ОС Android на 7 травня 2019

Треба враховувати всі операційні системи, якими користується ваша цільова аудиторія. Так, системні елементи управління можуть виглядати

зовсім по-різному на різних версіях Android і різних оболонках однієї і тієї ж версії Android.

UI / UX повинен враховувати не тільки різні розміри екранів пристроїв, але і роботу в режимі багатовіконності, і щільність пікселів екранів: тонкий шрифт на неякісних дисплеях буде спотворений або зовсім зникне.

Також потрібно враховувати архітектуру самого додатка. На відміну від iOS, де додатки архітектурно представляють собою щось єдине ціле, в Android вони збираються з логічно самостійних і відокремлених частин - активують і фрагментів.

Такий підхід був обраний саме для того, щоб забезпечити роботу додатків на абсолютно різних пристроях, в тому числі з дуже малим об'ємом оперативної пам'яті і дуже слабкими процесорами. Якщо частини програми незалежні, будь-яку з них можна в потрібний момент викинути з пам'яті і не витрачати на підтримку її життєвого циклу дорогоцінні ресурси.

Саме це дає зрозуміти чому, наприклад, далеко не всі типи додатків можливо реалізувати кросплатформно: якщо це щось об'ємне по функціоналу, то воно повністю вивантажується з пам'яті при нестачі місця і на слабких пристроях можливість роботи з ними просто відсутня.

Андроид додатки можна заробляти за допомогою мов програмування Java і Kotlin та в травні 2019 року компанія Google оголосила Kotlin кращою мовою для Android-розробки.

Java є рідною мовою, що використовується Android, програми, які взаємодіють з операційною системою та безпосередньо використовують апаратне забезпечення, використовує Java. Ця мова дозволяє створювати будь-які програми і підтримує майже всі типи машин, а OS X, будь то Android, Windows або Linux. Java була розроблена Sun Microsystems (нині власністю Oracle), і з Java можна використовувати мікросервіси.

Kotlin − це молода мова, натхненна Java, але це її вдосконалена версія з такою кількістю додаткових функцій. Він чистий, відносно простий і містить менше формальностей та правил порівняно з Java та іншими мовами

програмування. Будь-яка частина коду, написана на Kotlin, набагато менша порівняно з Java, оскільки вона менш детальна, а менше коду означає менше помилок. Kotlin компілює код у байт-код, який може бути виконаний у JVM. Таким чином, усі бібліотеки та фреймворки, створені на Java, можна переміщувати та запускати в проекті Kotlin. Для того, щоб використовувати цю мову для програмування програм для Android, розробники все ще повинні розуміти основні концепції та структури програмування.

В даний момент існує 3 найбільш популярні середовища розробки під Android – Eclipse, IntelliJ IDEA та Android Studio.

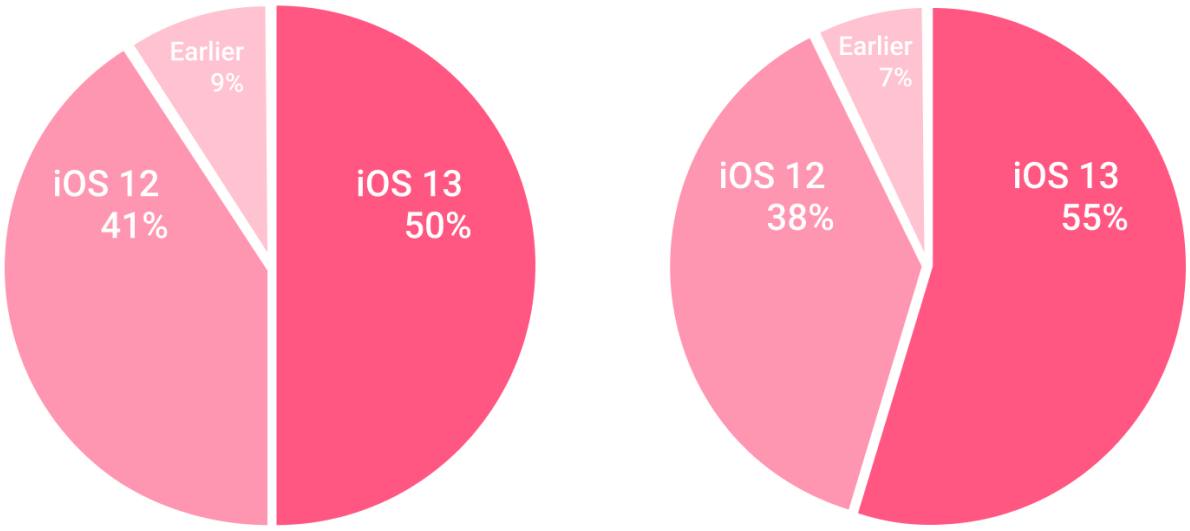
2.3 iOS

Операційна система iOS була випущена компанією Apple в 2007 році. До 2019 року на ній функціонували як iPhone, так і iPad, але зараз це змінилося − для iPad розробили власну ОС.

Перша і головна відмітна риса iOS додатків − то, що варіативність пристроїв значно менше, ніж кількість смартфонів на Android. Це означає, що адаптувати зовнішній вигляд мобільного застосування під актуальні на ринку айфони має бути простіше.

У той же час з виходом кожного нового пристрою і оновленням операційної системи переважна кількість мобільних додатків потрібно адаптувати під нові умови. Статистика App Store показує, що користувачі айфонів охоче оновлюються до актуальної версії ОС. Тому додатки повинні відповідати її вимогам, наприклад, підтримувати темну тему, представлену в пристроях в 2019 році.

На діаграмах нижче видно, що за один місяць з моменту презентації iOS 13 до неї оновилося 50% користувачів.



а) б)

Рисунок 2.2 – Статистика оновлень операційної системи на iOS пристроях за 2019 рік: а) всі пристрої; б) пристрої, за останні 4 роки.

Екрани сучасних iOS пристроїв при цьому мають гарний дозвіл. Це дозволяє використовувати тонкі шрифти: вони не спотворюються, як це буває на дисплеях низької якості.

Також однотипна архітектура пристроїв дозволяє не проводити додаткових перевірок при старті програми: не потрібно перевіряти наявність камери, GPS датчика або акселерометра.

Apple доволяє створювати додатки для iOS на мовах програмування Objective-C та Swift що з’явилася червні 2014 року і прийшла на зміну Objective-C. Мова для створення iOS-додатків відноситься до об'єктно-орієнтованого програмування (ООП) та успішно виконує основні парадигми: наслідування, поліморфізм, інкапсуляцію та абстракцію. ООП - це стиль написання коду, який дозволяє розробляти групування подібних завдань у класах. Код відповідає принципу DRY (don’t repeat yourself − не повторюй самого себе) і стає легким для супроводу.

Objective-C − мова програмування для iOS додатків, створений на початку 1980-х років минулого століття шляхом схрещування C з популярним в той час Smalltalk (зв'язок з об'єктами через повідомлення).

Objective-C спочатку сприймався, як проста надбудова над мовою C, що модифікує його деякі синтаксичні конструкції, але після того, як за ліцензування взялася спочатку компанія Next Step, а потім на правах наступника і Apple, Objective-C став одним з найбільш популярних мов для розробки додатків для iPhone і iPad. Це основна мова, що використовується компанією Apple, знання якої дозволяє писати під будь-які платформи Apple, в тому числі macOS.

Swift − молода, відкрита мова програмування програмування загального призначення. Офіційно представлена компанією Apple 2 червня 2014 року. Поєднує в собі все краще від C і Objective-C, але позбавлена обмежень останнього, що були додані для сумісності з C. У Swift використовуються сувора типізація об'єктів, що зменшує кількість помилок ще на етапі написання коду. Також в Swift додані сучасні функції, такі як дженерики, замикання, множинні повернені значення і багато іншого, що перетворюють створення додатка в більш гнучкий і захоплюючий процес.

Найбільш популярні середовища розробки додатків на iOS, або IDE (Integrated Development Environmetn) - це Xcode від Apple і AppCode від JetBrains.

3 ГІБРИДНІ ДОДАТКИ

Гібридні або кросплатформні додатки, можуть працювати в різних операційних системах. Після написання коду програми його можна розгорнути на різних пристроях і платформах, не турбуючись про проблеми несумісності. Це універсальний підхід, який широко використовується для економії часу і грошей на розробку. Часто для цього використовуються спеціалізовані кросплатформні фреймворки як Xamarin, React Native, Flutter, Cordova, Ionic та інші. Найголовніша перевага розробки кросплатформних додатків є простота повторного використання коду. Замість того, щоб випускати кілька власних програм для кожної платформи, гібридні додатки дають можливість повторно використовувати значну кількість коду і мати той самий додаток, розгорнутий на декількох платформах. Деякі фреймворки дозволяють створювати додатки не обмежуючи iOS та Android, але і підтримують операційні системи розумних телевізорів та наприклад, розумних годинників [21].

Серед переваг гібридного підходу також слід відзначити:

- один код доступний для повторного використання на інших платформах;
- розробка гібридних додатків економічно ефективна;
- просте і швидке розгортання;
- гібридні додатки покривають більш широку аудиторію;
- гібридні додатки допускають однаковий інтерфейс і UX.

Основною перевагою гібридної розробки мобільних додатків є той факт, що один і той же код може використовуватися на різних мобільних платформах. На відміну від розробки нативного додатки, для гібридної програми не потрібне використання окремого технічного стека для кожної операційної системи.

Повторне використання коду дозволяє легко розгортати додаток на

іншій платформі, так як можливості програми, реалізовані на одній платформі, будуть працювати і на інших платформах.

Одна команда може реалізувати потрібну ідею відразу на всіх платформах, використовуючи єдиний технологічний стек. Це призводить до менших витрат ресурсів.

Розробникам гібридних додатків не потрібно вивчати кілька технологічних стеків різних платформ перед створенням своїх додатків, їм потрібно добре освоїти один стек розробки та особливості його застосування. Оскільки немає необхідності створювати різні кодові бази, код пишеться один раз, команді розробників доведеться докласти половину зусиль, що обернеться зменшенням витрат на виробництво, обслуговування та найм. Початкове розгортання на цільових платформах відбувається набагато швидше. Згодом, менше часу, витраченого на розробку, означає, що ви зможете швидше надати цінність своїм клієнтам і скоріше налаштувати свої програми відповідно до їхніх відгуків. Крім того, майбутні зміни в додатку можуть виконуватися одночасно, без внесення індивідуальних змін на кожній платформі.

Гібридні додатки пропонують розробникам більше можливостей для охоплення ширшої аудиторії, оскільки такі програми досягають користувачів всіх типів і мобільних пристроїв, незалежно від їх операційної системи. Це значно рентабельніше для бізнесу, ніж присутність тільки на одній платформі.

Тоді як продуктивність важлива для будь-якого мобільного додатка, його зовнішній вигляд (UI) і відчуття (UX) так само важливі. Використання єдиної спільної команди розробників і єдиного коду дозволяє компаніям використовувати однаковий зовнішній вигляд програми на всіх платформах. Тобто один і той же користувальницький інтерфейс і UX буде однаково виглядати на всіх платформах.

Серед недоліків гібридного підходу слід відзначити:

- гібридні додатки не є такими гнучкими, як нативні додатки;

- гібридні програми не працюють так само добре, як нативні додатки;
- можлива невідповідність UI в різних платформах;
- відправка гібридних додатків до відповідних магазинів додатків може мати складнощі.

Хоча завдання програми будуть реалізовуватися на всіх платформах, швидше за все ви не зможете адаптувати готове додаток для використання максимальних можливостей кожної з платформ. Робота з уніфікованим стеком технологій не забезпечить такий же гнучкості настройки і оптимізації, як застосування стека технологій, індивідуального для кожної ОС.

Використання одного універсального стека технологій приносить в жертву гнучкість. Однак втрата гнучкості в розробці означатиме втрату можливості поліпшити продуктивність. Оскільки гібридні додатки відмовляються від деякої гнучкості, ці програми не будуть працювати так само добре, як нативні додатки. Відмінності здебільшого складають хвилину, але звісно що розробка мобільної гри з Flutter або React Native не буде відповідати нативній технології.

Зовнішній вигляд інтерфейсу програми і правильна настройка UI для відповідності функціонала в обох системах може доставити проблем. Наприклад, у кожної системи є свої вимоги до дизайну елементів UI. У певних випадках ці вимоги можуть виявитися взаємовиключними.

Механізм додавання вашого застосування, що є гібридним, в Apple App Store і в Google Play Store буде відрізнятися. Вимоги цих магазинів додатків до представлених у них продуктів різні. Проходження всіх перевірок і виконання всіх правил для відповідності обом магазинам будуть викликати певні складності.

На рисунку 3.1 можна ознайомитися з популярними кросплатформними мобільними фреймворками [3].

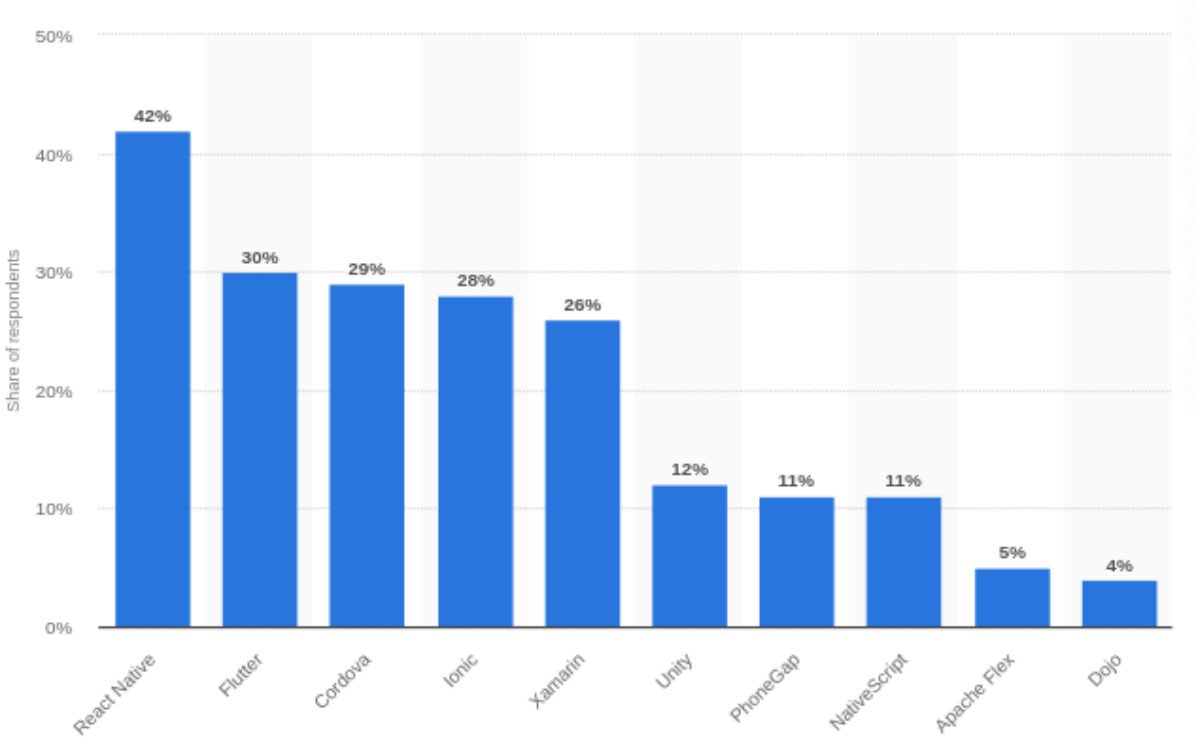


Рисунок 3.1 – Кросплатформні мобільні фреймворки, які використовували розробники програмного забезпечення у всьому світі в 2020 році

Якщо вибір зроблено на користь гібридних фреймворків, ми радимо звернути увагу на наступні аспекти роботи:

- оцінка;
- дизайн;
- CI/CD;
- splash screen;
- верстка;
- паралельна розробка web і mobile;
- debug;
- робота програми з файлами системи;
- доставка збірок клієнту.

В період оцінки тестувати потрібно всі задіяні платформи (iOS, Android). Важливо об'єктивно оцінити рівень знань і досвід усіх учасників проекту, щоб оцінка в роботи не виявилася заниженою. Передбачте ризик

появи багів в самих фреймворками React Native і Flutter під час розробки.

При дизайні деякі елементи складно (або зовсім неможливо) відрендерити в Flutter або React Native. З цієї причини дизайн обов'язково узгодити з розробниками − причому до того, як замовник прийме дизайн.

CI/CD. У React Native не виключені специфічні проблеми з автозбиранням (наприклад, через встановлення бібліотек на різні платформи). Потрібно закласти більше ризикового буферу.

Реалізація splash screen на Flutter відбувається швидше, ніж на React Native, де цей елемент можна відмалювати лише нативно, з великою ймовірністю виникнення багів. При використанні React Native на splash screen з усіма багофіксами бажано закласти більше часу.

При використанні React Native верстку на iOS і Android потрібно проводити одночасно, щоб надалі уникнути проблем при адаптації верстки під одну з систем.

Якщо веб-версія додатка написана на React, менше витрата часу на розробку мобільного додатку на React Native - за рахунок однакової логіки компонентів.

Якщо додаток великий, на React Native простіше провести тестування і юніт тести. На Flutter потрібно закладати більше часу на багофікс, оскільки логи не інформативні.

При роботі програми з файлами системи потрібно запрошувати дозвіл до sd-card, при цьому не з кожним файлом можливо отримати ім'я та шлях. Для відправки файлу потрібно використовувати ContentResolver. Для того щоб мінімізувати ризики, закладіть час на всі операції, пов'язані з файлової системою.

Тут немає істотних відмінностей від нативної розробки, можна вибрати будь-який зручний сервіс: Crashlytics, TestFairy, TestFlight [20].

3.1 React Native

React Native − це фреймворк мобільних додатків з відкритим кодом, створений Facebook, Inc у березні 2015 року. Він використовується для розробки програм для Android, Android TV, iOS, macOS, tvOS, Web, Windows та UWP, дозволяючи розробникам використовувати React фреймворк та JavaScript разом із можливостями власної платформи.

При створенні додатків на React Native також необхідно реалізовувати нативну частину, яка ініціалізує JavaScript-движок і свій JavaScript-код. Далі JS-додаток починає створювати нативні об'єкти за допомогою React Native і керує ними. Так як архітектура React Native дозволяє здійснювати оновлення JS-коду без перезавантаження додатки, то оновлення кросплатформеної частини допускається без обов'язкової перепублікація в магазині додатків.

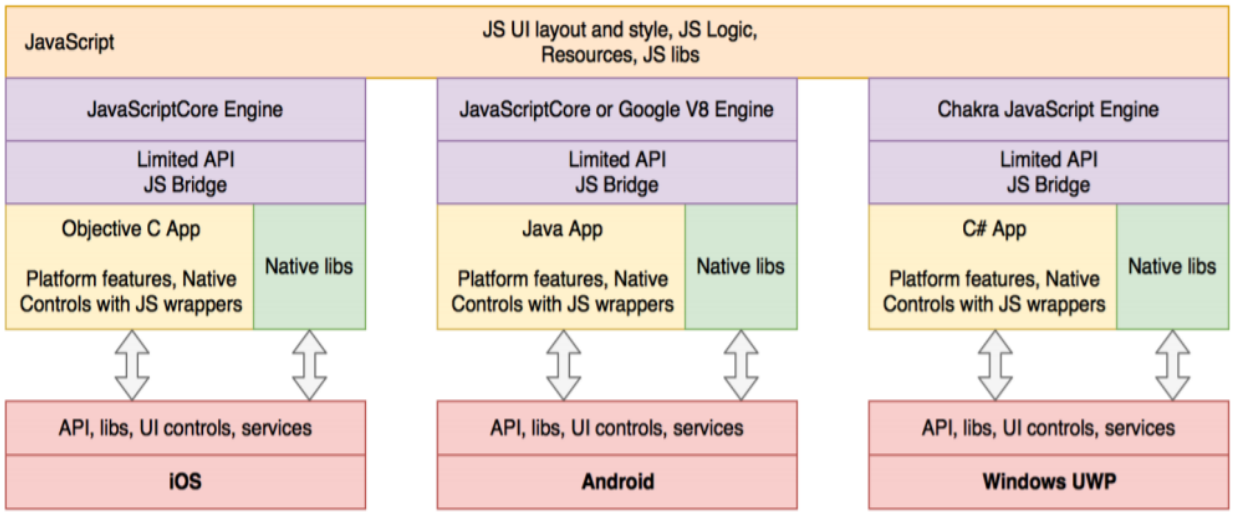


Рисунок 3.2 – Архітектура React Native

При створенні додатків на React Native потрібен досвід JavaScript, а також добре знання iOS і Android. інтеграцію нативної і кроссплатформеної частини легко зробити по офіційній документації. Інтерфейс є повністю нативним, але має обмеження і особливості при стилізації з JS-коду [6].

У кожному додатку React Native запущено два важливі потоки. Один з них − основний потік, який також працює у кожному стандартному рідному додатку. Він обробляє відображення елементів інтерфейсу користувача та обробляє користувальницькі жести.

Інший − специфічний для React Native. Його завдання − виконати код JavaScript в окремому механізмі JavaScript. JavaScript має справу з бізнес-логікою програми. Він також визначає структуру та функціональні можливості інтерфейсу користувача.

Ці два потоки ніколи не спілкуються безпосередньо і ніколи не блокують один одного. Між цими двома потоками знаходиться так званий міст, який є ядром React Native. Міст має три важливі характеристики [10].

Він асинхронний − це забезпечує асинхронний зв'язок між потоками. Це гарантує, що вони ніколи не блокують одне одного.

Також, він серійний – це означає що він оптимізовано передає повідомлення з одного потоку в інший.

Міст React Native має два потоки які ніколи не обмінюються одними даними та не працюють з ними. Натомість вони обмінюються серіалізованими повідомленнями.

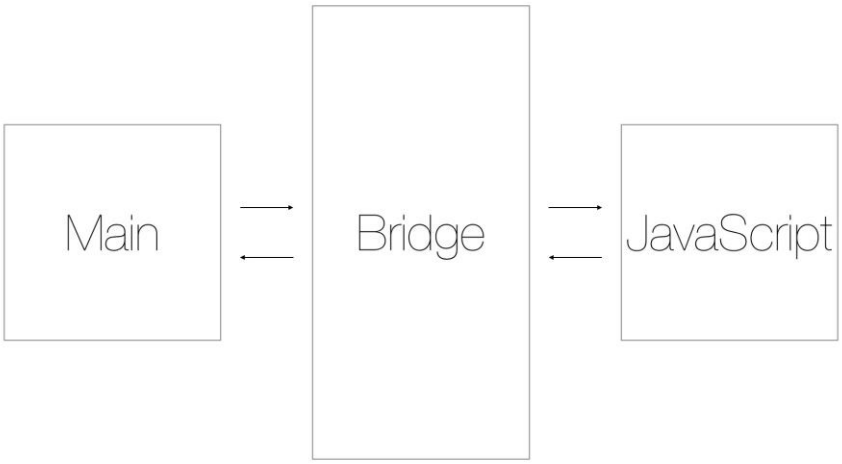


Рисунок 3.3 – Архітектура React Native

Ключовим поняттям у React є компонент. Компонент − це певна частина інтерфейсу користувача, наприклад настутний приклад використовує Text, так у вебі ми використовуємо span примітив.

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

const Greeting = () => {
      return <Text style={styles.greeting}>Hi!</Text>;
};

const styles = StyleSheet.create({
      greeting: {
            fontSize: 100,
      },
});

export default Greeting;
```

Приклад 3.1 – Простий приклад компоненту у React Native

Такий компонент не містить коду для певної платформи. React Native надає Android − TextView, а iOS – UIView замість Text.

Крім того, розробляти додаток за допомогою React Native досить швидше і зручніше для розробників, оскільки це фреймворк з відкритим кодом і дозволяє повторно використовувати кодову базу для розробки додатків. Тож загалом, React Native ідеальний інструмент для розробки додатків на рівні підприємства.

Основні випадки використання React Native: Facebook та Instagram.

3.2 Flutter

Flutter – це програмний каркас із відкритим кодом, для створення додатків для платформ Android та iOS, а також на вебі, розроблений компанією Google. Google запустив Flutter у 2018 році, він добре підходить для сучасного середовища розробки додатків. Це набір для розробки програмного забезпечення з відкритим кодом (SDK), що пропонує широкий

вибір елементів інтерфейсу, віджети та високопродуктивний механізм візуалізації, що дозволяє розробникам налаштовувати програму за допомогою плавної анімації.

Оскільки Flutter – дитина Google, то вона і надалі зростатиме і пропонуватиме активну підтримку спільноти. Ця платформа цікава своєю простотою порівнянної з розробкою веб-додатків, і швидкістю роботи на рівні з нативними додатками. Висока продуктивність програми і швидкість розробки досягається за рахунок декількох технік.

На відміну від багатьох відомих на сьогоднішній день мобільних платформ, Flutter не використовує JavaScript ні в якому вигляді. В якості мови програмування для Flutter вибрали Dart, який компілюється в бінарний код, за рахунок чого досягається швидкість виконання операцій порівнянна з Objective-C, Swift, Java, або Kotlin.

Flutter не використовує нативні компоненти, знову ж таки, ні в якому вигляді, так що не доводиться писати ніяких прошарків та мостів для комунікації з ними. Замість цього, подібно до ігрових движків він рендерить весь інтерфейс самостійно. Кнопки, текст, медіа-елементи, фон − все це промальовується всередині графічного движку в самому Flutter. Після вищесказаного варто відзначити, що "Hello World" додаток на Flutter займає зовсім небагато місця: iOS ≈ 2.5Mb і Android ≈ 4Mb.

Для побудови UI під Flutter використовується декларативний підхід, натхненний веб-фреймворком ReactJS, на основі віджетів. Для ще більшого приросту в швидкості роботи інтерфейсу віджети перемальовуються по необхідності − тільки коли в них щось змінилося (подібно до того як це робить Virtual DOM в світі веб-фронтенду) [17].

Окрім цього, Flutter має функцію гарячого перезавантаження, яка дозволить розробникам переглядати зміни в кодуванні без необхідності їх збереження.

Flutter складається з:

- Flutter рушію для рендерингу;
- базової бібліотеки (Foundation library);
- віджетів.

Рушій Flutter написаний в основному на C++ з використанням графічної бібліотеки Google Skia. Він також використовує SDK платформ Android або iOS;

Бібліотека складається з класів та функцій (написані на Dart), які використовують для побудови Flutter програм, для взаємодії із Flutter рушієм;

Дизайн інтерфейсу користувача у Flutter будують з віджетів. Віджет у Flutter являє собою незмінний об'єкт, який описує частину інтерфейсу користувача. Вся графіка, текст, фігури та анімації створюють за допомогою віджетів. Складні віджети створюють шляхом об'єднання простих.

На поточний час Flutter містить два набори віджетів, які відповідають відповідним принципами побудови:

- віджети Material Design використовують дизайн Google;
- віджети Cupertino імітують дизайн Apple iOS.

Flutter розроблений як розширювана багатошарова система. Він існує як серія незалежних бібліотек, кожна з яких залежить від базового рівня. Жоден шар не має привілейованого доступу до шару нижче, і кожна частина рівня фреймворку спроектована як необов’язкова та замінна (рисунок 3.2).

В основі Flutter лежить механізм Flutter, який здебільшого написаний на C++ і підтримує примітиви, необхідні для підтримки всіх додатків Flutter. Двигун відповідає за растеризацію складених сцен, коли потрібно намалювати новий кадр. Він забезпечує низькорівневу реалізацію основного API Flutter, включаючи графіку (через Skia), макет тексту, файлові та мережеві вводи-виводи, підтримку доступності, архітектуру плагінів, а також середовище виконання та компіляції Dart [11].

Механізм піддається дії Flutter через dart: ui, який обгортає базовий код C ++ у класах Dart. Ця бібліотека містить примітиви найнижчого рівня, такі

як класи для керування введенням, графікою та підсистемами візуалізації тексту.

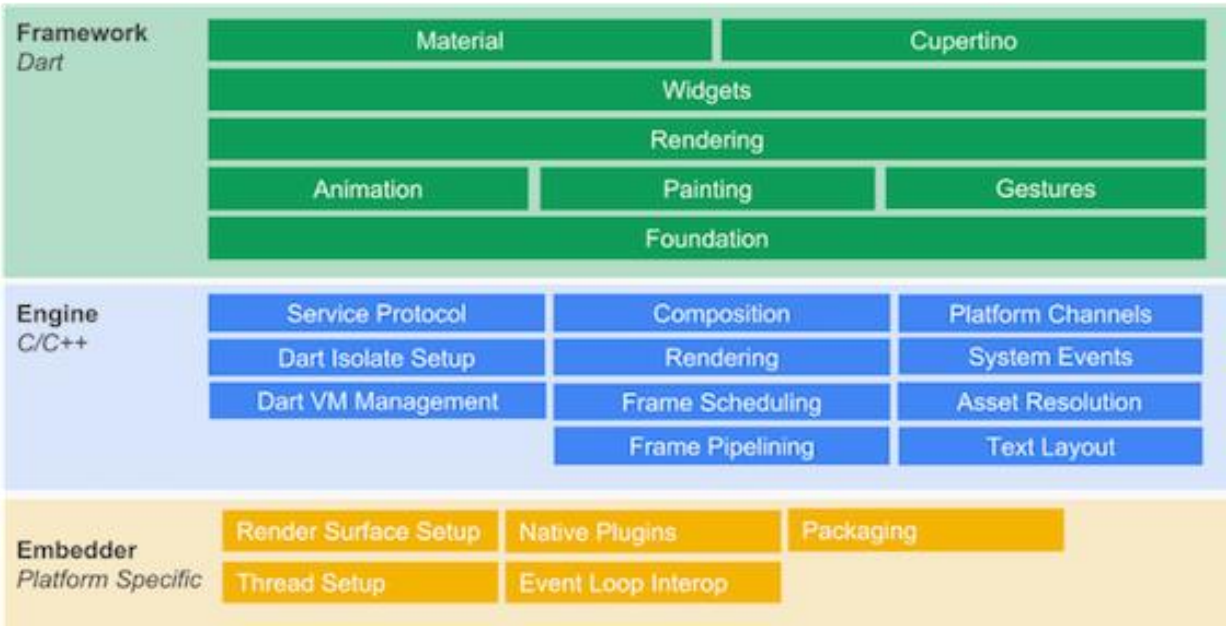


Рисунок 3.4 – Архітектура Flutter

Як правило, розробники взаємодіють з Flutter через фреймворк, який забезпечує сучасну реактивну структуру, написану мовою Dart. Він включає багатий набір бібліотек платформи, верстки та основоположних бібліотек, що складається з низки шарів.

Основні основоположні класи та такі основні послуги, як анімація, малювання та жести, які пропонують часто використовувані абстракції над основним фундаментом.

Шар рендерингу забезпечує абстракцію для роботи з макетом. За допомогою цього шару ви можете побудувати дерево об’єктів, що відображаються. Ви можете динамічно маніпулювати цими об’єктами, дерево автоматично оновлює макет, щоб відобразити ваші зміни.

Шар віджетів є абстракцією композиції. Кожен об'єкт візуалізації на рівні візуалізації має відповідний клас на рівні віджетів. Крім того, шар віджетів дозволяє визначити комбінації класів, які можна використовувати

повторно. Це рівень, на якому представлена модель реактивного програмування.

Бібліотеки Material і Cupertino пропонують вичерпні набори елементів керування, які використовують примітиви складу віджета для реалізації мов дизайну Material або iOS.

Каркас Flutter порівняно невеликий; багато функцій вищого рівня, які можуть використовувати розробники, реалізовані у вигляді пакетів, включаючи плагіни платформи, такі як камера та веб-перегляд, а також функції агностики платформи, такі як символи, http та анімації, що базуються на основних бібліотеках Dart та Flutter. Деякі з цих пакетів походять із ширшої екосистеми, що охоплює такі послуги, як оплата в додатках, аутентифікація Apple та анімація.

Основні випадки використання Flutter: Google Ads та Alibaba.

3.3 Apache Cordova

Apache Cordova − це основа для розробки мобільних додатків з відкритим кодом. Це дозволяє використовувати стандартні веб-технології − HTML5, CSS3 та JavaScript для крос-платформної розробки. Додатки працюють в орієнтованих на платформу обгортках і покладаються на прив'язки API, що відповідають стандартам, для доступу до можливостей кожного пристрою, таких як датчики, дані, стан мережі тощо. Apache Cordova підходить для таких завдань як поширення мобільних додатків на більш ніж одній платформі з використанням однієї кодової бази.

Для розгортання програми, яка вже упакована для розповсюдження в магазинах різних операційних систем, а також для створення змішаного додатка з використанням як власних технологій, так і WebView.

Архітектура програм Cordova складається з декількох компонентів. WebView − це вбудований браузер, який власна програма може використовувати для відображення веб-вмісту. Тобто програми, написані за

допомогою WebView, відкриваються власними браузерами мобільних додатків, але виглядають не як вікна браузера, а як окремі програми. Вбудований Cordova WebView призначений для відображення користувацького інтерфейсу в програмі. У деяких випадках цей інструмент може бути частиною програми, яка робить ці програми додатками змішаного типу, які використовують як WebView, так і власні компоненти. WebApp - це частина коду, де знаходиться наш додаток. Сама програма реалізована як веб-додаток, за замовчуванням виконується локальний файл під назвою index.html, на який посилаються CSS, JavaScript, зображення, мультимедійні файли або інші ресурси, необхідні для його запуску. Додаток працює в корневому WebView, що відкрито на весь єкран мобільного пристрою. Готовий додаток поширюється за допомогою інтернет-магазинів додатків.

Архітектура програм, створених за допомогою Cordova забражена на рисунку 3.5.

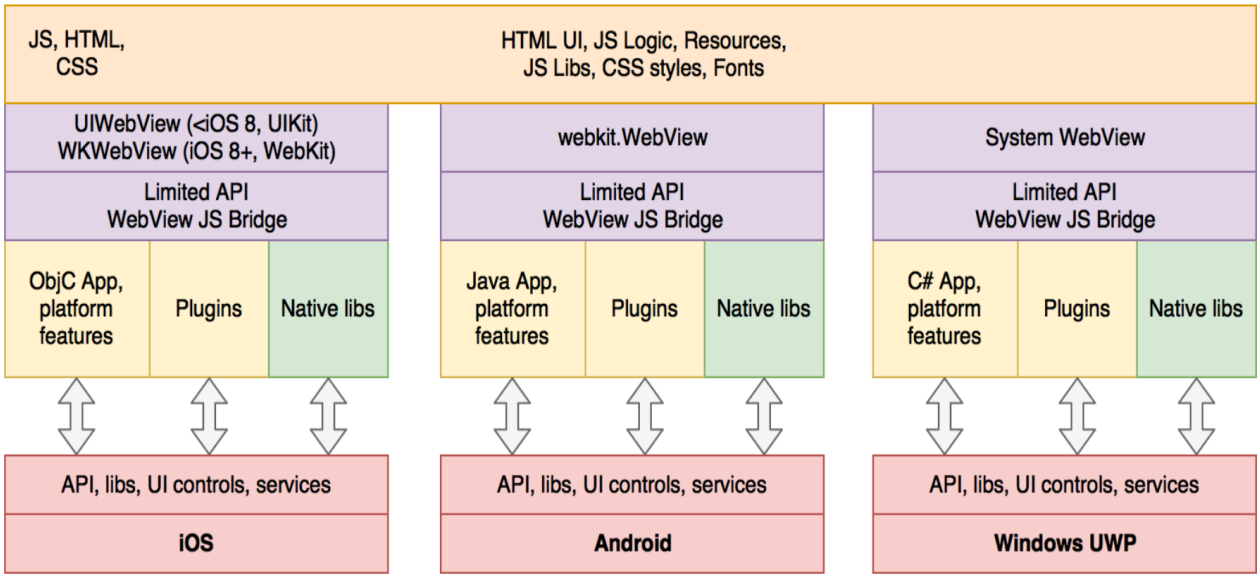


Рисунок 3.5 – Принцип роботи додатку

Плагіни забезпечують інтерфейс для Cordova та власних компонентів для взаємодії між собою та прив’язки до стандартних пристроїв API. Це дозволяє вам викликати власний JavaScript. Проект Apache Cordova

підтримує набір плагінів під назвою Core Plugins. Ці основні плагіни надають програмі доступ до можливостей пристрою, таких як акумулятор, камера, контакти тощо. На додаток до основних плагінів, існує кілька сторонніх плагінів, які надають додаткові прив'язки до функцій, які не обов'язково доступні на всіх платформах. Також можна створити власні плагіни за допомогою Посібника з розробки плагінів Apache.

Рішення на базі Apache Cordova вживають WebView і є простими з точки зору реалізації: створюється невелике нативное додаток, яке здійснює виклик нативного функціоналу з JavaScript в локальному WebView додатки. Таким чином, програма може мати один універсальний код на JavaScript і реалізовані на різних платформах єдиний API для доступу до нативном функціонал. Apache Cordova дозволяє розділяти весь код між платформами і вимагає реалізації нативной частини на мовах програмування Objective C, Java і C #. При розробці додатків в даному середовищі так само потрібен досвід роботи з такими мовами, як HTML, JavaScript, CSS, і хороші інженерні знання для інтеграції нативной і кроссплатформенной частин. Apache Cordova підходить для невеликих додатків, так доступ до нативної частини досить ускладнений.

Основними перевагами Cordova можна вважати такі особливості:

- простота розробки;
- ідеально підходить для невеликих додатків, що активно не використовують нативні можливості пристроїв;
- доступність для усіх фахівців із різними навичками.

Cordova дозволяє вести розробку використовуючі різні фреймворки, такі як React, Angular, Vue.js. Також існує можливість вести розробку на мові програмування typescript, що гарантує безпечність типів та перевірки під час компіляції.

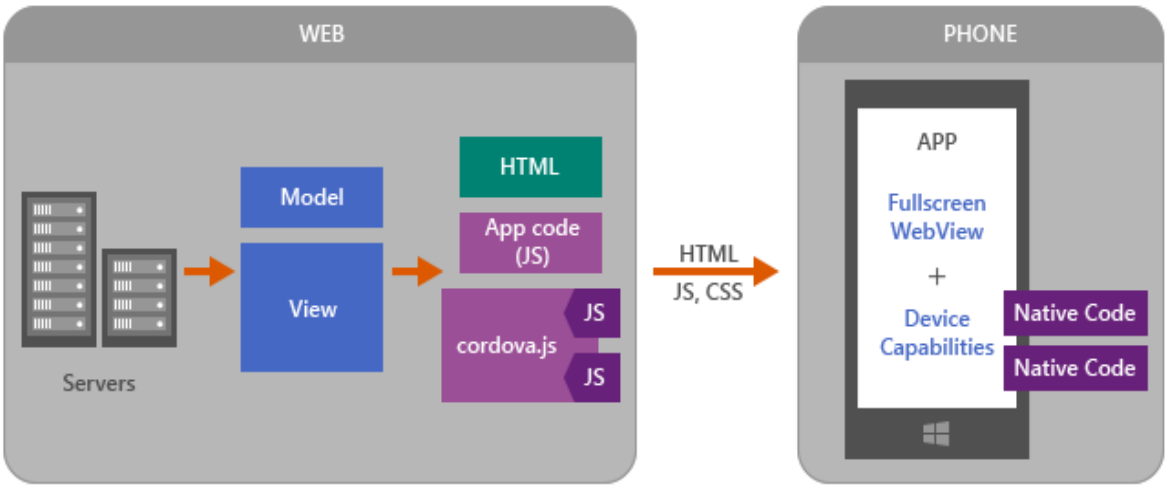


Рисунок 3.6 – Структура клієнт-серверного додатку, що базується на Cordova

Але існують і недоліки, такі як:

- ускладний алгоритм роботи із нативними API пристроїв;
- складність розробки та підтримки додатків із великою кількістю функціоналу;
- нижча продуктивність додатку, ніж для нативного додатку, так як код не компілюється а інтерпретується по ходу виконання.

Фреймворк використовується у таких компаніях як Adobe, BlackBerry, Google, IBM, Intel, Microsoft, Mozilla, та інші.

3.5 Xamarin

Xamarin – це компанія, що була заснована в травні 2011 року інженерами, що створили Mono, Mono for Android та MonoTouch, що є реалізаціями Common Language Infrastructure (CLI) та Common Language Specifications(що часто називають Microsoft.NET). Історія заснування та розвитку: На початку XXI століття Microsoft вперше оголосила про створення .NET Framework – спеціальний фреймворк, що дозволяв писати програми під ОС Windows, використовуючи різні мови програмування. Мігель Де Іказа з компанії Ximian підняв питання про реалізацію даного

фреймворку для платформи Linux. Ця ідея мала розвиток у вигляді проекту з відкритим вихідним кодом, що називається Mono, який був запущений 19 липня 2001 року. Однак, дана ідея хоч і мала розвиток, проте через декілька років, все жтаки розробку Mono було призупинено.

У 2011 році Мігель Де Іказа анонсував в своєму блозі про те, що Mono буде розвиватися далі і підтримуватися компанією Xamarin, що була створена у тому ж році і основним напрямком розробки даної компанії буде розробка під мобільні пристрої. Він також повідомив про те, що значна частина людей, що працювала над Mono перейшла до нової компанії.

Наступного року, компанією Xamarin було випущено Xamarin.Mac – плагін для IDE MonoDevelop, що дозволяв розробникам писати C# додатки для операційної системи Apple OS X і публікувати їх в Apple App Store. Згодом було анонсовано створення нової версії Xamarin 2.0. Даний реліз влюкчав у себе два основні компоненти – Xamarin Studio, що заміняла MonoDevelop та плагін для інтеграції у Visual Studio – IDE від Microsoft для .Net Framework розробників. Завдяки цьому C# розробники змогли створювати свої додатки для iOS, Android аналогічно Windows та Windows Phone(WP).

У 2016 році компанії Xamarin та Microsoft анонсували те, що Microsoft підписала угоду, за якою вона повністю купує компанію Xamarin, хоча точну цифру за яку було придбано Xamarin не називають, в одному з журналів припустили, що сума варіюється десь в межах 400 - 500 мільйонів доларів.

Однак, є й погані новини, як було зазначено у першому розділі, Xamarin колись придбала усі права на RoboVM, що є аналогічним фреймворком для Xamarin.Forms для Java. Однак, після придбання компанії Xamarin компанією Microsoft, Microsoft анонсували, що не будуть підтримувати даний продукт і усі контракти буде анульовано після 30 квітня 2017 року.

На конференції Build 2016 було анонсовано те, що Xamarin тепер є повністю безкоштовним фреймворком, підтримується Microsoft, також буде

повністю реліцензійовано Mono.

Основними продуктами, що було створено за цей період часу є:

- Xamarin Platform;
- Xamarin.Forms;
- Xamarin Test Cloud;
- Xamarin for Visual Studio;
- Xamarin.Mac;
- Xamarin Studio;
- .Net Mobility Scanner.

Xamarin Platform − платформа, що дозволяє використовувати нативні API конкретної операційної системи для написання додатків.

Xamarin.Forms − дозволяє будувати інтерфейс з використанням C# та мови XAML замість AngularJS для побудови UI.

Xamarin Test Cloud − даний сервіс дозволяє розробнику протестувати свій додаток на різних телефонах, не маючи його. Достатньо лише мати підключення до інтернету та підписку (дана функція є платною), щоб мати можливість проводити тестування своїх додатків у великій базі фізично існуючих смартфонів.

Xamarin for Visual Studio − підтримка Xamarin у Visual Studio дозволяє розробнику писати додатки на iOS та Android не використовуючи Mac та Xamarin Studio. Хоча для того, щоб розробляти iOS додатки, все одно потрібно мати підключення до OS X пристрою.

Xamarin.Mac − фреймворк, що дозволяє писати додатки для Mac;

Xamarin Studio − основна IDE, що використовується для написання додатків під iOS чи Mac з використанням Xamarin.

.Net Mobility Scanner − спеціальний сервіс, що дозволяє просканувати бібліотеку, написану на C#, щоб дізнатися, чи підходить вона для додатків на Xamarin.iOS, Xamarin.Android і т.д.

Існує 2 основних підходи до проєктування додатків:

- нативні;

- Xamarin Forms.

У нативному підході для кожної платформи окремо створюється дизайн, додаток проєктується враховуючи особоливості роботи конкретної платформ;

Для спрощення процессу розробки додатків у Xamarin було додано підтримку технології Xamarin forms.

Xamarin.Forms – це зовсім несхожа технологія на дві інші, що були описані раніше. Дана технологія більше схожа на JavaScript фреймворки, які було описано в першому розділі. Різниця в тому, що Xamarin.Forms дозволяє писати нативні додатки з більшою кількістю спільного коду. Наприклад, інтерфейс програми буде спільний для усіх додатків(по структурі та по написанню, проте матиме дещо різний вигляд на платформах) оскільки для побудови інтерфейсу користувача використовується спеціальна мова розмітки – XAML. Тобто розробнику не потрібно знати особливості кожної платформи, розробники із Xamarin все зробили за вас.

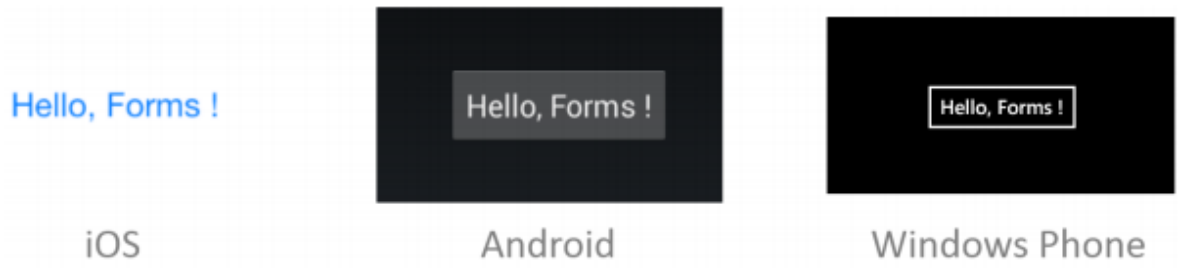


Рисунок 3.7 – Стандартний вигляд кнопок для кожної платформи в Xamarin.Forms

Xamarin.Forms надає розробнику абстракції графічного інтерфейсу, що використовуються для кожної платформи. Дана технологія використовує нативні компоненти під час роботи додатку, тобто код графічного інтерейсу на різній платформі має різне підґрунтя, а саме його буде перетворено на нативний код окремої конкретної платформи. Код Xamarin.Forms має можливість взаємодіяти з операційною системою, використовувати її

програмний інтерфейс.

Основними типами компонентами, що використовуються в розробці з використанням Xamarin.Forms є:

- представлення;
- макет;
- сторінка.

Представлення – це основний блок графічного інтерфейсу, бо саме з цих елементів його побудовано. Прикладом представлення є кнопка, поле вводу і т.д.;

Макети визначають, як представлення будуть розташовані на екрані, їх розмір при різних розмірах екрану, поведінку представлень в певних ситуаціях;

Сторінка має дві основні функції. Перша − сторінка представляє собою контейнер для представлень та макетів, тобто сторінка і є екраном телефону, також сторінка має і іншу функцію – навігаційну. Саме сторінка дозволяє перехід від однієї сторінки до іншої.

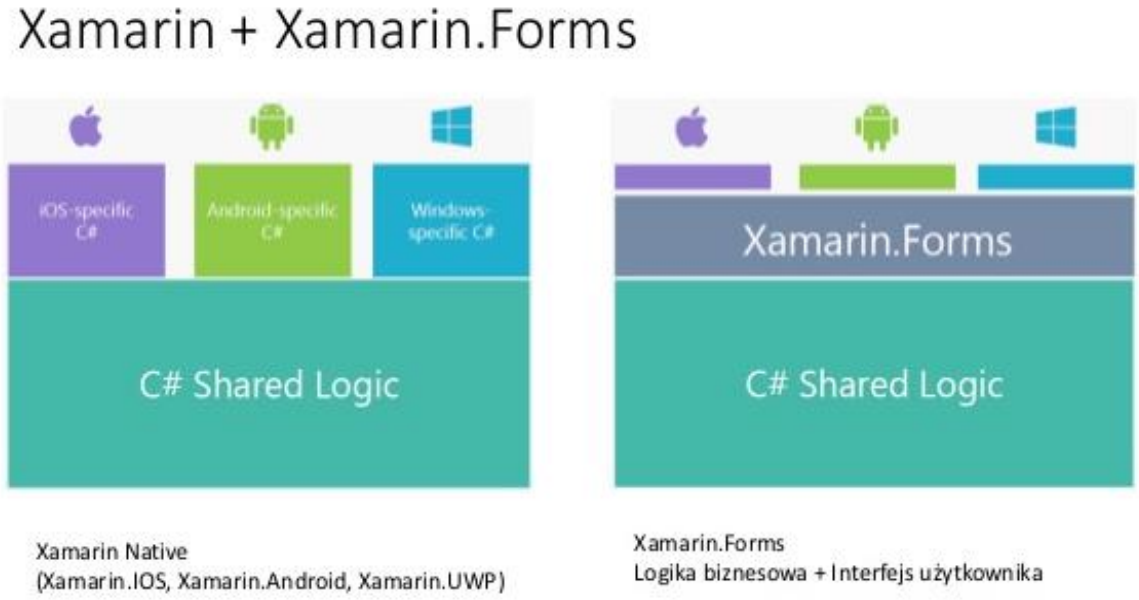


Рисунок 3.8 – Порівняння двох підходів до побудови додатків використовуючи технологію Xamarin

Основними блоками, що дозволяють писати логіку додатку є клас Device, центр обміну повідомленнями(message center) та сервіс залежностей (dependency service). Клас Device надає методи для використання платформоспецифічних можливостей додатку та побудови поведінки додатку з точки зору ОС. Центр обміну повідомленнями надає можливість використовувати методи publish/subscribe – тобто підписуватися на якусь подію, що сталася в ОС, чи, наприклад, зіставити метод, що буде спрацьовувати при натисканні кнопки.

До випадків використовування технології можна віднести: Alaska Airlines, Oro, MRW та AOX.

4 ВЕБ-ДОДАТКИ

Поняття мобільного веб-додатки можна трактувати по-різному. Мобільна версія або m(dot) – історично перший мобільний веб-рішення. Зазвичай це урізана версія сайту зі спрощеної навігацією, меншою кількістю контенту і сторінок.

Коли користувачі тільки почали заходити в веб зі смартфонів, сайти працювали повільно, а іноді і взагалі не працювали на маленьких екранах. Щоб вирішити цю проблему, компанії почали створювати мобільні версії сайтів. При спробі користувача зайти на сайт з телефону, вони автоматично перенаправляються на сайт приставкою "m." в адресі.

Мобільні версії були хорошим виходом, коли мобільний браузер тільки з'явився. Але зараз це вважається застарілою технологією, яка не дозволяє ефективно поліпшити клієнтської досвід або підвищити онлайн конверсію.

Деякі джерела трактують мобільний веб-додаток як сайт з чуйним або адаптивним дизайном, це досить популярне рішення на зараз, у якого є безліч плюсів. Такий додаток доступний для пристрою на будь-якій операційній системі в якому є браузер, він простий в обслуговуванні і швидкий в розробці.

Однак у таких додатків немає доступу до функціоналу мобільного пристрою та його не можна зберегти як мобільний додаток, однак такий додаток це гарна основа для прогресивних web-додатків.

4.1 PWA

Прогресивний web-додаток (англ. Progressive Web App, PWA) – технологія в web-розробці, яка візуально і функціонально трансформує сайт в додаток (мобільний додаток в браузері).

Статистика говорить про те, що 66% користувачів не скачують

жодного додатка в місяць (дані comScore від 2014 року - в середньому за три місяці). Більшу частину свого часу − приблизно 85% − користувач проводить в п'яти улюблених додатках. Як правило, це месенджери, соцмережі, відеохостінги [5].

При цьому мобільний браузер також багато в чому не є пріоритетною формою виходу в Інтернет. За даними comScore, у 2017 році користувачі смартфонів і планшетів витратили 87% свого часу на додатки − в порівнянні з 13% в браузері.

PWA є гібридним рішенням і дозволяє відкрити програму за допомогою мобільного браузера. При цьому повністю зберігається функціонал нативного додатку:

- відправка push-повідомлень;
- робота в режимі офлайн;
- доступ до апаратного забезпечення пристрою (з обмеженнями);
- установка ярлику (іконки) на робочий стіл мобільного пристрою, візуально не відрізняється від ярлика нативного додатки, тощо.

Один з варіантів впровадження PWA скористатися готовим безкоштовним фреймворком з відкритим кодом. Серед них виділяються Ionic що був розглянутий вище або Vue Storefront [13].

Vue Storefront – це безкоштовний фреймворк для PWA інтернет-магазину з відкритим вихідним кодом. Написаний на Vue.js. Він досить гнучкий і адаптивен, що робить його досить універсальним рішенням для інтеграції з Pimcore/CoreShop, BigCommerce, PrestaShop, Shopware або, наприклад, Magento через API.

Також одним з універсальних рішень для впровадження PWA є фреймворк Quasar.

Крім використання фреймворка, можна інтегрувати технологію PWA, використовуючи плагіни. Для кожної CMS існує свій стандартний плагін. У цьому випадку спочатку необхідно перевірити, щоб сайт відповідав вимогам:

- Service Worker;

- архітектура application shell (оболонка для швидкого завантаження з Service Worker);
- Web App Manifest;
- Push Notifications;
- SSL-сертифікат для передачі даних по протоколу HTTPS.

Service Worker − це JavaScript-файл, який запускається у фоновому режимі як автономний сервіс. Він не пов'язаний з DOM (Document Object Model) або web-сторінками, працює на іншому потоці і отримує доступ до DOM за допомогою API postMessage.

З точки зору користувача Service Worker дозволяє виконувати такі дії, як, наприклад, відправка push-повідомлень і попередня завантаження матеріалів для перегляду в автономному режимі офлайн.

Application shell – це віртуальна оболонка. Подібно оболонці нативного додатки, вона завантажується при його запуску, а далі динамічна інформація завантажується на неї з мережі [8].

Web App Manifest надає інформацію про програму в текстовому JSON-файлі. Необхідний, щоб web-додаток було завантажено і візуально відображалося для користувача аналогічно нативному додатком.

Може містити наступні елементи: background_color, categories, description, dir, display, iarc_rating_id, icons, lang, name, scope, screenshots, serviceworker, short_name, start_url, theme_color тощо. Всі вони відповідають за інформацію, яку користувач зазвичай бачить після установки: назва, колір фону, створення іконки на екрані смартфона і т. д.

Крім цього, PWA вимагає, щоб всі ресурси сайту передавалися по HTTPS-протоколу.

PWA поєднує в собі властивості нативного додатку та функціонал браузера, що має свої переваги:

- PWA підтримується найбільш популярними ОС: Windows, iOS, Android. При цьому завантажити можна на десктоп, смартфон, планшет, термінал в торговому залі;

- новий функціонал і поновлення додаються розробниками віддалено. Користувачі бачать зміни і поліпшення, але їм не потрібно завантажувати ці оновлення самостійно;

- PWA індексується Google і іншими пошуковими системами;

- завдяки сценарієм Service Worker, який запускається браузером в фоновому режимі, і стратегії кешування забезпечується можливість роботи офлайн;

- front відділений від back'a. Менше часу і ресурсів витрачається на розробку і переробку дизайну і логіки взаємодії PWA з клієнтом;

- PWA можна встановити без Google Play Store і App Store, а також всупереч забороні встановлювати додатки з невідомих джерел. Лояльно ставляться до PWA і антивірусні програми. Одночасно з цим передача даних відбувається по протоколу HTTPS, тому PWA безпечно;

- з лютого 2019 року PWA можна додавати в App Store і Google Play, даючи користувачеві можливість завантажити додаток зі звичного джерела.

Технологія PWA неуніверсальна і має ряд недоліків:

- не всі пристрої і не всі операційні системи підтримують повний функціонал PWA;

- неможливо налагодити активну участь користувачів iOS (наприклад додаток може зберігати локальні дані і файли розміром тільки до 50 Мбайт, немає доступу до In-App Payments (вбудовані платежі) і багатьом іншим сервісам Apple, немає інтеграції з Siri), підтримка iOS починається з версії 11.3;

- робота офлайн обмежена;

- роботу PWA може обмежувати неповний доступ до апаратних компонентів;

- немає достатньої гнучкості щодо «спеціального» контенту для користувачів (наприклад програми лояльності);

- при використанні PWA збільшується витрата заряду батареї мобільного пристрою.

Інтернет-магазини активно використовують PWA. Більше 60% від світового обсягу роздрібних продажів в e-Commerce виробляються через онлайн-канали. До 2021 року очікується зростання цього показника до 73% (за даними Statista) [9].

Функціонал PWA дозволяє працювати з e-Commerce офлайн і підвищувати конверсію за рахунок безперервності сесій. З PWA користувач може управляти push-повідомленнями, входити в «Особистий кабінет» в один клік, оплачувати покупки банківськими картами або системами Apple Pay і Google Pay.

Простота впровадження залежить від того, чи використовує інтернет-магазин платформу з відкритим вихідним кодом. Для SaaS-рішень потрібно кастомними розробка PWA від постачальника.

Одна з платформ, доступних для впровадження PWA, - Magento CMS. Технологія PWA набула широкого поширення не так давно, проте вже накопичено досвід інтеграції PWA з Magento.

Як правило, для прискореного впровадження використовується вбудоване рішення − PWA Studio. Однак в ньому до цих пір не реалізована технологія SSR (server-side rendering), що дозволило б видавати пошуковій системі готову сторінку за запитом. У цьому випадку можливе використання Vue Storefront. Для пошукового робота сторінка на Vue Storefront виглядає так само, як і звичайний сайт, − це важливо для SEO.

Мережа Starbucks розробила PWA для збору замовлень на додаток до звичайного мобільного додатку. При майже однаковій інтерфейсі вага PWA виявився менше на 99,84%. В результаті кількість замовлень через мобільний Інтернет подвоїлася і майже зрівнялася з кількістю замовлень через десктоп.

Кейс по впровадженню PWA став успішним і для AliExpress. Показник конверсії для нових користувачів підвищився на 104%. Функціональність PWA також допомогла їм генерувати вдвічі більше відвідувань сторінок за сеанс. Час сеансу збільшилася в середньому на 74% у всіх браузерах (за даними developers.google.com).

5 НАТИВНІ МОДУЛІ У ГІБРИДНИХ ДОДАТКАХ

За даними Statista, 42% розробників все ще розглядають можливість розробки додатків з React Native, оскільки це дозволяє писати модулі на основних рідних мовах, включаючи C ++, Java, Swift, Objective та Python [7].

Власний модуль − це набір функцій javascript, які реалізовані власноруч для кожної платформи (у нашому випадку це iOS та Android). Наприклад іноді додатку потрібен доступ до API платформи, для якого React Native ще не має відповідного модуля. Можливо, ви хочете повторно використати якийсь існуючий код Java, не потребуючи його повторного впровадження в JavaScript, або написати якийсь високопродуктивний багатопотоковий код, наприклад, для обробки зображень, бази даних або будь-якої кількості вдосконалених розширень. Якщо React Native не підтримує потрібну вам рідну функцію, ви зможете створити її самостійно.

Власні модулі, як правило, поширюються як пакети npm, крім типових файлів javascript та ресурсів, вони будуть містити наприклад проект бібліотеки Android та, або iOS. Можно встановити вже стрворенный модуль або створити власний.

Наприклад ми хотіли б мати доступ до календаря iOS із JavaScript, мы можемо використати API календаря iOS .

Власний модуль − це клас Objective-C, який реалізує RCTBridgeModule протокол.

Клас також повинен також включати RCT_EXPORT_MODULE() макрос. Це бере необов’язковий аргумент, який вказує ім’я, до якого модуль буде доступний, як у вашому коді JavaScript. Якщо не вказати ім'я, ім'я модуля JavaScript буде відповідати імені класу Objective-C. Та React Native не буде піддавати JavaScript будь-яким методам, CalendarManager якщо це прямо не сказано. Це робиться за допомогою RCT_EXPORT_METHOD() макросу:

```
# import <React / RCTBridgeModule.h>

@interface CalendarManager : NSObject < RCTBridgeModule >
RCT_EXPORT_MODULE ();
RCT_EXPORT_METHOD(addEvent:(NSString *)name location:(NSString
*)location)
{
  RCTLogInfo(@"Pretending to create an event %@ at %@", name, location);
}
@end
```

Приклад 5.1 – Клас календаря (CalendarManager.h)

Тепер із файлу JavaScript можна викликати метод таким чином:

```
import { NativeModules } from 'react-native';
var CalendarManager = NativeModules.CalendarManager;
CalendarManager.addEvent(
  'Birthday Party',
  '4 Privet Drive, Surrey'
);
```

Приклад 5.2 – Клас календаря

Модуль CalendarManager екземпляром створюється на стороні Objective-C за допомогою виклику [CalendarManager new]. Тип повернення мостових методів завжди void. Мост React Native є асинхронним, тому єдиний спосіб передати результат у JavaScript - це використання зворотних викликів або випромінювання подій.

React Native пропонує також безліч віджетів власного інтерфейсу, готових до використання в найновіших програмах − деякі з них є частиною платформи, інші доступні як сторонні бібліотеки. У React Native є кілька найважливіших компонентів платформи, які вже обгорнуті, як ScrollView і TextInput, але не завжди ці компоненти відповідають нашим вимогам, на щастя React Native дозволяє розширити та імплементувати також компоненти інтерфейсу користувача.

Наприклад розширеня існуючого ImageView компонента для Android, доступного в основній бібліотеці React Native.

По-перше створюємо клас менеджера подань ReactImageManager, який

розширює SimpleViewManager типу ReactImageView. ReactImageView − це тип об’єкта, яким керує менеджер, це буде власне нативне відобра. Ім'я, яке повертає getName, використовується для посилання на власний тип відображення з JavaScript.

```
public class ReactImageManager extends SimpleViewManager<ReactImageView>
{
  public static final String REACT_CLASS = "RCTImageView";
  ReactApplicationContext mCallerContext;

  public ReactImageManager(ReactApplicationContext reactContext) {
    mCallerContext = reactContext;
  }

  @Override
  public String getName() {
    return REACT_CLASS;
  }
```

Приклад 5.3 – Підклас ViewManager

Представлення створюються методом createViewInstance, відображення має ініціалізуватися у стані за замовчуванням, будь-які властивості будуть встановлені за допомогою наступного виклику updateView.

```
@Override
public ReactImageView createViewInstance(ThemedReactContext context) {
    return new ReactImageView(context,
                              Fresco.newDraweeControllerBuilder(),
                              null,
                              mCallerContext);
}
```

Приклад 5.4 – Метод createViewInstance

Далі описуємо властивості відображення за допомогою анотації @ReactProp (або @ReactPropGroup).

```
      @ReactProp(name = "src")
        public void setSrc(ReactImageView view, @Nullable ReadableArray
sources) {
          view.setSource(sources);
        }
      @ReactProp(name = "borderRadius", defaultFloat = 0f)
        public void setBorderRadius(ReactImageView view, float borderRadius) {
          view.setBorderRadius(borderRadius);
        }
```

Приклад 5.5 – Властивості установників за допомогою @ReactProp

Останнім кроком Java є реєстрація ViewManager у програмі, це відбувається подібно до власних модулів за допомогою функції-члена пакета програм createViewManagers.

```
      @Override
        public List<ViewManager> createViewManagers(
                                  ReactApplicationContext reactContext) {
          return Arrays.<ViewManager>asList(
            new ReactImageManager(reactContext)
          );
        }
```

Приклад 5.6 – Регістрація ViewManager

Самим останнім кроком є створення модуля JavaScript, який визначає рівень інтерфейсу між Java та JavaScript для користувачів вашого нового подання.

```
      import { requireNativeComponent } from 'react-native';

      module.exports = requireNativeComponent('RCTImageView');
```

Приклад 5.7 – Імплементація JavaScript модуля

Надалі є можливість загрузити бібліотеку до npm модулів, та опублікувати код.

Власні нативні модулі у React Native надає можливість створювати одну кодову базу для багатьох платформ та дозволяє впровадити потрібну

рідну функцію що не підтримується у React Native за замовчуванням.

Це дає досить широкі можливості бізнесу, наприклад, можливість підтримувати специфічні платформи як Xbox, PS4, одразу розробляти рішення для веб та мобайл та впроваждувати, наприклад, OTT рішення для Roku, AppleTV, Amazon FireTV, Chromecast та інших розумних телевізорів.

Розробка React Native OTT стала популярним рішенням по всьому світу. Це швидко, надійно та гнучко. OTT рішення схоже на мобільний додаток, оскільки забезпечує передачу вмісту, наприклад потокового відео чи ігор, через Інтернет. Різниця полягає в тому, що програма OTT передає цей вміст на ваш телевізор, на відміну від вашого мобільного пристрою. Якщо у вас є розумний телевізор, ви використовували додатки OTT. Такі постачальники послуг, як кабельне, мовне та супутникове телебачення, традиційно виступали воротарями доставки контенту. Програми OTT - це спосіб обійти цю стару гвардію.

У React Native буда реалызована підтримка телевізійних пристроїв з наміром змусити існуючі програми React Native працювати на Apple TV і Android TV, з невеликими змінами коду JavaScript для додатків або без них.

Та наприклад використовуючи вже існуюче рішення – ReNative дозволяє створювати універсальні кросплатформні додатки з React Native і і включає останні ОС як iOS, tvOS, Android, Android TV, Web, Tizen TV, Tizen Watch, LG webOS, macOS / OSX, Windows, KaiOS, Firefox OS та Firefox TV платформи.

Також окрім власних нативних модулів ReNative підтримує стандартні плагіни, реалізовані спільнотою, які можна використовувати для покращення функціональності програм, підтримує інтеграцію різних служб та інфраструктур розгортання додатків. Але, саме написання власного коду, дає змогу доповнити будь-які непідтримувані специфічні для пристрою функції, які React Native не підтримує безпосередньо.

Одна з переваг використання саме React Native це те що він постачається з попередньо завантаженими кількома ключовими функціями,

котрі потрібні кожному додатку OTT. Він обробляє реєстрацію, вхід, потокове передавання відео та інші основні функції. Надійний набір функцій React Native може різко скоротити час і витрати на виведення на ринок програми OTT.

Тож основна перевага розробки саме React Native OTT полягає у тому, що це економить час і гроші. Якщо ваш проект не бачить цих переваг, ця структура може не відповідати вашому проекту.

Якщо у бізнесу вже є програма OTT, побудована власноруч або на іншій платформі то в такому випадку може мати сенс зберегти вже виконану роботу та додати до неї. Економія грошей та швидший вихід на ринок є найважливішими міркуваннями при плануванні розвитку проекту.

Коли програмі OTT потрібна функціональність із більш глибокими рівнями інтеграції пристроїв, слід задуматися про власну розробку. Наприклад, якщо ваш додаток повинен інтегруватися з власним 3D SDK для iOS або Android, і кожна платформа вимагає використання іншого механізму 3D, можливо, немає сенсу ділитися кодовою базою. У цьому випадку більшість файлів коду та проектів будуть сильно відрізнятися. Однак іноді ви можете додавати модулі для доповнення більшості інтеграцій пристроїв, які не підтримуються React Native.

Якщо ваш проект зосереджений на невеликій функції, специфічній для пристрою OTT, яку React Native не підтримує. У цьому випадку може бути більш економічно вигідним побудувати власну програму, а не створювати реагуючу нативну програму і необхідні користувацькі модулі для підтримки цієї однієї функції.

Нарешті, якщо планується випустити додаток на Roku, доведеться писати його, використовуючи власну мову сценаріїв Roku, Brightscript. Roku − це закрита система, і все, що створено для Roku, повинно бути написано її власною мовою сценаріїв - принаймні до тих пір, поки React Native чи інша система швидкого розвитку не додасть підтримку платформи Roku.

ВИСНОВКИ

В ході виконання робити був проведений аналіз предметної області, а саме, актуальності мобільних пристроїв та мобільних операційних систем.

Згідно якого у сучасному світі все більше веб-трафіку припадає на мобільні та планшети. Пошукові движки почали знижувати у видачі сайти, які не оптимізовані під смартфони. Тож розробка мобільних додатків нині є однією з найпопулярніших завдань в сфері інформаційних технологій. Зростає потреба в розробці якісного і відповідного сучасним тенденціям продукту.

Був проведений порівняльний аналіз існуючих методологій для розробки мобільних додатків, а також були виявлені їх основні недоліки та переваги.

Також був проведений огляд нативних, кросплатформних та веб рішень для розробки мобільних додатків.

Розробка нативних, гібридних та веб мобільних додатків має свої переваги й недоліки, які враховуються бізнесом і виконавцем при виборі технології. У числі найбільш значущих критеріїв - терміни і вартість розробки і супроводу, відповідність завданню, безпека і перспективність, рівень розвитку спільноти.

Було запропоноване рішення розробки конкурентоспроможного мобільного додатку та метод вибору технології розробки мобільного додатку оптимального для бізнесу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Стаття AIN.UA «Що змінилося з ростом мобільного трафіку і як бізнесу просуватися в нових умовах» [Електронний ресурс] – Режим доступу: www / URL: https://ain.ua/2018/08/13/mobilnyi-trafik-rulit – 30.11.2019 г. – Загол. з екрану.

2. Infopulse: Enterprise approach to mobile applications: Native vs. Web vs. Hybrid vs. Cross-plafform [Електронний ресурс] – Режим доступу: www / URL: https://www.infopulse.com/blog/enterprise-approach-to-mobile-applications-part-2-native-vs-web-vs-hybrid-vs-cross-platform – 30.11.2020 г. – Загол. з екрану.

3. Gerardus Blokdyk, Mobile «Application Development Platform A Complete Guide» [Текст] / Gerardus Blokdyk: Kobo, 2019. – 609 p.

4. Лучшие фреймворки для разработки кроссплатформенных мобильных приложений [Електронний ресурс] – Режим доступу: www / URL: https://blog.sibirix.ru/2020/08/25/crossplatform-frameworks/ – 30.11.2020 г. – Загол. з екрану.

5. Mobile and tablet internet usage exceeds desktop for first time worldwide [Електронний ресурс] – Режим доступу : www/ URL: http://gs.statcounter.com/press/mobile-and-tablet-internet-usage-exceeds-desktop-for-first-time-worldwide – 30.06.2020 г. – Загол. з екрану.

6. Medium: 7 Popular Cross-Platform App Development Tools That Will Rule in 2020 [Електронний ресурс] – Режим доступу: www / URL: https://medium.com/datadriveninvestor/7-popular-cross-platform-app-development-tools-that-will-rule-in-2020 – 30.11.2020 г. – Загол. з екрану.

7. Wikipedia: Progressive web application [Електронний ресурс] – Режим доступу: www / URL: https://en.wikipedia.org/wiki/Progressive_web_app – 30.11.2020 г. – Загол. з екрану.

8. Akshat Paul, Abhishek Nalwaya, React Native for Mobile Development

[Текст] / Akshat Paul, Abhishek Nalwaya: Apress, 2019. – 360 p.

9. John M. Wargo: Learning Progressive Web Apps [Текст] / John M. Wargo: Addison-Wesley Professional, 2020. – 200 p.

10. Progressive Web Apps [Електронний ресурс] – Режим доступу: www / URL: https://web.dev/progressive-web-apps – 30.11.2019 г. – Загол. з екрану.

11. React Native [Електронний ресурс] – Режим доступу: www / URL: https://reactnative.dev/docs – 30.11.2020 г. – Загол. з екрану.

12. Flutter [Електронний ресурс] – Режим доступу: www / URL: https://flutter.dev/docs – 30.11.2020 г. – Загол. з екрану.

13. Apache Cordova [Електронний ресурс] – Режим доступу: www / URL: https://cordova.apache.org/docs – 30.11.2020 г. – Загол. з екрану.

14. Ionic [Електронний ресурс] – Режим доступу: www / URL: https://ionicframework.com/docs – 30.11.2020 г. – Загол. з екрану.

15. Hermes, D. Building Xamarin.Forms Mobile Apps Using XAML [Текст] / D. Hermes, N. Mazloumi : Bookmetrix, 2019. – 456 p.

16. Xamarin University [Електронний ресурс] – Режим доступу: www / URL: https://university.xamarin.com/ – 30.11.2020 г – Загол. з екрану.

17. Karlsson, J. Xamarin.Forms Projects: Build seven real-world cross-platform mobile apps with C# and Xamarin.Forms, [Текст] / J. Karlsson, D. Hindrikes : Packt Publishing, 2018. – 416 p.

18. Flutter Vs. React Native: Let’s See Who the Winner [Електронний ресурс] – Режим доступу: www / URL: https://www.mindinventory.com/blog/flutter-vs-react-native – 30.11.2020 г. – Загол. з екрану.

19. Kotlin vs Java: Which is Better for Android App Development? [Електронний ресурс] – Режим доступу: www / URL: https://www.xenonstack.com/blog/kotlinandriod/#:~:text=Kotlin%20Java– 30.11.2020 г. – Загол. з екрану.

20. Habr: Нативная разработка, React Native и Flutter: критерии выбора [Електронний ресурс] – Режим доступу: www / URL:

https://habr.com/ru/company/simbirsoft/blog/460030/ – 30.11.2020 г. – Загол. з екрану.

21. Варианты кроссплатформенной разработки мобильных приложений [Електронний ресурс] – Режим доступу: www / URL: https://dou.ua/lenta/articles/cross-platform-mobile-development/ – 30.11.2020 г. – Загол. з екрану.

22. Finding a React Native OTT Dev Shop [Електронний ресурс] – Режим доступу: www / URL: https://www.studiolabs.com/finding-a-react-native-ott-dev-shop/ – 30.11.2020 г. – Загол. з екрану.

23. Кобзар, М. С., Єрьоміна Н. С. Методи розробки мобільних додатків [Текст] / Кобзар М. С., Єрьоміна Н. С. // Журнал «Проблеми інформатизації». – 2020. – 45 с.