Міністерство освіти і науки України

Український державний університет науки і технологій

Факультет Комп'ютерні технології та системи
Кафедра Комп'ютерні інформаційні технології

Пояснювальна записка
До кваліфікаційної роботи
магістра

на тему: «Аналіз швидкодійності відтворення інтерфейсу мобільних додатків на кросплатформених платформах Flutter та React Native»
за освітньою програмою інформаційні технології
зі спеціальності: 121 Інженерія програмного забезпечення.

Виконав: студент групи ПЗ2121М: /Валентин БЕРДАШЕНКО/

Керівник: /Олександр ІВАНОВ/

Нормоконтролер: /Світлана ВОЛКОВА/

Консультанти:
Економічний розділ /Микола ГНЕНИЙ/

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент

Дніпро – 2022 рік

Міністерство освіти і науки України

Український державний університет науки і технологій

Факультет Комп'ютерні технології та системи
Кафедра Комп'ютерні інформаційні технології

Пояснювальна записка
До кваліфікаційної роботи
магістра

на тему: «Аналіз швидкодійності відтворення інтерфейсу мобільних додатків на кросплатформених платформах Flutter та React Native»
за освітньою програмою інформаційні технології
зі спеціальності: 121 Інженерія програмного забезпечення.

Виконав: студент групи ПЗ2121М: /Валентин БЕРДАШЕНКО/

Керівник: /Олександр ІВАНОВ/

Нормоконтролер: /Світлана ВОЛКОВА/

Консультанти:
Економічний розділ /Микола ГНЕНИЙ/

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент

Дніпро – 2022 рік

Ministry of Education and Science of Ukraine

Ukrainian State University of Science and Technologies

Faculty Computer technologies and systems
Department Computer information technology

Explanatory Note
To Master's Thesis
master

on the topic: « Analysis of the rendering speed of the mobile application interface on the cross-platform Flutter and React Native platforms»
according to educational curriculum information technologies
in the Speciality: 121 software engineering.

Done by the student of the group ПЗ2121М: / Valentyn BERDASHENKO /

Scientific Supervisor: / Oleksandr IVANOV /

Normative controller: /Svitlana VOLKOVA/

Supervisors:
Economic part /Mykola GNENNIYN/

Dnipro – 2022

Міністерство освіти і науки України

Український державний університет науки і технологій

Факультет: Комп'ютерних технологій і систем
Кафедра: Комп’ютерні інформаційні технології
Рівень вищої освіти: магістр
Освітня програма: Інформаційні технології
Спеціальність: Інженерія програмного забезпечення

«ЗАТВЕРДЖУЮ»

Завідувач кафедри

_____ Вадим ГОРЯЧКІН

___ грудня 2022 р.

ЗАВДАННЯ

На кваліфікаційну роботу ОС Магістр

студента групи _____ _____
(номер групи) (ПІБ)

1 Тема дипломної роботи: аналіз швидкодійності відтворення інтерфейсу мобільних додатків на кросплатформених платформах Flutter та React Native
Керівник роботи:_____
затверджена наказом по університету від «11» лютого 2022 р. № 173ст.
2 Термін подання студентом закінченої роботи_____
3 Вихідні дані до дипломної роботи _____
_____
_____
_____
4 Зміст пояснювальної записки (перелік питань до розробки)
Теоретичні засади по швидкодійності відтворення інтерфейсу мобільних додатків на кросплатформених платформах. Опис програмного додатку. Етапи розробки та створення додатку. Дослідження області роботи та результати данного дослідження.
5 Перелік демонстраційного матеріалу
Пояснювальна записка;
Технічне завдання;
Робочий проект;
Вимоги до технічного забезпечення;
Функціонал;
Висновки.

6. Консультанти (з назвами розділів):

Розділ	Консультант	завдання видав	завдання прийняв
Техніко-економічні розрахунки			

КАЛЕНДАРНИЙ ПЛАН

№ пор.	Назва розділів дипломної роботи	Термін виконання розділів роботи	Примітка
1	Вступ		
2	Аналіз сучасного стану дослідження проблеми за науковими літературними джерелами	заполнить	від 70 джерел
3	Аналіз сучасного стану програмно-апаратного забезпечення, яке потребує вдосконалення для вирішення проблем дослідження	заполнить	
4	Постановка задачі, технічне завдання	заполнить	30%
5	Техніко-економічні показники	заполнить	
6	Розробка інструментальних засобів дослідження	заполнить	
7	Виконання досліджень	заполнить	60%
8	Оформлення тез доповідей	заполнить	
9	Оформлення статті у фаховий журнал	заполнить	
10	Подання кваліфікаційної роботи до кафедри	заполнить	
11	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	заполнить	100%

Студент _____ Валентин БЕРДАШЕНКО

Керівник роботи _____ Олександр ІВАНОВ

ЗМІСТ

СПИСОК СКОРОЧЕНЬ

React Native — це фреймворк JavaScript для написання справжніх мобільних додатків для iOS та Android. Він заснований на React, бібліотеці JavaScript від Facebook для створення користувальницьких інтерфейсів, але не для браузерів, а для мобільних платформ [1].

WebView — це вбудований веб-браузер, який вбудовані програми можуть використовувати для відображення веб-вмісту [2].

JSCore — це фреймворк, який надає механізм JavaScript для реалізації

WebKit і надавати цей тип сценаріїв в інших контекстах на macOS [3].

Власницьке програмне забезпечення - (від Proprietary Software англійською мовою) програмне забезпечення, на яке поширюються немайнові та майнові авторські права.

Фреймворк — це реальний або концептуальний фреймворк, призначений для використання як фреймворк або посібник для побудови чогось, таким чином розширюючи фреймворк у щось корисне.

API (Application Programming Interface) — це набір загальнодоступних методів і властивостей для взаємодії з іншими об’єктами в програмі.

ВСТУП

Сьогодні для успішного ведення бізнесу, орієнтованого на взаємодію з великою кількістю людей, необхідно створювати власні додатки або інтегруватися в існуючі системи, такі як соціальні мережі та конструктори сайтів. Створення сучасної програмної системи вимагає поширення серед найбільшої бази користувачів, включаючи веб-додатки, мобільні додатки для iOS та Android.

Підприємства зацікавлені у створенні та збереженні великої кількості постійних клієнтів. Тобто збереження високого рівня лояльності та надання якісного інструменту вирішення проблем клієнтів – користувачів електронних додатків.

Є конкуренція у сфері інформаційних технологій. Тому існують подібні платформи, які потрібно адаптувати до своїх конкретних стандартів. Тому виникають проблеми, пов'язані зі створенням і підтримкою окремих програм для багатьох платформ. По-перше, це впливає на вартість розробки. По-друге, це спричиняє великі витрати часу на спілкування між командами розробників і сумісність між додатками платформи. Ці фактори призводять до втрати гнучкості розробки та збільшення ціни внесення змін на будь-якому етапі.

Більшість сучасних платформ кросплатформенної розробки використовують компонент системи WebView для створення додатків. Але вони мають свої обмеження і часто приховують реалізацію чогось. Наприклад, створення остаточної збірки для спеціальної програми відбувається на віддаленому сервері, і розробник не має доступу до вихідного тексту програми. Підтримка невеликої кількості цільових платформ, створення додаткових модулів для взаємодії з системними сервісами вимагає специфічних знань цільової платформи.

Тому об’єктом дослідження даної роботи є фреймворк для створення кросплатформних користувальницьких додатків.

Тема дослідження полягає в тому, як інтегрувати «рідні» API

Мобільні платформи та архітектури.

Тому основними цілями цієї роботи є дослідження існуючих рішень для розробки кросплатформних користувальницьких додатків, розробка методу написання кросплатформенних користувальницьких додатків за допомогою WebView та розробка архітектури на основі модульного підходу проектування системи.

Тому, виходячи з цілей написання та розробки прикладного програмного забезпечення, були поставлені та вирішені наступні завдання:

– огляд і аналіз існуючих аналогів, особливо програмних рішень і відомих технологій, для виділення їх сильних і слабких сторін, на цій основі визначення вимог до розробки програм;

– стандартизація модулів, що взаємодіють із системними сервісами платформи;

– впровадження програмного забезпечення та результати, отримані в результаті тестування та аналізу.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ДЛЯ РОЗРОБКИ КРОСПЛАТФОМНИХ МОБІЛЬНИХ ДОДАТКІВ

1.1 Теоретичні основи

Розробка кросплатформних мобільних додатків передбачає створення вихідного коду додатка, який дозволяє працювати на різних платформах, включаючи iOS, Android тощо. Перевагою такого підходу є швидкість розробки технології та загального процесу розробки. Швидкість досягається за рахунок можливості повторного використання тексту програми. Підхід «напиши один раз, запусти де завгодно» дозволяє розробникам використовувати той самий текст програми на кількох платформах, що значно скорочує капітальні витрати та час розробки порівняно з розробкою власних програм. Можливість повторного використання тексту програми на кількох платформах дозволяє скоротити витрати на ресурси розробки на 50-80%.

Витрати часу також зменшуються, оскільки розробникам кросплатформних програм не потрібно вивчати кілька технологій перед створенням програми. Початкове розгортання на цільовій платформі відбувається набагато швидше, оскільки немає необхідності створювати іншу текстову бібліотеку програми. Крім того, майбутні зміни в програмі можна вносити одночасно, не вимагаючи окремих змін на кожній платформі. Крім того, немає проблеми синхронізації змін між різними платформами.

Такі можливості дозволяють швидко й легко охопити більше платформ і пристроїв, збільшуючи вашу цільову аудиторію. Але, як і будь-яка технологія, кросплатформенний підхід має свої недоліки.

Оскільки кожна платформа має свій власний API для взаємодії з системними службами, неможливо написати програмний текст, який працює скрізь. У тексті програми необхідно враховувати платформу, на якій зараз працює програма, і використовувати різні канали для взаємодії. Тому для технології часто

створюється екосистема, включаючи оболонки на кожній платформі та спеціальні середовища виконання для клієнтських програм. Такі оболонки стають проміжною ланкою між користувачами додатків і системними API, що може викликати проблеми з продуктивністю додатків і реакцією на дії користувача.

Слід також зазначити, що кожна платформа має власні специфічні компоненти інтерфейсу користувача, які можуть взаємодіяти по-різному. Проблема в тому, що потрібен додатковий час, щоб забезпечити поведінку користувача та зробити текст програми платформоорієнтованим. Це суперечить ідеї кросплатформного підходу до розробки спеціальних програм. З іншого боку, використання компонентів, незнайомих користувачам, може негативно вплинути на досвід користувача [4].

Для розробки кросплатформних програм можна використовувати різні методи. Наприклад, мова програмування C, веб-технології, C# (Xamarin) тощо. Для цієї роботи було вирішено проаналізувати рішення за допомогою веб-технологій. Тому що вони працюють практично на будь-якій платформі: десктоп, мобільний, планшет, автомобіль, холодильник тощо. Це досягається за допомогою спільних API на основі веб-стандартів, які є спільними для цих платформ. Далі в цьому розділі представлено кілька таких рішень.

1.2. Flutter

Flutter — це програмний SDK для створення iOS, Android і веб-додатків з єдиною кодовою базою. Мета полягає в тому, щоб дозволити розробникам створювати високопродуктивні програми, які природно почуваються на різних платформах. Мова розробки — Dart. Як і React Native, Flutter використовує адаптивні компоненти дисплея. Однак, якщо React Native перекладається на рідні

віджети, Flutter завжди збиратиме «рідний» текст програми. Flutter контролює кожен піксель на екрані, щоб уникнути проблем.

Рис. 1.1 – Flutter

Продуктивність через необхідність мосту JavaScript.

Dart має такі характеристики:

- надає Open Text Programs, розширювану мову програмування для створення веб-, серверних і мобільних додатків;
- надає об’єктно-орієнтовану мову з єдиним успадкуванням, яка використовує синтаксис у стилі C і AOT скомпільована в рідну мову;
- можна перекласти на JavaScript;
- підтримує інтерфейси та абстрактні класи.

В iOS більшість створення інтерфейсу користувача виконується за допомогою об’єктів перегляду, які є екземплярами класу UIView. Вони можуть діяти як контейнери для інших класів UIView, які складають макет. У Flutter

приблизним еквівалентом UIView є віджет. Віджети не зовсім точно представляють компоненти iOS, але коли ознайомитеся з тим, як працює Flutter, то можете думати про них як про «спосіб оголошення та створення інтерфейсів користувача». Але вони відрізняються від UIView кількома параметрами. По-перше, віджети мають інший життєвий цикл: вони незмінні й існують лише до тих пір, поки їх не потрібно змінити. Кожного разу, коли змінюється віджет або його стан, фреймворк Flutter створює нове дерево екземплярів віджетів. Навпаки, перегляди iOS не рендеряться, коли вони змінюються, а є змінними об’єктами, які малюються лише один раз і не перемальовуються, доки setNeedsDisplay() не визнає недійсним.

Крім того, на відміну від UIView, віджети Flutter є легкими, частково через їх незмінність. Тому що вони безпосередньо не переглядають і не малюють щось саме по собі, а опис інтерфейсу користувача та його семантику, які «роздуті» у фактичний об’єкт перегляду під капотом.

Flutter містить бібліотеку Material Components. Це віджети, які реалізують вказівки щодо матеріального дизайну. Material Design — це гнучка система дизайну, оптимізована для всіх платформ, включаючи iOS [10].

Flutter включає сучасну адаптивну структуру, 2D-візуалізацію, готові віджети та інструменти розробки. Ці компоненти працюють разом, щоб допомогти створювати, тестувати та налагоджувати програми. Все організовано за принципом: «Все — віджет».

Віджети є основними компонентами інтерфейсу користувача Flutter. Кожен віджет є незмінною частиною інтерфейсу користувача. На відміну від інших фреймворків, які розділяють подання, контролери подання, макети та інші властивості, Flutter має узгоджену однооб’єктну модель: віджети.

Віджет можна визначити як:

- структурні елементи (наприклад, кнопки або меню);
- стилістичні елементи (наприклад, шрифти або кольорові схеми);

– аспекти планування (наприклад, оббивка).

Віджети утворюють ієрархію на основі композиції. Кожен віджет розміщується всередині та успадковує властивості від свого батька. Немає окремого об'єкта "програма". Замість цього є кореневий віджет, який виконує цю роль [11].

1.3. React Native

React Native — це платформа JavaScript для написання мобільних додатків для iOS та Android. Він заснований на React, бібліотеці JavaScript від Facebook для створення користувальницьких інтерфейсів, але замість браузерів він націлений на мобільні платформи. Іншими словами: веб-розробники можуть писати мобільні додатки, які виглядають і відчуваються справді «рідними» — все в рамках бібліотеки JavaScript (React). Крім того, оскільки більша частина написаного коду може використовуватися між платформами, React Native спрощує розробку як для Android, так і для iOS.

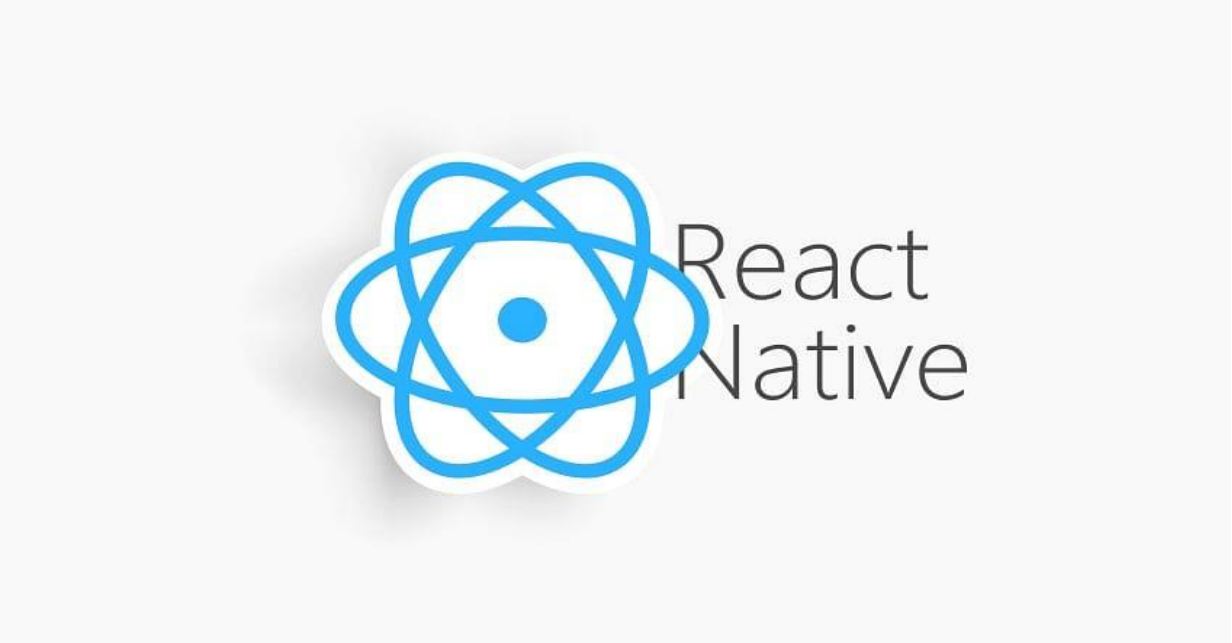

Рис. 1.2 – React Native

Подібно до React для Інтернету, програми React Native написані з використанням суміші розмітки JavaScript і XML під назвою JSX. Потім «міст» React Native викликає вбудований API візуалізації в Objective-C (для iOS) або Java (для Android). Таким чином програма відображатиметься за допомогою компонентів реального мобільного інтерфейсу користувача замість веб-переглядів і виглядатиме як будь-яка інша мобільна програма. React Native також надає інтерфейс JavaScript для API платформи, тому додаток React Native може отримати доступ до таких функцій платформи, як камера телефону або місцезнаходження користувача.

Наразі React Native підтримує iOS та Android і має потенціал для поширення на більшість майбутніх платформ.

Текст програми, написаний у React Native, буде кросплатформним. Такі компанії, як Facebook, Palantir і TaskRabbit, уже використовують його для розробки програм, орієнтованих на користувача.

Той факт, що React Native фактично виконує рендеринг за допомогою стандартного API рендерингу своєї хост-платформи, виділяє його серед більшості існуючих підходів до кросплатформенної розробки додатків, таких як Cordova або Ionic. Існуючі методи написання мобільних програм із використанням комбінації JavaScript, HTML і CSS зазвичай використовують візуалізацію веб-браузером. Хоча цей підхід працює, він має недоліки, особливо з точки зору ефективності. Крім того, вони часто не мають доступу до «рідних» елементів інтерфейсу хост-платформи. Коли ці фреймворки намагаються імітувати природні елементи інтерфейсу користувача, результати часто трохи «відчуваються»; зворотне проектування всіх тонкощів, таких як анімація, є величезною роботою, яка може швидко застаріти.

Навпаки, React Native фактично перетворює вашу розмітку на справжні власні елементи інтерфейсу користувача, використовуючи наявні інструменти візуалізації перегляду будь-якої платформи, яку використовуєте. Крім того,

React працює окремо від основного потоку інтерфейсу користувача, тому програма може залишатися продуктивною без шкоди для продуктивності. Цикл оновлення в React Native такий самий, як і в React: коли параметри або стан змінюються, React Native знову перемальовує інтерфейс. Основна відмінність між React Native і React у браузері полягає в тому, що React Native робить це за допомогою бібліотеки інтерфейсу користувача на своїй хост-платформі замість розмітки HTML і CSS.

Для розробників, які звикли працювати в Інтернеті з React, це означає, що вони можуть використовувати знайомі інструменти для створення мобільних додатків із продуктивністю та зовнішнім виглядом «рідних» додатків. React Native також означає вдосконалення звичайних мобільних пристроїв

Розвиток у двох інших сферах: досвід розробника та потенціал розвитку платформи [1].

Використання React Native може значно скоротити ресурси, необхідні для створення мобільних додатків. Будь-який розробник, який знає, як писати програми за допомогою React, тепер може використовувати той самий набір навичок для Інтернету, iOS та Android. Усунувши необхідність синхронізації розробників на різних цільових платформах, React Native дозволяє команді швидше вносити зміни та ефективніше ділитися знаннями та ресурсами.

Окрім здорового глузду, можна поділитися великою частиною програмного тексту. Не весь письмовий текст програми є кросплатформним, тому розробникам час від часу може знадобитися використовувати Objective-C або Java, залежно від функціональних можливостей, необхідних для певної платформи. Але за допомогою React Native дуже легко повторно використовувати текст програми на різних платформах. Наприклад, як описано на React Europe 2015, додаток Facebook Ads Manager для Android ділиться 87% своєї кодової бази з версією для iOS.

Як і у всіх речах, використання React Native не позбавлене недоліків, і чи буде використання React Native як технології дійсно ефективнішим, залежить від індивідуальної ситуації.

Найбільший ризик, ймовірно, полягає в зрілості React Native, оскільки проект відносно молодий. Підтримка iOS була випущена в березні 2015 року, а підтримка Android – у вересні 2015 року. Документацію, безумовно, можна вдосконалити, і вона все ще розвивається. Деякі функції в iOS і Android досі не підтримуються, і спільнота все ще шукає найкращі методи. Перевагою є те, що в переважній більшості випадків підтримку відсутніх API можна реалізувати незалежно.

1.4. Огляд програмних аналогів

PhoneGap народився в Nitobi Software влітку 2008 року. У жовтні 2012 року він став частиною Apache Software Foundation (ASF) під назвою Apache Cordova. Це робить PhoneGap постійно відкритим для інших розробників і ліцензованим під Apache версії 2.0.

PhoneGap має два переконання:

1. Проблеми між платформами можна вирішити за рахунок Інтернету.

2. Усі технології з часом знецінюються [5].

PhoneGap — це технологія контейнерних програм, яка дозволяє створювати нативні програми для мобільних пристроїв.

Інтерфейс користувача програми PhoneGap створений за допомогою HTML, CSS і JavaScript. Рівень інтерфейсу користувача програми PhoneGap — це WebView, який займає 100% ширини та 100% висоти пристрою. Він відтворює вміст HTML, не прикрашаючи вікно звичайного веб-браузера. Потрібно створити програму, щоб скористатися цим простором і розмістити елементи навігації/взаємодії/вмісту програми в інтерфейсі користувача на основі HTML і CSS.

WebView, який використовується PhoneGap, є тим самим WebView, що використовується у рідній ОС. На iOS це клас Objective-C UIWebView; на Android це Android і webwekit.WebView. Оскільки існують відмінності в механізмі візуалізації WebView між операційними системами, важливо переконатися, що реалізуєте свій інтерфейс з урахуванням цього.

PhoneGap надає інтерфейс прикладного програмування (API), який дозволяє отримати доступ до функціональної операційної системи за допомогою JavaScript. Логіка програми побудована за допомогою JavaScript, а PhoneGap API обслуговує зв'язок із рідною операційною системою. Також можете додати власні спеціальні «нативні плагіни» до функцій стандартного API PhoneGap за допомогою механізму зв'язку JavaScript-to-Native. Рідний плагін PhoneGap дозволяє писати власні класи та відповідні інтерфейси JavaScript для використання у вашій програмі.

Програми PhoneGap розробляються з використанням HTML, CSS і JavaScript, але кінцевим продуктом програми PhoneGap є бінарний архів програми, який можна поширювати через стандартну екосистему програми. Формат вихідного файлу — IPA (архів додатків iOS) для додатків iOS, APK (пакет Android) для додатків Android, файли XAP (пакет додатків) для Windows Phone тощо. Це ті самі формати додатків, які використовуються у «рідних» додатках, і їх можна поширювати через відповідні екосистеми (iTunes Store, Android Market, Amazon Market, BlackBerry App World, Windows Phone Marketplace тощо).

PhoneGap використовується для написання клієнтських програм. Зв'язок між клієнтами та серверами додатків може базуватися на стандартних запитах HTTP для вмісту HTML, службах RESTful XML, службах JSON або SOAP (або веб-сокетах, якщо це підтримується вашою операційною системою). Це ті самі методи, які використовуєте в настільних браузерах на основі AJAX [6].

Apache Cordova — проект з відкритим кодом для розробки мобільних додатків. Він дозволяє розробляти кросплатформенність за допомогою стандартних веб-технологій - HTML5, CSS3 і JavaScript. Програми працюють у специфічних для платформи оболонках і покладаються на стандарти зв'язування API для доступу до можливостей кожного пристрою, таких як датчики, дані, стан мережі тощо.

На рисунку 1.3 показана архітектура програми Cordova.

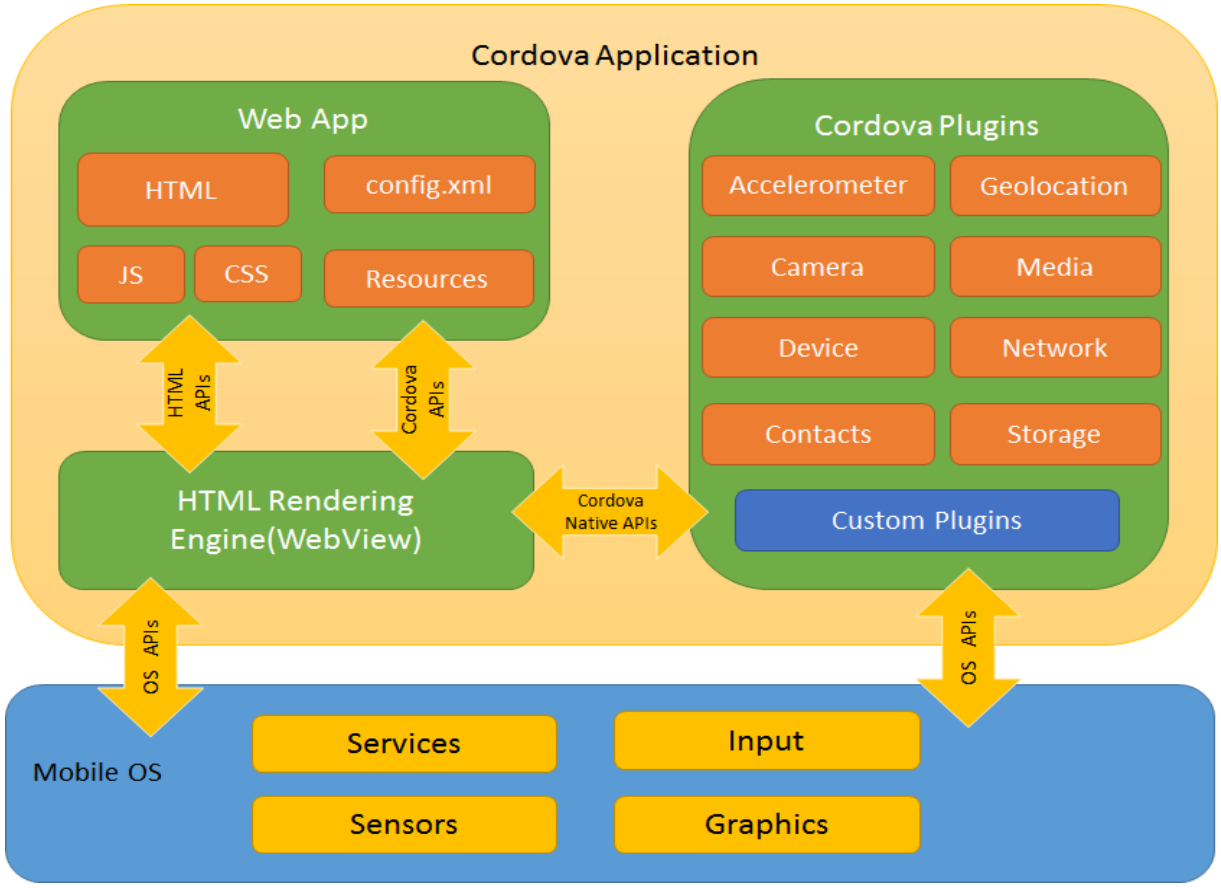

Рисунок 1.3. Архітектура Cordova

Перегляд мережі. WebView з підтримкою Cordova може надати програмі повний інтерфейс користувача. На деяких платформах він також може бути компонентом у більшій гібридній програмі, яка поєднує WebView з рідними компонентами програми.

Веб-додаток. Це розділ, де знаходиться ваша програма. Сама програма реалізована як веб-сторінка, за замовчуванням локальний файл під назвою index.html, на який посилаються CSS, JavaScript, зображення, медіафайли чи інші ресурси, необхідні для його запуску. Програма працює в WebView вашої рідної програми, яка розповсюджується в магазині програм.

Цей контейнер має один дуже важливий файл, файл config.xml, який надає інформацію про програму та визначає параметри, які впливають на її роботу, наприклад, чи реагує вона на зміни орієнтації.

Плагіни є невід'ємною частиною екосистеми Cordova. Вони забезпечують інтерфейс для Cordova та «власних» компонентів для зв'язку один з одним і прив'язки до стандартних API пристроїв. Це дозволяє викликати власний текст програми з JavaScript.

Проект Apache Cordova підтримує набір плагінів, які називаються основними плагінами. Ці основні плагіни надають вашому додатку доступ до таких функцій пристрою, як акумулятор, камера, контакти тощо.

На додаток до основного плагіна існує кілька сторонніх плагінів, які надають додаткові прив'язки для функцій, які не обов'язково доступні на всіх платформах [7].

Ionic Framework/Capacitor

Ionic було засновано в 2012 році, коли використання веб-технологій як засобу створення користувальницьких програм було ще в зародковому стані. Основна ідея полягає в тому, щоб створити кращий спосіб для веб-розробників використовувати свої наявні навички під час створення додатків для смартфонів.

Рис. 1.4 – Ionic Framework

Ionic Framework — це безкоштовний проект із відкритим вихідним кодом, випущений за дозвільною ліцензією MIT. Це означає, що його можна безкоштовно використовувати для особистих або комерційних проектів. MIT — це та сама ліцензія, яку використовують такі популярні проекти, як jQuery та Ruby on Rails.

Ionic Framework — це набір інструментів інтерфейсу програмування програм із відкритим кодом для створення мобільних і настільних додатків за допомогою веб-технологій (HTML, CSS і JavaScript).

Фреймворк Ionic фокусується на інтерфейсі користувача або взаємодії з інтерфейсом користувача програми (елементи керування, взаємодії, жести, анімація). Його можна легко інтегрувати з іншими бібліотеками, такими як Angular, або використовувати окремо.

Фреймворк Ionic наразі має офіційну інтеграцію з Angular і React, а підтримка Vue знаходиться в розробці [8].

Одним із найпоширеніших випадків використання Ionic є створення програми, яку можна завантажити з App Store і Play Store. Комплекти програмного забезпечення для iOS і Android (SDK) надають WebViews, які можуть відтворювати будь-яку програму Ionic і надають повний доступ до SDK.

Такі проекти, як «Capacitor» і «Cordova», часто використовуються для надання додаткам Ionic доступу до Native SDK. Це означає, що розробники можуть швидко створювати додатки, використовуючи звичайні інструменти веб-розробки, і при цьому мати доступ до власних функцій, таких як акселерометр пристрою, камера, GPS тощо [9].

Capacitor — це кросплатформна технологія виконання додатків, яка дозволяє створювати веб-додатки, які працюють на iOS, Android, Electron і в Інтернеті.

Capacitor надає послідовний веб-інтерфейс, орієнтований на колекцію, який дозволяє додаткам максимально наблизитися до веб-стандартів, отримуючи доступ до багатьох власних можливостей пристроїв на платформах, які їх підтримують. Власні функції можна легко додати за допомогою простих API плагінів для Swift на iOS, Java на Android і JavaScript для Інтернету. Capacitor є наступником Apache Cordova та Adobe PhoneGap, натхненний іншими популярними крос-платформними інструментами розробки, такими як React Native і Turbolinks, але повністю зосереджений на полегшенні роботи сучасних веб-додатків на всіх основних платформах. Capacitor забезпечує зворотну підтримку багатьох існуючих плагінів Cordova.

1.5. Висновки до розділу 1

У цьому розділі аналізуються існуючі рішення, розроблені на різних платформах. Отримана інформація систематизована для використання в подальшій роботі щодо визначення вимог до розробки програмного забезпечення, що дозволить створювати кросплатформні користувальницькі додатки.

2. ТЕХНІЧНЕ ПОРІВНЯННЯ ЗАСОБІВ РОЗРОБКИ

2.1. Постановка задачі

Данна робота присвячена вивченню технік Flutter та React Native для створення мобільних додатків.

Акцент робиться на вивченні нових технологій і вивченні сильних і слабких сторін відомих аналогів.

Мета цієї роботи — ознайомитися з мовою програмування Dart та JavaScript і фреймворком Flutter та ReactNative, отримати нові навички та застосувати.

Для досягнення поставленої мети були розроблені наступні завдання:

1. Ознайомитеся з документацією та освоїти основи Flutter і Dart та React Native і JaveScript.

2. Дослідження переваг і недоліків.

3. Набуття практичних навичок.

4. Створити мобільні додатки на різних фреймворках та порівняти їх швидкодійності.

2.2. Теоретичне порівняння Flutter та React Native

Метою системи розробки є створення додатків для платформ з використанням єдиної кодової бази. Це веб-платформи, iOS і Android. Необхідно створити окремі програму на різних мовах Flutter та React.

Більшість розробок припадає на веб-додатки, адже вони містить інтерфейс користувача, бізнес-логіку та зв’язок із віддаленими серверами.

Веб-додатки для системи збірки розробляються так само, як і звичайні веб-додатки для браузера. Але візьміть до уваги, що доступ до всіх API браузера повинен проходити через «плагіни». Оскільки API браузера є частиною середовища виконання програми, він не буде доступний на інших платформах.

Існує два способи запуску JavaScript у браузері. Спочатку завантажте окремі файли .js для різних функцій. Це рішення важко масштабувати, оскільки завантаження занадто великої кількості файлів .js може спричинити вузькі місця. Другий варіант полягає у використанні одного великого файлу .js, який містить увесь процедурний текст проекту. Але це призводить до проблем із обсягом, розміром, читабельністю та зручністю обслуговування [19].

Перевагою більшості рішень є повторне використання тексту програми для запуску програми на різних платформах. Також однією з переваг є мова розробки – JavaScript, оскільки вона проста у вивченні та найпопулярніша серед розробників станом на жовтень 2019 року [12]. Недоліком більшості розглянутих рішень є потреба в обгортках, які працюють всередині екосистеми та важко розширюються. Більш детальні результати за критеріями порівняння наведені в таблиці. 2.1.

Таблиця 2.1. – Порівняння існуючих рішень

Критерій	Flutter	React Native
Мова програмування	Dart	JavaScript
Рік розробки платформи	2017	2015
Платформи, що підтримуються	iOS, Web, Android	iOS, Android
Складність написання «власних» плагінів	Середнє	Лекго
Рівень покриття «рідного» API плагінами	Низький	Високий
Повторне використання коду програми	На всіх платформах	Наявний специфічний для платформи код програми
Середовище виконання		JS Thread(JS Core)

Цілі та вимоги до програмних продуктів, розроблених у цій роботі, сформовані на основі вивчення кросплатформної розробки та порівняння існуючих рішень зі стандартами.

Метою даної роботи є розробка додатків.

Для написання кросплатформних користувальницьких програм і розробки.

2.3. Вимоги до технічного забезпечення

ОС і середовище швидко змінювалися з останніх 30 років, що робить речі застарілими, перш ніж вони зможуть стати продуктивними протягом свого життя. Наприклад, Microsoft випустила понад 20 основних випусків ОС за останні 25 років. Кожна мова та середовище випускають новий випуск кожні шість місяців. Коли ОС або її середовище змінюються, це має ефект пульсації, що призводить до застарілих програм. Не забезпечуючи жодної перевірки, одним із можливих підходів є уникнення операційної системи та її середовища. У деяких областях обчислень це робиться повільно, непомітно.

Після завантаження ПК він перебуває в реальному режимі, де адресація обмежена 1 МБ і немає захисту вашого коду. Це просто в режимі дискової операційної системи (DOS). Щоб завантажувати та запускати більші програми та мати доступ до більшої пам’яті, потрібно навчитися переходити в захищений режим за допомогою інструкцій на мові асемблера. Однак усі виклики базової системи введення/виводу (BIOS) доступні лише в реальному режимі. Якщо хочете використовувати BIOS для вводу-виводу, потрібно мати механізми перемикання із захищеного режиму в реальний для виконання викликів BIOS. Таким чином, ваша програма або якась керуюча програма повинні виконувати це перемикання, прозоре для користувача. Якщо плануєте використовувати програмні переривання (їх 255) замість переривань BIOS, то потрібно написати власний код на зборці для вирішення всіх операцій введення-виводу. В розробці було використано як BIOS, так і програмні переривання і написали перемикач

захищеного режиму в режим реального для роботи в обох режимах. Це нетривіальна річ, яку потрібно зробити під час складання, яка вимагає від вас ознайомлення з документом специфікації архітектури Intel, який доступний на їх веб-сайті. Цей документ є корисним ресурсом для розуміння та реалізації переривань, схем адресації, засобів завдань, пасток і винятків.

Тестування проводиться зазвичай у кількох етапах життєвого циклу, коли розробляється програмне забезпечення. Технологія тестування програмного забезпечення містить такі етапи:

- визначення функціоналу, що підлягає і не підлягає тестуванню;
- формулювання підходів, що використовуються для даного продукту;
- написання тест кейсів;
- розробка критерію проходження тестів;
- визначення вимог середовища проведення тестування;
- проведення тестування та оцінка результатів;
- звітність результатів.

Системні вимоги:

- операційна система Windows 7 чи наступних версій;
- Android
- iOS

Апаратні вимоги:

- Досить вільної пам'яті комп'ютера для установки програми;
- 32 або 64-розрядний процесор із тактовою частотою 1GHz;
- Оперативна пам'ять не менше 1Gb.

Тестування проводилося на ноутбуці компанії Lenovo із процесором Intel Core i5-6060U, тактовою частотою 2GHz, операційною системою Windows 10.

2.4. Висновки до розділу 2

У цьому розділі детально обговорюються методи використання Flutter та React Native як середовища для виконання нестандартних програм на мобільних пристроях. Крім того, система розроблена таким чином, що програму також можна запускати у веб-браузері настільного комп’ютера. Також ідеться порівняння Flutter та React Native.

Крім того, усвідомлення веб-середовища як іншої цільової платформи можна вважати модифікацією. Зрештою, взаємодія з орієнтованою на платформу частиною тексту програми відбувається в тому ж процесі, а не через вторинні канали зв’язку. І ця взаємодія відрізняється від стандартного підходу.

3. ЗАСОБИ ДЛЯ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ

Ця частина роботи присвячена детальному опису розроблених програмних модулів. Розроблена система має такі основні компоненти:

- веб-додатки;
- мобільні додатки;
- компонент WebView в React Native;
- канал зв’язку між веб-сторінками та React Native;
- плагіни;
- менеджер плагінів.

3.1. Платформо-орієнтовані додатки

Метою системи розробки є створення додатків для трьох платформ з використанням єдиної кодової бази. Це веб-платформи, iOS і Android. Створіть окрему програму для кожної платформи.

Більшість розробок припадає на веб-додатки. Адже він містить інтерфейс користувача, бізнес-логіку та зв’язок із віддаленими серверами.

Веб-програми та Webpack

Веб-додатки для системи збірки розробляються так само, як і звичайні веб-додатки для браузера. Але візьміть до уваги, що доступ до всіх API браузера повинен проходити через «плагіни». Оскільки API браузера є частиною середовища виконання програми, він не буде доступний на інших платформах.

Існує два способи запуску JavaScript у браузері. Спочатку завантажте окремі файли .js для різних функцій. Це рішення важко масштабувати, оскільки завантаження занадто великої кількості файлів .js може спричинити вузькі місця. Другий варіант полягає у використанні одного великого файлу .js, який містить

увесь процедурний текст проекту. Але це призводить до проблем із обсягом, розміром, читабельністю та зручністю обслуговування [19].

Ця проблема вирішується за допомогою статичних модулів, які перетворюють дерево взаємозалежних класів в один файл. Система Webpack для розробки.

У нашій розробці використовуються додаткові плагіни, які наведені в таблиці. 3.1.

Таблиця 3.1. – Застосування плагінів Webpack

Назва плагіну	Що робить
HotModule ReplacementPlugin	Модулі замінюються, додаються або видаляються під час роботи програми, що не вимагає перезавантаження всієї сторінки. використовується під час розробки.
clean-webpack-plugin	Очищує каталог збірки перед наступною збіркою.
optimize-css-assets- webpack-plugin	Шукає ресурси CSS і оптимізуйте/зменште їх під час компіляції webpack.
terser-webpack-plugin	Згортає текст програми JavaScript для збирання.
copy-webpack-plugin	Копіює окремі файли або цілі каталоги до каталогу збірки.
moment-locales- webpack-plugin	Видаляє невикористані локалізації Moment.js під час створення.
webpack-bundle- analyzer	Візуалізує розмір вихідних файлів збірки за допомогою інтерактивної шкали.
NamedModulesPlugin	Відображає відносні шляхи до модулів під час розробки.
DefinePlugin	Дозволяє створити глобальну константу, яку можна налаштувати під час компіляції.

Webpack — це постачальник статичних модулів для сучасних програм JavaScript. Коли webpack обробляє програму, він внутрішньо будує графік залежностей, який відображає кожен модуль, потрібний для проекту, і генерує один або кілька вихідних комплектів - файлів .js.

Webpack — це інструмент, який дозволяє об’єднувати програми JavaScript. Його можна розширити для підтримки багатьох різних активів, таких як: зображення, шрифти та таблиці стилів [19].

Крім того, плагіни CommonsChunkPlugin або SplitChunksPlugin часто використовуються для оптимізації швидкості завантаження сторінки. Але в нашому випадку їх використання недоцільно. Причина описана нижче.

Мобільні програми та WebView

Що стосується мобільних додатків для iOS і Android, то отримимані «рідні» додатки, розміщені на серверах розповсюджувачів додатків для смартфонів: AppStore і PlayMarket.

Мобільна програма працює у своєму рідному середовищі виконання (Objective-C/Java) та інтерпретує текст програми, який знаходиться на стороні React Native і написаний на JavaScript. На стороні ReactNative запит на відкриття WebView надсилається до «рідної» частини програми. За допомогою цього запиту у веб-браузер завантажується програмний текст HTML сторінки, яка використовується для створення дерева елементів, і текст програми JavaScript скомпільованої веб-програми, яка відображає інтерфейс програми користувача та виконує бізнес-логіку.

Вимога API для компонента WebView полягає в тому, що текст програми HTML і JavaScript завантажується як рядок. Ось чому для компіляції веб-програми не використовується плагін.

3.2. Канали комунікації

Канали зв'язку реалізовані на основі архітектури клієнт-сервер. Клієнти та сервери в цій системі реалізовані за допомогою бібліотеки mole-rpc. Він забезпечує інтерфейс, незалежний від транспорту та його реалізації. Крім того, сервер може мати кілька переказів одночасно. Повідомлення надсилаються у форматі JSON RPC.

Транспорт, розроблений для цієї системи, базується на генераторі подій onmessage, вбудованому в браузер. Для цього використовуються наступні інтерфейси: вікно глобальної змінної в середовищі WebView та його метод onmessage та компонент із React Native <WebView> , посилання на нього та його властивість onMessage.

Також слід зазначити, що для обробки потоків подій застосовано шаблон дизайну Pub/Sub. Де вікно глобальних змінних і компоненти <WebView> діють як видавці, а екземпляри сервера з обох сторін є передплатниками.

Повідомлення надсилаються у форматі JSON RPC. Опис полів запиту та відповіді наведено в таблиці. 3.2.

Таблиця 3.2. – Опис поля запиту JSON-RPC

Поле	Значення
Запит	
jsonrpc	версія JSON RPC
id	ID запиту
method	Один із способів реєстрації на сервері за допомогою методу expose
params	Параметри для виконання операцій на сервері
Поле запиту params[1]	
pluginId	ID плагіна запиту, який виконується
method	Один із доступних методів плагіна

args	Параметри для виконання запитів плагіна
Відповідь	
jsonrpc	версія JSON RPC
id	ID запиту
result	дані результату

3.2.1. проміжне програмне забезпечення

За допомогою проміжного програмного забезпечення, яке є основним компонентом каналу зв'язку, клієнт і сервер можуть обмінюватися повідомленнями. Для зв’язку між веб-сторіною та стороною React Native створено спеціальне проміжне програмне забезпечення для інтеграції з бібліотекою mole-rpc. І щоб інші розробники могли використовувати це проміжне програмне забезпечення, воно перетворюється на бібліотеку та надсилається до NPM – реєстру програмного забезпечення.

На рисунку 3.1 показано клас проміжного ПЗ. Відповідно до вимог API бібліотеки mole-rpc, проміжне програмне забезпечення реалізує два методи: onData і sendData. Перший використовується для сповіщення об’єкта «Сервер» або «Клієнт» про отримання повідомлення. Другий - для генерації повідомлення клієнтом.

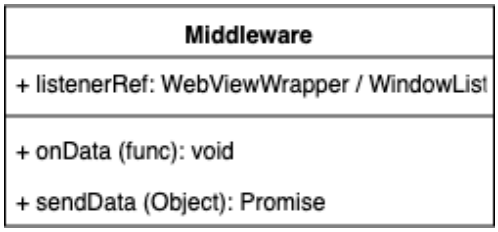

Рисунок 3.1. Клас проміжного ПЗ

Крім того, клас проміжного програмного забезпечення має поле listenerRef, яке зберігає посилання на один із вбудованих генераторів подій. Активно

використовуйте його в методах onData і sendData для пересилання відповідного типу повідомлення.

WebViewWrapper і WindowListener

Щоб дозволити іншим розробникам використовувати транспорт mole-rpc, додайте компонент WebViewWrapper і клас WindowListener до бібліотеки вище.

На рисунку 3.2 схематично ілюструє поля та методи доповнення та надання вбудованих структур.

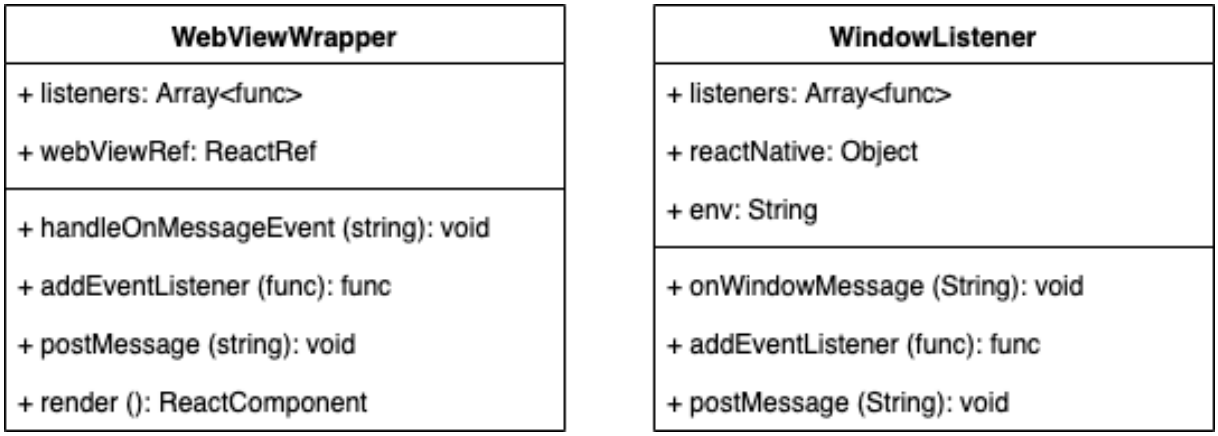

Рисунок 3.2 Компонент WebViewWrapper і клас WindowListener

Створені структури схожі, але націлені на різні платформи та середовища виконання, тому вони мають свої особливості.

Обидва класи мають поле слухачів. Він містить набір функцій зворотного виклику, які викликаються, коли надходить повідомлення. Викличте слухача в методах класу handleOnMessageEvent і onWindowMessage. Ці методи також є функціями зворотного виклику, але реєструються у вбудованому генераторі подій платформи. Параметр, який передається під час виклику функції слухача, — це рядок повідомлення.

Ви можете додати слухача, викликавши метод addEventListener і передавши функцію зворотного виклику як перший параметр. Метод addEventListener повертає функцію, яка видаляє функцію зворотного виклику з масиву слухачів.

За функціональність отримання повідомлень відповідають описані вище структурні компоненти, а для їх відправки використовуються поля webViewRef / reactNative і метод postMessage. Поля webViewRef і reactNative відповідно містять посилання на об’єкт генератора глобальних подій кожної платформи. Метод postMessage захищає той самий метод генератора глобальних подій ззовні.

Клас каналу зв'язку

Щоб полегшити використання серверів і клієнтів у mole-rpc і створених транспортах, створюється клас, який об’єднує ці три структурні компоненти системи – канали зв’язку. на рис. На малюнку 12 показано поля та методи класу каналу зв’язку кожної сторони.

Як описано в розділі 2.4 цієї роботи, кожна сторона має бути і сервером, і клієнтом. Тому категорія каналу зв'язку містить посилання на відповідний екземпляр структури. Крім того, кожен клас містить метод ініціалізації, який зареєстровано в розробленому транспорті класу MoleServer і генераторах подій глобального слухача.

Кожен тип каналу зв'язку також має метод виконання. Цей метод генерує повідомлення для надсилання контрагенту. Тобто метод callMethod класу MoleClient викликається всередині цього методу. Першим параметром є рядок «execute» — ідентифікатор методу, зареєстрованого в MoleServer. Другий параметр - об'єкт, який буде розміщено в полях запиту - params.

Якщо показати разом усі компоненти каналу зв’язку в системі розробки, то отримаємо діаграму класів, показану на рисунку 3.3.

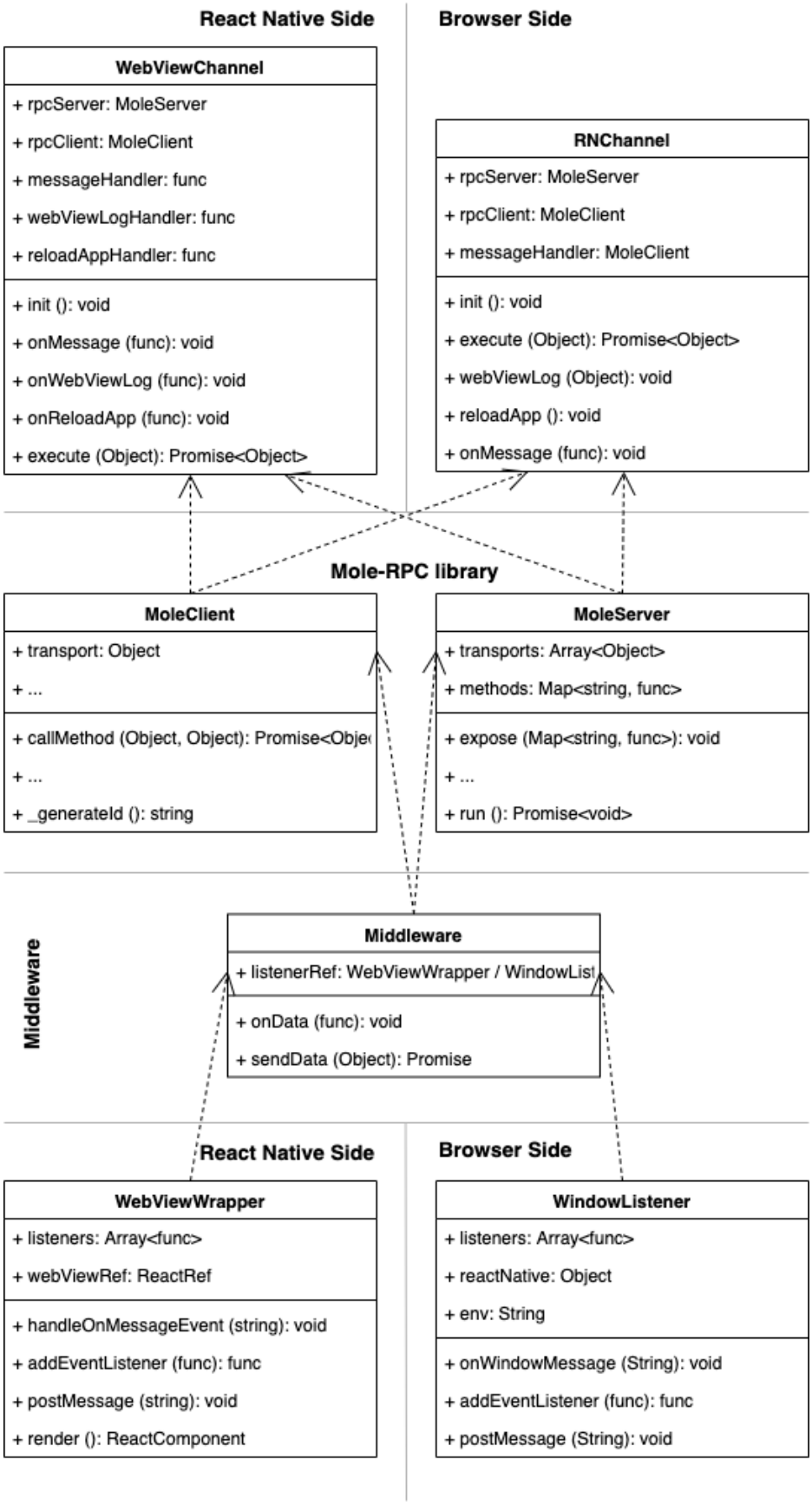

Рисунок 3.3. Діаграма класів каналу зв'язку

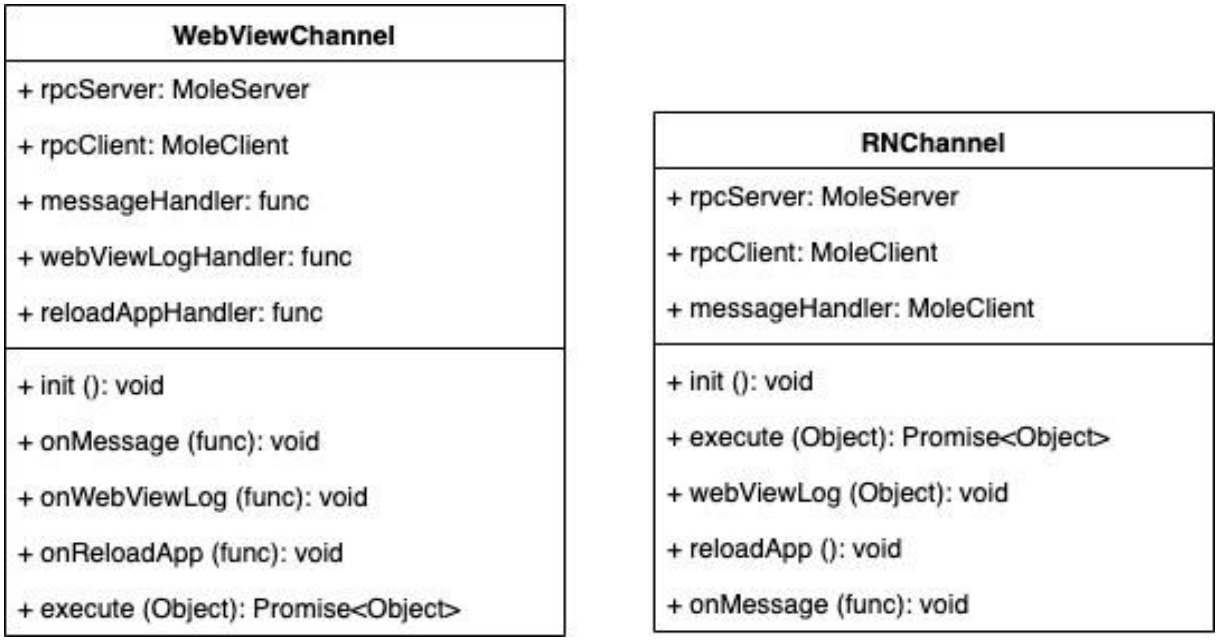

Рисунок 3.4. Клас каналу зв’язку між React Native і веб-сторіною

Інший такий самий метод — onMessage. Цей метод виконує роль обробника налаштування для подій типу "виконання", зареєстрованих на сервері. Функція зворотного виклику, передана як перший параметр, буде записана в поле messageHandler і викликана, коли відповідне повідомлення прийде на сервер. Методи onWebViewLog і onReloadApp, а також поля webViewLogHandler і reloadAppHandler служать тій же меті. Вони відповідають за обробку подій із серверів "webViewLog" і "reloadApp" відповідно.

У клієнтському каналі на стороні браузера реалізовані методи генерації подій типу "webViewLog" і "reloadApp" - webViewLog і reloadApp відповідно. Вони працюють подібно до методів виконання цих класів.

3.3. Плагіни

Плагіни в системі діють як архітектурні компоненти для конкретного середовища виконання платформи. Це тому, що WebView не має повного API

звичайних браузерів [13]. Наприклад, WebView не може отримати доступ до API localStore або browserHistory, які є частиною будь-якої веб-програми. З іншого боку, мобільні програми мають більше можливостей використовувати API операційних систем смартфонів, ніж браузери. Наприклад, доступ до списку контактів, камери, галереї, відомостей про підключення до Інтернету, файлової системи тощо [13].

Тому для кожного середовища виконання програми необхідно написати окремий плагін. За допомогою диспетчера плагінів можна отримати доступ до нього з будь-якої точки програми, незалежно від контексту, в якому наразі виконується текст програми.

Плагіни бувають двох типів: утиліти та компоненти. Службові програми — це класи для роботи з API, такими як локальне сховище або інформація про стан підключення смартфона до Інтернету. Компоненти — це плагіни, які мають графічне представлення. Наприклад, камера, модальне вікно для вибору дати та часу, відеоплеєр тощо.

Обов'язкові властивості

Обидва типи плагінів повинні містити статичне поле pluginOptions, яке відображає тип плагіна та його унікальний ідентифікатор. Крім того, кожен плагін отримує функцію зворотного виклику onInitEnd. У випадку плагінів утиліт ця функція буде передана класу в конструкторі через поле властивостей для плагінів компонентів. Функцію onInitEnd необхідно викликати після ініціалізації плагіна. Його метою є реєстрація загальнодоступних методів плагінів у менеджері плагінів для зовнішнього виклику.

У приведеному нижче коді містить шаблони для створення обох типів плагінів.

```
function ComponentPlugin(props) {
    const { onInitEnd }
= props;
```

```
    const _method1 = ()
=> {
        // some stuff
    };

    render() {
        return (
            // component visualization
        );
    }
}
ComponentPlugin.propType s = {
    onInitEnd : PropTypes.func.isRequire d
};

ComponentPlugin.pluginOp tions = {
class UtilPlugin { constructor({ onInitEnd }) {

onInitEnd(UtilPlugin.pluginOptio ns.id, {
            publicMethodName : this._method1
        });
    }

    static pluginOptions = {
        id  : 'plugin-util-id', type : 'util'
    };

    _method1 = () => {
        // some stuff
}
}

 export default UtilPlugin;

    id :
'componeent-
plugin-id',
    type : 'component'
};

export default
ComponentPlugin;
```

Внутрішній стан плагіна

Слід також виділити спеціально розроблені допоміжні функції, які надають плагінам можливість маніпулювання станом. Вони призначені для стандартизації сповіщень користувальницьких програм, які знаходяться в

WebView, про зміни у внутрішньому стані плагіна або API платформи, які взаємодіють із цим плагіном. Наприклад, сповіщення про зміну типу підключення до Інтернету з WiFi на мобільний Інтернет ініціюється операційною системою смартфона.

Створення на основі аналогічної функціональності компонентів класу

Компоненти від React.

У зв'язку із закриттям реалізовано співпрацю з державою. Тому маніпуляції станом можна досягти лише за допомогою спеціальних функцій, які повертаються викликами допоміжних функцій.

Основним завданням розробленої допоміжної функції є маніпулювання внутрішнім станом і надсилання повідомлень про зміни стороні веб-додатку. Такі повідомлення мають чітко визначені поля для структур, описаних у таблиці. 3.3.

Таблиця 3.3. – Поле сповіщень про зміни внутрішнього стану плагіна

Поле	Значення
pluginId	system
Method	updateState
Args	{ pluginId, prevState, newState }

Розроблена допоміжна функція має наступну сигнатуру. В якості вхідних даних для параметра вони приймають змінну типу Object з полями: id і state, де перше поле — це унікальний ідентифікатор плагіна, а друге — початковий внутрішній стан. Наступні допоміжні функції повертають екземпляри типу Object, які містять посилання на такі функції: getState, updateState, resetState.

Функція getState не приймає аргументів і повертає поточне значення внутрішнього стану плагіна.

Функція updateState приймає як перший параметр змінну типу Object, яка містить карту полів стану, які потрібно змінити. ключ — це ім'я поля, а значення

— це значення, яке потрібно встановити у відповідному внутрішньому полі стану. Під час цього процесу на веб-сторіну буде надіслано сповіщення про оновлення внутрішнього стану плагіна.

Функція resetState скидає внутрішнє значення стану до початкового значення та сповіщає систему про відповідні зміни в плагінах.

Реєстрація плагіна

Плагіни реєструються в менеджері плагінів під час запуску програми. Реєстрація плагінів утиліт відбувається під час запуску менеджера плагінів. Зрештою, йому просто потрібно створити екземпляр класу. Що стосується плагінів-компонентів, слід зазначити, що процес реєстрації є більш складним. Це пояснюється тим, що такі плагіни мають графічне представлення, тому їх потрібно додати до дерева інтерфейсу DOM (Document Object Model).

Ця проблема вирішується за допомогою функції React HOC (компоненти вищого порядку).

HOC - injectPlugins був створений для розроблених систем. Його мета — відобразити вміст сторінки, переданий параметрами, і відобразити абсолютне розташування плагінів-компонентів на одному рівні. Цей помічник отримує доступ до класу менеджера плагінів і отримує від нього список компонентів для відображення.

3.4. Менеджер плагінів

Менеджер плагінів у нашій системі — це клас, який містить реєстр усіх доступних плагінів і функцій для додавання, видалення та доступу до них. Екземпляри цього класу створюються як на стороні React Native, так і на стороні системи веб-розробки. Крім того, слід зазначити, що з усіма плагінами, доступними для «рідних» платформ, також має сенс скопіювати їх у Інтернет. Це

робиться як частина текстової сумісності програм, що працюють у всіх трьох середовищах. Наприклад, при створенні плагіна, який відповідає за визначення географічного розташування користувача програми, було надано різні API для трьох платформ. У випадку, коли впроваджуємо плагін, який відповідає за моніторинг стану інтернет-з'єднання, не маємо відповідного API у браузері, але можливо отримати доступ до відповідного плагіна в програмі користувача. Тому має сенс створити плагін із незмінними даними за замовчуванням.

Клас менеджера плагінів існує на обох кінцях системи розробки, але його реалізація має одну важливу відмінність. Менеджер плагінів на веб-сторінці визначає пріоритет плагінів із боку платформи. Це дозволяє створювати плагіни на веб-сторіні з тим самим унікальним ідентифікатором, що й на платформі. І якщо текст програми користувача виконується на платформі смартфона, веб-менеджер плагінів перевизначає зв'язок ідентичності від екземпляра веб-плагіна до екземпляра, орієнтованого на платформу.

На рисунку 3.5 показано поля та методи класу менеджера плагінів.

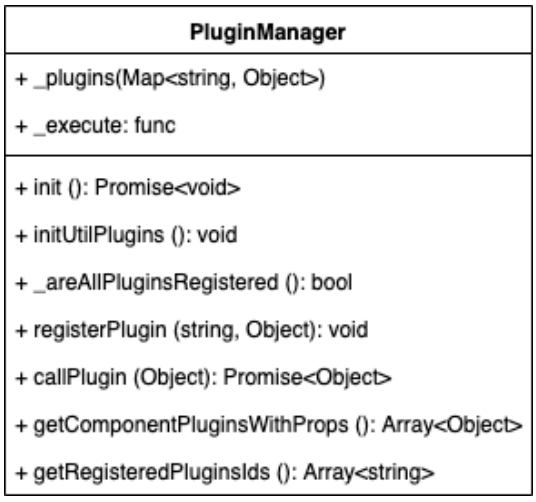

Рисунок 3.5 Клас PluginManager

Він містить поле _plugins. Це карта, ключі якої є унікальними ідентифікаторами для доступних плагінів, а значеннями є об'єкти, що

складаються з двох полів: стан і метод. де статус може приймати одне з таких значень: В ОЧІКУВАННІ, ІНІЦІАЛІЗОВАНО. А методи — це карта з ключами — іменами методів, доступних у плагіні, і значеннями — посиланням на функцію для виклику.

Поле _execute містить посилання на метод виконання класу зв’язку, описаного в розділі 3.2.3.

Метод init внутрішньо викликає метод initUtilPlugins, який створює екземпляр класу для службових плагінів і очікує ініціалізації всіх оголошених плагінів. Сигнал запуску плагіна — це функція зворотного виклику, яка передається екземпляру плагіна разом із властивістю під назвою onInitEnd. Ця функція зворотного виклику є методом registerPlugin менеджера плагінів.

Оскільки клас диспетчера плагінів по суті є реєстром плагінів, взаємодія з плагіном здійснюється через метод його інтерфейсу — callMethod.

Цей метод перевіряє реєстр на наявність плагіна, з яким він має намір взаємодіяти, і чи існує метод, зареєстрований для цього плагіна. Якщо необхідні дані відсутні, метод callMethod генерує відповідну помилку. В іншому випадку він очікує на результат виклику відповідного плагіна та повертає результат.

3.5. Висновки до розділу 3

Тому було розроблено вдосконалений підхід до розробки мобільних додатків за допомогою WebView шляхом створення та використання таких структурних компонентів:

– Компонент відображення інтерфейсу користувача - WebView, переважно для всіх сучасних платформ, включаючи iOS та Android.

– Канал зв’язку між React Native і WebView на основі архітектури клієнт-сервер і механізму глобального генератора подій, вбудованого в браузер. Канал зв’язку має кілька компонентів, таких як клієнт, сервер, проміжне

програмне забезпечення, клас-оболонка для генераторів глобальних подій з обох сторін і клас, який об’єднує всі компоненти.

– Плагіни надають доступ до специфічних для платформи API. Вони забезпечують однаковий інтерфейс незалежно від того, на якій платформі зараз працює програма. Таким чином, розробнику не потрібно перевіряти поточний тип середовища виконання на кожному кроці.

– Менеджер плагінів, який обробляє різні типи плагінів (утиліти та компоненти), зберігає їх у реєстрі та надає до них доступ.

Завдяки своїй взаємодії веб-додатки можуть використовувати API платформи, надані операційною системою смартфона, як якщо б вони були «рідними» мобільними додатками. Крім того, не втрачається сумісність із веб-середовищем.

4. РОЗРОБКА ПРОГРАМ

4.1. Ідея розробки додатку

Після роботи над модифікованим підходом до розробки мобільних додатків за допомогою WebView проаналізовано, чи відповідає він вимогам користувача, описаним у першому висновку

У першому розділі аналізується якість програмного тексту та вимірюється час, необхідний для запуску програми, з точки зору кількості підключених плагінів.

Розроблений метод:

- Використовує WebView для виконання в ньому тексту програми, відповідального за відображення інтерфейсу користувача та бізнес-логіки;
- Має стандартизовану архітектуру для взаємодії з «рідними» API платформа;
- Надає базову оболонку для мобільної платформи у формі програми React Native, яка діє як перекладач: для тексту клієнтської програми вона надає єдиний API для взаємодії з системними службами та виконує необхідні команди API залежно від платформи. на ньому зараз запущена програма;
- Виконується на мобільних пристроях Android та iOS і веб-браузерах.

Слід також зазначити, що програми, написані в React Native, можна скомпільувати для наступних платформ: Universal Windows Platform, Windows Presentation Foundation для Windows Phone і Desktop з ОС Windows, DOM для Web, Desktop для Desktop Ubuntu OS, macOS – настільні комп'ютери, що використовують MacOS OS, tvOS – адаптовано до Apple tvOS [21]. Таким чином, методи, розроблені з використанням основних методів React Native, обіцяють збільшити охоплення сучасних пристроїв за допомогою єдиної кодової бази.

Одним із важливих показників цієї платформи є швидкість ініціалізації програми. У розвинених системах на цей показник може впливати кількість підключених «різних» плагінів. Результати вимірювань часу показані на графіку на рисунку 4.1.

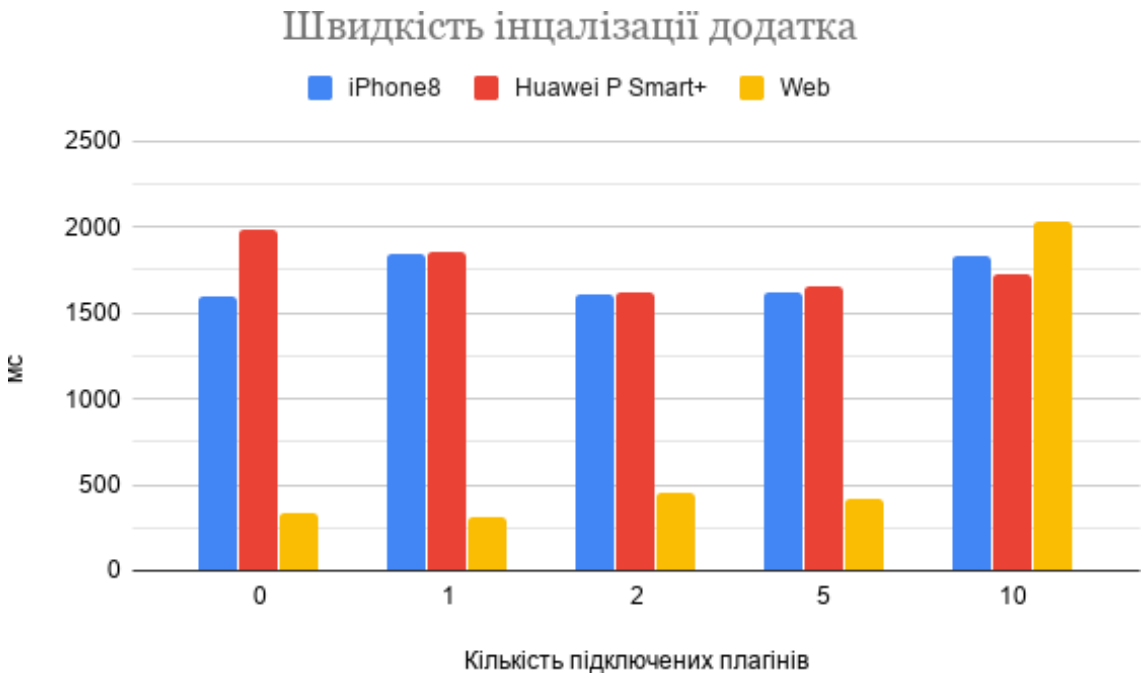

Рисунок 4.1. Графік швидкості ініціалізації програми в залежності від кількості плагінів і цільової платформи

Як видно з малюнка, швидкість ініціалізації програми не залежить від кількості плагінів, підключених до мобільної платформи. Не можу сказати багато про веб-програми. Але такий результат можна вважати задовільним, оскільки при роботі з веб-додатком немає необхідності використовувати всі плагіни одночасно. Крім того, програмний текст, завантажений для веб-додатків, можна розділити на частини та завантажувати поступово. Таким чином прискорюється «час першої взаємодії» користувача з додатком.

Цей показник є важливим і особливо включений до списку показників, які використовуються для аналізу ефективності веб-сайту. Максимально допустиме значення для цього показника досягає 5-15 секунд [22]. На нашому графіку

видно, що показник не перевищує норму для жодної платформи, що є задовільним результатом.

Ще один важливий показник — розмір програми. Адже, по-перше, обсяг пам'яті в смартфоні обмежений. По-друге, це для веб-програми

Показники впливають на швидкість завантаження. на рис. 4.2 та рис. 4.3 Показано результати аналізу розміру збірки платформ: Android і веб.

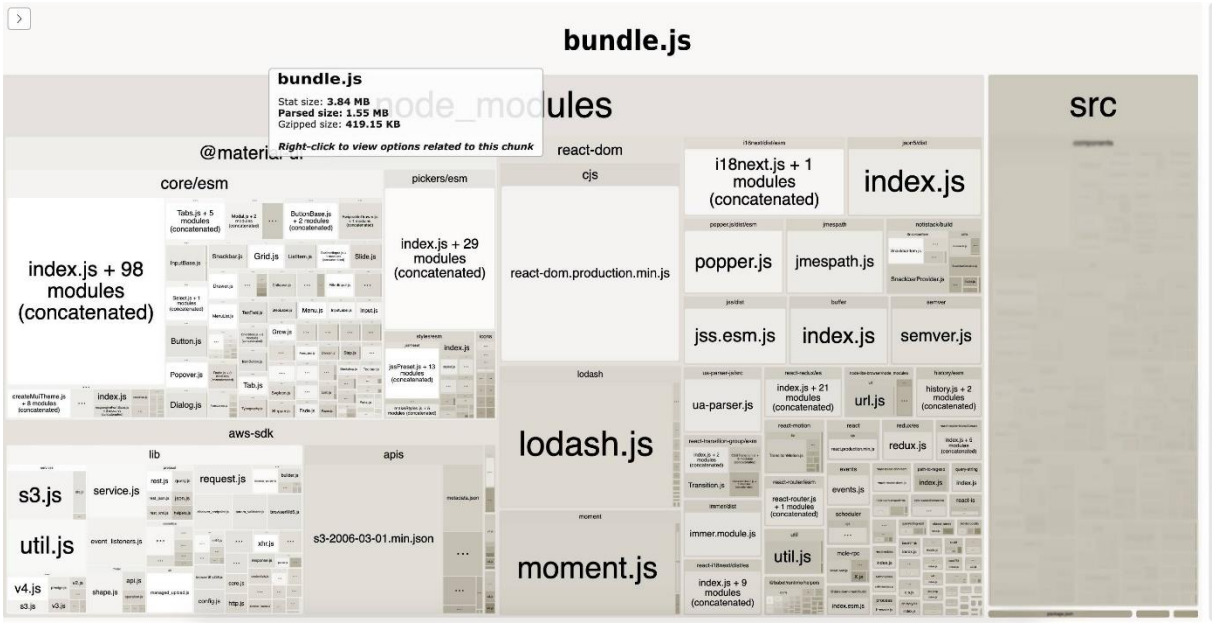

Рисунок 4.2. Результати аналізу веб-складання. Рекомендований розмір складання для веб-додатку становить 244 Кб.

Розроблений додаток важить 419,5 Кб, що приблизно в два рази більше. З малюнка видно, що більшість з них зайняті сторонніми бібліотеками. Тому їх розмір слід зменшити, включивши в збірку лише необхідні частини тексту програми замість усієї сторонньої бібліотеки. Безпосередньо текст програми платформи становить 67,91 Кб, що є задовільним результатом. Однак слід зазначити, що зображення, які містяться в збірці, важать 21,26 Кб, що становить

третину всього тексту програми платформи. Тому його потрібно виключити зі збірки і завантажити окремо.

Про платформу Android, з рисунку. 4.3 бачимо, що розмір завантаженого файлу становить 27,5 МБ. Зверніть увагу, що додаток скомпільовано у форматі *.aab.

Android App Bundle – це новий формат завантаження, який включає весь текст додатка та ресурси додатка, але відкладає створення файлу APK і підпис у Google Play.

Нова модель служби додатків Google Play під назвою «динамічна доставка» використовує *.aab для створення та обслуговування оптимізованих файлів APK для кожної конфігурації пристрою, тому вони завантажують лише текст програми та ресурси, необхідні для запуску програми[23].

Тому розмір файлу, завантаженого з Google Play, становить 13 МБ замість 28 МБ.

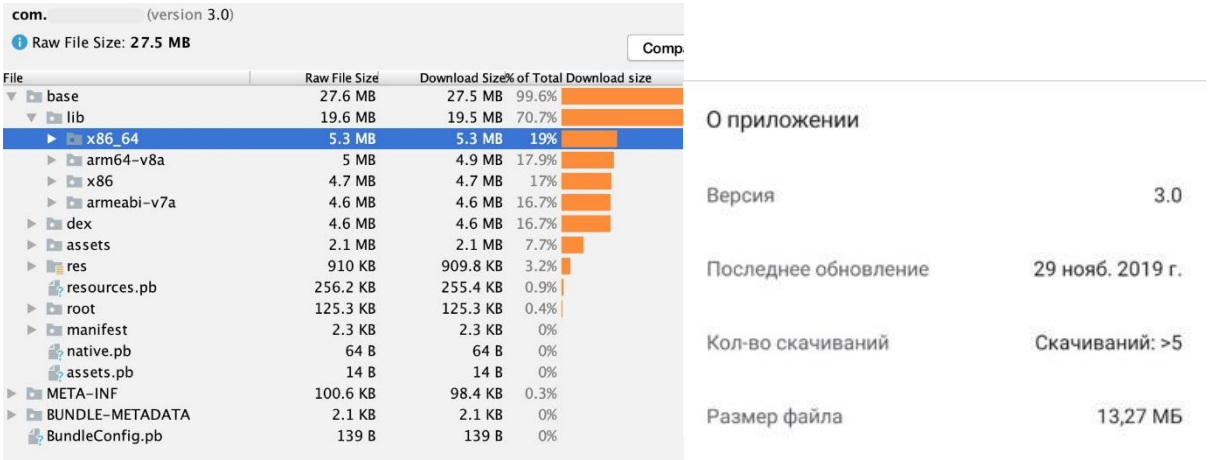

Рисунок 4.3. Результати аналізу збірки для платформи Android

Виходячи з результатів аналізу компонентів розробленої системи в попередній частині цієї роботи, перше покращення буде за рахунок оптимізації розміру зображення та зменшення розміру компонентів кінцевої платформи, що містить лише програмний текст сторонніх бібліотек, які використовуються в

збірці. Це пришвидшить завантаження програми в Інтернеті та на цільових мобільних платформах. Крім того, це прискорить роботу всієї програми.

Крім того, важливим удосконаленням є можливість налаштувати список використовуваних плагінів. Кількість «рідних» впливає на розмір збірки та час завантаження додатків як для веб-, так і для мобільних цільових платформ. Цю проблему можна вирішити шляхом створення модулів для webpack (web), gradle (Android) і cocoapods (iOS).

Розроблена система складається з двох ключових компонентів, які можна і потрібно публікувати в Реєстрі програмного забезпечення – NPM. Це транспортні класи каналу зв'язку для бібліотеки Mole-RPC і всієї системи.

Транспорти мають бути скомпільовані в бібліотеки, встановлені для кожного веб-проекту та проекту React Native, і забезпечувати можливість обмінюватися даними за допомогою технології Remote Procedure Call (RPC) і повідомлень JSON.

У контексті цієї роботи система означає дві програми: веб і React Native, які встановили канали зв'язку для функціональності залежного від платформи процедурного тексту – плагінів. Вся система повинна бути розроблена як шаблонний додаток і завантажена на платформу GitHub у загальнодоступний спосіб. А основні команди для створення та налаштування таких шаблонів внесені до консольної утиліти та опубліковані в Реєстрі програмного забезпечення - NPM.

Ще одним напрямком вдосконалення платформи є впровадження модулів віртуальної та доповненої реальності. Ці технології зараз дуже активні і скоро отримають широке поширення. Для React Native створені модулі для роботи з віртуальною та доповненою реальністю [24].

4.2. Опис алгоритму тестування для вимірів продуктивності інтерфейсу

Графічний інтерфейс користувача є важливою частиною будь-якої комп'ютерної програми чи системи. Від інтерфейсу залежить швидкість, з якою користувачі освоюють систему, і якість виконання роботи. Графічний інтерфейс повинен звести до мінімуму можливість помилки користувача. Тому до розробки інтерфейсу слід підходити серйозно і приділяти більше часу його розробці, оскільки важливість інтерфейсу користувача важко переоцінити.

Проте, здається, усі знають, яким має бути інтерфейс. Це тому, що всі висувають до нього вимоги. І ці вимоги зазвичай відрізняються від людини до людини. Але є і загальні моменти, яких необхідно дотримуватися. Іншими словами, інтерфейс повинен бути інтуїтивно зрозумілим. І це має бути самодокументування, очевидно, не інвазивне. Так, наприклад, не варто допомагати користувачеві, показуючи багато підказок. З іншого боку, має бути належний контроль над даними, які вводить користувач.

Тестування зручності використання перевіряє здатність користувача виконувати певні повсякденні завдання та проблеми, з якими він стикається при цьому. Результати такого тестування допомагають виявити проблеми, які ускладнюють розуміння та використання продукту, а також успішні рішення.

Тестування дозволяє:

- знати, наскільки добре працює інтерфейс, що може спонукати вас покращити його, або, якщо це достатньо добре, заспокоїтися; у будь-якому випадку, користь;
- порівняйте якість старого та нового інтерфейсів, щоб обґрунтувати зміни чи впровадження;
- ідентифікуйте та ідентифікуйте проблемні фрагменти інтерфейсу та, якщо розмір вибірки достатній, оцініть їх частоту.

Водночас U-тестування не може перетворити поганий продукт на хороший. Це лише покращує продукт.

Тестування зручності використання здійснюється на всіх етапах життєвого циклу програмного продукту.

Плануйте та готуйте. На цьому етапі вони намагаються виявити можливі проблеми зручності використання до початку розробки, щоб мінімізувати ризик під час розробки. Тут використовуються такі методи, як аналіз контексту, аналіз конкурентів, сортування карток, сценарії.

Активна фаза розробки програмного продукту. На цьому етапі повноцінний додаток розроблений і готовий до випуску. Застосовувати евристичні та експертні оцінки; паперові прототипи; розкадровки для виконання оцінок прототипів.

прийняття. Об’єктивна оцінка зручності використання необхідна для прийняття рішення про випуск програми. Фокус-група, евристичний метод оцінювання.

розвитку. На цьому етапі необхідно швидко виявити та усунути вузькі місця та труднощі, з якими стикаються користувачі програми під час роботи та через зміни. Перевірка здійснюється шляхом спостереження за користувачами (маніфест; опитування користувачів).

Під час тестування необхідно дотримуватися комплексних методів. Тому прийнято тестувати всі три основні сфери.

1. Інформаційна архітектура, яка повинна включати:

- інтуїтивно зрозуміла структура інформації;
- зручні інструменти для навігації та виклику функцій програми;

- Наочний спосіб представлення результату дії.

2. Робочий процес і взаємодія. Перевірте тут:

- використовувати логіку та логіку робочого процесу;

– ефективне використання додатків, виявлення та усунення зайвих операцій;
– простота використання, інтуїтивне розуміння та швидкість адаптації користувача;
– час реакції на дії користувача, короткі безперешкодні шляхи користувача (пользувальні шляхи);
– плавний процес виконання сценарію, належна обробка помилок і відмовостійкість програми.

3. Графічний дизайн. обстеження:

– легко сприймати та засвоювати інформацію, що відображається;
– повнота та одноманітність представлення функціональних і графічних елементів, що використовуються у всій програмі.

Для виявлення проблем із зручністю використання програми, у тому числі на ранніх етапах планування та розробки програмного забезпечення, використовуються методи подвійної перевірки (рисунок):

– вивчати досвід взаємодії користувача з програмою шляхом імітації поведінки користувача;
– перевірка дотримання принципів забезпечення зручності використання та коректного візуального представлення в контексті функціональних вимог шляхом експертної оцінки.

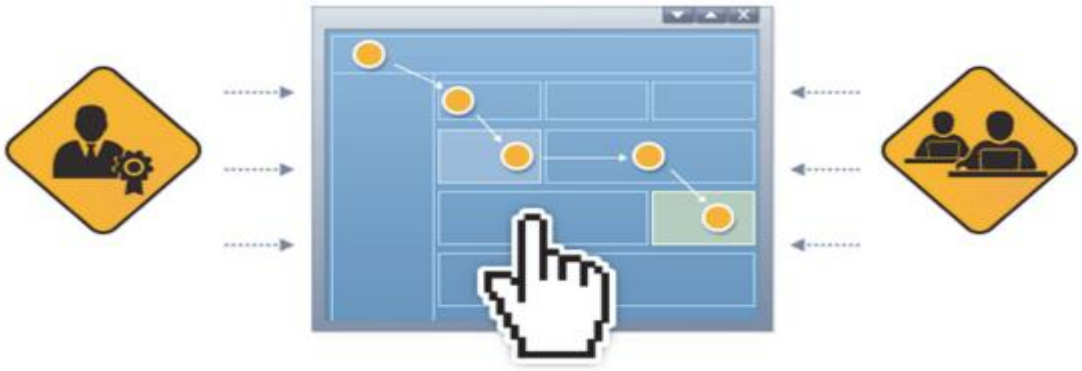

Рис. 4.4 - тестування зручності використання

Експертна оцінка програми базується на цілях проекту, функціональних і нефункціональних вимогах програмного забезпечення. Процес рецензування включає:

- визначення та вивчення можливих варіантів використання та шляхів користувача (шляхів користувача) у контексті бізнес-цілей та функціональності програми;
- проаналізувати інформаційну архітектуру програми;
- аналіз інтерфейсу та елементів інтерфейсу;
- аналіз функціональної відповідності.

Імітація поведінки користувача. Перевірте поведінку програми, імітуючи поведінку користувача. Таким чином, буде надана вся інформація, необхідна для швидкого усунення виявлених дефектів, які можуть негативно вплинути на зручність використання програми або загальний досвід користувача.

Юзер виявив серйозні проблеми з інтерфейсом. Але він не може показати, наприклад, помилковий фрагмент інтерфейсу, який недостатньо серйозний, щоб викликати людську помилку, але сповільнює роботу.

Експертний досвід дозволяє визначити їх без праці. Один або кілька експертів, професійних дизайнерів інтерфейсів або фахівців з юзабіліті оцінюють зручність використання системи за певним сценарієм або показником, усувають виявлені проблеми та (за бажанням) дають свої рекомендації.

Юзабіліті-тестування — це набір дій, які виконує користувач (респондент) над тестованим об’єктом. На підставі поведінки респондентів експерт з юзабіліті (модератор) робить висновки про проблеми юзабіліті, які існують в інтерфейсі, та їх характер.

Під час тестування юзабіліті користувачам пропонується вирішити основне завдання розробки продукту в «лабораторних» умовах, і їх просять коментувати під час виконання цих тестів.

Хід перевірки фіксується в протоколі (журналі) та/або аудіо- та відеоапаратурі. Якщо перевірка виявить будь-які труднощі, розробник повинен вдосконалити продукт і повторити тест.

Методи дослідження UX можна розділити на два табори: кількісні та якісні.

Кількісне дослідження – це будь-яке дослідження, яке можна виміряти чисельно. Він відповідає на такі запитання, як "Скільки людей тут клацнули?" або «Який відсоток користувачів здатні знайти заклик до дії?». Важливо розуміти статистичну ймовірність і те, що відбувається на сайті чи в програмі.

Якісне дослідження іноді називають «м'яким» дослідженням. Це допомагає нам зрозуміти, чому люди роблять те, що вони роблять, і часто має форму інтерв'ю чи розмов.

Поширені запитання: «чому люди не бачать заклик до дії» і «що ще люди помічають на сторінці?». Мета цих методів – допомогти зрозуміти мотивацію поведінки та логіку користувача. Кількісне дослідження спрямоване на опитування якомога більшої кількості респондентів і отримання числових результатів.

Типи юзабіліті тестування

Існує кілька типів юзабіліті тестування:

1) U-тестування «Коридор»;

2) розміщено дистанційне тестування;

3) Неконтрольоване дистанційне тестування;

4) A/B тестування.

Коридорний тест. Назва «коридор» походить від початкової ідеї випадкового захоплення 5-6 людей, які проходять по коридору. Респонденти виконують завдання, поставлені їм ведучим, на комп'ютері чи іншому пристрої. Як правило, сеанси записуються за допомогою спеціального програмного забезпечення та обладнання.

Варіант: Тестування в повністю обладнаній лабораторії (лабораторне тестування), живі зустрічі, включаючи спостереження за користувачами на їхніх робочих місцях.

Модеровано дистанційне тестування. Тестування проводиться дистанційно один на один. Респонденти та господар спілкуються віддалено (через телефон, Skype тощо). Респонденти виконували завдання зі своїх комп'ютерів, підключаючись до спільного віддаленого робочого столу.

Ведучий ставить завдання, контролює хід їх виконання, ставить уточнюючі запитання. Завдання модератора – змусити співрозмовника зосередитися на завданні (а не «допомагати» йому з цими проблемами) і з'ясувати:

– які непотрібні чи непотрібні дії користувач виконав на веб-сайті;

– що заважає йому швидше знайти потрібний об'єкт (інформацію);

– контент, який відхиляється від основної мети (наприклад, надто яскравий дизайн можна виправити після перевірки зручності використання сайту).

Дистанційне тестування без модерації. Респонденти самостійно виконують тести, створені в одній із спеціальних систем. Тестові завдання формуються в одній із дистанційних тестових систем. Спеціальні служби автоматично виконують завдання, збирають показники та відгуки без участі модератора. Посилання на тест буде надіслано респондентам, які проходили тест самостійно.

У системі можна поставити будь-яке питання (як в анкеті). Роботи проводяться на місці.

Використані фіксовані показники:

1) Виконання завдання (за налаштованими умовами);

2) Шлях користувача через веб-сайт;

3) Термін виконання;

4) Відповіді на запитання;

5) Теплова карта руху миші на веб-сайті.

Доступні інструменти для виконання цього тесту:

- Loop '11 (https://www.loop11.com);
- Usabilla (https://usabilla.com/);
- User Zoom (http://www.userzoom.com/);
- Webnographer (http://www.webnographer.com/);
- Remoteresear (http://remoteresear.ch/tools/).

A/B-тестування (A/B-тестування, спліт-тестування) — це автоматизоване тестування двох або більше версій одного контенту для однієї аудиторії.

Показувати кілька версій вмісту (сторінки, зображення, листи) з невеликими відмінностями для великої кількості користувачів. На основі результатів статистичної вибірки виміряйте продуктивність кожного випуску та з’ясуйте, які зміни покращують ціль

Різновидом A/B тестування є багатоваріантне тестування. При цьому тестуються не два повних варіанти, а одночасно кілька елементів виробу або компонентів об'єкта дослідження в різних комбінаціях, причому кожен тестовий елемент може бути двох типів (А або Б).

4.3. Опис функціоналу

React Native та Flutter працюють та виглядають однаково, за винятком того, що стиль кожної платформи відрізняється (крім кількох стилістичних аспектів, які є специфічними для кожної платформи). Під час створення додатків за допомогою React Native найбільшою проблемою для користувачів є те, що його час виконання є складнішим, ніж керування окремими процесами для кожної архітектури. Це означає, що за допомогою React можна наблизитися до продуктивності рідної програми, але не можете її досягти. Flutter не має таких же переваг, як React Native, щодо підтримки вже існуючих кодових баз JavaScript і

можливості повторного використання загальних частин додатків для iOS і Android.

Плюси та мінуси нативної продуктивності програми

Новіша віртуальна машина JavaScript у React Native швидша за V8, оскільки має JIT-компілятор. Крім того, це попередньо скомпільований фреймворк, який дозволяє публікувати будь-яку кодову базу, оскільки вона буде перетворена у рідний виконуваний файл React. React Native може досягти такої ж продуктивності розробки, як і програми для iOS, не змінюючи налаштувань збірки iOS. Це робить його таким же швидким на практиці, як чистий мобільний додаток React.

Коли ваш проект буде готовий, вбудований компілятор Ahead-of-Time Flutter згенерує код для iOS і Android. Подібно до React, не потрібно включати всю кодову базу у свій пакет додатків, щоб отримати адаптивну продуктивність Native.

Плюси та мінуси рідного розміру програми

Програми React Native можна зменшити, змінивши такі налаштування, як обхід відступів і використання режиму розробника на true. Обхід заповнення повідомляє React Native пропустити заповнення свого віртуального DOM результатом порівняння з рідним UI react. Більшість додатків мають приблизно 300 Кб часу виконання JavaScript. Тому зображення займає менше пам'яті та погіршує його якість. Завдяки передовому компілятору Flutter розробники можуть відправляти лише ту кодову базу, яка їм потрібна для програми, яку вони створюють, без будь-яких пакувань. Якщо необхідно, щоб ваша програма займала менше місця, можно запустити Flutter у віртуальній машині JavaScript, яка у вас уже є.

React Native має вбудований налагоджувач, який може підключатися до програм, які вже запущені на iOS або Android. Це дозволяє розробникам попередньо переглядати поточний стан віртуальної машини JavaScript і надає

інструменти для перегляду використання пам’яті та внесення змін на льоту. Крім того, Flutter має вбудований налагоджувач, який може підключатися до запущеної програми на iOS або Android. Це дає розробникам попередній перегляд поточного стану механізму візуалізації та доступ до інструментів для перевірки використання пам’яті або внесення змін на льоту.

React Native поставляється з власним набором API, які можна використовувати для створення програм для iOS і Android. Якщо є бажання, то можна написати частини програм для iOS і Android для обох платформ. Однак більшість компаній, які використовують React Native, спочатку розробляють свої програми для однієї платформи, а потім переносять їх на іншу. Ваші додатки для iOS і Android не зможуть надавати спільний доступ до коду, оскільки додатки Flutter створено з кодом, специфічним для кожної платформи. З іншого боку, сторонні бібліотеки легко знайти, і вони можуть легко повторно використовувати існуючі компоненти React Native.

Обидва фреймворки мають механізми, які дозволяють збільшити швидкість розробки додатків.

Однією з ключових функцій Flutter є Hot Reload, яка дозволяє розробникам швидко побачити ефект від змін коду. Ця функція дуже зручна для спільної роботи розробників і дизайнерів, і така паралельна робота значно збільшує швидкість розробки.

Крім того, програми Flutter складаються з віджетів: розробники можуть просто перекинути один віджет в інший і так далі. Цей принцип прискорює і спрощує роботу. Є багато готових віджетів, але розробники також можуть написати власні. Flutter також легко оновити: просто напишіть у консолі «flutter upgrade» — усі проекти та компоненти оновляться автоматично.

Що стосується React Native, він має функцію швидкого оновлення, яка дозволяє розробникам вставляти новий код безпосередньо в запущену програму. Але є деякі проблеми під час використання React Native. Наприклад, спроба

оновити версію React Native часто порушує роботу бібліотеки. Інструменти автоматичного оновлення версій іноді не працюють, тому розробникам доводиться оновлювати деякі компоненти вручну.

React Native також не такий хороший, коли справа стосується безпеки. Наприклад, будь-хто може отримати доступ до коду JavaScript у збірці випуску, що становить серйозну загрозу для всієї програми.

Швидкість розробки Flutter проти React Native: Flutter перемагає.

4.4. Тестування розробленого програмного забезпечення

У Flutter є три типи тестів:

- юніт-тест для окремої функції, методу чи класу;
- тест віджету для одного віджету (аналогічний тесту компонентів в інших фреймворках);
- інтеграційний тест для всієї програми або її частини.

Тестування відбувається дуже швидко: наприклад, тести віджетів у Flutter швидше, ніж тести аналогічних компонентів iOS. Також є список поширених помилок Flutter з інструкціями щодо їх виправлення (табл. 4.1)

Таблиця 4.1. – Тестування результатів

Вам потрібно розробити	Flutter	React Native
Швидкий прототип додатку	+	+
MVP мобільного додатка	+	+
Додаток з безліччю екранів та складною логікою	+	-

Додатки, які добре виглядатимуть навіть на старих пристроях	+	-
Не тільки мобільний додаток, а й веб-інтерфейс	+	+
Ігровий додаток	-	-
Додаток з AR	-	-

У більшості випадків Flutter краще підходить. Але якщо хочете створити гру або додаток AR, необхідно піти шляхом розробки нативних програм. У цьому випадку можна створити додаток на Flutter, але потрібено єдиний інтерфейс користувача, одним з елементів якого є AR.

Можна використовувати як Flutter, так і React Native для розробки програм для таких областей:

- мобільний банкінг (наприклад, у нас є додаток для Росбанка на Flutter);
- соціальна мережа (Insight Timer - соціальна мережа на тему уважності);
- месенджер;
- програми лояльності (наприклад, впровадження програми лояльності в додатку мережі аптек «Рігла»);
- електронна комерція (Xianyu — програма для електронної комерції, розроблена Alibaba);
- сервіси потокового передавання (прикладом є наш додаток на Flutter для платформи потокового відео The Hole);
- додатки для стилю життя (Reflectly — програма для усвідомленості);
- освітні послуги (додаток AIteacher від китайської компанії Tencent).

Тестування програм, створених на React Native, може бути досить складним, оскільки React Native використовує різні движки JavaScript для нормального режиму та режиму налагодження. Так що код може поводитися по-різному.

Тестування Flutter vs. React Native: Flutter виграє.

4.5. Висновки до розділу 4

У цьому розділі описано методику тестування розробленої системи, проаналізовано результати роботи та визначено напрями подальшого вдосконалення системи в майбутньому.

За результатами тестування можна сказати, що розроблена система не має помилок, які могли б призвести до нештатних ситуацій під час роботи користувача. Тому його можна вважати готовим до залучення бета-тестерів.

Аналіз системи дав позитивні результати, а саме незалежність початкового завантаження програми від кількості використовуваних плагінів і метрики продуктивності веб-програми – «час до першої взаємодії». Однак також було встановлено, що система має дефекти за монтажними розмірами, і перераховані заходи щодо усунення проблеми.

Крім того, за результатами аналізу встановлено, що система повністю відповідає вимогам системного аналізу моделювання.

В останній частині цього розділу визначено основні напрямки вдосконалення розробленої системи. який є:

- усунути дефекти, виявлені під час системного аналізу розробки;
- забезпечувати публічний доступ до модулів розробки, розміщуючи їх на відповідних ресурсах у вигляді бібліотек, шаблонів і консольних утиліт;
- адаптувати розроблену систему до інших цільових платформ, що підтримуються технологією React Native;
- запровадити модулі віртуальної та доповненої реальності.

5. РЕЗУЛЬТАТИ ВИПРОБУВАНЬ

5.1 Порівняння результатів виводу текстового списку

Для порівняння виводу текстового списку було реалізовано такий же самий інтерфейс у React Native та Flutter (на Android та iOS). Також було реалізовано автоматичну швидкість прокручування за допомогою RecyclerView.SmoothScroller на Android. В iOS і React Native використовуємо таймери та програмне прокручування. У Flutter використовуємо ScrollController для плавного прокручування списку. У кожному випадку в списку є 1000 елементів, і заміряємо час прокручування до останнього елемента в списку. Детальніше у додатку Б.

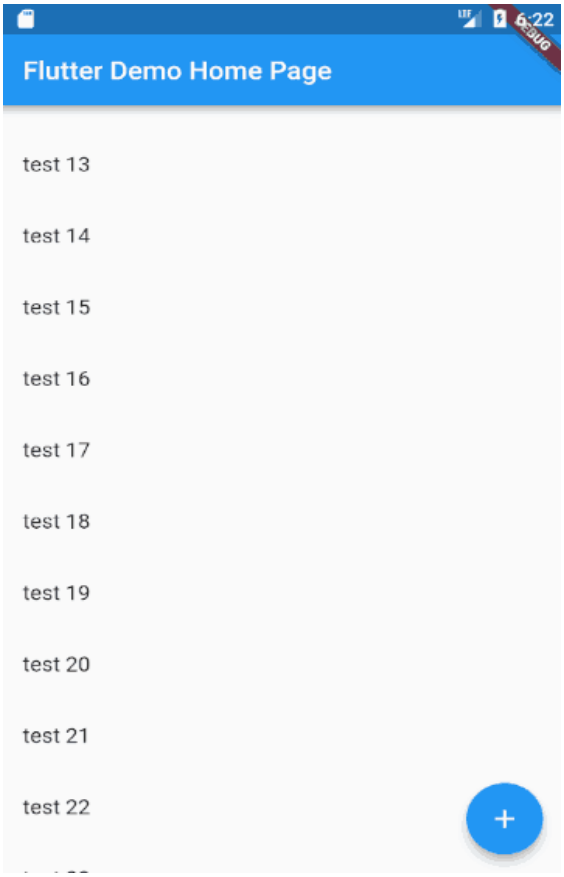

Рисунок 5.1 – Тестування списків

В таблиці 5.1 наведено порівняльна характеристика виводу текстового списку Flutter та React Native на Android, а в таблиці 5.2 на IOS.

Таблиця 5.1 – порівняльна характеристика Flutter та React Native на Android

Android	FPS	CPU%	Max Memory Mb	Battery mAh
React Native	60	2.4	58	49.7mAh
Flutter	60	5.4	114	65.28mAh

Додаткові результати тестування для Android

Всі тести показують приблизно однаковий FPS.

Порівняно з Flutter і React Native, Android Native використовує вдвічі менше пам’яті.

React Native найбільше використовує ЦП. Причина полягає в тому, що JS Bridge використовується між JS і рідним кодом, що призводить до марної витрати ресурсів серіалізації та десеріалізації.

Що стосується часу автономної роботи, Android Native отримав найвищий бал. React Native відстає від Android і Flutter. Запуск безперервної анімації на React Native споживає більше енергії акумулятора.

Таблиця 5.2 – порівняльна характеристика Flutter та React Native на IOS

IOS	FPS	CPU%	GPU%	Battery Mb
React Native	60	12.72	21.24	154
Flutter	60	33.3	10.75	159

FPS. У React Native результати гірші, ніж у Flutter. Причина полягає у неможливості використання IoT-компіляції на iOS.

Обсяг пам'яті. Flutter можна порівняти зі React щодо споживання пам'яті, але сильніше вантажить процесор. У цьому тесті React Native сильно відстає від Flutter.

Flutter vs React: Flutter активно використовує CPU, React активно використовує GPU.

5.2 Порівняння результатів виводу списку з малюнками

Для порівняння виводу списку із малюнків було реалізовано той самий інтерфейс у React Native та Flutter (на Android та iOS). Також реалізували автоматичну швидкість прокручування за допомогою RecyclerView.SmoothScroller на Android. В iOS і React Native використовуємо таймери та програмне прокручування. У Flutter використовуємо ScrollController для плавного прокручування списку. У кожному випадку в списку є 1000 елементів, і заміряємо час прокручування до останнього елемента в списку.

У кожному випадку було використовано кеші зображень з різними бібліотеками для кожної платформи. Детальніше у додатку Б.

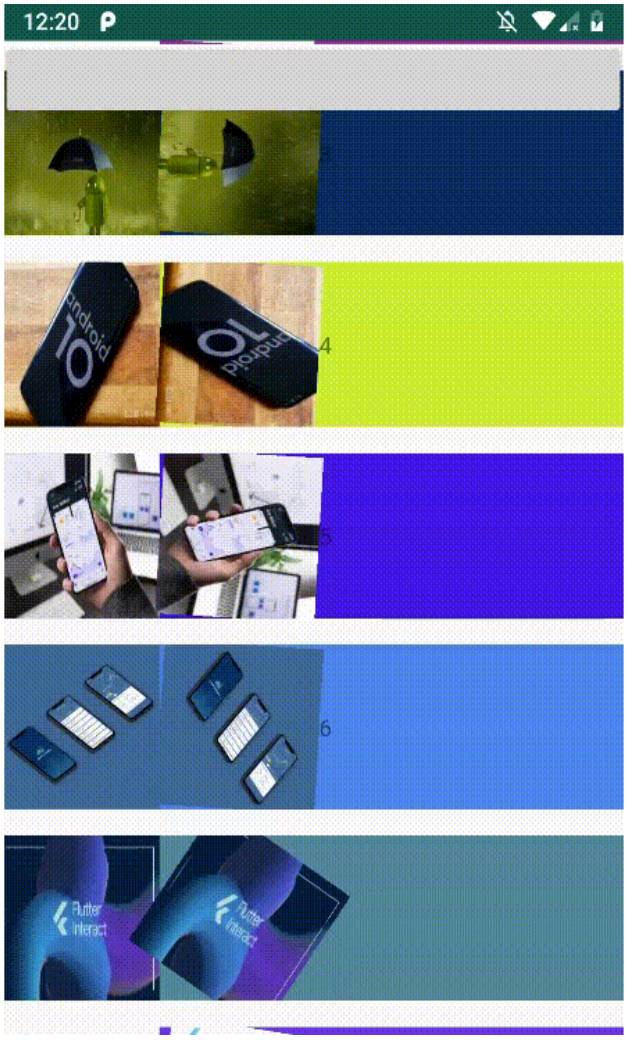

Рисунок. 5.2 – тестування списку з малюнками

В таблиці 5.3 наведено порівняльна характеристика виводу списку зображень Flutter та React Native на Android, а в таблиці 5.4 на IOS.

Таблиця 5.3 – порівняльна характеристика Flutter та React Native на Android

Android	FPS	CPU%	Max Memory Mb	Battery mAh
React Native	58	6.53	80	59.7mAh
Flutter	19	10.28	168	69.28mAh

Таблиця 5.4 – порівняльна характеристика Flutter та React Native на IOS

IOS	FPS	CPU%	GPU%	Battery Mb
React Native	59	61	48.28	158
Flutter	59	69	81.91	191

Результати тестів для Android

Android Native і React Native можна порівняти за продуктивністю. Це очевидно тому, що Lottie для React Native використовує нативні виклики (16–19% CPU, 30–29 FPS).

Результат Flutter показав гарний результат під час тестування (12% CPU та 9 FPS).

Android Native вимагає найменшого обсягу пам'яті (205 Мб); React Native вимагає 280 Мб, а Flutter - 266 Мб.

Холодний старт програми. За цим показником Flutter є лідером (2 секунди). Для Android Native та React Native він займає близько 4 секунд.

Було виявлено, що видалення однієї анімації із сітки збільшує FPS до 40% у Flutter. Було зроблено припущення, що Flatter важкий і недостатньо оптимізований для таких завдань. Саме тому у Flutter і було таке падіння FPS.

Результати тестів для iOS

Результати iOS та React Native у цьому тесті майже такі самі, як у Lottie для React Native.Нативний код для iOS тут потребує найменшого обсягу пам'яті (48 Мб). React Native вимагає 135 Мб, а Flutter - 117 Мб;Холодний старт програми. За цим показником Flutter є лідером (2 секунди). Для нативного коду iOS та React Native він займає близько 10 секунд.Було використано з Flutter іншу бібліотеку, яка набагато важча порівняно з тими, що використовували для інших платформ, і це може спричинити падіння FPS для Flutter.

5.3 Порівняння результату виводу анімацій

Зараз більшість телефонів Android та iOS мають потужне апаратне забезпечення. У більшості випадків при використанні звичайних бізнес-додатків падіння FPS не спостерігалося. Ось чому було обрано провести важке тестування анімації. Для створення векторної анімації було використано Lottie для Android, iOS, React Native і Flare для Flutter:

Рисунок 5.3 – тестування анімації

Таблиця 5.5 – порівняльна характеристика Flutter та React Native на Android

Android	FPS	CPU%	Memory Mb	Battery mAh
React Native	30	18.9	205	15.97mAh
Flutter	9	12.8	266	14.11mAh

Таблиця 5.6 – порівняльна характеристика Flutter та React Native на IOS

IOS	FPS	CPU%	GPU%	Memory Mb	Battery mAh
React Native	25	15.1	62.9	48	16.00mAh
Flutter	8	12.3	57.71	117	17.00mAh

Результати тестів для Android

Android Native показав найвищу продуктивність та найбільш ефективне споживання пам'яті.

Flutter показав дуже близький до Native fps і вдвічі більшу витрату пам'яті, але все ж таки пристойну продуктивність.

React Native показав низьку продуктивність цього тесту.

Результати тестів для iOS

iPhone досить потужний, щоб не допустити падіння FPS у всіх трьох випадках.

Робота нативного коду вимагала менше ресурсів, оскільки в основному використовувався GPU.

React Native для рендерингу переважно використовував процесор, а Flutter використовував GPU.

Крім того, React Native з'їв трохи більше пам'яті.

5.4. Порівняння адаптивності

Адаптивність

Обидві платформи забезпечують високу адаптивність, але вони в будь-якому випадку повільніші, ніж технології нативної розробки. Але це компроміс, для мінімізації витрат на розробку та час виведення продукту на ринок, з цим кросплатформові технології справляються на відмінно. Треба сказати, що користувачі навряд чи помітять відмінності, тому що обидва фреймворки забезпечують плавні скролінг, анімацію та переходи між екранами.

Основною характеристикою продуктивності програми є FPS – кількість кадрів, що відображаються за секунду. Золотий стандарт – 60 кадрів на секунду: якщо цей показник досягнуто, анімація та переходи між екранами будуть плавними. Коли FPS впаде, скажімо, до 30, це буде помітно для користувача.

Вимірювання, проведені на різних пристроях, показали, що Flutter і React Native демонструють адаптивність, аналогічну до нативних додатків, коли йдеться про просте прокручування: нативні додатки на iOS та Android показують 60 FPS, Flutter – 60 FPS, React Native – 58-59 FPS . Але React Native вимагає більше пам'яті пристрою та заряду акумулятора в порівнянні з Flutter та нативними програмами.

Але коли справа дійшла до важкої анімації з ротацією, масштабуванням та згасанням, продуктивність React Native впала до 7 кадрів за секунду для Android, а Flutter продемонстрував 19 FPS.

Є й інші дані, які говорять на користь Flutter. Наприклад, було проведено порівняльні тести адаптивності додатків на iPhone 6 та Xiaomi Redmi Note 5. І отримали наступні результати.

iOS: За результатами тесту Гауса додаток на Flutter працює в 20 разів швидше за програму на React Native.

Android: додаток на Flutter працює у 15 разів швидше, ніж на React Native, і навіть випереджає нативний додаток на Swift.

Адаптивність React Native vs. Flutter: Flutter перемагає.

5.5. Порівняння недоліків

Основні недоліки які були помічені при розробці програми на Flutter наступні:

— Кінцевий інсталяційний пакет більший, оскільки додано віртуальну машину Dart.

Таким чином, існують файли Flutter і додані віртуальні машини залежно від того, чи компілюєте програму для iOS чи Android.

Віртуальна машина має власний графічний механізм, який промальовує інтерфейс програми через усі переходи між екранами, діалогами, фрагментами тощо. Це суттєва різниця між розробкою Flutter та розробкою Xamarin і React Native, які використовують справжні компоненти Android та iOS. У їхньому випадку неможливо використовувати специфічні для платформи компоненти (якщо є така необхідність, необхідно створити два варіанти інтерфейсу). З Flutter при виборі дизайну достатньо орієнтуватися на одну платформу (наприклад, Android).

- Інтерфейси створюються за допомогою коду, тому грань між логікою та дизайном набагато тонша.

З іншого боку, такий підхід полегшує розділення екрана на окремі компоненти. Насправді будь-який вкладений блок елементів інтерфейсу можна перетворити в один віджет за кілька кроків, що набагато простіше, ніж створювати користувацькі перегляди та фрагменти.

- Нестабільний.

Очевидно, що з новим фреймворком у вашому розпорядженні набагато менше бібліотек, ніж з нативною розробкою на Android/iOS. Проте бібліотек Flutter все ще досить багато, і вони все ще з’являються з великою швидкістю. Так, наприклад, багато бібліотек було додано у другій половині 2018 року, мабуть, під час підготовки до першого стабільного випуску, тоді як найважливіші (Google Analytics, Firebase, Maps тощо) існували до цього.

Стосовно недоліків React Native, вони були наступні:

- Проблеми з налагодженням і сумісністю.

Незважаючи на всі чудові функції, React Native все ще знаходиться в бета-версії! Ось чому він все ще має деякі кричущі проблеми, такі як труднощі з налагодженням додатків, і обмеження, включаючи проблеми сумісності. Використання React native під час налагодження може бути проблематичним.

- Низьке управління пам'яттю.

React Native створює потужні програми. Однак React Native може бути не найкращою платформою для створення програм, які ефективно керують апаратними ресурсами.

Оскільки керування пам’яттю в React Native не відповідає стандартам, то не варто вибирати цю платформу для створення високопродуктивних програм. Якщо потрібно створити програму, яка потребує складних обчислень, краще пошукати альтернативну платформу розробки.

- Продуктивність і функціональність.

React Native чудово підходить для створення простих програм із привабливими візуальними ефектами. Однак, коли потрібно додати складну функціональність, досягти високої продуктивності стає важко. Тому, можна, обрати одне з двох: функція чи інтерфейс.

5.6. Висновки до розділу 5

І Flutter, і React Native – чудові способи створення кросплатформових додатків. Між ними є деякі подібності, але є і суттєві відмінності, про які необхідно знати, перш ніж вирішити, який із них використати. Вибір кращого кроссплатформенного фреймворку для вашої корпоративної програми або нового стартапу залежить від ваших навичок розробки, якості вашої команди розробників та компонентів react-native, які має використовувати ваш проект.

Flutter або React Native - є сучасними, дуже популярними і можуть бути використані для висококласних кросплатформових проектів.

Якщо необхідно, щоб частина інтерфейсу програми була нативною, то використовувати React Native. Якщо в більшому пріоритеті дизайн то краще використовувати Flutter як рішення.

ЗАГАЛЬНІ ВИСНОВКИ

Для бізнес-додатків із простою анімацією та гарним зовнішнім виглядом вибір технології для розробки не є критичним. Не рекомендується використовувати React Native для завдань, що інтенсивно використовують процесор, оскільки Flutter чудово підходить як для процесора, так і для пам’яті.

Вибір засобу залежить від конкретного продукту та ситуації. Якщо розробити MVP для платформи, можна вибрати нативну розробку, але Flutter дозволяє створювати програми для мобільних пристроїв і Інтернету. Тож, можливо, найближчим часом Flutter стане лідером ринку кросплатформенної розробки. Сьогодні Flutter є дуже гідним конкурентом нативним інструментам розробки, особливо якщо невеликий бюджет розробки, але необхідно забезпечити прийнятний рівень продуктивності програми.

Існує багато факторів, які впливають на процес розробки та стандарти для кожної технології. Основною ціллю є максимізувати прозорість процесу, надаючи кожній програмі єдине тестове середовище та уніфікований набір інструментів вимірювання продуктивності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Eisenman, B. Learning React Native [Text] / Bonnie Eisenman. 2nd edition. Sebastopol: O'Reilly Media, 2017. - 242 p.

2. Kirupa, Understanding WebViews [Електронний ресурс] / Kirupa. Режим доступу: https://www.kirupa.com/apps/webview.htm. Дата доступу: 15.09.2022.

3. WebKit Org., The WebKit Opensource Project [Електронний ресурс] / WebKit Org. Режим доступу: https://webkit.org/project/. Дата доступу: 15.09.2022.

4. Garbade, M. Native vs. cross-platform app development: pros and cons [Електронний ресурс] / Dr. Michael J. Garbade. Режим доступу: https://codeburst.io/native-vs-cross-platform-app-development-pros- and-cons-49f397bb38ac. Дата доступу: 15.09.2022.

5. LeRoux, B. PhoneGap Beliefs, Goals, and Philosophy [Електронний ресурс] / Brian LeRoux. Режим доступу: https://phonegap.com/blog/2012/05/09/phonegap-beliefs-goals-and-philosophy/.Дата доступу: 15.09.2022.

6. Trice, A. PhoneGap Explained Visually [Електронний ресурс] / Andrew Trice. Режим доступу: https://phonegap.com/blog/2012/05/ 02/phonegap-explained-visually/. Дата доступу: 15.09.2022.

7. Wargo, J. Apache Cordova 4 Programming (Mobile Programming) [Text] / John M. Wargo. 1st edition. Boston: Addison-Wesley Profeccional, 2015. - 560 p.

8. Griffith, C. Mobile App Development with Ionic, Revised Edition: Cross-Platform Apps with Ionic, Angular, and Cordova [Text] / Chris Griffith. 1st edition. Sebastopol: O'Reilly Media, 2017. - 292 p.

9. Flutter, Flutter for iOS developers [Електронний ресурс] / Flutter. Режим доступу: https://flutter.dev/docs/get-started/flutter-for/ios-devs. Дата доступу: 15.09.2022.

10. Napoli, M. Beginning Flutter: A Hands On Guide to App Development [Text] / Varco L. Napoli. 1st edition. Birmingham: Wrox, 2019. 528 p.

11. Aggarwal, R. 10 top Programming Languages in 2019 for Businesses [Електронний ресурс] / Ruchika Singh Aggarwal. Режим доступу: https://codeburst.io/10-top-programming-languages-in-2019-for- developers-a2921798d652. Дата доступу: 15.09.2022.

12. Google Inc., Building web apps in WebView [Електронний ресурс] / Google Inc. Режим доступу: https://developer.android.com/guide/w ebapps/webview. Дата доступу: 15.09.2022.

13. Kumar, S. How React Native Works? [Електронний ресурс] / Saket Kumar. Режим доступу: https://www.codementor.io/saketkumar95/ how-react-native-works-mhjo4k6f3. Дата доступу: 15.09.2022.

14. Buckler, C. JavaScript HTML5 Programming [Text] / Craig Buckler. 2nd edition. Sebastopol: O'Reilly Media, 2012. 241 p.

15. Morley, M. JSON-RPC 2.0 Specification [Електронний ресурс] / Matt Morley. Режим доступу: https://www.jsonrpc.org/specification. Дата доступу: 07.10.2022.

16. Turskyi, V. MOLE-RPC [Електронний ресурс] / Viktor Turskyi. Режим доступу: https://www.npmjs.com/package/mole-rpc. Дата доступу: 07.10.2022..

17. Facebook, Communication between native and React Native [Електронний ресурс] / Facebook. Режим доступу: https://facebook.github.io/react-native/docs/communication-ios. Дата доступу: 07.10.2022.

18. Vepsäläinen, J. SuviveJS – Webpack and React: From apprentice to master [Text] / Juho Vepsäläinen. 2st edition. Scotts Valley: CreateSpace Independent Publishing Platform, 2016. 284 p.

19. Turskyi, V. Yet another JSON RPC Library? [Електронний ресурс] / Viktor

Turskyi. Режим доступу: https://docs.google.com/presentation/d/1NCmVJalBJp0Gliyc8KCdExHTl-xztDz3v50V4Y2k0bQ/edit#slide=id.g43ac3735fc_0_757. Дата доступу: 07.10.2022.

20. Facebook, Out-of-Tree Platforms [Електронний ресурс] / Facebook. Режим доступу: https://facebook.github.io/react-native/docs/out-of- tree-platforms. Дата доступу: 07.10.2022.

21. Souders, S. Even Faster Web Sites: Performance Best Practices for Web Developers [Text] / Steve Souders. 1st edition. Sebastopol: O'Reilly Media, 2009. 256 p.

22. Google Inc., About Android App Bundles [Електронний ресурс] / Google Inc. Режим доступу: https://developer.android.com/guide/ap p-bundle. Дата доступу: 07.10.2022.

23. ViroMedia, Overview [Електронний ресурс] / ViroMedia. Режим доступу: https://docs.viromedia.com/docs/viro-platform-overview. Дата доступу: 07.10.2022.

24. Gasston, P. The Modern Web: Multi-Device Web Development with HTML5, CSS3, and JavaScript [Text] / Peter Gasston. 1nd edition. San Francisco: No Starch Press, 2013. 264 p.

25. Simpson, K. You Don't Know JS: Async & Performance [Text] / Kyle Simpson. 1nd edition. Sebastopol: O'Reilly Media, 2015. 296 p.

26. Simpson, K. You Don't Know JS: Up & Going [Text] / Kyle Simpson. 1nd edition. Sebastopol: O'Reilly Media, 2015. 88 p.

27. Ionic, Core Concepts [Електронний ресурс] / Ionic. Режим доступу: https://ionicframework.com/docs/intro/concepts. Дата доступу: 15.09.2022.

ДОДАТКИ

ДОДАТОК А

ЕКОНОМІЧНА ЧАСТИНА

До цієї статті належать витрати на основну та додаткову заробітну плату розробникам, інженерно-технічним працівникам, тестувальникам, та іншим працівникам, безпосередньо зайнятим у відтворенні інтерфейсу мобільних додатків на кросплатформених платформах Flutter та React Native, обчислені за посадовими окладами та тарифними ставками, відрядними розцінками для робітників, включаючи преміальні виплати. Вихідні дані наводяться в табл. А.1.

Середньоденна ставка зарплати ($Сд_i$) для кожного з виконавців (i) дорівнює:

$$Сд_i = ЗП_i/F_р, \quad (6.1)$$

де $F_р$ – місячний фонд робочого часу (24 дні).

Для розрахунку витрат на оплату праці виконавців даної НДДКР (табл. 4.2) визначається трудомісткість роботи кожного з працівників (T_i) (в людино-днях).

Таблиця А.1. – Вихідні дані для розрахунку заробітної плати

№ з/п	Посада виконавців	Місяч-ний оклад, грн. (основ. ЗП)	Місячна тарифна ставка, грн. (основ. ЗП)	Додаткова зарплата, грн.	Премія, грн.	Середньо-місячна ЗП, грн. (сума 3-6)	Середньо-денна ставка, грн./дні
1	Розробник (КР)	9500	–	2000	–	11500	479
2	Інженерно-технічний працівник	6600	–	2400	–	9000	375
3	Тестувальник	3000	–	–	–	3000	125

Таблиця А.2. – Розрахунок витрат на оплату праці виконавців

№ з/п	Посади виконавців	$Сд_i$ грн./дні	T_i люд./дні	Витрати на оплату праці ($В_{оп}$), грн.
1	Розробник	479	5	2395
2	ІТ працівник	375	1	375
3	Тестувальник	125	35	4375
Разом:				7145

Відрахування на соціальні заходи

Єдиний соціальний внесок у частині нарахувань установлюється в розмірі 22%:

$B_{сз} = B_{оп}$ *0,22 = 7145*0,22 = 1571,90 (грн.).

Розрахунок амортизаційних відрахувань та витрат на енергію для наукових цілей

Вартість однієї машино-години $Ц_м$ обраховується за формулою:

$$Ц_м = B \cdot H_а /(100 \cdot N) + P_к \cdot Ц_е, \quad (6.2)$$

де N – кількість годин роботи комп'ютера (принтера); B – вартість комп'ютера (принтера); $H_а$ – амортизаційні відрахування; $P_к$ – потужність кВт/год; $Ц_е$ – ціна, грн./кВт/год.

Порахуємо для одного комп'ютера:

$Ц_{м.к} = 20000 \cdot (33/(100 \cdot 2400)) + 0,1 \cdot 1,68 = 2,91$ (грн/год).

Порахуємо для принтера:

$Ц_{м.пр} = 6000 \cdot (20/(100 \cdot 2400)) + 0,5 \cdot 1,68 = 1,71$ (грн/год).

Результати розрахунків витрат на амортизацію та енергію для наукових цілей занесено в табл. А.3, а розрахунку витрат на носії інформації – в табл. А.4.

Таблиця А.3. – Розрахунок витрат на амортизацію та енергію для тестування на кроссплатформах ($B_{ен}$)

№ з/п	Назва устаткування	$Ц_м$, грн./ кВт×год.	N, маш/год	$B_{ен}$, грн.
1	Комп'ютер	2,91	20	58,2
2	Принтер	1,71	10	17,1
	Разом:			75,3

Таблиця А.4. – Розрахунок витрат на носії інформації для кроссплатформ

№ з/п	Найменування (вид) матеріалу	Одиниця виміру	$H_і$, од.	$Ц_і$, грн./од.	$B_{мі}$, грн.
1	Папір	пачка	1	90	90
2	Флеш-пам'ять	штука	1	250	250
3	Ручки	штука	3	5	15
Р а з о м:					340

Накладні витрати

До складу накладних витрат ($B_н$) відносяться:

- витрати, пов’язані з управлінням мобільними додатками, де проводиться розробка;

- витрати на науково-технічну інформацію;

- витрати на забезпечення нормальних умов праці і техніки безпеки;

- витрати на інші загальногосподарські потреби тощо.

Накладні витрати розраховуються у відсотках до витрат на оплату праці ($B_{оп}$):

$$B_н = B_{оп} \times \alpha / 100, \quad (6.3)$$

де α – середньостатистичний відсоток накладних витрат в організації (150-200%). Згідно даних НДЧ НУ “Львівська політехніка” приймаємо $\alpha = 155\%$.

$$B_н = 7145 \times 155/100 = 11074{,}75 \text{ (грн.)}.$$

Розрахунок калькуляції кошторисної вартості розробки

Результати розрахунку по всіх статтях наводяться в табл. 4.5 і складають кошторисну вартість виконання НДДКР (K):

$$K = B_{оп} + B_{сз} + B_{су} + B_і + B_н.. \quad (6.4)$$

Розрахунок витрат на експлуатацію проектного рішення роботи додатків на кросплатформених платформах Flutter та React Native

Експлуатаційні одноразові витрати (E) для проектного рішення включають вартість підготовки даних ($E1$) і вартість машино-годин роботи ПЕОМ ($E2$) (грн.):

$$E = E1 + E2. \quad (6.5)$$

Таблиця А.5. – Калькуляція кошторисної вартості проектного рішення

№ з/п	**Статті витрат**	**Сума, грн.**
1.	Витрати на оплату праці	7145,00
2.	Відрахування на соціальні заходи	1571,90
3.	Інші витрати, в т.ч. :	415,30
3.1	Амортизаційні відрахування і енергія для наукових цілей	75,30
3.2	Витрати на носії інформації	340,00
4.	Накладні витрати	11073,75
	Всього:	**20 621,25**

Річні експлуатаційні витрати

$$E_{\text{Р}} = E \cdot N\ , \tag{6.6}$$

де N – періодичність експлуатації кросплатформ, раз/рік (N = 35 разів/рік – для введення змін у створені форми замовлень).

Вартість підготовки даних для роботи на Flutter та React Native ($E1$) визначимо за формулою :

$$E1 = n \cdot t \cdot c, \tag{6.7}$$

де n – чисельність співробітників, які підготовлюють дані, осіб; t – час роботи співробітників із підготовки даних, год.; c – середньогодинна ставка співробітника, грн./год. (c = 125,00/8 = 15,62 грн./год. – див. табл. 4.1).

Витрати на експлуатацію платформ ($E2$) визначимо за формулою:

$$E2 = t \cdot S_{\text{м-г}},\ , \tag{6.8}$$

де t – машино-години роботи платформ для одноразової експлуатації проектного рішення, год.; $S_{\text{м-г}}$ – вартість 1 машино-години роботи платформ конкретного типу, грн./год. ($S_{\text{м-г}}$ = 2,61 грн./год.)

В нашому випадку: n = 1; t = 7 год. – максимальний час необхідний для підготовки даних.

Отже, $E1 = 1 \cdot 7 \cdot 15{,}62 = 109{,}34$ (грн.);

$E2 = 7 \cdot 2{,}61 = 18{,}27$ (грн.).

Тоді $E = 109{,}34 + 18{,}27 = 127{,}61$ (грн.),

$E_{\text{Р}} = 125{,}61 \cdot 35 = 4466{,}35$ (грн.).

Розрахунок ціни споживання проектного рішення роботи додатків на кросплатформених платформах Flutter та React Native

Ціна споживання ($Ц_{\text{с}}$) – це витрати на придбання і експлуатацію проектного рішення за весь строк його служби:

$$Ц_{\text{с(п)}} = Ц_{\text{п}} + B_{\text{(е)п рv}},\ , \tag{6.9}$$

де $Ц_{\text{п}}$ – ціна придбання проектного рішення, грн.:

$$Ц_{\text{п}} = K \cdot (1 + П_{\text{р}}/100) + K_{\text{о}} + K_{\text{к}},\ , \tag{6.10}$$

де K – витрати на розробку і впровадження проектного рішення; $П_{\text{р}}$ – норма

рентабельності (20-35%, взято 28 %); $К_о$ – витрати на прив'язку та освоєння проектного рішення на конкретному об'єкті.

За даними у відкритих інформаційних порталах $К_о$ включає:

- інсталяцію програмного продукту на ПК (555 грн.);
- документацію на програмний продукт (1250 грн.);
- демонстрацію роботи програмного продукту (655 грн.).

Загальна сума витрат на прив’язку та освоєння проектного рішення роботи додатків на кросплатформених платформах Flutter та React Native на конкретному об’єкті :

$$К_о = 555 + 1250 + 655 = 2460 \text{ (грн.)}.$$

де $К_к$ – витрати на доукомплектування технічних засобів на об'єкті (дорівнює нулю).

Таким чином, ціна придбання проектного рішення дорівнює:

$$Ц_п = 20\,621{,}25 \cdot (1 + 0{,}28) + 2460 = 28\,855{,}20 \text{ (грн.)}.$$

Теперішня вартість витрат на експлуатацію проектного рішення (за весь час його експлуатації ($В_{(е)п\ pv}$), грн.:

$$В_{(е)п\ pv} = \sum_{t=0}^{T} \frac{В_{(е)п\_t}}{(1+R)^t},, \qquad (6.11)$$

де $В_{(е)п\_t}$ – річні експлуатаційні витрати в t-ому році, грн.; T – строк служби проектного рішення, 5 років; R – річна облікова ставка НБУ (тут приймається рівним 0,08 і залишається незмінною протягом експлуатації).

Для даних R і T отримаємо наступне значення коефіцієнта дисконту:

$$pv = \frac{1}{(1+0{,}08)^1} + \frac{1}{(1+0{,}08)^2} + \frac{1}{(1+0{,}08)^3} + \frac{1}{(1+0{,}08)^4} + \frac{1}{(1+0{,}08)^5} =$$

$$= 0{,}925 + 0{,}857 + 0{,}793 + 0{,}735 + 0{,}680 = 3{,}99$$

Якщо впродовж всього строку експлуатації $B_{(е)п\_t}$ = const, то:

$$B_{(е)п}{}^{pv} = B_{(е)п} \cdot \sum_{t=0}^{T} \frac{1}{(1+R)^t} = pv \cdot Ер\ , \qquad (6.12)$$

де *pv* – коефіцієнт дисконту на період *T*, який визначається залежно від процентної ставки *R* і періоду експлуатації *T* (для нашого випадку *T* =5 років).

$$B_{(е)п}{}^{pv} = 3{,}99 \cdot 4466{,}35 = 17820{,}73 \text{ (грн.)}$$

Тоді ціна споживання проектного рішення роботи додатків на кросплатформених платформах Flutter та React Native дорівнюватиме:

$$Ц_{с(п)} = 28\ 855{,}20 + 17820{,}73 = 46\ 675{,}93 \text{ (грн.)}.$$

В залежності від виду виконуваних в дипломній роботі досліджень та поставлених цілей, для підсумкової оцінки результатів вибирається один з перелічених ефектів, а інші використовуються як додаткові характеристики. Як правило, в межах виконуваних досліджень характерним є отримання наукового та науково-технічного ефекту, який оцінюється коефіцієнтом науково-технічної ефективності:

$$K_{НТЕ} = \frac{\sum_{j=1}^{n} \alpha_j^{н} \overline{Б}_{jk}}{\sum_{j=1}^{n} \alpha_j^{н} Б_{jk\,\max}} \begin{matrix} j = 1,2,..., n \\ k = 1,2,..., l, \end{matrix} \qquad (6.13)$$

де $\alpha_j^{н}$ – нормована величина коефіцієнта вагомості *j*-го фактору науково-технічної ефективності;

$\overline{Б}_{jk}$ – середнє значення бальної оцінки, яка виставляється експертами *k*-ій якості *j*-го фактору; $Б_{jkmax}$ – максимально можлива величина бальної оцінки; *l* – кількість якостей, які характеризують *j*-ий фактор; *n* – кількість факторів.

Чим більше значення розрахованого $K_{НТЕ}$, тим вищий рівень науково-технічної ефективності проведеного дослідження по кроссплатформам. Оцінка $K_{НТЕ}$ здійснюється експертним шляхом не менш як трьома експертами за

десятибальною шкалою. $\overline{Б}_{jk}$ визначається як середнє арифметичне.

Нормовані значення коефіцієнтів вагомості для факторів науково-технічної ефективності наведено в табл. А.6, а результати оцінки науково-технічної ефективності проектного рішення роботи додатків на кросплатформених платформах Flutter та React Native – в табл. А.7.

Таблиця А.6. – Нормовані значення коефіцієнтів вагомості для факторів науково-технічної ефективності пропонованого проектного рішення роботи додатків на кросплатформених платформах Flutter та React Native

№ з/п	Фактор (*j*)	$\alpha_j^н$
1	Новизна очікуваних або одержаних результатів	0,20
2	Глибина наукової проробки	0,17
3	Ступінь ймовірності успіху	0,13
4	Перспективність використання результатів	0,18
5	Масштаб можливої реалізації результатів	0,08
6	Завершеність одержаних результатів	0,24
Разом		**1,0**

Таблиця А.7. – Результати оцінки науково-технічної ефективності пропонованого проектного рішення роботи додатків на кросплатформених платформах Flutter та React Native

№ з/п	Фактори науково-технічної ефективності	Якість фактора	Експертні оцінки			$\overline{Б}_{jk}$	$Б_{jk\ max}$
			1	2	3		
1	Новизна очікуваних або одержаних результатів	середня	3	7	5	5	6
2	Глибина наукової проробки	середня	1	1	3	1,6	3
3	Ступінь ймовірності успіху	помірна	7	6	5	6	7
4	Перспективність використання результатів	достатня важливість	9	8	9	9	9
5	Масштаб можливої реалізації результатів	середня	5	6	6	6	6

6	Завершеність одержаних результатів	середня	6	3	6	5	6
$К_{НТЕ}$ = 0,870							

$$\boldsymbol{K}_{\text{НТЕ}} = \frac{\mathbf{0,22 \cdot 5 + 0,18 \cdot 1,6 + 0,12 \cdot 6 + 0,20 \cdot 9 + 0,07 \cdot 6 + 0,21 \cdot 5}}{\mathbf{0,22 \cdot 6 + 0,18 \cdot 3 + 0,12 \cdot 7 + 0,20 \cdot 9 + 0,07 \cdot 6 + 0,21 \cdot 6}} = \mathbf{0,870}$$

1. Для визначення витрат на проведення дослідження роботи додатків на кросплатформених платформах Flutter та React Native складена калькуляція кошторисної вартості робіт за статтями витрат на оплату праці, відрахування на соціальні заходи, накладні витрати. Кошторисна вартість досліджень складає 20 621,25 грн.

2. Річні витрати на експлуатацію проектного рішення становлять 4466,35грн., а ціна його споживання дорівнює 46 675,93 (грн.).

3. Також було здійснено оцінку науково-технічної ефективності роботи додатків на кросплатформених платформах Flutter та React Native. Отримання наукового і науково-технічного ефекту визначено за допомогою коефіцієнта науково-технічної ефективності, окремі фактори якого оцінені експертним шляхом. Значення вище згаданого коефіцієнта становить 0,870, що свідчить про високий рівень науково-технічної ефективності проведених робіт.

Короткі відомості

ДОДАТОК Б

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ДЛЯ РОЗРОБКИ КРОСПЛАТФОМНИХ МОБІЛЬНИХ ДОДАТКІВ

У цьому розділі аналізуються існуючі рішення, розроблені на різних платформах. Отримана інформація систематизована для використання в подальшій роботі щодо визначення вимог до розробки програмного забезпечення, що дозволить створювати кросплатформні користувальницькі додатки.

2. ТЕХНІЧНЕ ПОРІВНЯННЯ ЗАСОБІВ РОЗРОБКИ

У цьому розділі детально обговорюються методи використання Flutter та React Native як середовища для виконання нестандартних програм на мобільних пристроях. Крім того, система розроблена таким чином, що програму також можна запускати у веб-браузері настільного комп’ютера. Також ідеться порівняння Flutter та React Native.

Крім того, усвідомлення веб-середовища як іншої цільової платформи можна вважати модифікацією. Зрештою, взаємодія з орієнтованою на платформу частиною тексту програми відбувається в тому ж процесі, а не через вторинні канали зв’язку. І ця взаємодія відрізняється від стандартного підходу.

3. ЗАСОБИ ДЛЯ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ

У даному розділі було розроблено вдосконалений підхід до розробки мобільних додатків за допомогою WebView шляхом створення та використання таких структурних компонентів:

– Компонент відображення інтерфейсу користувача - WebView, переважно для всіх сучасних платформ, включаючи iOS та Android.

– Канал зв’язку між React Native і WebView на основі архітектури клієнт-сервер і механізму глобального генератора подій, вбудованого в браузер. Канал зв’язку має кілька компонентів, таких як клієнт, сервер, проміжне програмне забезпечення, клас-оболонка для генераторів глобальних подій з обох сторін і клас, який об’єднує всі компоненти.

– Плагіни надають доступ до специфічних для платформи API. Вони забезпечують однаковий інтерфейс незалежно від того, на якій платформі зараз працює програма. Таким чином, розробнику не потрібно перевіряти поточний тип середовища виконання на кожному кроці.

– Менеджер плагінів, який обробляє різні типи плагінів (утиліти та компоненти), зберігає їх у реєстрі та надає до них доступ.

Завдяки своїй взаємодії веб-додатки можуть використовувати API платформи, надані операційною системою смартфона, як якщо б вони були «рідними» мобільними додатками. Крім того, не втрачається сумісність із веб-середовищем.

4. РОЗРОБКА ПРОГРАМ

У цьому розділі описано методику тестування розробленої системи, проаналізовано результати роботи та визначено напрями подальшого вдосконалення системи в майбутньому.

За результатами тестування можна сказати, що розроблена система не має помилок, які могли б призвести до нештатних ситуацій під час роботи користувача. Тому його можна вважати готовим до залучення бета-тестерів.

Аналіз системи дав позитивні результати, а саме незалежність початкового завантаження програми від кількості використовуваних плагінів і метрики продуктивності веб-програми – «час до першої взаємодії». Однак також було встановлено, що система має дефекти за монтажними розмірами, і перераховані заходи щодо усунення проблеми.

Крім того, за результатами аналізу встановлено, що система повністю відповідає вимогам системного аналізу моделювання.

В останній частині цього розділу визначено основні напрямки вдосконалення розробленої системи. який є:

– усунути дефекти, виявлені під час системного аналізу розробки;

– забезпечувати публічний доступ до модулів розробки, розміщуючи їх на відповідних ресурсах у вигляді бібліотек, шаблонів і консольних утиліт;

– адаптувати розроблену систему до інших цільових платформ, що підтримуються технологією React Native;

запровадити модулі віртуальної та доповненої реальності.

5. РЕЗУЛЬТАТИ ВИПРОБУВАНЬ

І Flutter, і React Native – чудові способи створення кросплатформових додатків. Між ними є деякі подібності, але є і суттєві відмінності, про які необхідно знати, перш ніж вирішити, який із них використати. Вибір кращого кроссплатформенного фреймворку для вашої корпоративної програми або нового стартапу залежить від ваших навичок розробки, якості вашої команди розробників та компонентів react-native, які має використовувати ваш проект.

Flutter або React Native - є сучасними, дуже популярними і можуть бути використані для висококласних кросплатформових проектів.

Якщо необхідно, щоб частина інтерфейсу програми була нативною, то використовувати React Native. Якщо в більшому пріоритеті дизайн то краще використовувати Flutter як рішення.