Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

магістра
(назва освітньо-кваліфікаційного рівня)

студента	*Кунденка Павла Руслановича*
	(ПІБ)
академічної групи	*122М-21-1*
	(шифр)
спеціальності	*122 Комп'ютерні науки*
	(код і назва спеціальності)
освітньої програми	*«122 Комп'ютерні науки»*
	(назва освітньої програми)
на тему:	*Дослідження ефективності фреймворку React Native при розробці мобільних додатків*

_____*П.Р. Кунденко*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
розділів кваліфікаційної роботи				
спеціальний	*проф. Мещеряков Л.І.*	96	відмінно	
Рецензент				
Нормоконтролер	*проф. Лактіонов І.С.*			

Дніпро
2022

Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 20 22 Року

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

студенту *122м-2-1* *Кунденко Павлу Руслановичу*
(група) (прізвище та ініціали)

Тема кваліфікаційної роботи *Дослідження ефективності фреймворку React Native при розробці мобільних додатків*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 00.00.2022 р. №

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – процес розробки мобільних додатків для мобільних операційних систем Android та iOS, використовуючи фреймворк React Native, з метою визначення його ефективності.

Предмет досліджень – методи та підходи фреймворку React Native у розробці програмного забезпечення для мобільних пристроїв.

Мета роботи – дослідження та вдосконалення процесу розробки мобільних додатків під найпоширеніші операційні системи з використанням фреймворку React Native.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Новизна запропонованих рішень полягає у вдосконаленні методів визначення ефективності мобільного програмного забезпечення створеного з використанням програмного засобу React Native. Розробка мобільних додатків за допомогою React Native отримала подальший розвиток у відстеженні швидкодії додатку.

Практична цінність полягає у створенні програмного пакету для фреймворку React Native, що дозволяє відстежувати ефективність та швидкодію роботи мобільного додатку. Програмний пакет також може бути використаний у будь-якому проекті побудованому за допомогою програмного засобу React Native, тим самим поліпшуючи якість та підвищуючи швидкість процесу розробки.

Вихідні дані для проведення роботи – теоретичні та експериментальні дослідження, демонстраційний застосунок та програмний пакет для оцінки ефективності.

4 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Результати досліджень мають бути подані у вигляді, що дозволяє побачити та оцінити ефективність роботи додатку, розробленого на React Native. В результаті роботи повинен бути проведений аналіз ефективності роботи мобільних додатків та розроблений програмний пакет для оцінки швидкодії роботи додатку.

5 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець**
Аналіз теми та постановка задачі	12.09.2022-30.09.2022
Дослідження методів для вирішення поставленого завдання	01.10.2022-31.10.2022
Експериментальні дослідження	01.11.2022-01.12.2022

Завдання видав _____ *Мещеряков Л.І.*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Кунденко П.Р.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *05.09.2022 р.*

Термін подання кваліфікаційної роботи до ЕК *09.12.2022*

РЕФЕРАТ

Пояснювальна записка: 119 стор., 45 рис., 3 додатки, 50 джерел.

Об’єкт досліджень: процес розробки мобільних додатків для операційних систем Android та iOS, використовуючи фреймворк React Native, з метою визначення його ефективності.

Предмет досліджень: методи та підходи фреймворку React Native у розробці програмного забезпечення для мобільних пристроїв.

Мета магістерської роботи: дослідження та вдосконалення процесу розробки мобільних додатків під найпоширеніші операційні системи з використанням фреймворку React Native.

Методи дослідження. Для вирішення поставлених задач використані методи: теорія системного аналізу, принципи функціонального програмування, принципи об’єктно орієнтованого програмування.

Новизна отриманих результатів дипломної роботи полягає у вдосконаленні методів визначення ефективності мобільного програмного забезпечення створеного з використанням програмного засобу React Native. Розробка мобільних додатків за допомогою React Native отримала подальший розвиток у відстеженні швидкодії додатку.

Практична цінність отриманих результатів полягає у створенні програмного пакету для фреймворку React Native, що дозволяє відстежувати ефективність та швидкодію роботи мобільного додатку. Програмний пакет також може бути використаний у будь-якому проєкті побудованому за допомогою програмного засобу React Native, тим самим поліпшуючи якість та підвищуючи швидкість процесу розробки.

Область застосування. Розроблене програмне забезпечення може застосовуватися при розробці мобільних додатків з використанням React Native.

Значення роботи та висновки. Результати дослідження ефективності фреймворку React Native розробленим програмним пакетом на розробленому демонстраційному додатку показують, що додаток працює ефективно навіть при великому навантаженні.

Прогнози щодо розвитку досліджень. Покращити програмний пакет для тестування ефективності додатків методом доповнення додатковими метріками оцінки швидкодії.

Список ключових слів: мобільний додаток, кросплатформенний додаток, фреймворк, React Native, Javascript, Typescript, iOS, Android.

ABSTRACT

Explanatory note: 119 pages, 45 figures, 3 applications, 50 sources.

Object of research: the process of development mobile applications for mobile operating systems Android and iOS, using the React Native framework, to determine its effectiveness.

Subject of research: React Native framework techniques and approaches in mobile software development.

The purpose of the master's work: research and improvement of the mobile application development process for the most common operating systems using the React Native framework.

Research methods. The following methods are used to solve the problem: theory of system analysis, principles of functional programming, principles of object-oriented programming.

The novelty of the obtained results of the thesis consists in improving the methods of determining the effectiveness of mobile software created using the React Native framework.

The practical value of the obtained results lies in the creation of a software package for the React Native framework, which allows monitoring the efficiency and performance of the mobile application. The software package can also be used in any project built using the React Native software tool, thereby improving the quality, and increasing the speed of the development process.

Scope. The developed software can be used in the development of mobile applications using React Native.

The value of the work and conclusions. The results of the research on the effectiveness of the React Native framework by the developed software package on the developed demo application show that the application works efficiently even being high loaded.

Further research and development. Improve the software package for testing the effectiveness of applications by supplementing it with additional evaluation metrics.

Keyword list: mobile application, cross-platform application, framework, React Native, Javascript, Typescript, iOS, Android.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОС – операційна система;

БД – база даних;

RN – React Native;

HTML – Hyper Text Markup Language;

CSS – Cascading Style Sheets;

JS – JavaScript;

UI – User Interface;

ART – Android Runtime;

HAL – Hardware Abstraction Level;

DEX – Dalvik Executable;

JIT – Just In Time compilation;

AOT – Ahead Of Time compilation;

FPS – Frames Per Second;

JSX – Javascript XML;

SPA – Single Page Application;

DOM – Document Object Model;

API – Application Programming Interface;

CLI – Command Line Interface;

SQL - Structured Query Language.

ЗМІСТ

РЕФЕРАТ 4

ABSTRACT 5

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 6

ЗМІСТ 7

ВСТУП 9

РОЗДІЛ 1. АНАЛІЗ ПОТОЧНИХ ЗАСОБІВ ТА ІНСТРУМЕНТІВ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ МОБІЛЬНИХ ПРИСТРОЇВ 11

1.1 Аналіз мобільних операційних систем 11

1.1.1 Операційна система Android 11

1.1.2 Операційна система iOS 17

1.2 Аналіз методів та засобів розробки нативних мобільних застосунків. 23

1.3 Аналіз методів та засобів створення гібридних мобільних додатків ... 25

1.4 Кросплатформенні мобільні додатки 27

1.5 Висновок до першого розділу 32

РОЗДІЛ 2. АНАЛІЗ МЕТОДІВ РОБОТИ ТА РОЗБІР ФРЕЙМВОРКУ REACT NATIVE 33

2.1 Розбір роботи бібліотеки React 33

2.1.1 Потік даних у React 35

2.1.2 Алгоритм Virtual DOM у React 36

2.1.3 Життєвий цикл компоненту у React 37

2.1.4 Управління станом додатку в React 38

2.2 Розбір роботи фреймворку React Native 40

2.2.1 Архітектура фреймворку React Native 41
2.2.2 Основні потоки в React Native 42
2.2.3 Методи створення проєкту React Native 43
2.2.4 Компоненти React Native 46
2.2.5 Аналіз поширеності React Native серед розробників 48
2.3 Висновок до другого розділу 50
РОЗДІЛ 3. РОЗРОБКА ДЕМОНСТРАЦІЙНОГО ДОДАТКУ ТА ПРОГРАМНОГО ПАКЕТУ ДЛЯ МОНІТОРИНГУ ЕФЕКТИВНОСТІ РОБОТИ ДОДАТКУ 51
3.1 Розробка демонстраційного додатку 51
3.1.1 Технології використані при розробці додатку 51
3.1.2 Архітектура додатку 56
3.1.3 Інтерфейс додатку 58
3.2 Розробка програмного пакету для вимірювання ефективності роботи додатку 67
3.3 Тестування додатку за допомогою пакету оцінки ефективності 70
3.4 Висновок до третього розділу 71
ВИСНОВКИ 73
СПИСОК ВИКОРИСТАННИХ ДЖЕРЕЛ 75
Додаток А. КОД ПРОГРАМИ 79
Додаток Б. ВІДГУК КЕРІВНИКА 115
Додаток В. РЕЦЕНЗІЯ 117
Додаток Г. ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ 119

ВСТУП

На сьогодні є очевидною тенденція на зростання попиту на мобільні додатки та розробку мобільного програмного забезпечення. Кількість мобільних пристроїв, смартфонів, планшетів, смарт-годинників стрімко зростає та вже значно перевищує кількість класичних комп'ютерів.

Серед мобільних операційних систем перші позиції безперечно займають Android та iOS. Android є open-source операційною системою та використовується на більшості мобільних пристроїв у світі, а також планшетів та на багатьох пристроях Smart TV. Його доля становить 71%, тоді як доля iOS становить 28% [1].

Відповідно до цієї тенденції на перехід споживача до мобільного контенту, бізнес також зацікавлений в швидкому наданні своїх послуг через мобільні додатки. Навіть держава не є виключенням, адже яскравим прикладом діджиталізації та переходу до мобільних сервісів є додаток «Дія», розроблений міністерством цифрової трансформації України, та яким вже користуються мільйони Українців.

Це свідчить про те, що мати «представництво» на ринку мобільних додатків для бізнеса вже не є опціональним, а скоріш необхідним кроком для різних підприємств.

Існує багато програмних засобів створення мобільних додатків, але складність сучасної мобільної розробки полягає у тому, що дві домінуючі мобільні операційні системи потребують різних підходів та навіть мов програмування для розробки додатків. Однак створення програмного забезпечення це досі дорогий процес, і утримувати одразу дві команди розробки для того, щоб створити один додаток одразу під дві різні платформи, є досить важкою задачею для бізнесу. Саме відповідями на ці виклики стали такі не так давно створені фреймворки як React Native або Flutter, які дозволяють створювати кросплатформенні додатки.

Метою роботи є дослідження ефективності мобільного фреймворку React Native від компанії Facebook як засобу створення мобільних кросплатформенних додатків.

Для досягнення мети дослідження було поставлено наступні задачі:

1. Проаналізувати існуючі методи та інструменти для розробки кросплатформенних мобільних додатків.

2. Здійснити детальний аналіз найпопулярнішого фреймворку для розробки кросплатформенних мобільних додатків React Native.

3. Розробити окремий універсальний пакет для визначення ефективності роботи програмного забезпечення для мобільних пристроїв.

4. Розробити демонстраційний додаток, що реалізує всі поширені паттерни сучасних мобільних додатків.

5. Зробити висновки щодо найкращих програмних інструментів для розробки кросплатформенних мобільних додатків.

Кваліфікаційна робота складається з трьох розділів. У першому розділі був проведений аналіз предметної області, проаналізовані основні мобільні операційні системи Android та iOS, а також розглянуті різні підходи та інструменти для створення мобільного програмного забезпечення. У другому розділі були детально проаналізовані підходи, методи, особливості фреймворку React Native. В третьому розділі було розроблено програмний пакет для оцінки ефективності додатку, а також було розроблено демонстраційний додаток.

РОЗДІЛ 1
АНАЛІЗ ПОТОЧНИХ ЗАСОБІВ ТА ІНСТРУМЕНТІВ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ МОБІЛЬНИХ ПРИСТРОЇВ

1.1 Аналіз мобільних операційних систем

На сьогодні провідними мобільними операційними системами є Android, що займає 70,96% ринку мобільних пристроїв, та iOS, що займає 28,43% [1]. Також слід зазначити що на ринку мобільних операційних систем присутні KaiOS, Samsung OS, Windows Mobile тощо. Але їх загальна доля в цілому складає лише близько 1% ринку (рис. 1.1).

Рис 1.1. Розподіл ринку мобільних операційних систем

1.1.1 Операційна система Android

Android (рис. 1.2), як система, є операційною системою на основі Java, яка працює на ядрі Linux 2.6 [2]. Програми для Android розроблені з використанням

Java і можуть бути досить легко перенесені на нові платформи. Інші особливості Android включають прискорений 3-D графічний механізм (на основі апаратної підтримки), підтримка бази даних на основі SQLite та вбудований браузер [3].

Рис. 1.2 Логотип ОС Android

Архітектура операційної системи Android містить велику кількість компонентів для підтримки будь-яких потреб пристроїв Android. Програмне забезпечення Android містить ядро Linux з відкритим вихідним кодом, яке містить набір бібліотек C/C++, які доступні через служби програми.

Серед усіх компонентів ядро Linux надає основні функції операційної системи для смартфонів, а віртуальна машина Dalvik (DVM) забезпечує платформу для запуску програми Android [4].

Основні компоненти архітектури Android такі (рис. 1.3):

- Додатки.
- Фреймворк програми.
- Android Runtime.
- Бібліотеки платформи.
- Ядро Linux.

Докладніше про кожен з компонентів:

1. Linux Kernel - цей рівень є базовим в архітектурі Android, тому що вся система Android побудована на ядрі Linux з деякими архітектурними змінами. Ядро містить драйвери, необхідні взаємодії з апаратним забезпеченням. Наприклад, візьмемо Bluetooth. На сьогодні складно знайти пристрій, який би не

включав цю функцію. Тому ядро має містити драйвер для роботи з ним. На схемі нижче (рис. 1.3) перераховані всі драйвери, що входять в ядро Linux [5].

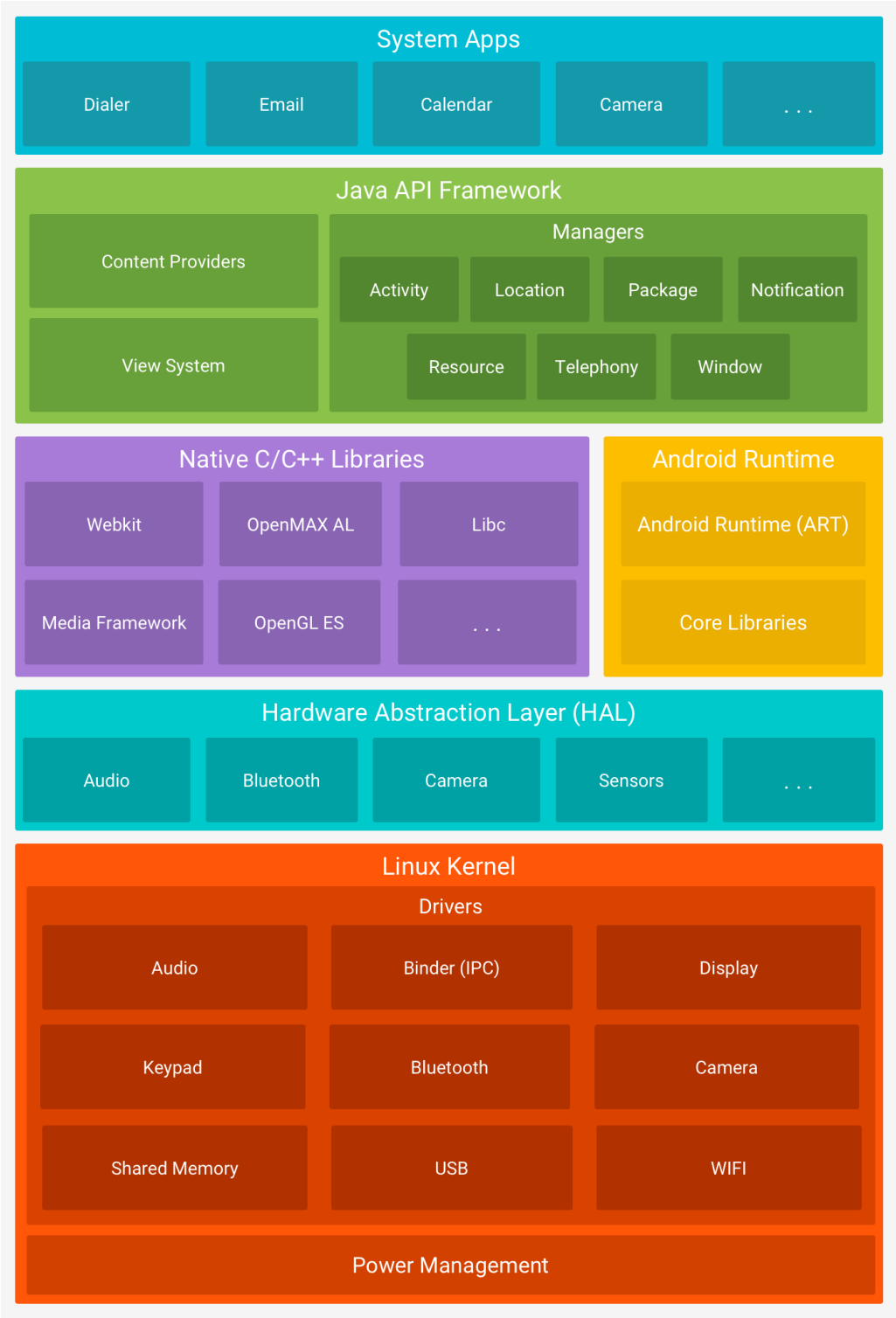


Рис. 1.3 Архітектура Android

Power Management – це свого роду система управління живленням. Вона надає різні засоби, за допомогою яких програма може реагувати на режими живлення пристрою, а також підтримувати необхідні компоненти пристрою активними. Наприклад, при читанні книги було б зручно, якби екран залишався постійно активним. Або коли користувач вмикає музику і вона продовжує програватися у фоновому режимі при вимкнутому екрані.

Крім перерахованого вище, ядро Linux забезпечує управління пам'яттю і процесом. При запуску різних програм ядро гарантує, що простори пам'яті, які вони використовують, не конфліктує і не перезаписує один одного. Також воно перевіряє, що всі програми отримують достатній обсяг пам'яті для своєї роботи, і в той же час стежить, щоб жодна програма не займала занадто багато місця.

Кожна програма в Android працює в окремому процесі. Ядро відповідає за управління цими процесами, а саме за створення, призупинення, зупинку, або завершення процесів, за одночасне виконання декількох процесів, обмін даними між процесами, запуск процесів у фоновому режимі. Крім цього, ядро розподіляє роботу та ресурси між процесорами пристрою, що максимізує продуктивність пристроїв з багатоядерними процесорами [6].

2. Hardware Abstraction Layer (HAL) забезпечує зв'язок між драйверами та бібліотеками. Складається він із кількох бібліотечних модулів, кожен з яких реалізує інтерфейс для певного апаратного компонента (Bluetooth, Камера тощо). І коли до обладнання пристрою звертаються через API-інтерфейс, завантажується необхідний його роботи модуль.

Коли від програми надходить якесь повідомлення, HAL його обробляє таким чином, щоб воно стало зрозумілим для драйверів і навпаки.

3. Android Runtime (ART) - основною мовою Android був обраний Java, оскільки це одна з найпопулярніших мов програмування. Для Java існує багато напрацювань і фахівців, а написані на ньому програми переносяться між операційними системами.

Але для того, щоб програма працювала на Java, необхідна віртуальна машина – Java Virtual Machine. В Android використовується віртуальна машина Android Runtime (ART). Ця машина спеціально оптимізована для роботи на мобільних пристроях: з нестачею пам'яті, з постійним вивантаженням та завантаженням додатків тощо. У версіях Android нижче 5.0 Lollipop використовувалася віртуальна машина Dalvik, що є минулою реалізація віртуальної машини для Android.

В ART, як і в Dalvik, використовується свій формат байт-коду - DEX (Dalvik executable). При складанні програми вихідні файли спочатку компілюються у файли типу class звичайним компілятором, а потім конвертуються спеціальною утилітою в DEX.

І Dalvik, і ART оптимізують код, що виконується, використовуючи механізм компіляції just-in-time (JIT) - компіляція відбувається під час виконання програми, що дозволяє оптимізувати код для виконання на конкретному пристрої. При цьому байт-код програми можна переносити на інші пристрої.

ART може компілювати байт-код заздалегідь, а не під час виконання, використовуючи ahead-of-time (AOT). Система сама вирішує, коли та які програми необхідно скомпілювати. Наприклад, коли пристрій не завантажено та підключено до заряджання. При цьому ART враховує інформацію про програму, зібрану під час попередніх запусків, що дає додаткову оптимізацію.

4. Native C/C++ Libraries - набір бібліотек, написаних мовами C або C++ та використовуваних різними компонентами ОС.

Приклади бібліотек:

– WebKit являє собою движок веб-браузера та надає доступ до інших функцій, пов’язаних з браузером.

– Media Framework надає медіа-кодеки, що дозволяють записувати та відтворювати різні медіа-формати.

– OpenGL – використовується для відображення 2D та 3D графіки.

— SQLite - движок бази даних, що використовується в Android для зберігання даних додатків на пристрої.

5. Java API Framework (Application Framework) - набір API, написаний на мові Java і надає розробникам доступ до всіх функцій Android. Ці API-інтерфейси утворюють будівельні блоки, необхідні для створення додатків, спрощуючи повторне використання основних, модульних, системних компонентів та сервісів, таких як [7]:

— Activity Manager надає доступ до керування життєвим циклом програми та забезпечує загальний навігаційний стек зворотних викликів.

— Window Manager надає доступ до керування вікнами додатків та є певною абстракцією бібліотеки Surface Manager.

— Content Providers дозволяє додатку отримувати доступ до даних інших програм або обмінюватися власними даними, тобто надає механізм обміну даними між поточними додатками.

— View System містить будівельні блоки для створення інтерфейсу програми (списки, тексти, кнопки і т. д.), а також управляє подіями елементів інтерфейсу користувача.

— Package Manager керує різними видами інформації, пов'язаними з пакетами програм, які в даний час встановлені на пристрої.

— Telephony Manager дозволяє програмі використовувати можливості телефонного сервісу пристрою.

— Resource Manager забезпечує доступ до таких ресурсів, як локалізовані рядки, растрові зображення, графіка та макети.

— Location Manager надає можливість позиціонування.

— Notification Manager відповідає за повідомлення у рядку стану.

6. System Apps – поверхневий рівень в архітектурі Android, який включає ряд системних (передвстановлених) додатків і багато інших додатків, які встановлює користувач.

Системні програми на всіх пристроях різні, але всі вони є встановленими виробниками пристрою (додаток для SMS-повідомлень, календар, карти, браузер, контакти тощо).

Цей рівень використовує всі рівні нижче, якщо дивитися на схему, для правильного функціонування програм.

1.1.2 Операційна система iOS

iOS (рис. 1.4) — це мобільна операційна система, створена та розроблена Apple Inc. виключно для свого апаратного забезпечення [8].

Рис. 1.4 Логотип iOS

Багатошаровість архітектури iOS передбачає упаковку технології у фреймворки [9]. Фреймворк зазвичай включає файли заголовків, зображення та всі необхідні загальні бібліотеки.

У ньому є чотири рівні абстракції (рис. 1.5) [10]:

- Core OS.
- Core Services.
- Media.
- Cocoa Touch.

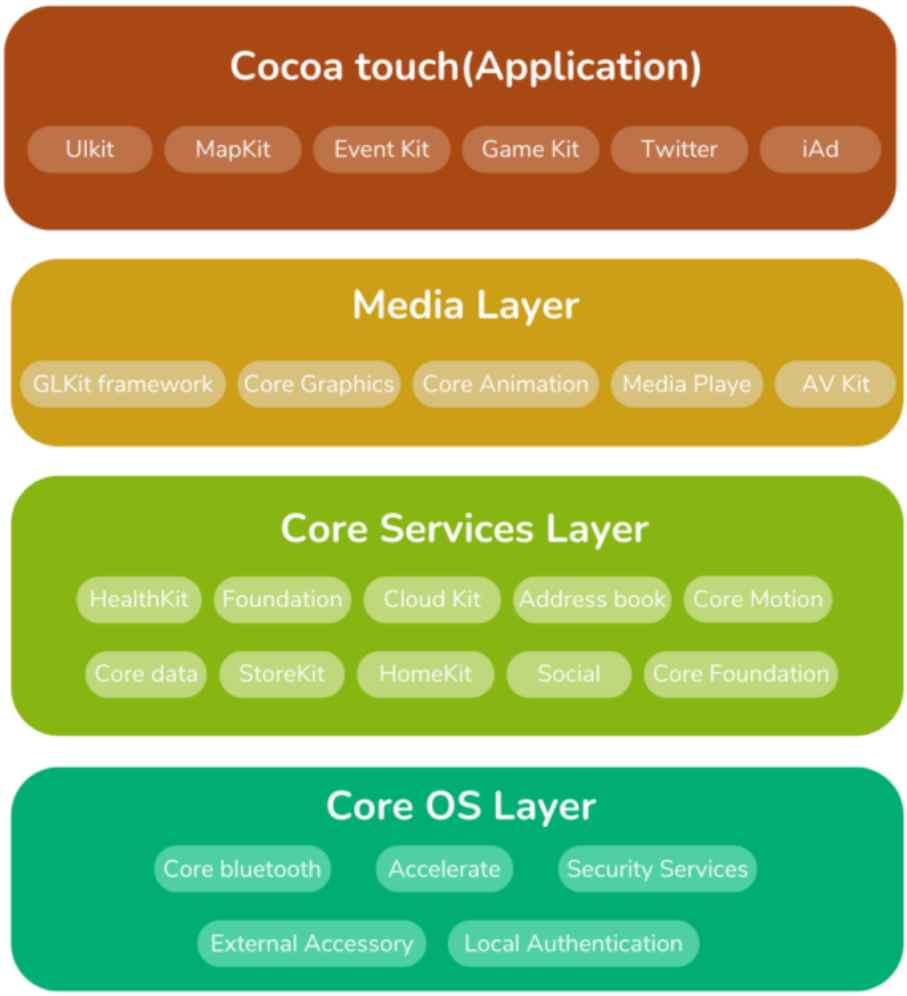


Рис. 1.5 Архітектура iOS

1. Core OS є нижнім рівенем у стеку iOS, він знаходиться безпосередньо поверх апаратного забезпечення пристрою. Функції низького рівня, які є основою всіх функцій iOS, надаються базовим рівнем ОС. На додаток до стандартних функцій основної операційної системи, таких як керування пам'яттю, обробка файлової системи та потоки, вона також пропонує ряд послуг, у тому числі низькорівневу мережу та доступ до зовнішніх аксесуарів.

Функції нижнього рівня, на яких базується більшість інших технологій, зберігаються на рівні Core OS:

— Основна структура Bluetooth: ця структура взаємодіє з пристроями BR/EDR («Класичний») і пристроями Bluetooth з низьким енергоспоживанням.

— External Accessories Framework обмінюється даними з аксесуарами пристроїв, підключеними через Bluetooth або роз'єм Apple Lightning.

— Accelerate Framework виконує масштабні математичні обчислення та обчислення зображень, оптимізовані для високої продуктивності та низького енергоспоживання.

— Платформа служб безпеки контролює доступ до програми та даних, які вона зберігає.

— Локальна структура авторизації автентифікує користувачів за допомогою їхніх біометричних даних або паролю.

2. Core Services - сервіси, які пропонує рівень Core OS, абстрагуються рівнем Core Services. Він надає основні служби, якими можуть користуватися всі програми. Рівень основних служб, як і інші рівні, надає набір інфраструктур:

— Accounts дозволяє користувачам отримувати доступ до своїх зовнішніх облікових записів і керувати ними безпосередньо з програми без необхідності вводити інформацію для входу.

— Address Book дозволяє отримати доступ до контактної інформації користувача.

— Ad Support надає програмам доступ до ідентифікатора для реклами.

— CFNetwork керує змінами конфігурації мережі та доступом до мережевих служб.

— Core Data - технологія, яка використовується для керування моделлю даних програми Model View Controller (MVC).

— Core Foundation надає інтерфейси для програм iOS, які забезпечують базові функції керування даними та обслуговування.

— Core Location підтримує надання даних розташування програми та заголовка.

— Core Media використовує основні типи даних для представлення аудіовізуального вмісту на основі часу.

— Core Motion використовується для доступу до всіх даних на основі руху на пристрої.

— Core Telephony надає інформацію про доступ до постачальника послуг стільникового зв’язку користувача, наприклад підтримку оператором VoIP та унікальну ідентифікацію.

— EventKit забезпечує доступ до даних календаря та нагадувань, щоб користувачі могли створювати, отримувати та редагувати елементи календаря у програмі.

— Foundation забезпечує базовий рівень функціональності для додатків і фреймворків, включаючи зберігання та збереження даних, обробку тексту, обчислення дати й часу, сортування та фільтрацію та роботу в мережі. Класи, протоколи та типи даних, визначені Foundation, використовуються в пакетах SDK для macOS, iOS, watchOS і tvOS.

— Mobile Core Services забезпечує доступ до функцій служби запуску та ідентифікації, і керування ними.

— NewsstandKit допомагає розвивати клієнтську сторону програми Newsstand. За допомогою Newsstand користувачі можуть переглядати газети та журнали, оптимізовані для перегляду на мобільних пристроях.

— PassKit створює та розповсюджує пропуски для програми Wallet і приймає платежі (наприклад, Apple Pay) у програмі.

— Quick Look створює попередній перегляд файлів для використання у програмі або легко редагує попередній перегляд.

— Social використовує загальні системні інтерфейси, розміщує матеріали на підтримуваних сайтах соціальних мереж.

— StoreKit framework надає доступ до покупок в програмі та підтримує взаємодію з AppStore.

— System Configuration дозволяє програмам отримувати доступ до параметрів конфігурації мережі на пристрої, перевіряти доступність пристрою, наприклад, чи активне з’єднання Wi-Fi або мобільний зв’язок.

3. Media Layer надає можливість використовувати мультимедійні послуги з мультимедійного рівня на iPhone. Це дозволяє розробнику працювати з графічними елементами, такими як анімація, зображення, відео та аудіо. Apple часто пропонує залишити відгук про мультимедійний досвід, особливо про якість аудіо та відео. Медіарівень стека iOS, який надає iOS доступ до аудіо, відео, графіки та можливостей AirPlay, по суті, виконує цей обов'язок. Подібно до інших рівнів системи, медіа-рівень має низку фреймворків, які можуть використовувати програмісти:

— Assets Library надає доступ до медіа-бібліотеки ресурсів користувача.

— AV Foundation працює з аудіовізуальними ресурсами, керує налаштуваннями камери, редагує аудіо та налагоджує взаємодію зі звуком.

— Core Audio надає можливість взаємодії інтерфейсом аудіоапаратного забезпечення пристрою.

— Core Graphics є власним механізмом рендерінгу додатка iOS і підтримує спеціальний 2D-вектор і візуалізацію на основі зображень.

— Core Image забезпечує розширену підтримку для керування відео та нерухомими фотографіями

— Core MIDI надає API для зв'язку з пристроями MIDI включаючи апаратні клавіатури та синтезатори.

— Core Text надає низькорівневий інтерфейс програмування для розміщення тексту та обробки шрифтів. Механізм макета Core Text має декілька переваг, а саме високу швидкодію, простоту використання та тісну інтеграцію з Core Foundation. API макета тексту забезпечує високоякісний набір тексту, включаючи перетворення символів у гліф із лігатурами, кернінгом тощо.

— Core Video використовує API на основі пайплайнів з підтримкою Metal і OpenGL для обробки цифрового відео, включаючи покадрове редагування.

— Image I/O забезпечує доступ до метаданих зображення, а також читає та записує більшість типів файлів зображень.

— GLKit керує розширеним двовимірним і тривимірним рендерингом за допомогою API з апаратним прискоренням.

— Media Player знаходить і відтворює пісні, аудіоподкасти, аудіокниги та інші медіафайли у програмі.

— OpenAL framework є стандартною технологією обробки аудіо.

— OpenGL ES керує потужним двовимірним і тривимірним рендерингом за допомогою інтерфейсів з апаратним прискоренням.

— Quartz Core дозволяє користувачам переглядати, змінювати та зберігати фотографії за допомогою слайд-шоу та фільтрів Core Image.

4. Рівень Cocoa Touch пропонує рівень абстракції, який робить доступними різні бібліотеки для програмування iPhone та інших пристроїв iOS. Важлива група фреймворків Objective-C, створених за допомогою Mac OS X Cocoa API, включена до рівня Cocoa Touch. Цей рівень підтримує сповіщення, багатозадачність, сенсорне введення, усі системні служби високого рівня та інші технології. Він також надає базову інфраструктурну підтримку програми.

Нижче наведено список основних фреймворків, які часто використовуються на цьому рівні:

— Address Book UI - ця структура отримує контакти користувачів і представляє їх у графічному інтерфейсі.

— Event Kit UI описує загальний системний інтерфейс, який використовує контролери перегляду для відображення та зміни подій.

— Game Kit дозволяє користувачам ділитися своїми даними, пов’язаними з грою, онлайн через Game Center.

— iAd - цей фреймворк дозволяє відображати банерну рекламу у додатку, що є основним методом монетизації.

— Map Kit надає прокручувану карту, яку можна включити в інтерфейс користувача програми.

— Message UI створює інтерфейс для електронної пошти та текстових повідомлень, щоб користувачі могли оновлювати та надсилати повідомлення, не виходячи з програми.

— UI Kit: цей фреймворк забезпечує важливу основу для розробки графічних, керованих подіями програм для iOS.

1.2 Аналіз методів та засобів розробки нативних мобільних застосунків

Розробка нативного додатку означає створення мобільної програми, яка адаптована та призначена лише для певної платформи, наприклад iOS або Android, та пишеться відповідномовою програмування [11].

Оскільки нативні додатки створено спеціально для операційної системи, вони забезпечують кращий користувацький досвід, ніж гібридні додатки. Нативні мобільні додатки зазвичай працюють і виглядають краще, ніж їхні веб-аналоги, які мають обслуговувати численні платформи. Крім того, нативні мобільні додатки мають доступ до розробки обладнання та можливостей, таких як датчики та камери, які недоступні через інтерфейс мобільного браузера.

Нативні мобільні програми, на відміну від веб-сайтів і веб-додатків, не працюють у браузері. Їх потрібно завантажувати з магазинів додатків для певної платформи, таких як Apple App Store і Google Play.

Нативні програми створювати складніше, ніж веб-сайти для мобільних пристроїв. Однак нативні програми надають розробникам можливість використовувати власні можливості мобільних операційних систем, щоб створити більш багатий користувацький досвід.

Переваги розробки нативних програм [12]:

1. Нативні додатки мають найкращу швидкодію. Додаток створено та оптимізовано для певної платформи. Як наслідок, додаток може похвалитися

значною швидкодією. Нативні додатки є швидкими, оскільки вони розроблені для однієї платформи та скомпільовані з їх основною мовою програмування та API. В результаті додаток стає значно ефективнішим та швидким. Програма зберігається на гаджеті, що дозволяє програмі ефективно використовувати обчислювальну потужність пристрою. Матеріальні та візуальні аспекти нативного мобільного додатку вже зберігаються на телефонах користувачів, що призводить до швидкого завантаження.

2. Нативні програми безпечніші. Веб-додатки створюються за допомогою різних технологій та можуть виконуватися в будь-якому браузері. Саме фактор «плаваючої» середи виконання додатку погано впливає на безпеку додатку. Розробка нативної мобільної програми є чудовим способом гарантувати постійну безпеку даних користувачів.

3. Нативні додатки більш інтерактивні та інтуїтивно зрозумілі. Нативні мобільні додатки значно краще реагують на введення та виведення даних користувачами. Ці програми враховують ОС своїх пристроїв, створюючи враження, що вони є невід'ємною частиною пристрою.

Найвагоміша перевага нативних мобільних додатків — кращий досвід користувача. Нативні мобільні додатки розроблені спеціально для конкретної операційної системи. Вони дотримуються суворих стандартів, які зрештою покращують і синхронізують роботу користувача з конкретною операційною системою. Як результат, робота додатку є більш зрозумілою для користувача, оскільки кожна платформа має свої критерії інтерфейсу користувача.

Користувачі можуть взаємодіяти з програмами за допомогою дій і жестів, з якими вони вже знайомі, якщо вони дотримуються певних стандартів.

4. Розробники мають можливість отримати доступ до повного набору апаратних функцій, що надає їм пристрій.

Нативні програми написані спеціально для платформи та використовують всі можливості пристрою. Ці програми можуть отримати миттєвий доступ до апаратного забезпечення пристрою, наприклад GPS, камери та мікрофона, що

робить їх більш швидкими у виконанні. Push-сповіщення є ще однією значною перевагою вибору розробки нативної програми. Розробник матиме доступ до всіх API та інструментів, які пропонує платформа, для якої ви розробляєте.

5. Краща підтримка магазину додатків. Нативні програми часто легше публікувати та займати вищі позиції в магазині додатків, ніж гібридні програми, оскільки вони забезпечують чудову швидкодію і користувацький досвід.

Але у нативних додатків є і мінуси, а саме:

1. Високі затрати у часі та ресурсах – при створенні власної програми, запуск як для iOS, так і для Android може бути дорогим. Відповідно замовнику потрібно буде найняти дві окремі команди для роботи на різних платформах.

2. Оновлення. Розробники часто вводять нові версії в нативних програмах з різних причин. Найпоширенішою причиною є усунення помилок і несправностей. Як наслідок, оновлення повинні будуть завантажені користувачем, щоб запрацювати, тоді як, наприклад, веб-додатки не потребують завантажувати нічого додатково.

3. Вимоги до завантаження. Завантаження нативної програми передбачає кілька процедур. Користувачі повинні знайти програму, переглянути умови та положення, а потім завантажити її. Процедура завантаження може зайняти багато часу, тому що нативні додатки зазвичай значно «важчі», якщо порівнювати їх з веб-додатками [13]. Однак, після завантаження та встановлення на відповідний пристрій, додаток запускається швидше, за рахунок зберігання необхідних ресурсів на самому пристрої. Тому не можна однозначно сказати, зо така модель роботи є недоліком нативних додатків.

1.3 Аналіз методів та засобів створення гібридних мобільних додатків

Гібридний мобільний додаток – це додаток, що поєднує в собі елементи нативного додатку і веб-додатку. Гібридні додатки — це, по суті, веб-додатки,

які розміщено у нативній оболонці. Після того, як їх завантажено з магазину додатків і встановлено локально, оболонка може підключитися до будь-яких можливостей, які надає мобільна платформа, через браузер, вбудований у програму. Браузер і його плагіни працюють на сервері та невидимі для кінцевого користувача [14]. Прикладом технології для створення гібридних додатків є Apache Cordova (рис. 1.6).

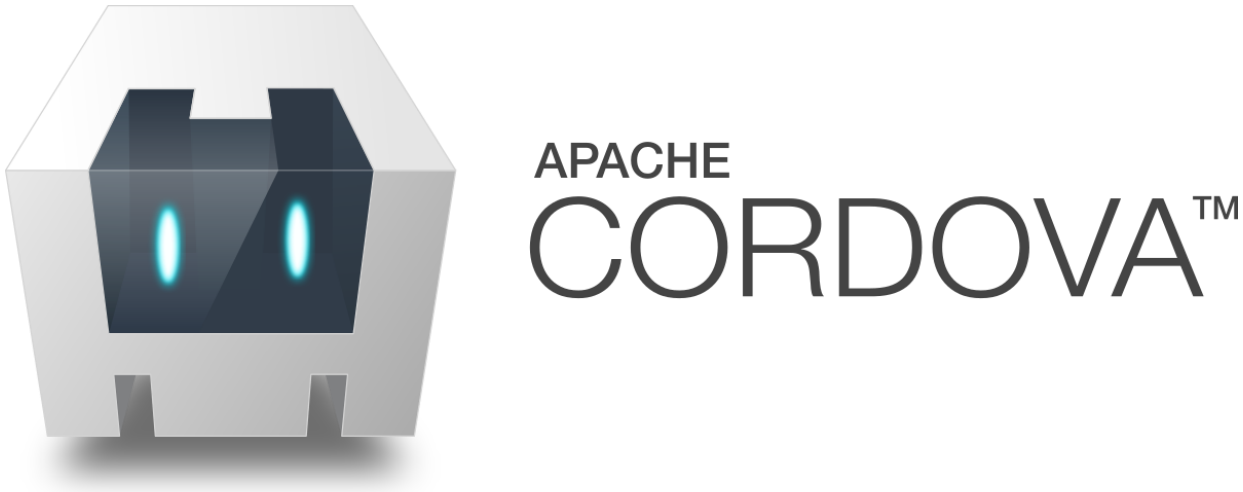


Рис. 1.6 Логотип Apache Cordova

Гібридні додатки запускають код усередині контейнера. Механізм браузера пристрою використовується для відтворення HTML і JavaScript, а також власних API для доступу до апаратного забезпечення пристрою [15].

Незважаючи на те, що гібридна програма зазвичай має веб-подібні елементи навігації, чи зможе програма працювати в автономному режимі, залежить від її функцій. Якщо програмі не потрібна підтримка з боку бази даних, її можна змусити працювати в автономному режимі.

Гібридні додатки популярні, оскільки вони дозволяють розробникам написати код для мобільного додатка один раз і все одно підтримують кілька платформ. Оскільки гібридні програми додають додатковий рівень між вихідним

кодом і цільовою платформою, вони можуть працювати дещо повільніше, ніж нативні або веб-версії тієї самої програми [16].

Переваги гібридних додатків включають:

— Буде працювати на різних платформах.

— Швидший час створення порівняно з нативними додатками.

— Дешевше розробити порівняно зі створенням двох версій нативної програми для двох різних платформ.

— Легше запускати виправлення та оновлення.

— Може працювати онлайн і офлайн.

Деякі недоліки, однак, включають:

— Зовнішній вигляд програми може відрізнятися залежно від платформи.

— Необхідність перевірити додаток на ряді пристроїв, щоб переконатися в належній роботі.

— Взаємодія з користувачем (UX) може погіршитися, якщо інтерфейс користувача (UI) не схожий на веб-переглядачі, до яких звик користувач, і не достатньо добре розроблений для цього.

— Гірша швидкодія, порівняно з нативними додатками.

1.4 Кросплатформенні мобільні додатки

Основна проблема розробки нативних програм полягає в тому, що кожен пристрій (iOS, Android, веб) потребує власної кодової бази.

Це може швидко стати великою проблемою обслуговування для компаній, оскільки вони фактично потроюють обсяг роботи, необхідний для підтримки свого додатка в актуальному стані.

Фреймворки для розробки кросплатформенні додатків (рис. 1.7) вирішують цю проблему, дозволяючи використовувати єдину кодову базу для всіх пристроїв, що значно полегшує процес розробки [17].

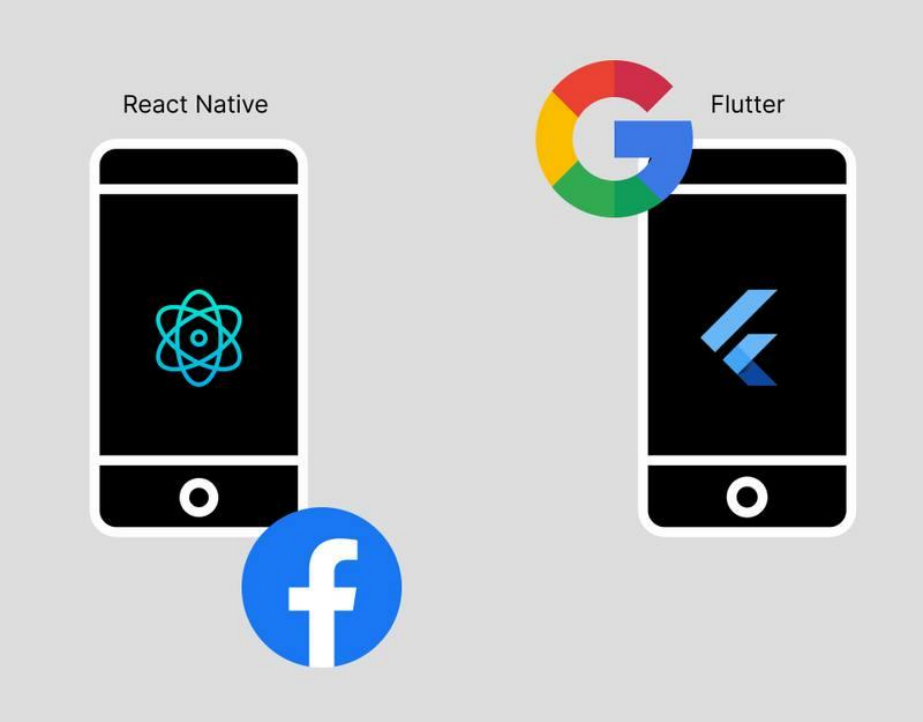


Рис. 1.7. Логотипи кросплатформенних фреймворків

Методи роботи кросплатформенного мобільного додатку залежать від реалізації та підходів конкретного фреймворку, адже кожен кросплатформенний фреймворк реалізує роботу у свій спосіб.

Переваги кросплатформенної розробки додатків:

– Зниження витрат на розробку та швидший час виходу на ринок.

– Покращений користувацький досвід на всіх пристроях. Ця перевага залежить від того, який фреймворк буде обрано, але в цілому взаємодія з користувачем буде більш узгодженою, оскільки код написаний з однієї кодової бази з наміром виглядати та діяти однаково.

– Простіше обслуговування та налагодження. Одна кодова база вимагає менше налагодження, ніж кілька кодових баз. Якщо помилка поширюється на різні кодові бази, виправлення однієї може не працювати на іншій.

– Менше витрат у часі для розробників, щоб бути в курсі останніх змін. Додатки розробляються із залежностями – бібліотеками та іншими

технологіями, які використовуються. Вони мають бути в актуальному стані, щоб забезпечити безперебійне функціонування. Набагато легше це зробити, коли ви керуєте лише однією кодовою базою.

Приклади кросплатформенних мобільних фреймворків:

1. React Native (рис. 1.8) - спочатку створений Facebook, він дозволяє розробникам створювати нативні програми для iOS і Android за допомогою єдиної кодової бази JavaScript [18].

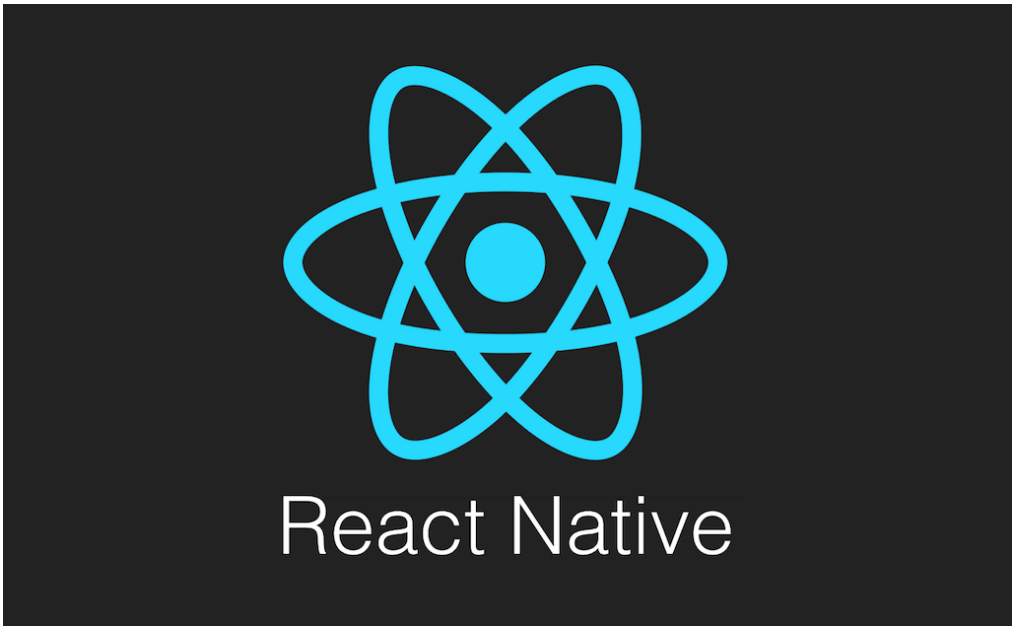


Рис. 1.8. Логотип React Native

Особливість React Native полягає в тому, що він працює за допомогою програмних мостів, що з’єднують потік Javascript та основний потік.

Оскільки Javascript бібліотека React, що використовується для розробки веб-застосунків має дуже велику популярність та велику спільноту розробників, то перехід з React до React Native для розробників є більш простим, через однакові методи і підходи в цих програмних засобах. Це одна з головних причин, чому він досяг такої популярності серед технічної спільноти.

Він не має власного механізму візуалізації, як Flutter, але він забезпечує менш заплутаний досвід розробника, поставляється з чудовими плагінами сторонніх розробників і має широку спільноту розробників.

2. Flutter — це кросплатформенний фреймворк для розробки мобільних додатків (рис. 1.9), створений Google. Flutter набуває все більшу популярність через те, що він забезпечує узгоджену взаємодію з користувачем на всіх пристроях, і його легше підтримувати та налагоджувати [19].

Рис. 1.9. Логотип Flutter

Flutter розроблено як розширювану багатошарову система. Він існує як ряд незалежних бібліотек, кожна з яких залежить від базового рівня. Жоден рівень не має привілейованого доступу до нижнього рівня, і кожна частина рівня структури створена як необов'язкова та замінна.

Flutter не потребує жодних специфічних для платформи компонентів інтерфейсу користувача для відтворення свого інтерфейсу. Це означає, що немає потреби в трансформуванні відображення властивостей для відображення кожної анімації у віджет певної платформи.

Спеціальна платформа для вбудовування забезпечує точку входу, координує роботу з базовою операційною системою для доступу до таких служб, як рендеринг інтерфейсу, доступність і ввід даних, і керує циклом подій повідомлення. Вбудовувач написано мовою, яка підходить для платформи: наразі Java і C++ для Android, Objective-C/Objective-C++ для iOS і macOS і C++

для Windows і Linux. Використовуючи embedder, код Flutter можна інтегрувати в існуючу програму як модуль або код може являти собою весь вміст програми. Flutter містить кілька вбудованих пристроїв для поширених цільових платформ, але існують і інші вбудовані пристрої.

3. Xamarin (рис. 1.10) — це потужний і багатофункціональний фреймворк для розробки програм. Його технологічний набір складається з C#, який дозволяє розробникам кодувати як для Android, так і для iOS однією мовою [20].

Рис. 1.10 Логотип Xamarin

Xamarin дотримується шаблону проектування Model-View-Controller, який знайомий більшості програмістів, які працювали з серверними програмами.

Програми Xamarin.Android компілюються з мови C# в проміжну мову (IL), який при запуску програми зазнає Just-in-Time-компіляції (JIT) в машинну сборку. Програми Xamarin.Android працюють у середовищі виконання Mono паралельно з віртуальною машиною середовища виконання Android (ART). Xamarin надає прив'язки .NET до просторів імен Android.* та Java.*.

Програми Xamarin.iOS проходять повну Ahead-of-Time-компіляцію (AOT) з мови C# у свій код збірки ARM. Xamarin використовує селектори для надання коду Objective-C керованого коду C# і Registrars для надання керованого коду C# коду Objective-C. Селектори і Registrars разом називаються "прив'язками" і забезпечують взаємодію між Objective-C і C#.

1.5 Висновок до першого розділу

В цьому розділі була проаналізована поточна ситуація в індустрії мобільних додатків, а саме були проаналізовані найпоширеніші операційні системи для мобільних пристроїв – Android та iOS, методи та підходи до розробки мобільних додатків на цих платформах.

Серед підходів були проаналізовані наступні:

- Розробка нативних мобільних додатків окремо під кожну ОС.
- Розробка гібридного додатку, що використовує технології веб-додатків, але запакований у нативну оболонку.
- Розробка кросплатформенного додатку.

Кожен з цих підходів має свої переваги та недоліки, наприклад розробка окремого нативного мобільного додатку для кожної платформи має найвищий потенціал у наданні користувачу найкращого користувацького досвіду, але вимагає найбільших витрат у ресурсах при розробці. В той же час гібридний мобільний додаток порівняно дешевий у розробці, але обмежений можливостями вбудованого браузера.

Через недоліки вищеперерахованих підходів, можна вважати, що підхід створення кросплатформенного додатку є оптимальним, через те що має єдину кодову базу, та водночас підтримує користувацький досвід на високому рівні.

РОЗДІЛ 2
АНАЛІЗ МЕТОДІВ РОБОТИ ТА РОЗБІР ФРЕЙМВОРКУ REACT NATIVE

2.1 Розбір роботи бібліотеки React

В основі фреймворку React Native лежить бібліотека React, що набула великої популярності в сфері розробки веб-додатків і досі займає лідируючі позиції серед бібліотек та фреймворків Javascript. Відповідно до глобального опитування State of JS, бібліотека займає перше місце по використанню розробниками вже 5 рік поспіль (рис. 2.1) [21].

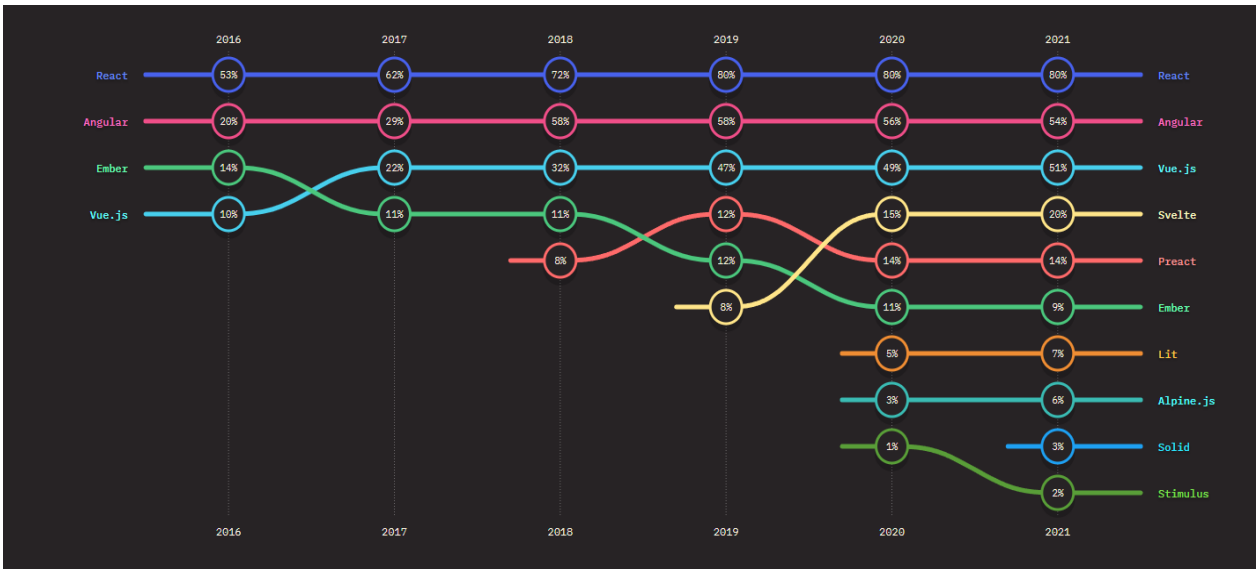


Рис. 2.1. Діаграма використання бібліотек та фреймворків розробниками

React — це бібліотека (рис 2.2), яка допомагає розробникам створювати користувацькі інтерфейси як дерево невеликих фрагментів, які називаються компонентами. Компонент — це суміш HTML і JavaScript, яка містить всю логіку, необхідну для відображення невеликої частини інтерфейсу [22].

Рис. 2.2. Логотип React

Сучасні веб-додатки, як правило, дотримуються так званої моделі односторінкових додатків (SPA). Це означає, що користувач ніколи не переходить на інші сторінки і навіть не перезавантажує сторінку. Натомість, контент додатку завантажується на ту саму сторінку (рис. 2.3) [23].

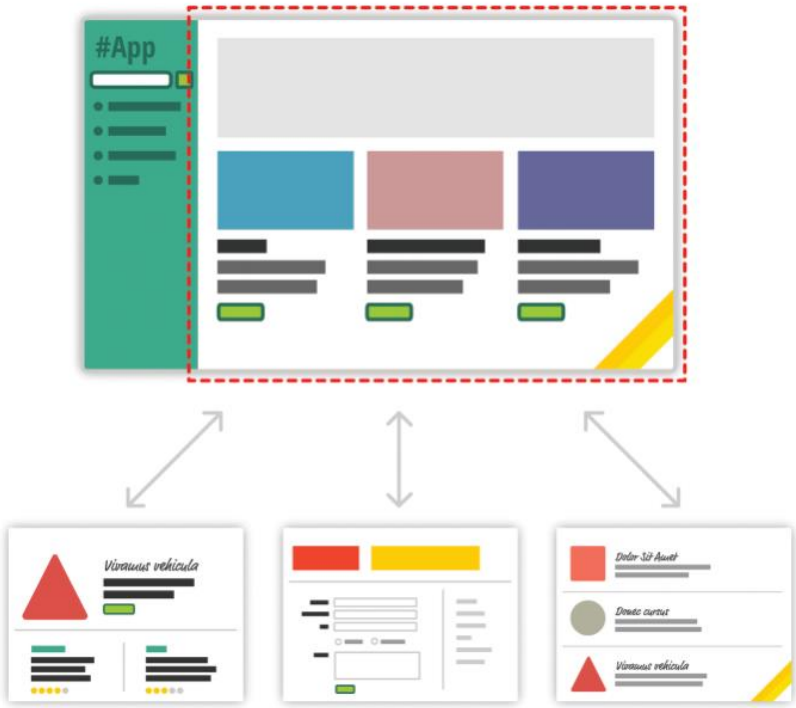


Рис. 2.3. Схема роботи односторінкового веб-додатку

З односторінковими додатками відстежувати інтерфейс користувача та підтримувати стан складно та вимагає багато часу. З React потрібно слідкувати лише за одним: поточним станом інтерфейсу користувача. А бібліотека вже буде слідкувати за ним та оновлювати відповідні частини інтерфейсу.

2.1.1 Потік даних у React

У React дані передаються в одному напрямку, від батьківського до дочірнього (рис. 2.4). Це допомагає компонентам бути простими та передбачуваними у своїй роботі [24].

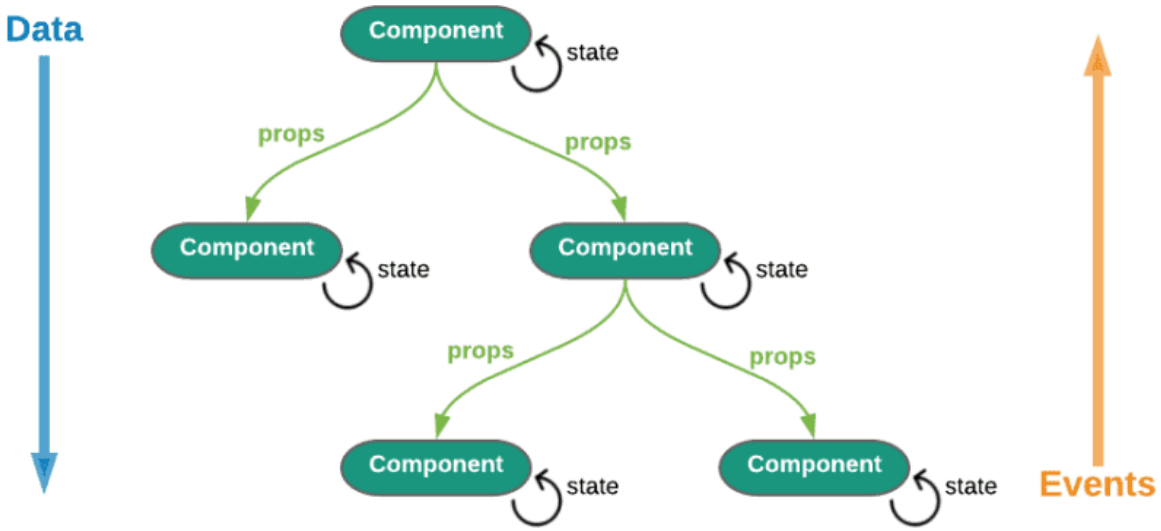


Рис. 2.4. Потік даних у React

Розглядайте компоненти React як прості функції, які отримують властивості, стан і повертають HTML. Коли дочірні компоненти отримують атрибути від своїх батьків, вони або застосовують модифікації (рендеринг), або передають їх іншому дочірньому компоненту, яка може їх використовувати.

2.1.2 Алгоритм Virtual DOM у React

Оскільки модифікації DOM дуже повільні, ви ніколи не змінюєте реальний DOM безпосередньо за допомогою React. Замість цього змінюється віртуальний DOM у пам’яті.

React використовує Virtual DOM, який схожий на полегшену копію фактичного DOM. Таким чином, для кожного об’єкта, який існує в оригінальній DOM, є об’єкт для нього в React Virtual DOM. Маніпулювання DOM відбувається повільно, але маніпулювання Virtual DOM відбувається швидко, оскільки на екрані нічого не перемальовується. Тому щоразу, коли відбувається зміна стану програми, спочатку оновлюється віртуальний DOM (рис. 2.5) [25].

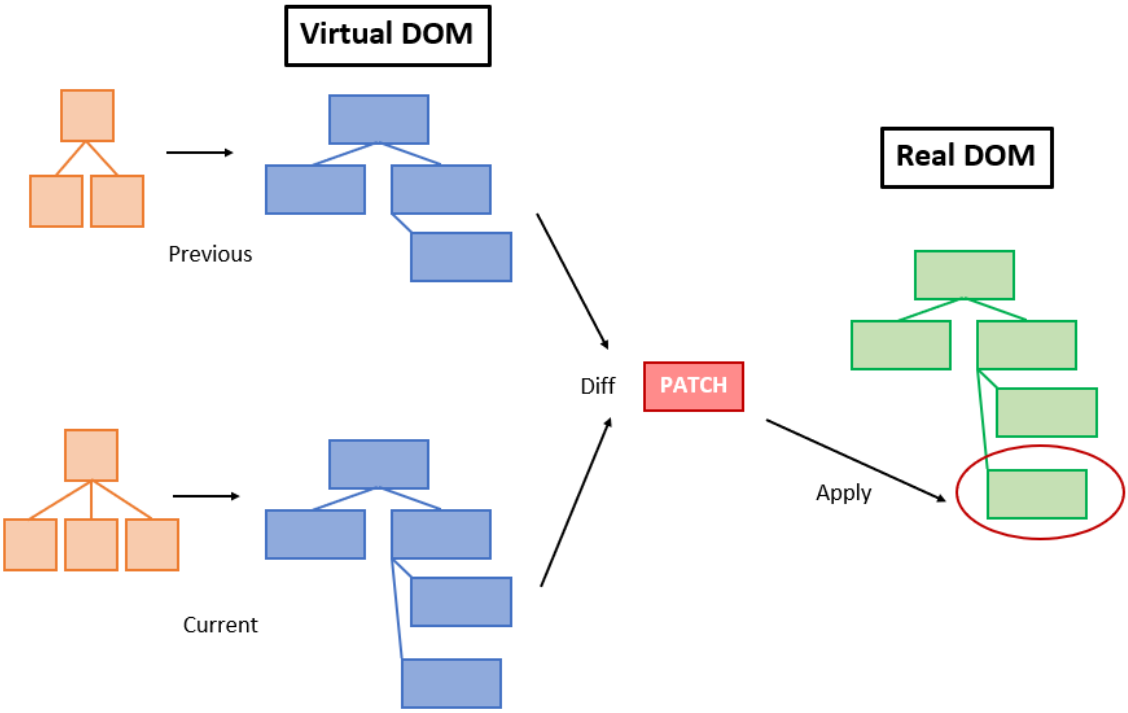


Рис. 2.5. Схема роботи Virtual DOM

У React все розглядається як компонент, будь то функціональний компонент або класовий компонент. Компонент може містити стан. Щоразу, коли щось змінюється в файлі JSX, або, кажучи простою мовою, щоразу, коли

змінюється стан будь-якого компонента, React оновлює своє віртуальне дерево DOM. Хоча може здатися, що це неефективно, але вартість не надто значна, оскільки оновлення віртуального DOM не займає багато часу. React підтримує дві віртуальні DOM кожного разу, одну містить оновлену віртуальну DOM, а іншу — лише версію до оновлення цієї оновленої віртуальної DOM. Тепер він порівнює версію до оновлення з оновленою віртуальною DOM і визначає, що саме змінилося в DOM, наприклад, які компоненти було змінено. Цей процес порівняння поточного дерева Virtual DOM з попереднім називається «відмінюванням». Коли React дізнається, що саме змінилося, він оновив лише ці об’єкти на реальному DOM. React використовує те, що називається пакетним оновленням, для оновлення справжнього DOM. Це просто означає, що зміни в справжньому DOM надсилаються пакетами замість надсилання будь-яких оновлень для однієї зміни стану компонента. Повторна візуалізація інтерфейсу користувача є найдорожчою частиною, і React вдається її оптимізувати гарантуючи, що Real DOM отримує пакетні оновлення для повторного рендерингу інтерфейсу. Весь цей процес перетворення змін у реальний DOM називається узгодженням та використовує евристичний алгоритм

Це значно покращує продуктивність і є основною причиною, чому React і його Virtual DOM дуже люблять розробники з усіх куточків світу.

2.1.3 Життєвий цикл компоненту у React

Веб-програми React насправді є набором незалежних компонентів, які працюють відповідно до взаємодії з ними. Кожен компонент React має власний життєвий цикл, життєвий цикл компонента можна визначити як ряд методів, які викликаються на різних етапах існування компонента. Компонент React може пройти чотири етапи свого життєвого циклу, як описано нижче [26].

1. Ініціалізація – це етап, на якому компонент створюється з заданими Props і стандартним станом. Це робиться в конструкторі класу компонентів.

2. Монтування – це етап візуалізації JSX, що повертається самим методом рендерингу.

3. Оновлення – це етап, на якому оновлюється стан компонента та програма перемальовується.

4. Демонтування є останнім етапом життєвого циклу компонента, на якому компонент видаляється зі сторінки.

React надає розробникам набір попередньо визначених функцій, які за наявності викликаються під час певних життєвих циклів компонента. Передбачається, що розробники замінюють функції за допомогою бажаної логіки для відповідного виконання (рис. 2.6) [27].

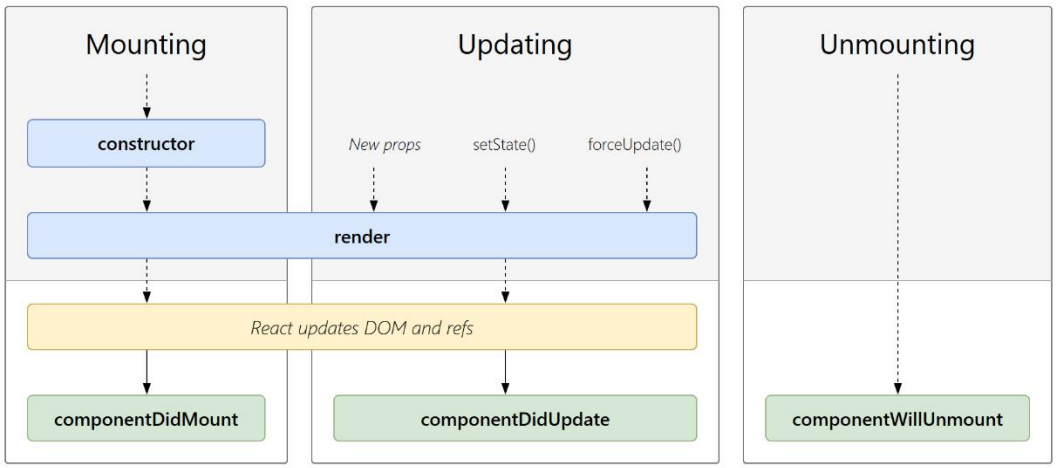


Рис. 2.6. Схема життєвого циклу та доступних методів

2.1.4 Управління станом додатку в React

Одна з головних задач, яка постає перед розробниками, що використовують React – це задача зберігання стану програми, а саме місця де його зберігати. Ця проблема лише загострюється з невпинним ростом додатку.

Зазвичай для вирішення проблеми збереження стану додатку, використовують сторонні бібліотеки, такі як Redux, MobX, Zustand, Recoil тощо.

Серед цих бібліотек найпопулярнішою є бібліотека Redux. Redux — це легкий інструмент керування станом, який допомагає компонентам у додатку спілкуватися один з одним. Проста концепція цього полягає в тому, що кожен стан компонента зберігається в сховищі, яке буде глобальним. Щоб кожен компонент міг отримати доступ до будь-якого стану з цього сховища [28].

Серед сильних сторін Redux можна відзначити:

– Доступ до стану: можна легко керувати станом програми, оскільки стан є глобальним.

– Простий зв’язок: допомагає компонентам легко спілкуватися один з одним, не надсилаючи жодних властивостей від одного компонента до іншого.

– Підтримка: Redux допомагає добре організувати кодову базу.

– Тестування та дебагінг: Redux допомагає легко тестувати код.

Redux складається з трьох частин (рис. 2.7) [29]:

1. Store – сховище зберігає стан усієї програми. Для однієї програми може бути лише одне сховище. Розробник може отримати доступ, оновити, зареєструвати або скасувати реєстрацію стану в сховищі.

2. Action - це подія. Це функція, що відповідно впливає на сховище і модифікує його.

3. Reducer — це чисті функції, які беруть поточний стан програми, виконують дію та повертають новий стан. Ці стани зберігаються як об’єкти, і вони визначають, як змінюється стан програми у відповідь на дію, надіслану до сховища.

Redux є доволі простою бібліотекою, але при розробці комплексних мобільних додатків часто потрібно взаємодіяти зі сторонніми сервісами, логувати роботу додатку тощо. Саме для цього в Redux є можливість створення middlewares.

Middleware Redux дозволяє перехоплювати кожну дію, надіслану редьюсеру, щоб мати змогу вносити зміни в дію або скасовувати дію. Middleware допомагає вести логи, повідомляти про помилки, робити асинхронні запити та багато іншого.

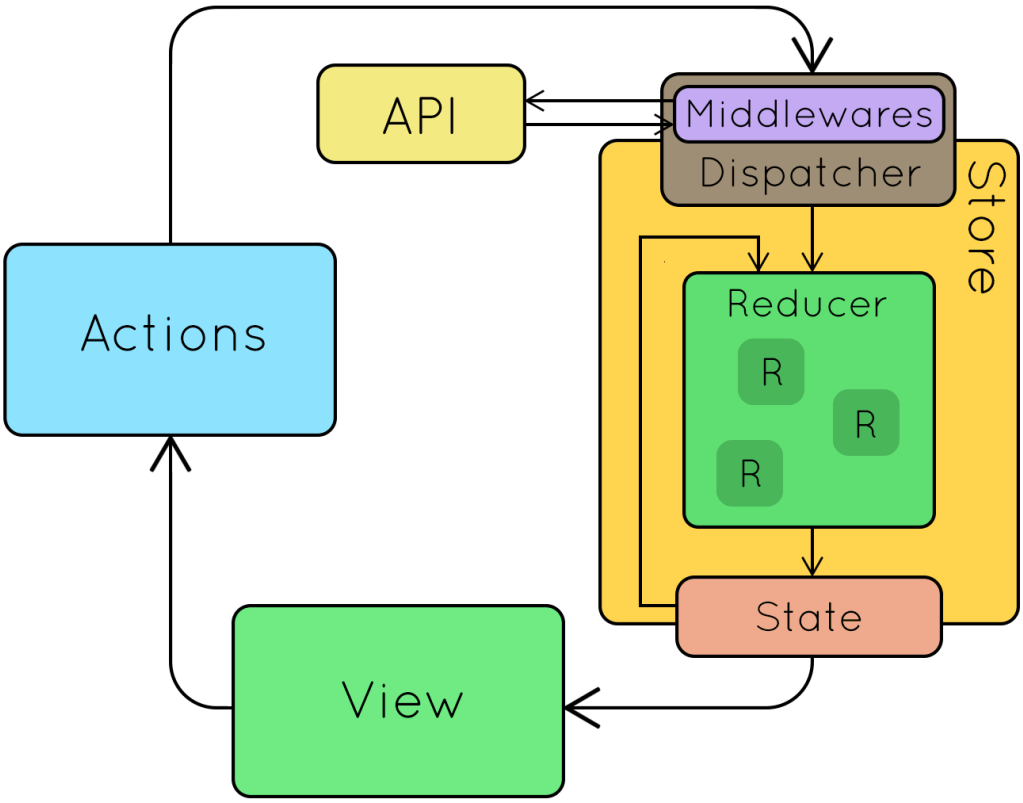


Рис. 2.7 Схема роботи Redux

2.2 Розбір роботи фреймворку React Native

React Native — це фреймворк JavaScript для написання мобільних додатків для iOS і Android, що відображаються нативно. Він заснований на React, наслідує всі його підходи і можливості, але замість того, щоб націлюватися на браузер, він націлений на мобільні платформи. Іншими словами: веб-розробники тепер можуть писати мобільні програми, які виглядають і відчуваються справді нативними, все це з використанням бібліотеки JavaScript. Крім того, оскільки

більша частина написаного коду може використовуватися між платформами, React Native дозволяє легко одночасно розробляти як для операційну систему Android, так і для iOS [30].

Подібно до React для веб-розробки, програми React Native написані з використанням суміші розмітки JavaScript і XML, відомої як JSX. Потім під капотом «міст» React Native викликає рідні API рендерингу в Objective-C (для iOS) або Java (для Android). Таким чином, програма відтворюватиметься за допомогою компонентів реального мобільного інтерфейсу користувача, а не веб-переглядів, і виглядатиме та працюватиме як будь-яка інша нативна мобільна програма. React Native також надає інтерфейси JavaScript для API платформи, тож програми React Native можуть отримувати доступ до таких функцій платформи, як камера телефону або місцезнаходження користувача.

Наразі React Native підтримує як iOS, так і Android і має потенціал для розширення на майбутніх платформах.

2.2.1 Архітектура фреймворку React Native

І iOS, і Android мають схожу архітектуру з тонкими відмінностями. Якщо розглядати загальну картину, то платформа React Native складається з трьох великих частин, що обмінюються інформацією між собою (рис. 2.8) [31]:

1. Нативний код/модулі: більшість нативного коду в iOS написано на Objective C або Swift, тоді як на Android — на Java або Kotlin. Але для написання додатка React Native навряд чи коли-небудь знадобиться писати нативний код для iOS або Android.

2. Javascript VM: віртуальна машина JS, яка запускає весь код JavaScript. На симуляторах і пристроях iOS/Android React Native використовує JavaScriptCore, який є механізмом JavaScript, який працює в Safari. JavaScriptCore — це двигун JavaScript з відкритим кодом, спочатку створений для WebKit. У

випадку iOS React Native використовує JavaScriptCore, наданий платформою iOS. Вперше він був представлений в iOS 7 разом з OS X Mavericks. У випадку Android React Native об'єднує JavaScriptCore разом із програмою. Це збільшує розмір програми.

3. React Native bridge — це міст C++/Java, який відповідає за зв'язок між нативним потоком і потоком Javascript. Для передачі повідомлень використовується спеціальний протокол.

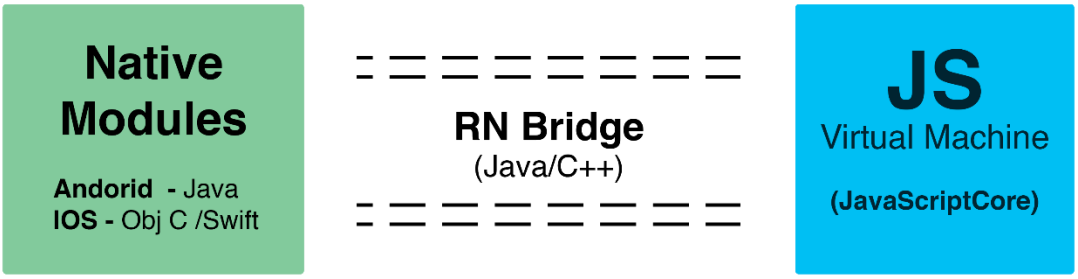


Рис. 2.8 Архітектура фреймворку React Native

2.2.2 Основні потоки в React Native

Коли програма React Native запускається, вона породжує наступні потоки, що взаємодіють між собою (рис. 2.9):

1. Native thread – це основний потік, який створюється, щойно програма запускається. Він завантажує програму та запускає потік JS для виконання коду Javascript. Рідний потік також прослуховує події інтерфейсу користувача, такі як «натискання», «дотик» тощо. Потім ці події передаються в потік JS через RN Bridge. Після завантаження Javascript потік JS надсилає інформацію про те, що потрібно відобразити на екрані. Ця інформація використовується потоком тіньового вузла для обчислення макетів. Потік тіней в основному схожий на математичний механізм, який остаточно вирішує, як обчислити позиції

перегляду. Потім ці інструкції передаються назад до основного потоку для відтворення представлення.

2. Потік Javascript (черга JS) – черга Javascript — це черга потоків, у якій виконується основний пакетний потік JS. Потік JS запускає всю бізнес-логіку, тобто код, який пишеться в React Native.

3. Користувальницькі нативні модулі. Окрім потоків, створених React Native, також можливо створювати потоки на спеціальних нативних модулях, які ми створюємо, щоб пришвидшити роботу програми. Наприклад, анімація обробляється в React Native окремим власним потоком, щоб розвантажити роботу з потоку JavaScript.

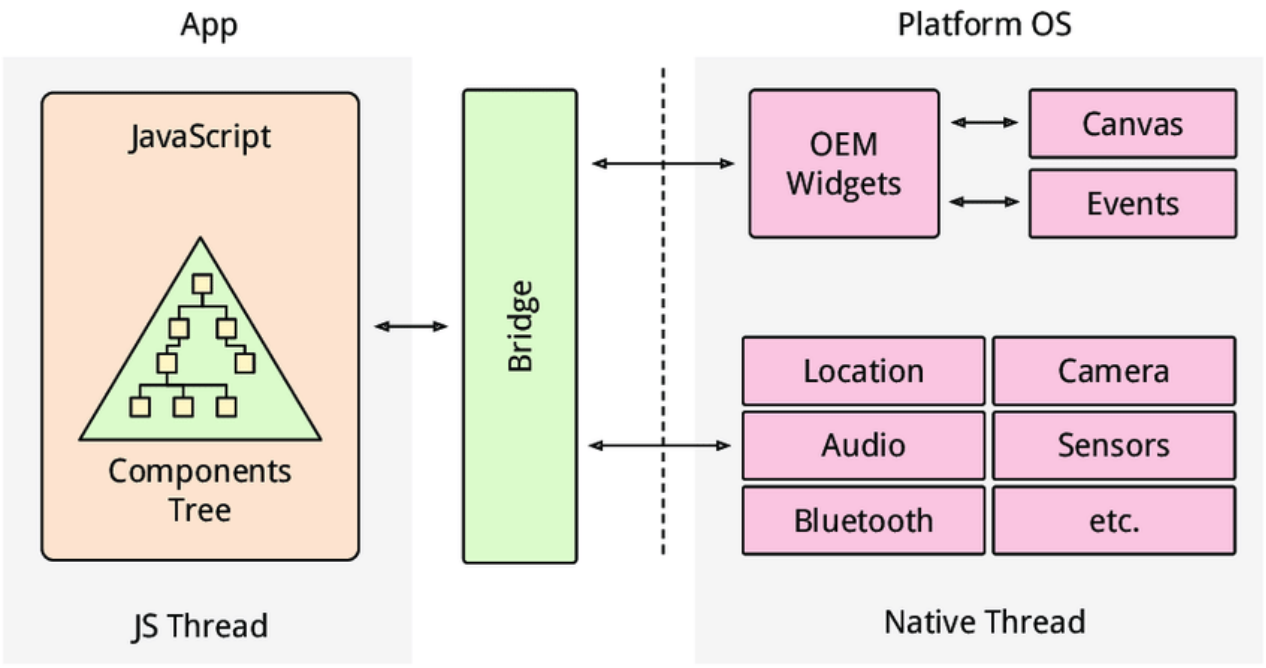


Рис. 2.9 Основні потоки React Native

2.2.3 Методи створення проєкту React Native

На сьогодні є два методи створення проєкту [32]:

1. React Native CLI — це вбудована функція, яка допомагає керувати проектом локально. За допомогою CLI можливо створювати та запускати програми. Він надає можливість створити проект, використовуючи команду: npx react-native init ProjectName.

Щоб запустити проект, можна виконати наступні команди:

- npx react-native start.
- npx react-native run-android.
- npx react-native run-ios.

За допомогою React Native CLI можливо додавати власні модулі, написані на Java/Objective-C, що є найсильнішою функцією.

Після створення проекту розробник має доступ до збірок, може контролювати їх і налаштовувати їх відповідно до своїх вимог. Таким чином, розробник має повний контроль над своєю програмою як на платформах Android, так і на IOS, та навіть може змінювати нативний код.

Найбільш очевидний недолік вражає користувачів Windows. Щоб створювати кросплатформенні мобільні програми за допомогою React Native CLI, знадобляться емулятори Andriod і IOS. Це означає, що потрібно мати Andriod Studio та XCode для запуску проектів, тоді як XCode дозволяється встановлювати лише на Mac. Використовуючи Windows можливо розробляти програми React Native і запускати їх на емуляторі Andriod Studio.

Ініціювання проекту за допомогою React Native CLI може зайняти багато часу. Крім того, це дуже тонкий процес, від якого залежить подальша робота.

В основному це пов’язано зі складною конфігурацією пристрою. Потрібно встановити змінні середовища та переконатися, що на поточному пристрої є потрібні SDK, щоб програма корректно працювала.

Базовий додаток «Hello World» створений за допомогою React Native CLI має наступну файлову структуру (рис. 2.10):

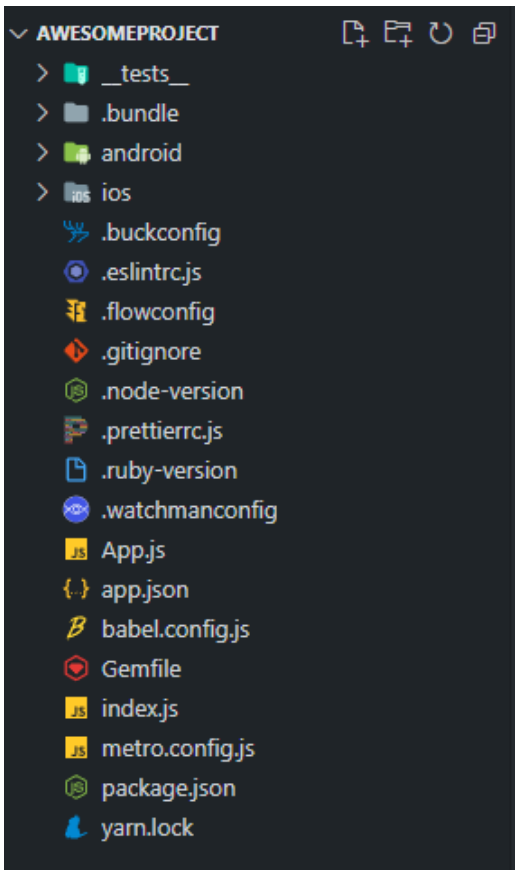


Рис. 2.10. Файлова структура додатку React Native CLI

2. Expo CLI побудовано на основі React Native, і це найшвидший спосіб налаштувати проект React Native. Він не вимагає налаштовувати конфігурації та оточення, а надає можливість створити проект і одразу почати роботу. Expo CLI можна встановити глобально через npm.

Створити та запустити програму React Native за допомогою Expo можна наступними способами:

— expo init ProjectName.

— npm start.

За допомогою Expo CLI можна легко налаштувати свій проект React. Не потрібен емулятор чи Mac, щоб запускати програми React Native із Expo. Можна завантажити клієнтську програму Expo для Android та IOS, а потім запустити проект на відповідному мобільному пристрої.

Після запуску програми за допомогою команди expo start або npm start інтерфейс DevTools у браузері відкриється з QR-кодом. Таким чином, можна поділитися цим QR-кодом і поділитися програмою в реальному часі протягом розробки. Відповідно можна уникнути використання файлів APK або IPA.

У Expo є деякі обмеження, наприклад неможливо додати нативні модулі. На відміну від React React Native CLI, розробник не може використовувати бібліотеки нативного коду, які використовують Objective-C/Java.

Базовий додаток «Hello World» створений за допомогою Expo має наступну файлову структуру (рис 2.11):

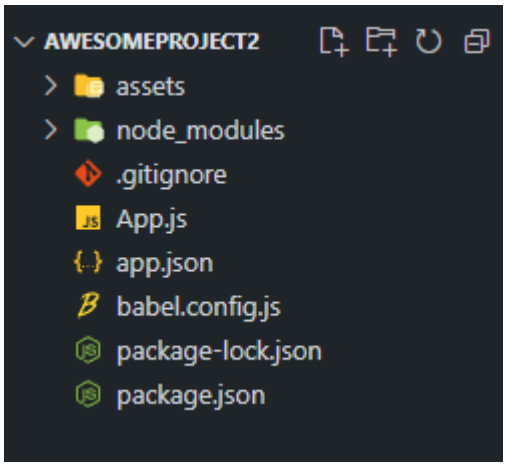
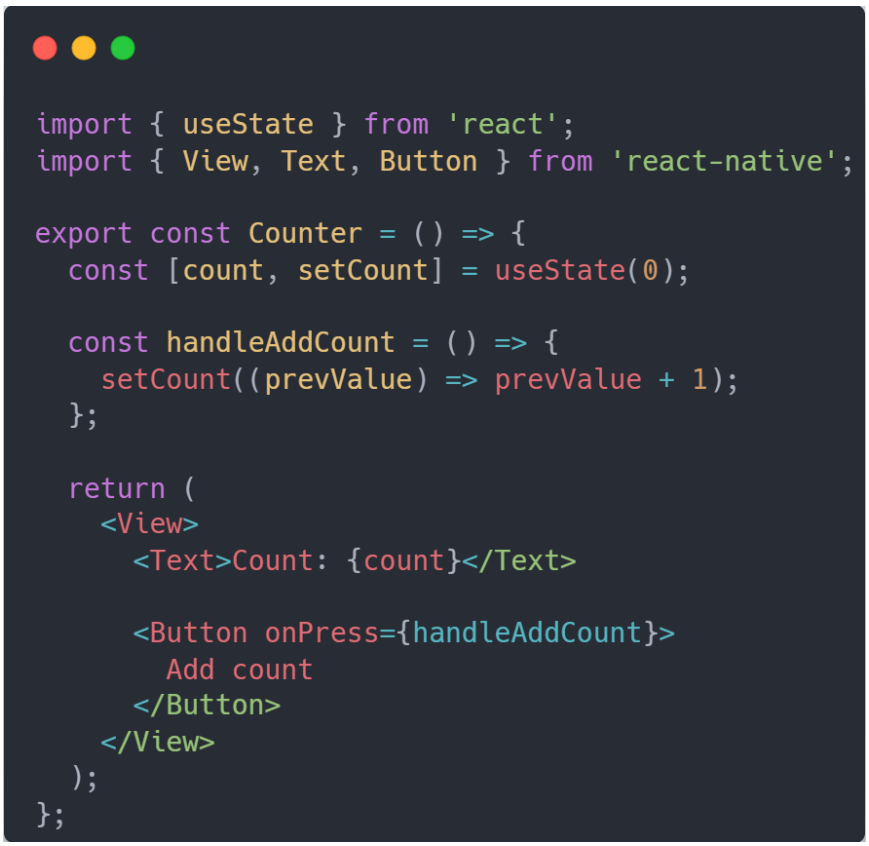


Рис. 2.11. Файлова структура проєкту Expo

2.2.4 Компоненти React Native

На відміну від React у вебі, де розробники використовують html-подібну розмітку, при розробці додатку за допомогою React Native розробник повинен використовувати компоненти з колекції вбудованих компонентів, а також будь-які методи та функції з бібліотеки React [33]. Втім, підключення деяких користувацьких компонентів, що залежать від реалізації платформи, можуть вимагати спеціальних конфігурацій.

Приклад простого компоненту React Native (рис. 2.12):



```
import { useState } from 'react';
import { View, Text, Button } from 'react-native';

export const Counter = () => {
  const [count, setCount] = useState(0);

  const handleAddCount = () => {
    setCount((prevValue) => prevValue + 1);
  };

  return (
    <View>
      <Text>Count: {count}</Text>

      <Button onPress={handleAddCount}>
        Add count
      </Button>
    </View>
  );
};
```

Рис. 2.12. Простий компонент React Native

Наприклад деякі з базових компонентів для побудови додатку:

– View є фундаментальним компонент для побудови інтерфейсу користувача, що обгортає всі інші компоненти.

– Text є компонентом для відображення текстової інформації.

– Image є компонентом для відображення різних типів зображень, включно з мережевими зображеннями, статичними ресурсами, тимчасовими локальними зображеннями та зображеннями з локального диска.

– TextInput є компонентом для введення текстової інформації в програму за допомогою клавіатури користувача.

– ScrollView є компонентом, який обгортає View платформи, одночасно забезпечуючи інтеграцію з системою блокування дотиків.

– Button є базовим компонентом кнопки, який має оптимально відображатися на будь-якій платформі.

– Switch відображає поле вводу для булевих перемикачів.

Кожен з цих компонентів за допомогою технологій React Native використовує нативний компонент платформи додатку. Однак є і компоненти специфічні для конкретних платформ.

Наприклад для Android це:

– DrawerLayoutAndroid.

– TouchableNativeFeedback.

А для iOS:

– InputAccessoryView.

– SafeAreaView.

2.2.5 Аналіз поширеності React Native серед розробників

Наразі React Native разом з Flutter є провідними фреймворками для розробки кросплатформенних мобільних додатків (рис. 2.13) [34].

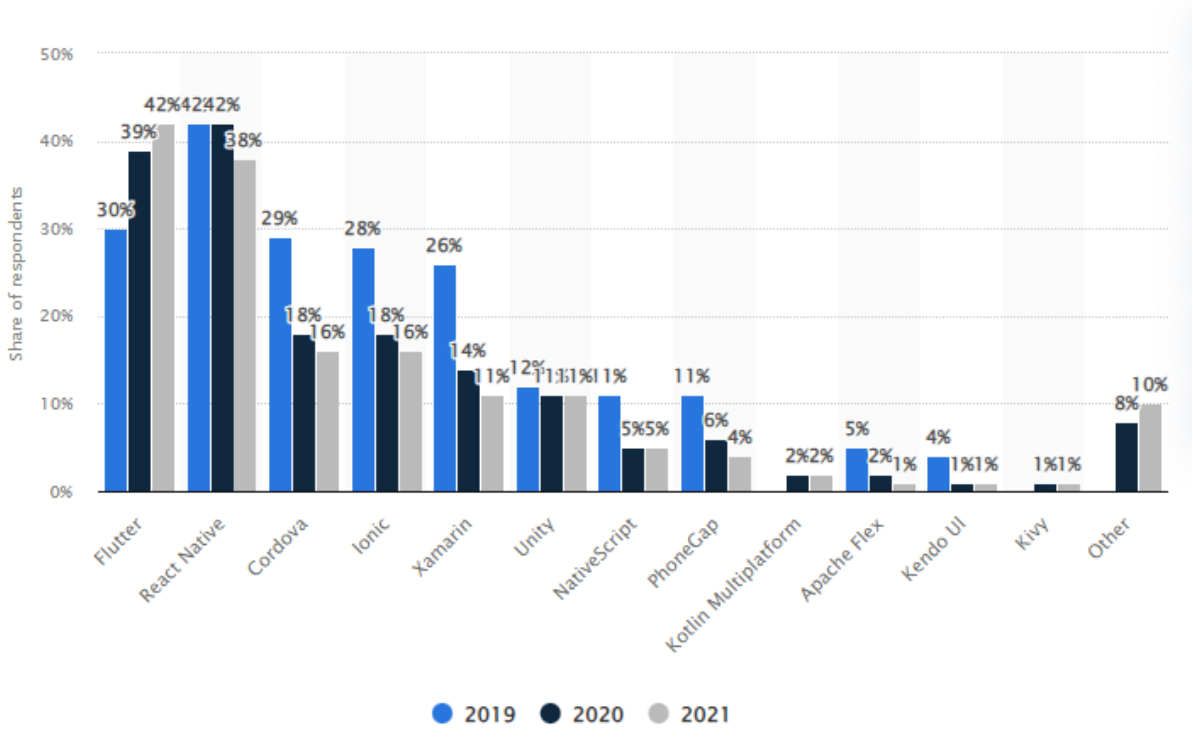


Рис. 2.13. Рейтинг використання технологій

Наприклад, React Native при розробці своїх додатків використовують наступні компанії:

- Instagram – більше 1 мільярда користувачів.
- Microsoft OneDrive – більше 1 мільярда користувачів.
- Netflix – більше 1 мільярда користувачів.
- Uber – більше 500 мільйонів користувачів.
- Discord – більше 100 мільйонів користувачів.

React Native найчастіше використовують додатки з домену «Food & Drink» (рис. 2.14) [35]. Втім розподілення серед доменів є доволі рівномірним, а сам фреймворк не має ніяких специфічних рис для конкретного домену, а є лише засобом реалізації додатку. Тому рейтинг використання по доменах корелюється з загальною популярністю додатків тих чи інших типів.

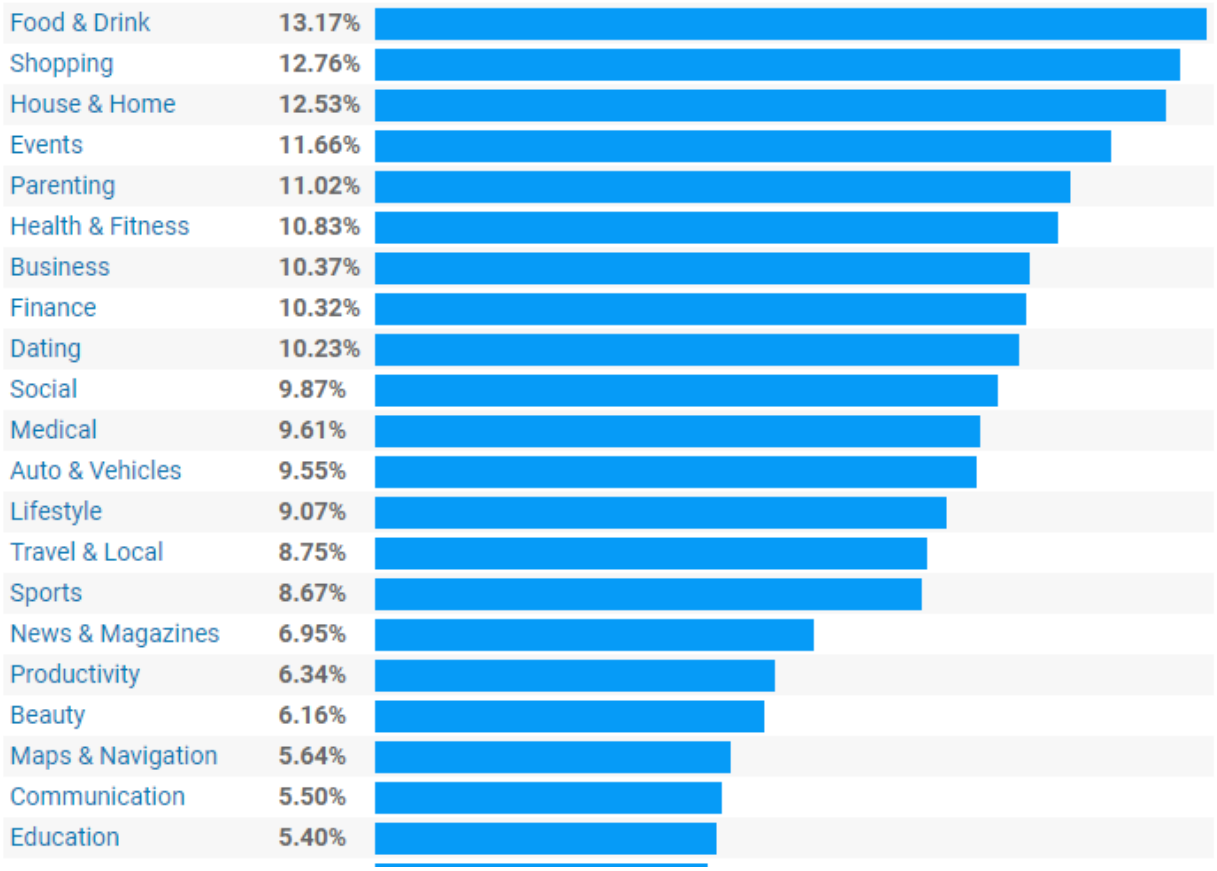


Рис. 2.14. Розподілення використання React Native серед доменів

2.3 Висновок до другого розділу

В другому розділі було проаналізовано методи роботи та архітектуру бібліотеки React та фреймворку React Native.

А саме було проаналізовано особливості та схеми роботи односторінкових веб-додатків (SPA), особливості бібліотеки React, такі як однонаправлений потік даних, технологія Virtual DOM, життєвий цикл та методи життєвого циклу компонентів, засоби зберігання стану додатку (Redux).

Також був проведений огляд архітектури фреймворку React Native, засобів створення додатку, розбір базових компонентів фреймворку та аналіз поширеності серед розробників та бізнесу.

Враховуючи проаналізовану інформацію, можна зробити висновок, що React Native є провідним фреймворком для розробки мобільних додатків.

РОЗДІЛ 3
РОЗРОБКА ДЕМОНСТРАЦІЙНОГО ДОДАТКУ ТА ПРОГРАМНОГО ПАКЕТУ ДЛЯ МОНІТОРИНГУ ЕФЕКТИВНОСТІ РОБОТИ ДОДАТКУ

3.1 Розробка демонстраційного додатку

В якості теми демонстраційного додатку було обрано галерею фільмів. Додаток повинен надавати користувачу наступні можливості:

- Перегляд актуального топу фільмів за популярністю.
- Перегляд детальної інформації про фільм.
- Пошук фільмів за назвою.
- Додавання фільму в список улюблених та збереження цього списку у локальної пам’яті пристрою.
- Перегляд списку улюблених фільмів.

3.1.1 Технології використані при розробці додатку

Окрім самого фреймворку React Native, робота якого була докладно описана в Розділі 2, при розробці демонстраційного додатку були використані наступні технології та бібліотеки:

1. Мова програмування TypeScript – це синтаксичний набір JavaScript, який додає статичний типізацію до Javascript. Це означає, що TypeScript додає синтаксис поверх JavaScript, дозволяючи розробникам типізувати код.

Сама по собі мова програмування Javascript має динамічну та слабку типізацію (рис. 3.1), що часто приводить до runtime помилок та труднощів при підтримці великих додатків.

Мова програмування TypeScript доповнює Javascript та переміщує його до когорти мов з сильною типізацією (рис. 3.2). Особливістю TypeScript є те, що він

працює лише на етапі розробки програми, допомагаючи розробникам робити код більш безпечним, а на етапі виходу програми в продакшн TypeScript компілюється в звичайний JavaScript [36].

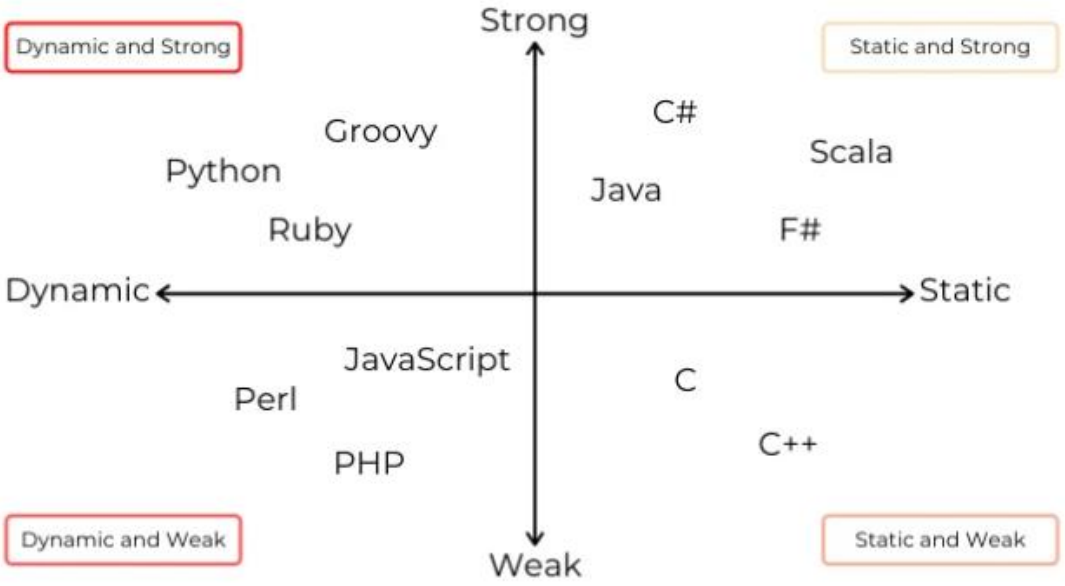


Рис. 3.1. Системи типізації в різних мовах програмування

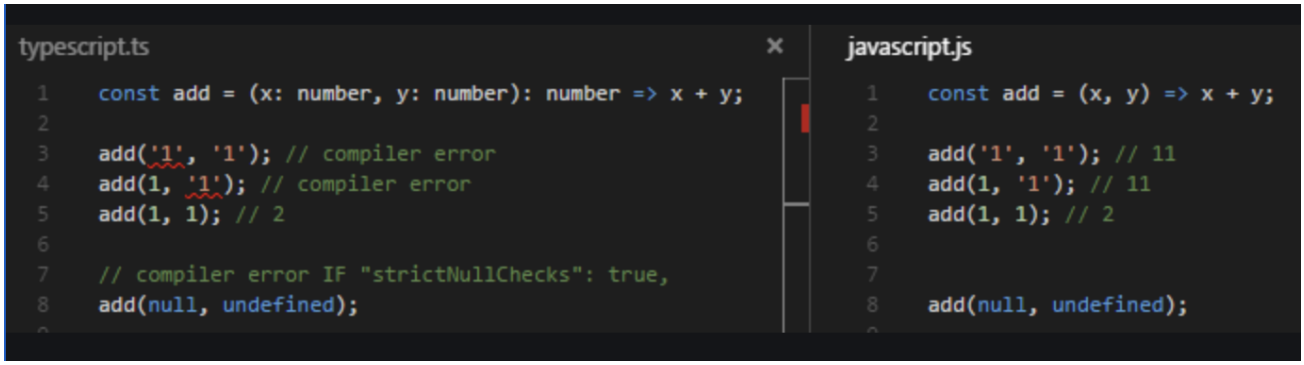


Рис. 3.2. Порівняння синтаксису та роботи Typescript і Javascript

Також Typescript додає інструменти для реалізації об’єктно орієнтованого підходу при створенні програм, додаючи можливість інкапсулювати поля за допомогою модифікаторів доступу, створювати абстрактні класи та методи, реалізовувати інтерфейси (рис. 3.3).

```typescript
interface Shape {
  getArea: () => number;
}

class Rectangle implements Shape {
  public constructor(protected readonly width: number, protected readonly height: number) {}

  public getArea(): number {
    return this.width * this.height;
  }
}

class Square extends Rectangle {
  public constructor(width: number) {
    super(width, width);
  }

  // getArea gets inherited from Rectangle
}
```

Рис. 3.3. Приклад класу у Typescript

TypeScript підтримується переважною кількість бібліотек, тому його популярність при розробці додатків стрімко зростає, і наразі він посідає 4 місце у рейтингу мов програмування згідно даних GitHub (рис. 3.4), та 3 місце в рейтингу найбільш стрімко зростаючих мов програмування (рис. 3.5) [37].

Такий стрімкий ріст можна пояснити великим попитом Javascript розробників на типізацію мови, оскільки вона є одним з ключових інструментів написання надійного та декларативного коду, а у випадку Typescript, він не лише додає типізацію, але також дає нові можливості у реалізації додатку, оскільки суттєво розширює ООП можливості та використання класового підходу створення додатків. Деякі великі фреймворки, вже взяли Typescript за свою основу. Наприклад, Angular – клієнтський фреймворк від Google, використовує саме Typescript як базову мову. Водночас серед серверних програмних інструментів Typescript також є досить поширеним, наприклад фреймворк Nest.js використовує його як базову мову, а платформа Deno навіть включила Typescript в базову архітектуру платформи, тобто не вимагає установки окремого пакету.

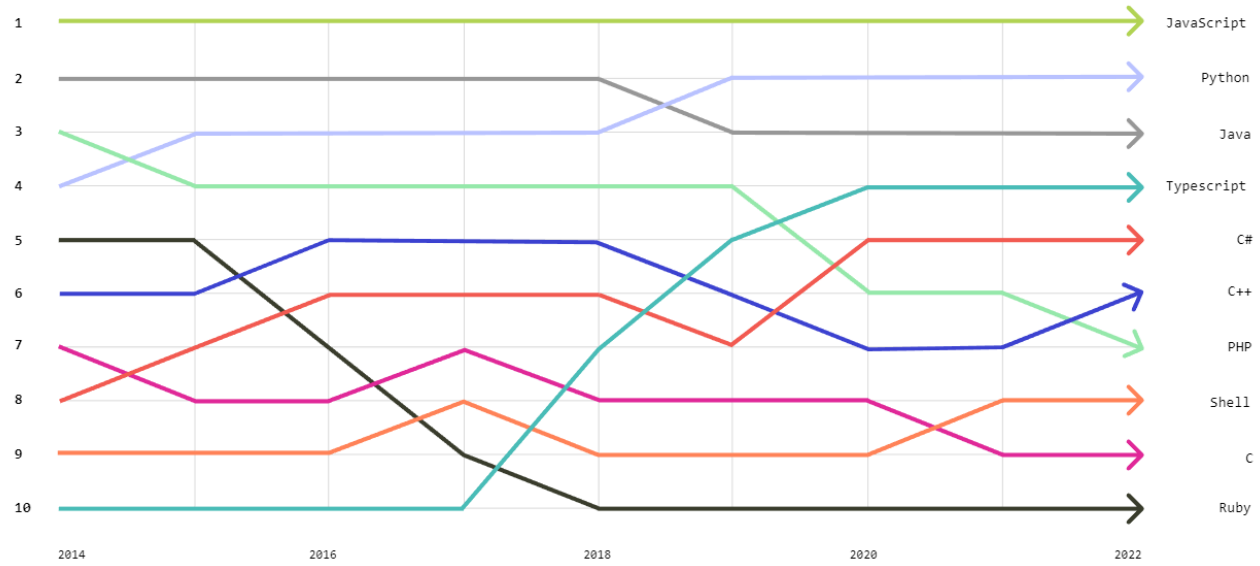


Рис. 3.4. Рейтинг мов програмування 2022

GROWTH IN PROGRAMMING LANGUAGES 2021-2022

01 HCL	56.1%
02 Rust	50.5%
03 TypeScript	37.8%
04 Lua	34.2%
05 Go	28.3%
06 Shell	27.7%
07 Makefile	23.7%
08 C	23.5%
09 Kotlin	22.9%
10 Python	22.5%

Рис. 3.5. Найбільш стрімко зростаючі мови програмування

2) Redux Toolkit - офіційно рекомендована бібліотека, що дозволяє писати ефективніший код, прискорювати процес розробки та автоматично застосовувати найкращі рекомендовані практики [38].

В основному його було створено для вирішення трьох основних проблем із Redux:

— Налаштування головного стору Redux надто складне.

— Щоб створити велику програму, потрібно додатково встановити багато бібліотек та пакетів.

— Redux вимагає занадто багато шаблонного коду, що ускладнює написання ефективного та чистого додатку.

3) RTK Query - надається як додаткове доповнення до пакету Redux Toolkit. Його було створено для вирішення випадків використання, отримання та кешування даних. RTK Query — це компактний і потужний набір інструментів, який використовується для визначення рівня взаємодії з API для програми.

Набір інструментів створено на основі Redux Toolkit. RTK Query надає додаткові можливості глобального керування стором, кешуванням, автооновленням даних [39].

4) Styled components - це специфічна для React CSS-in-JS бібліотека, яка надає можливість розробникам писати код CSS в коді JS для стилізації компонентів React, а також React Native (рис. 3.6) [40].

Перевагами styled-components є:

— Генерація CSS на льоту — застосовує лише стилі компонентів, які відображаються на сторінці.

— Унікальні імена класів — елементам DOM присвоюється унікальне ім’я класу, що дозволяє уникнути помилок і конфліктів імен класів.

— Просте видалення CSS — стилі додаються до окремого компонента, а не вставляються як ім’я класу, що полегшує видалення CSS.

— Простий динамічний стиль — він надає можливість додавати стилі до компонента залежно від атрибутів або глобальної теми.

— Легка та доступна підтримка — незалежно від того, наскільки великою є кодова база, обслуговування є простим, оскільки вам ніколи не доведеться переглядати багато файлів, щоб виявити стиль, який впливає на компонент.

— Автоматична префіксація – специфічні префікси для різних браузерів підставляються автоматично.

— Автоматичне очищення CSS коду, який не використовується, відповідно зменшуючи розмір файлу та пакету.

```
const SearchWrapper = styled.div`
  width: 100%;
  border-bottom: 1px solid ${constants.lightGrey};
  position: relative;
  padding-right: 2em;
  background: ${(props) => props.background};
`;
```

Рис. 3.6. Приклад синтаксису styled-components

4) Axios — це бібліотека Javascript, яка використовується для створення HTTP-запитів із node.js або XMLHttpRequests із браузера, яка також підтримує ES6 Promise API. На відміну від вбудованого методу fetch, axios є програмною обгорткою навколо XMLHttpRequest, що робить його більш гнучким у роботі та надає велику варіацію конфігурацій.

3.1.2 Архітектура додатку

Додаток складається з кількох модулів, а саме (рис. 3.7):

— Головного модулю із списком популярних фільмів.

— Модулю пошуку фільмів.

– Модулю улюблених фільмів.
– Модулю деталей фільму.

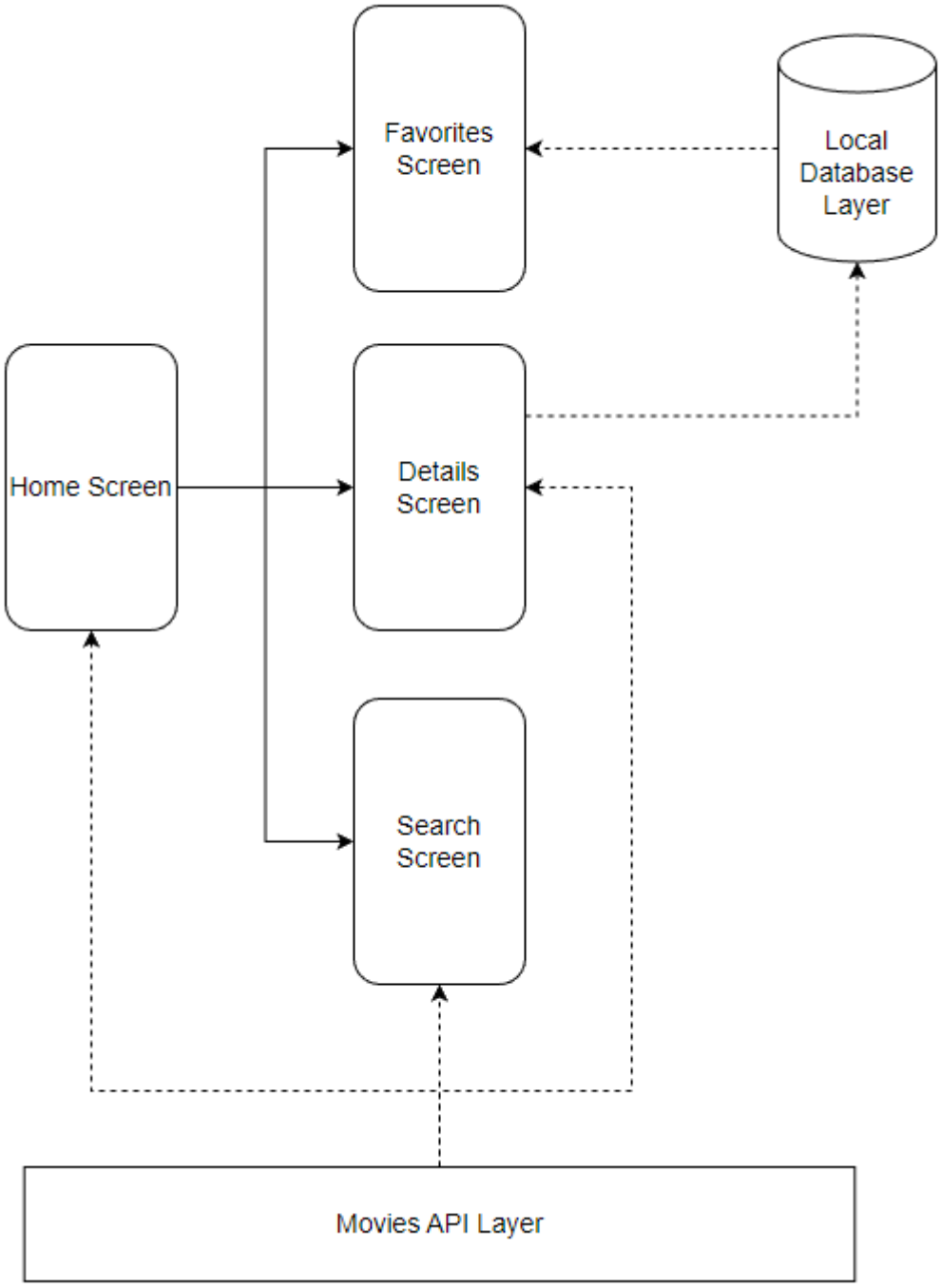


Рис. 3.7. Схема архітектури додатку

Всі модулі, окрім модулю улюблених фільмів потребують зовнішнього джерела даних. Для забезпечення зв’язку з сервером було створено сервіс API взаємодії з сервером, до якого можуть звертатися всі модулі. В цьому сервісі реалізовані запити на:

- Отримання пагінованої колекції фільмів.
- Отримання деталей окремого фільму.
- Отримання загального списку жанрів.
- Отримання базової конфігурації.

Слід зазначити, що запити сервіс API кешує інформацію, та при запиті з однаковими параметрами, будуть обрані дані з кешу. Також при використанні одного і того ж методу сервісу API в різних компонентах додатку одночасно, запити буде груповано, відповідно завдяки такому підходу покращена оптимізація.

Також для забезпечення роботи модулю улюблених фільмів, створений сервіс роботи з базою даних пристрою. Цей сервіс повністю інкапсулює в собі логіку звернень до БД через окремий пакет «react-native-async-storage». Завдяки подібній реалізації кожен з компонентів може викликати методи сервісу та взаємодіяти з сховищем улюблених фільмів.

3.1.3 Інтерфейс додатку

Головний екран додатку (рис. 3.8) містить топ популярних наразі фільмів, та відповідні елементи інтерфейсу що ведуть на сторінки пошуку фільмів та списку улюблених фільмів.

Топ фільмів є списком компонентів окремих фільмів, де кожен компонент містить наступну інформацію:

- Назву фільму.
- Постер.

– Рейтинг.

– Дату виходу фільму у прокат.

– Список жанрів.

– Скорочений опис фільму.

– Елементи інтерфейсу, що ведуть на сторінку деталей фільму, кнопка стрілка у рядку назви, та кнопка «Read more», якщо опис є скороченим.

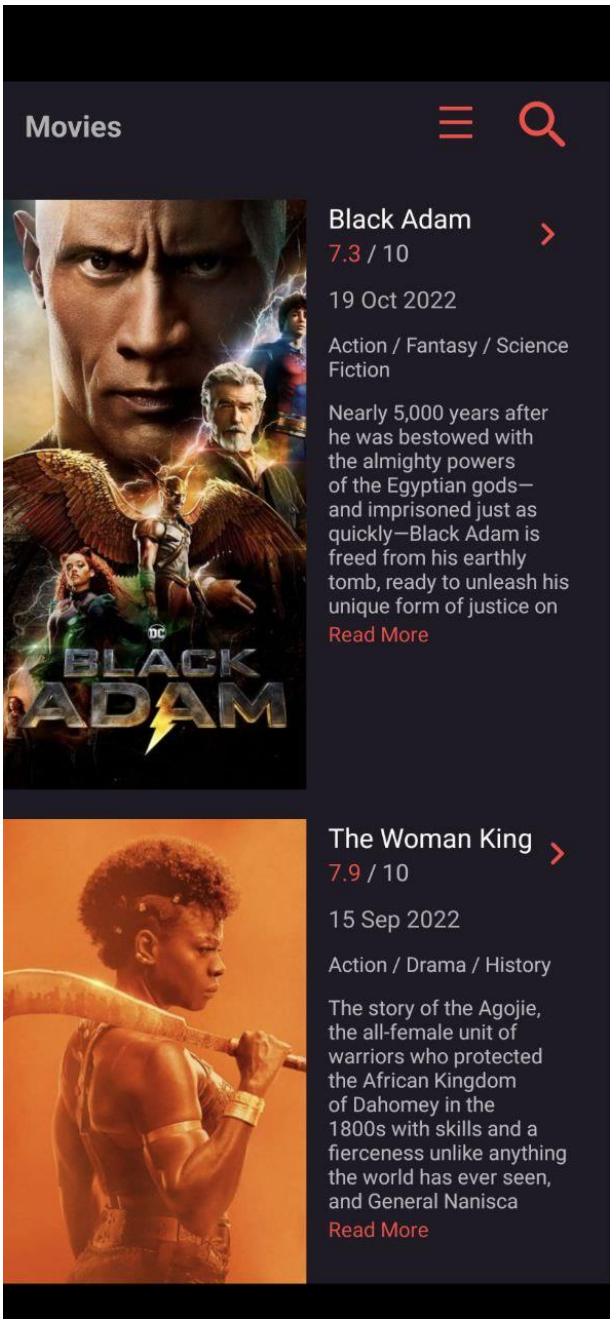


Рис. 3.8. Головний екран

Також на сторінці реалізовано механізм «Infinite scroll», що надає користувачу можливість прокручувати сторінку вниз нескінченно, і нові дані будуть завантажуватися автоматично разом з тим як користувач прокручує сторінку вниз (рис. 3.9). Це загальновідомий підхід, що дозволяє оптимізувати роботу додатку та завантажувати тільки потрібні на даний момент дані.

В React Native є вбудований компонент «VirtualizedList», що дозволяє реалізовувати такий підхід, та обробляти події досягання кінця списку, завантаження даних, відсутності даних тощо.

Також цей компонент оптимізований спеціально під відображення великих списків даних, завдяки механізму віртуалізації. Це означає, що одночасно побудоване дерево елементів буде лише для тих елементів списку, що знаходяться у полі зору користувача або близько до нього. Всі елементи списку, що знаходяться за полем зору користувача будуть лише займати місце для того, щоб прокручування сторінки відбувалось корректно. Механізм віртуалізації суттєво впливає на оптимізацію сторінки.

Віртуалізація значно покращує споживання пам’яті та швидкодію великих списків, зберігаючи обмежене вікно візуалізації активних елементів і замінюючи всі елементи поза вікном візуалізації порожнім простором відповідного розміру. Вікно адаптується до поведінки прокручування, а елементи відображаються поступово з низьким пріоритетом (після будь-яких запущених взаємодій), якщо вони знаходяться далеко від видимої області, або з високим пріоритетом в іншому випадку, щоб звести до мінімуму можливість бачити порожній простір.

Для завантаження даних використовується відкрите API «The movies DB». API надає можливість завантажувати колекції популярних наразі фільмів, включно зі всією необхідною інформацією, окрім інформації про акторів. Також надає можливість завантажувати дані про конкретний фільм, або шукати список фільмів які відповідають певним критеріям пошуку.

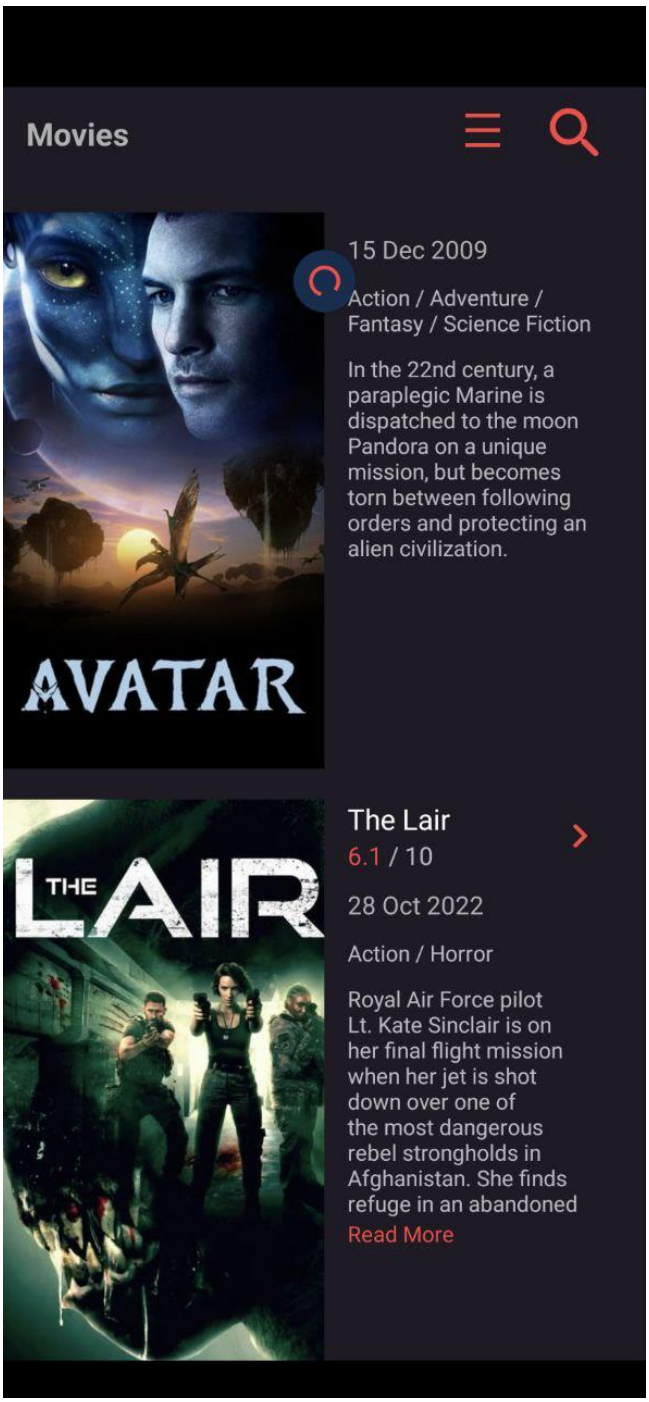


Рис. 3.9. Завантаження даних

Екран деталей фільму (рис. 3.10) містить повнорозмірний постер, а також більш розширену інформацію про фільм, а саме вона дублює всю ту інформацію, що була зображена на головному екрані, а також містить інформацію про бюджет та збори фільму, студію, а також повний опис фільму (рис. 3.11).

Рис. 3.10. Сторінка деталей фільму

Також екран деталей містить кнопку для додавання фільму в список улюблених або якщо фільм вже додано туди, то видалення фільму зі списку. Також містить кнопку навігації до головного екрану (рис.3.12).

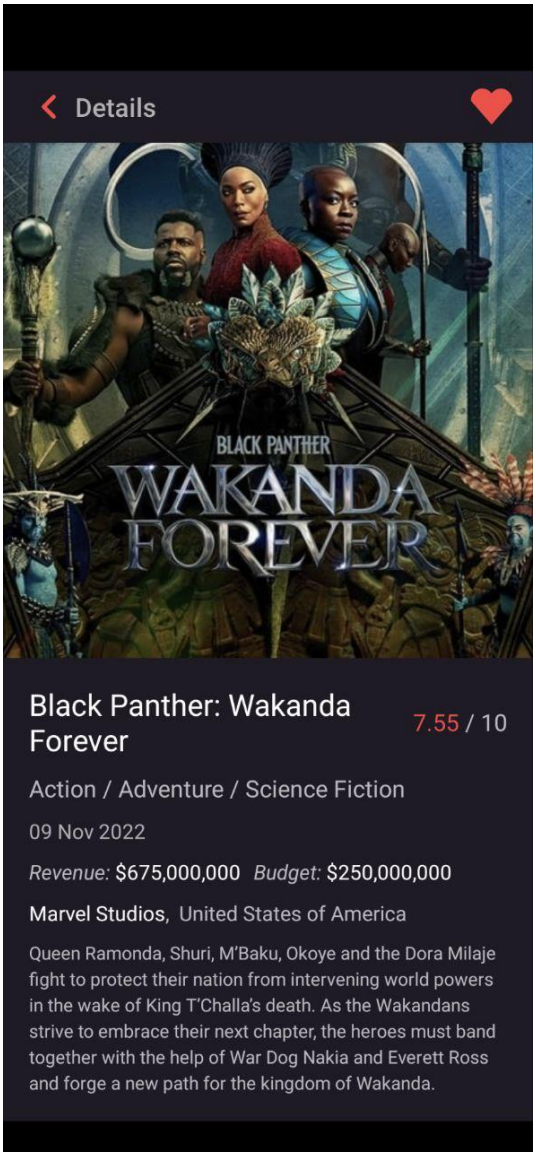


Рис. 3.11. Сторінка деталей фільму

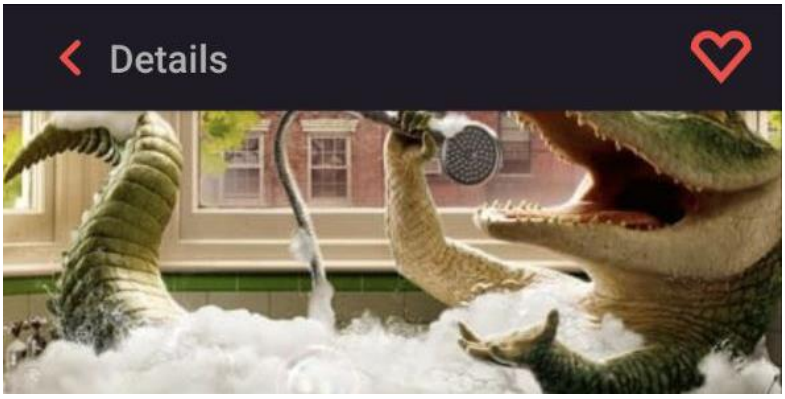


Рис. 3.12. Кнопка додавання до списку улюблених

Екран пошуку фільмів містить поле вводу, та список фільмів що задовольняють пошуковий запит. Список картин обмежений двадцятьма. Для виводу картин використовується вбудований компонент «FlatList», що є більш простішою версією компонента «VirtualizedList», однак добре оптимізований для рендерінгу обмежених списків.

За замовчанням сторінка пошуку не містить жодних фільмів, а робить запит на їх пошук після того як користувач вводить три і більше символів тексту пошукового запиту (рис. 3.13).

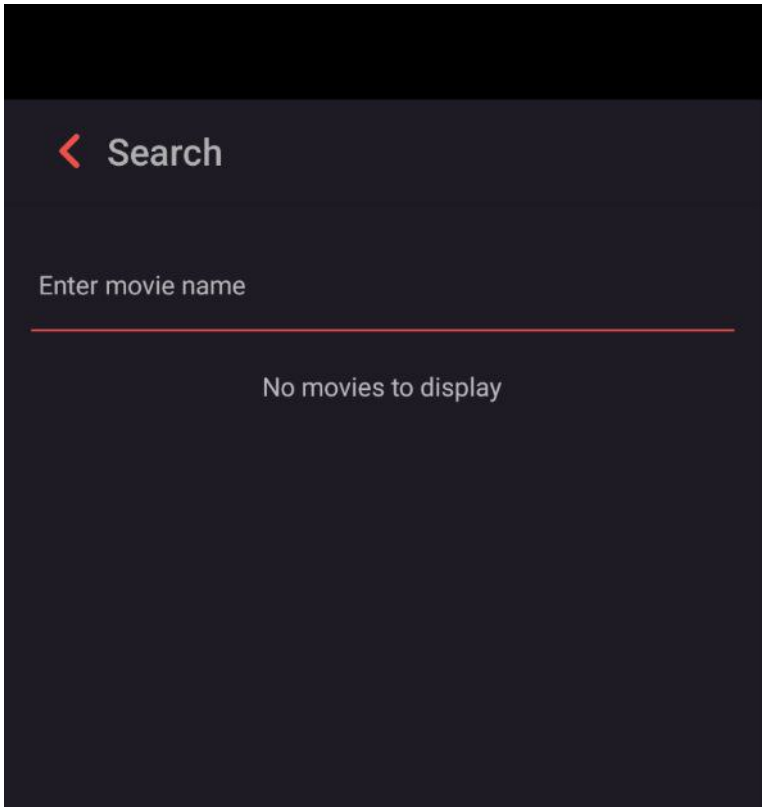


Рис. 3.13. Порожня сторінка пошуку

Також на сторінці реалізований підхід відкладання запитів (debounce). Цей підхід сприяє швидкодії та оптимізації, тому що не дає відсилати запити на кожну зміну поля вводу, а групую виклики функцій, відправляючи запит через пів секунди після останньої зміни у полі вводу (рис 3.14).

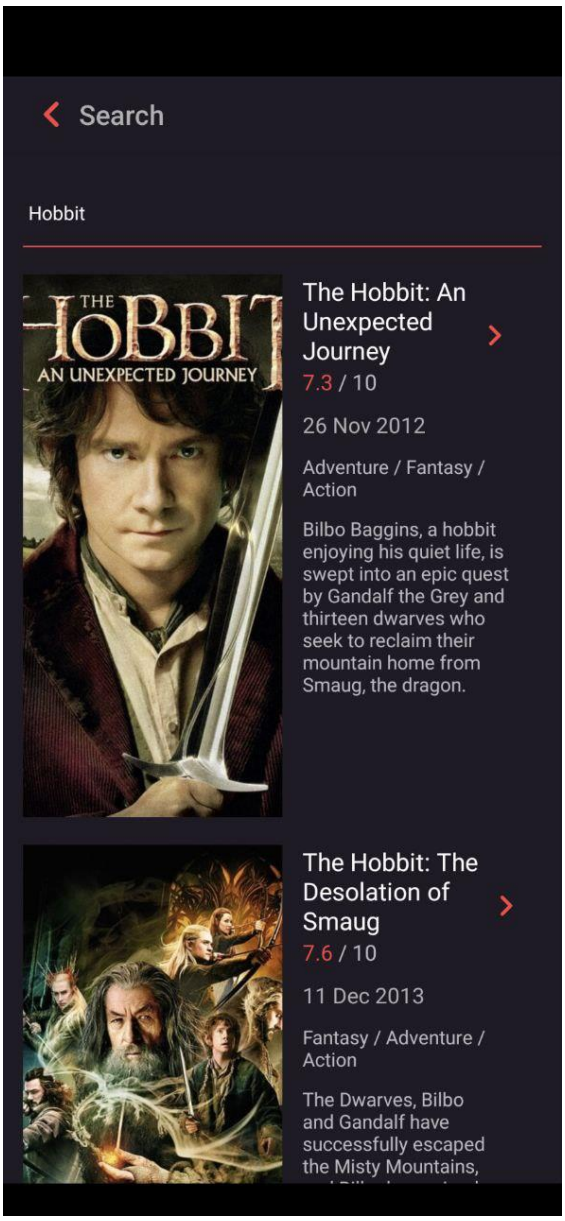


Рис. 3.14. Сторінка пошуку з уведеним пошуковим запитом

Сам список фільмів має схожий на головну сторінку інтерфейс, компоненти окремого фільму містять відповідну інформацію. Так сам використовуючи певні елементи інтерфейсу, користувач може перейти до екрану деталей будь-якого фільму.

Екран улюблених фільмів (рис. 3.15) містить список фільмів, що користувач власноруч додав до списку. За інтерфейсом він є схожим на головний екран, для виводу картин використовується вбудований компонент «FlatList».

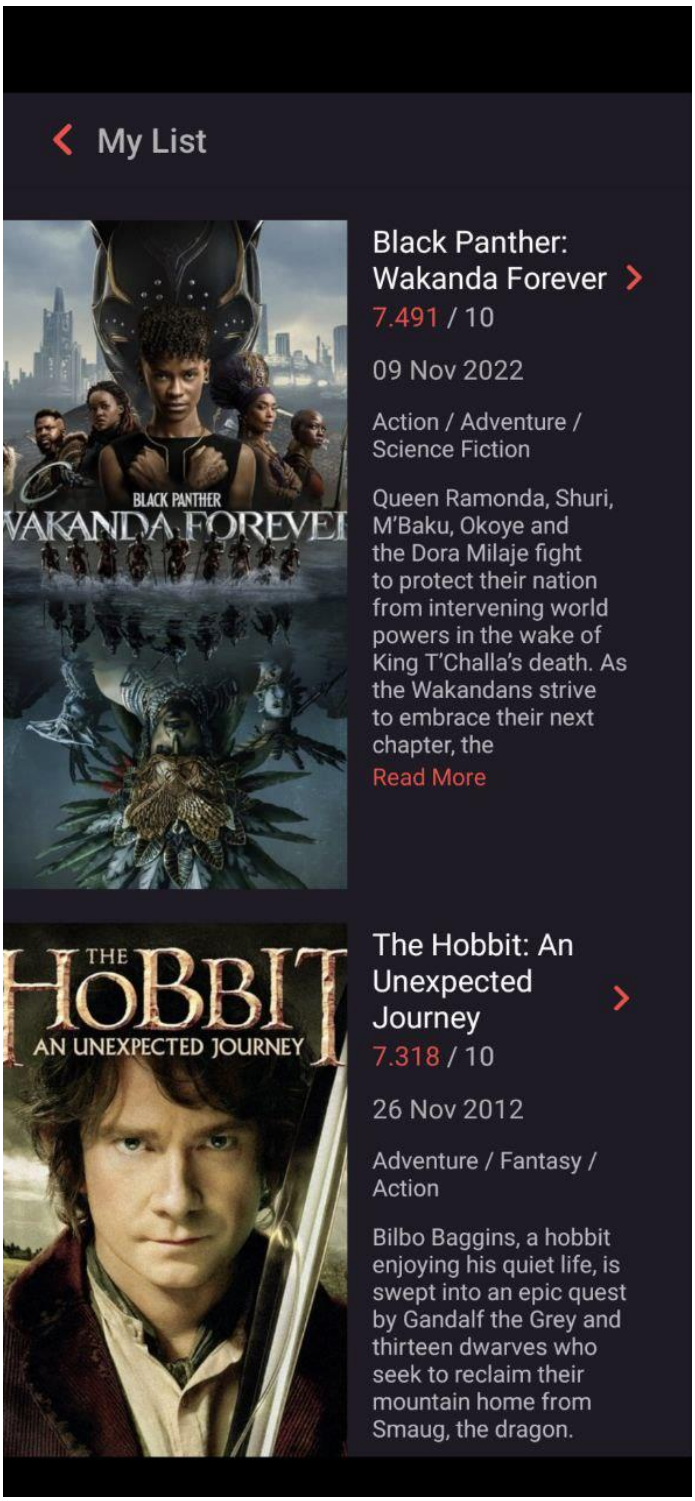


Рис. 3.15. Екран улюблених фільмів

Однак головною відмінністю цього екрану є те, що джерелом даних для нього є внутрішня база даних пристрою, відповідно цей список є унікальним для кожного пристрою, де встановлено додаток.

3.2 Розробка програмного пакету для вимірювання ефективності роботи додатку

Для оцінки ефективності роботи додатку на React Native наразі існує небагато інструментів. Головним недоліком вбудованих інструментів розробника є те, що вони вимагають змінити оточення для запуску.

А саме мова йде про двигун Javascript, що виконує Javascript код в React Native. При запуску програми в режимі дебагінгу, стандартний двигун JavascriptCore замінюється на браузерний V8. В деяких випадках, це може стати джерелом неочікуваної поведінки або програмних збоїв.

Саме для відстежування стану роботи програми у реальному часі при її нативних умовах роботи і було створено програмний пакет, що має змогу відстежувати наступні метрики швидкодії:

- Кількість кадрів.
- Середня кількість кадрів.
- Використана додатком оперативна пам’ять у мегабайтах.
- Використана додатком оперативна пам’ять у процентах від загальної кількості пам’яті пристрою.
- Час роботи додатку.

Програмний пакет реалізований у вигляді окремого ізольованого модулю, що може бути легко використаний з іншою програмою для виміру її ефективності, або може бути опублікований як окремий npm-пакет для React Native та бути використаним у будь-якому проекті.

Для того щоб додати пакет до додатку, треба його імпортувати та підключити як компонент «PerforamnceTools» в головний компонент App. Відповідно, ізоляція компонента необхідна, щоб його стан не викликав зайвих рендерів додатку, що негативно впливають на швидкодію (рис. 3.16).

Рис. 3.16. Підключення пакету

Пакет має декілька режимів роботи, а саме може виводити інформацію в консоль або виводити графічно в самому додатку, або одночасно. Для визначення цих режимів виводу інформації, компонент приймає декілька параметрів, а саме:

– Enabled відповідає за те, чи увімкнений пакет взагалі. Може бути використаний розробником, щоб регулювати роботу пакету, і, наприклад, вимкнути його в режимі продакшн збірки.

– enabledUiOutput відповідає за те, чи виводити інформацію графічно (рис. 3.17), за замовчанням false.

– enableConsoleOutput відповідає за те, чи виводити інформацію у консоль розробника (рис 3.18), за замовчанням true.

– updatePeriod відповідає за період оновлення інформації, за замовчанням становить 1000мс.

Головний функціонал пакету було реалізовано у вигляді спеціальної функції-хуку React «usePerfMonitor». Оскільки сам пакет має бути оптимізований, та вимагати мінімум ресурсів для роботи, то дані про поточну кількість кадрів, та інші зберігаються використовуючи механізм Ref в React, який жодним чином не впливає на життєвий цикл компоненту, тим самим не провокуючи зайвих рендерів інтерфейсу.

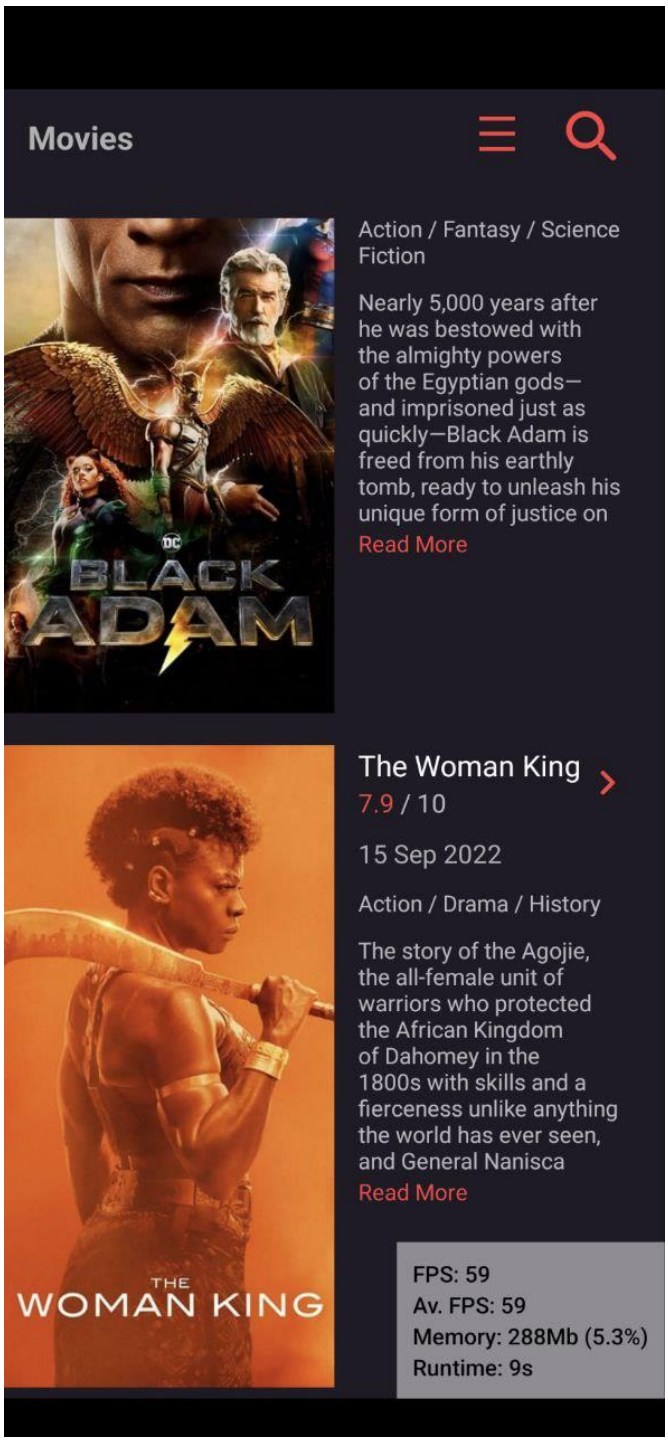


Рис. 3.17. Графічний інтерфейс показників щвидкодії

Показник поточної кількості кадрів вимірюється за допомогою вбудованої функції requestAnimationFrame, що викликається під час зміни кадру на пристрої. Також для оптимізації показника поточної кількості кадрів в секунду було використано механізм інтерполяції.

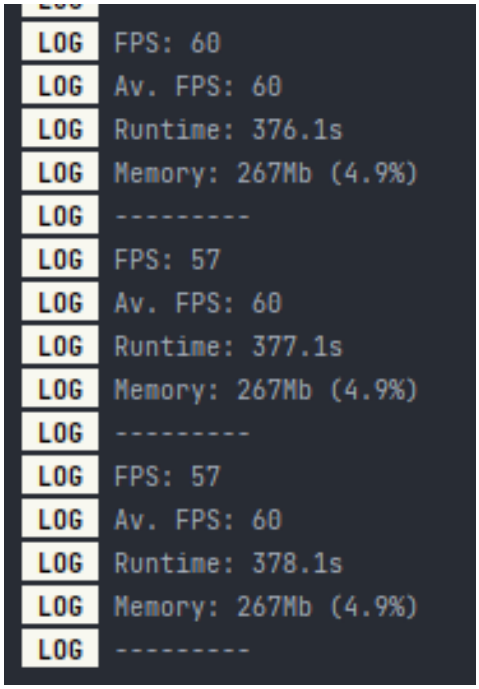


Рис. 3.18. Вивід показників в консоль

3.3 Тестування додатку за допомогою пакету оцінки ефективності

Під час тестування додатку був увімкнений консольний вивід інформації, що мінімізувати вплив самого пакету на швидкодію.

Отримані результати:

В стані спокою додаток видає стабільні 60 кадрів в секунду (рис. 3.19).

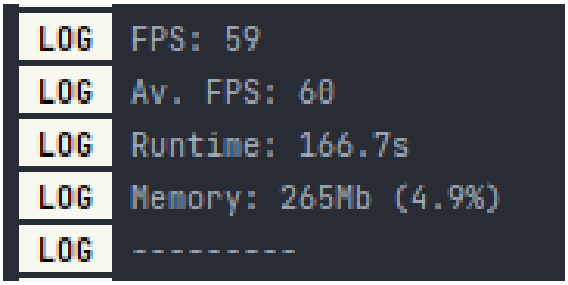


Рис. 3.19. Показники у стані спокою

При взаємодії користувача зі сторінкою, а саме введення тексту, помітних уповільнень в роботі додатку також не було виявлено (рис. 3.20).

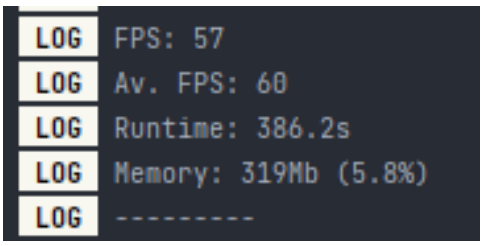


Рис. 3.20. Показники при взаємодії користувача з додатком

Однак при під час додаткового завантаження даних на головній сторінці при невпинній прокрутці екрану користувачем можуть траплятися зменшення частоти кадрів, що втім має мінімальний вплив на середній показник (рис. 3.21).

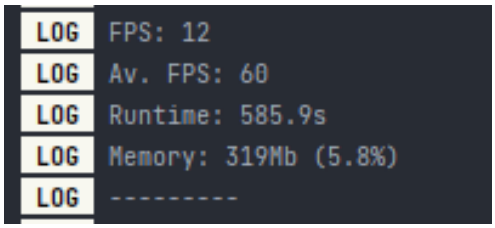


Рис. 3.21. Показники під час постійного завантаження даних

Навіть при рідких зменшеннях частоти кадрів, середній показник становить 60 кадрів в секунду, що є оптимальним показником для сучасних пристроїв. Споживання пам’яті варіюється від 260 Мб до 320 Мб.

3.4 Висновок до третього розділу

В ході виконання завдання третього розділу, був розроблений програмний пакет для вимірювання швидкодії додатку, що фіксує показники поточної частоти кадрів, середньої частоти кадрів, завантаження оперативної пам’яті у мегабайтах, та процентах від загальнодоступної пам’яті пристрою. Також виводиться інформація про поточний час роботи програми у секундах.

Пакет є ізольованим, що дає змогу його використовувати на будь-яких проектах React Native, а також опублікувати як окремий npm пакет. Оскільки

вплив самого пакета на швидкодію додатку повинен бути мінімальним, його було реалізовано через модифікацію ref значень, які на відміну від state не впливають на життєвий цикл компоненту, вимагаючи окремих рендерів.

Задля гнучкості у використані були реалізовані як вивід метрик швидкодії у консоль, так і окремий вивід через графічний інтерфейс. Через відповідні властивості компоненту розробник має змогу конфігурувати роботу пакету.

Також в ході виконання завдання третього розділу був розроблений додаток, на якому проводилося тестування пакету для виміру швидкодії. Задля того, щоб тестування було повноцінним, додаток повинен реалізовувати найпоширеніші функції сучасного додатку, а саме:

- Перегляд динамічно-завантаженого контенту.
- Взаємодію зі стороннім API.
- Взаємодію з користувачем через поля вводу.
- Взаємодію з локальною базою даних.

Відповідно до цих вимог в додатку були створені сторінки:

- Головна сторінка з механізмом Infinite Scroll, що забезпечує динамічне завантаження даних, реагуючи на дії користувача.
- Шар API для завантаження даних з серверу.
- Сторінка пошуку фільмів з полем вводу.
- Сторінка улюблених фільмів, що містить унікальний список для кожного користувача, та завантажує дані з локальної бази даних пристрою.

В результаті тестування додатку за допомогою розробленого пакету, середня частота кадрів становила 60, що є високим стандартом для сучасних мобільних пристроїв. Завантаження оперативної пам’яті також було помірним - від 260 до 320 мегабайтів при піковому навантаженні.

Враховуючи отримані показники, можна зробити висновки, що фреймворк React Native ефективно виконує поставлені задачі, забезпечуючи високу швидкодію комплексного мобільного додатку.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було проаналізовано методи та підходи до створення додатків, а саме нативні мобільні додатки, гібридні мобільні додатки, кросплатформенні мобільні додатки. Було встановлено, що найвищу якість продукту має нативний підхід, однак і найбільшу вартість розробки, оскільки вимагає утримання двох окремих команд розробки. Гібридний застосунок має меншу вартість розробки, однак і нижчу якість продукту. Найоптимальнішим варіантом виявився кросплатформенний застосунок, який поєднує в собі сильні сторони обох підходів, а саме порівняно низьку вартість розробки та високу якість розробленого рішення.

Серед інструментів для кросплатформенної розробки були детально розглянутий фреймворк React Native. У другому розділі кваліфікаційної роботи було проаналізовано його архітектуру, методи та підходи. Було встановлено, що React Native є одним із лідируючих інструментів для розробки кросплатформенних додатків. Від найближчого конкурента, фреймворка Flutter, його відрізняє те, що він використовує ті ж самі підходи, що і бібліотека для створення веб-застосунків React, що дає змогу веб-розробникам легко переходити у мобільну розробку. Також на відміну від Flutter, він використовує архітектуру мостів, за домогою якої код написаний на Javascript може використовувати нативні представлення та компоненти операційної системи, що у свою чергу може добре відзначитись на користувацькому досвіді, але може потребувати більше часу на створення додатку.

В третьому розділі кваліфікаційної роботи було розроблено програмний пакет для оцінки ефективності додатку React Native, який допомагає розробникам відстежувати основні показники швидкодії, не використовуючи вбудовані інструменти розробника React Native, що потенційно робить процес розробки надійнішим, оскільки не змінюється програмне оточення.

Також було розроблено демонстраційний додаток, який реалізовує всі основні функції та паттерни сучасних мобільних додатків. А саме додаток реалізує найпоширеніші функції: підхід Infinity Scroll для динамічного завантаження контенту при прокрутці, взаємодія зі стороннім API, взаємодія з локальною базою даних пристрою, підтримка вводу даних від користувача. Додаток був використаний для тестування розробленого програмного пакету.
Отримані результати показали, що додаток працює ефективно навіть у моменти пікового навантаження. З чого можна зробити висновок, що React Native є ефективним інструментом для розробки кросплатформенних мобільних додатків.

СПИСОК ВИКОРИСТАННИХ ДЖЕРЕЛ

1. Mobile Operating System Market Share Worldwide - [Електронний ресурс] - Режим доступу: https://gs.statcounter.com/os-market-share/mobile/worldwide

2. Ian F. Darwin: Android Cookbook: Problems and Solutions for Android Developers 2nd Edition. – Chapter 1 – 2017 – P. 20-21

3. Jerome DiMarzio: Android A Programmers Guide. – Chapter 1 – 2008 – P. 6

4. Android Architecture – [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/android-architecture/

5. How does Android work – [Електронний ресурс] – Режим доступу: https://bimlibik.github.io/posts/how-does-android-work/

6. Platform architecture – [Електронний ресурс] – Режим доступу: https://developer.android.com/guide/platform

7. Niranjan Kumar. Learn Android. – Chapter 3 – 2020 – P. 74-75.

8. Josh Berlin & Rene Cacheaux: Advanced iOS App Architecture – Chapter 1 – 2022 – P.17

9. Yair Carreno: Clean Architecture in iOS Principles and good design practices applied in iOS. – Chapter 3 – 2022 – P.109-112

10. iOS Architecture – [Електронний ресурс] – Режим доступу: https://redfoxsec.com/blog/ios-architecture/

11. Antonio Leiva: Kotlin for Android Developers : learn Kotlin the easy way while developing an Android App. – Chapter 1 – P. 5-6

12. What is a Native Mobile App Development? – [Електронний ресурс] – Режим доступу: https://mdevelopers.com/blog/what-is-a-native-mobile-app-development

13. Shaun Lewis & Mike Dunn: Native Mobile Development. – Chapter 1 – 2019 – P.35-38

14. Hybrid Application – [Електронний ресурс] – Режим доступу: https://www.techtarget.com/searchsoftwarequality/definition/hybrid-application-hybrid-app

15. Raymond K. Camden: Apache Cordova in Action. – Chapter 1 – 2015 – P.28

16. What Is a Hybrid App? (Detailed Guide for 2022) – [Електронний ресурс] – Режим доступу: https://www.upwork.com/resources/hybrid-app#hybrid-app

17. Cross Platform App Development: Developing an Application for Maximum Exposure – [Електронний ресурс] – Режим доступу: https://cleancommit.io/blog/cross-platform-app-development-developing-an-application-for-maximum-exposure/

18. React Native – one framework to rule them all – [Електронний ресурс] – Режим доступу: https://www.netguru.com/glossary/react-native

19. Carmine Zaccagnino: Programming Flutter: Native, Cross-Platform Apps the Easy Way. 1st Ed. – Chapter 1 – 2020 – P.11-12

20. Dan Bernes: Xamarin Mobile Application Development. – Chapter 1 – 2015 – P.15-16

21. Front-end Frameworks – [Електронний ресурс] – Режим доступу: https://2021.stateofjs.com/en-US/libraries/front-end-frameworks/

22. Alex Banks & Eve Porcello: Learning React. Modern Patterns for Developing React Apps. – Chapter 1 – 2020 – P. 1-2

23. Single Page Application – [Електронний ресурс] – Режим доступу: https://devopedia.org/single-page-application

24. One Way Dataflow – [Електронний ресурс] - https://tkssharma.gitbook.io/react-training/day-01/react-js-3-principles/one-way-data-flow

25. ReactJS Virtual DOM – [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/reactjs-virtual-dom/

26. David Griffiths, Dawn Griffiths: React Cookbook: Recipes for Mastering the React Framework. 1st Ed. – Chapter 4 – 2021 – P.174-178

27. React Lifecycle of Components – [Електронний ресурс] - https://www.geeksforgeeks.org/reactjs-lifecycle-components/

28. Alex Banks & Eve Porcello: Learning React Functional Web development with React and Redux. – Chapter 8 – 2020 – P.184-190

29. Introduction To Redux(using React Native) – [Електронний ресурс] – Режим доступу: https://medium.com/@abhayg772/introduction-to-redux-using-react-native-a8f1e8778333

30. Nader Dabit: React Native in Action. – Chapter 1 – 2019 – P. 4-5

31. React Native Internals – [Електронний ресурс] – Режим доступу: https://www.reactnative.guide/3-react-native-internals/3.1-react-native-internals.html

32. React Native Init vs Expo 2022: What Are the Differences? – [Електронний ресурс] – Режим доступу: https://fulcrum.rocks/blog/react-native-init-vs-expo

33. Core Components and APIs – [Електронний ресурс] – Режим доступу: https://reactnative.dev/docs/components-and-apis

34. Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021 – [Електронний ресурс] – Режим доступу: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/

35. React Native – Market Share By Category – [Електронний ресурс] – Режим доступу: https://www.appbrain.com/stats/libraries/details/react_native/react-native

36. Josh Goldberg: Learning Typescript: Enhance Your Web Development Skills Using Type-Safe JavaScript. – Chapter 1 – 2022 – P. 11

37. Top Languages Used in 2022 – [Електронний ресурс] – Режим доступу: https://octoverse.github.com/2022/top-programming-languages

38. What is Redux Toolkit – [Електронний ресурс] – Режим доступу: https://dev.to/maxinejs/what-is-redux-toolkit-b94

39. RTK Query Overview – [Електронний ресурс] – Режим доступу: https://redux-toolkit.js.org/rtk-query/overview

40. How To Use React Styled Components Efficiently – [Електронний ресурс] – Режим доступу: https://blog.copycat.dev/blog/styled-components-react/

41. Статистика зарплат – [Електронний ресурс] – Режим доступу: https://jobs.dou.ua/salaries/?period=2022-06&position=Middle%20SE&technology=JavaScript&experience=1-2

42. Fu Cheng. Flutter Recipes: Mobile Development Solutions for iOS and Android. – Chapter 5 – 2019 – P. 120-121

43. Christopher G. Lasater. Design Patterns. – Chapter 2 – 2006 – P. 93.

44. Eric Masiello, Jacob Friedmann. Mastering React Native. – Chapter 8 – 2017 – P. 270.

45. Frank Zammetti. Practical React Native: Build Two Full Projects. – Chapter 2 – 2018 – P. 40.

46. Roy Derks. React Projects: Build 12 real-world applications. – Chapter 10 – 2019 – P. 376.

47. Akshat Paul, Abhishek Nalwaya. React Native for Mobile Development. – Chapter 1 – 2019 – P.35

48. Akshat Paul, Abhishek Nalwaya. React Native for iOS Development. – Chapter 2 – 2015 – P. 60-62

49. Mahesh Panhale. Beginning Hybrid Mobile Application Development. – Chapter 9 – 2015 – P. 113-114.

50. Методичні рекомендації до виконання кваліфікаційних робіт здобувачами другого (магістерського) рівня вищої освіти спеціальностей 121 «Інженерія програмного забезпечення» та 122 «Комп'ютерні науки» / Б.І. Мороз, О.В. Іванченко, О.В. Реута, О.С. Шевцова; М-во освіти і науки України, Нац. техн. ун-т "Дніпровська політехніка". – Дніпро : НТУ «ДП», 2022. – 87 с.

Додаток А

КОД ПРОГРАМИ

Файл App.tsx:

```
import { Provider }                 from 'react-redux';
import { ThemeProvider }            from 'styled-components';
import { NavigationContainer }      from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import { store }                    from '@store/store';
import { BackPanel }                from '@components/BackPanel';
import { Gallery }                  from '@modules/Gallery';
import { Details }                  from '@modules/Details';
import { Search }                   from '@modules/Search';
import { Favorites }                from '@modules/Favorites';
import { PerformanceTools }         from '@modules/perfomance/PerformanceTools';
import { ConfigurationProvider }    from '@utils/configuration/ConfigurationProvider';
import { theme }                    from '@styles/theme';
import { INavigationStackPages }    from '@typings/navigation';
import { FavoritePanel }            from '@components/FavoritePanel';
import { MoviesHeader }             from '@components/MoviesHeader';

const { Screen, Navigator } = createNativeStackNavigator<INavigationStackPages>();

const App = () => (
  <>
    <NavigationContainer>
      <Provider store={store}>
        <ThemeProvider theme={theme}>
          <ConfigurationProvider>
            <Navigator>
              <Screen
                name      = "Movies"
                component = {Gallery}
                options={{
                  header    : MoviesHeader,
                  animation : 'slide_from_left',
                }}
              />

              <Screen
                name      = "Favorites"
                component = {Favorites}
                options={{
                  title           : 'My List',
                  headerTitleStyle : {
                    fontSize : theme.fontSizesNumeric.xxl,
                    color    : theme.colors.type.light,
                  },
                  headerStyle : {
```

```
            backgroundColor : theme.colors.primary.darkNavy,
           },
           animation  : 'slide_from_left',
           headerLeft : BackPanel,
          }}
         />

         <Screen
          name      = "Details"
          component = {Details}
          options   = {{
           title           : 'Details',
           headerTitleStyle : {
            fontSize : theme.fontSizesNumeric.xxl,
            color    : theme.colors.type.light,
           },
           headerStyle : {
            backgroundColor : theme.colors.primary.darkNavy,
           },
           animation   : 'slide_from_right',
           headerLeft  : BackPanel,
           headerRight : FavoritePanel,
          }}
         />

         <Screen
          name      = "Search"
          component = {Search}
          options   = {{
           title           : 'Search',
           headerTitleStyle : {
            fontSize : theme.fontSizesNumeric.xxl,
            color    : theme.colors.type.light,
           },
           headerStyle : {
            backgroundColor : theme.colors.primary.darkNavy,
           },
           animation  : 'slide_from_right',
           headerLeft : BackPanel,
          }}
         />
        </Navigator>
       </ConfigurationProvider>
      </ThemeProvider>
     </Provider>
    </NavigationContainer>

    <PerformanceTools
     enableConsoleOutput
     enabled
    />
   </>
);
```

```
export default App;
```

Файл Gallery.tsx:

```
import {
  useCallback,
  useEffect,
  useMemo,
  useState,
}                              from 'react';
import {
  RefreshControl,
  VirtualizedList,
}                              from 'react-native';
import styled, { useTheme }          from 'styled-components/native';
import { IShortMovieInfo }           from '@typings/movies';
import { useGetDiscoverMoviesQuery } from '@queries/movies';
import { MovieCard }                 from '@components/MovieCard';
import {
  NavigationProp,
  useNavigation,
}                              from '@react-navigation/native';
import { INavigationStackPages }     from '@typings/navigation';
import { useFavoriteMovies }         from '@utils/useFavoriteMovies';

export const Gallery = () => {
  const [movies, setMovies] = useState<IShortMovieInfo[]>([]);
  const [page, setPage]     = useState(1);
  const { colors }          = useTheme();
  const { checkIsFavorite } = useFavoriteMovies();

  const { navigate } = useNavigation<NavigationProp<INavigationStackPages>>();

  const {
    data,
    isFetching,
    refetch,
  } = useGetDiscoverMoviesQuery({ page });

  useEffect(
    () => {
      setMovies((prevData) => {
        const composedData = [...prevData, ...(data?.results ?? [])];

        return composedData.reduce(
          (acc, item) => {
            const isAlreadyIncluded = !!acc.find(({ id }) => id === item.id);

            if (!isAlreadyIncluded) {
              acc.push(item);
            }

            return acc;
          },
          [] as IShortMovieInfo[],
        );
```

```
    });
  },
  [data?.results, page],
 );

 const refreshControl = useMemo(
  () => (
   <RefreshControl
    colors              = {[colors.type.accent]}
    refreshing          = {isFetching}
    progressBackgroundColor = {colors.primary.navy}
    onRefresh           = {refetch}
   />
  ),
  [colors.primary.navy, colors.type.accent, isFetching, refetch],
 );

 const handleOpenDetails = useCallback(
  async (id: number) => {
   const isFavorite = await checkIsFavorite(id);

   navigate('Details', { id, isFavorite });
  },
  [checkIsFavorite, navigate],
 );

 const handleReachEnd = useCallback(
  () => {
   setPage((currentValue) => currentValue + 1);
  },
  [],
 );

 return (
  <Gallery.Wrapper>
   <VirtualizedList<IShortMovieInfo>
    data          = {movies}
    getItemCount  = {(list) => list.length}
    getItem       = {(list, index) => list[index]}
    keyExtractor  = {({ id }) => String(id)}
    refreshControl = {refreshControl}
    renderItem    = {({ item }) => (
     <MovieCard
      {...item}
      onOpenDetails={handleOpenDetails}
     />
    )}
    onEndReached  = {handleReachEnd}
   />
  </Gallery.Wrapper>
 );
};

Gallery.Wrapper = styled.View`
```

```
  min-height     : 100%;
  padding-top    : 20px;
  background-color: ${({ theme: { colors } }) => colors.primary.darkNavy};
`;
```

Файл Details.tsx:

```
import {
  ActivityIndicator,
  View,
}                          from 'react-native';
import styled              from 'styled-components/native';
import { NativeStackScreenProps }  from '@react-navigation/native-stack';
import { INavigationStackPages }   from '@typings/navigation';
import { useGetMovieDetailsQuery } from '@queries/movies';
import { POSTER_URL }      from '@env';
import { theme }           from '@styles/theme';
import { format }          from 'date-fns';

// eslint-disable-next-line @typescript-eslint/ban-ts-comment
// @ts-ignore
export const Details = ({ route }: NativeStackScreenProps<INavigationStackPages, 'Details'>) => {
  const { data } = useGetMovieDetailsQuery({ movie_id: route.params.id });

  const revenue = data?.revenue.toLocaleString('en-US', {
    style    : 'currency',
    currency : 'USD',
  }).slice(0, -3);

  const budget = data?.budget.toLocaleString('en-US', {
    style    : 'currency',
    currency : 'USD',
  }).slice(0, -3);

  return (
    <Details.Wrapper>
      {data ? (
        <View>
          <Details.Poster
            source = {{ uri: `${POSTER_URL}${data.poster_path}` }}
            style  = {{ resizeMode: 'cover' }}
          />

          <Details.ContentWrapper>
            <Details.TitleContainer>
              <Details.Title>
                {data.title}
              </Details.Title>

              {data.vote_average > 0 && (
                <Details.Rating>
                  <Details.RatingMark>{Math.round(data.vote_average * 100) / 100}</Details.RatingMark> / 10
                </Details.Rating>
              )}
```

```
          </Details.TitleContainer>

          <Details.GenreWrapper>
            {data.genres.map(({ name }) => name).join(' / ')}
          </Details.GenreWrapper>

          <Details.Date>
            {format(new Date(data.release_date), 'dd MMM u')}
          </Details.Date>

          <View style={{ flexDirection: 'row' }}>
            {data?.revenue ? (
              <Details.Revenue>
                Revenue: <Details.Emphasized>{revenue}</Details.Emphasized>
              </Details.Revenue>
            ) : null}

            {data?.budget ? (
              <Details.Revenue>
                Budget: <Details.Emphasized>{budget}</Details.Emphasized>
              </Details.Revenue>
            ) : null}
          </View>

          <Details.Creators>
            <Details.Emphasized>{data?.production_companies?.[0]?.name ?? ''}</Details.Emphasized>,
             
            {data?.production_countries?.[0]?.name ?? ''}
          </Details.Creators>

          <Details.Description>
            {data.overview}
          </Details.Description>
        </Details.ContentWrapper>
      </View>
    ) : (
      <ActivityIndicator
        size  = "large"
        color = {theme.colors.type.accent}
      />
    )}
  </Details.Wrapper>
 );
};

Details.Wrapper = styled.ScrollView`
 padding-bottom  : 20px;
 background-color: ${({ theme: { colors } }) => colors.primary.darkNavy};
 min-height      : 100%;
`;

Details.Poster = styled.Image`
 width: 100%;
 height: 700px;
```

```
`;

Details.ContentWrapper = styled.View`
  padding: 20px;
`;

Details.Title = styled.Text`
  width    : 70%;
  color    : ${({ theme: { colors } }) => colors.type.white};
  font-size: ${({ theme: { fontSizesNumeric } }) => `${fontSizesNumeric.xxl + 3}px`};
`;

Details.TitleContainer = styled.View`
  flex-direction : row;
  justify-content: space-between;
  align-items    : center;
`;

Details.Rating = styled.Text`
  font-size: ${({ theme: { fontSizes } }) => fontSizes.xl};
`;

Details.RatingMark = styled.Text`
  color: ${({ theme: { colors } }) => colors.type.accent}
`;

Details.Description = styled.Text`
  margin-top : 10px;
  line-height: 20px;
`;

Details.GenreWrapper = styled.Text`
  margin-vertical: 10px;
  font-size      : ${({ theme: { fontSizes } }) => fontSizes.xl};
`;

Details.Revenue = styled.Text`
  margin-right: 10px;
  font-size   : ${({ theme: { fontSizes } }) => fontSizes.l};
  font-style  : italic;
`;

Details.Emphasized = styled.Text`
  color      : ${({ theme: { colors } }) => colors.type.white};
  font-style  : normal;
`;

Details.Date = styled.Text`
  margin-bottom: 10px;
  font-size    : ${({ theme: { fontSizes } }) => fontSizes.l};
  color        : ${({ theme: { colors } }) => colors.type.light}
`;

Details.Creators = styled.Text`
```

```
  margin-top : 10px;
  font-size  : ${({ theme: { fontSizes } }) => fontSizes.l};
`;
```

Файл Favorites.tsx:

```
import {
  useCallback,
  useEffect,
  useState,
}                          from 'react';
import { FlatList }          from 'react-native';
import styled                from 'styled-components/native';
import {
  NavigationProp,
  useNavigation,
}                          from '@react-navigation/native';
import { IShortMovieInfo }     from '@typings/movies';
import { INavigationStackPages } from '@typings/navigation';
import { MovieCard }         from '@components/MovieCard';
import { useFavoriteMovies }   from '@utils/useFavoriteMovies';

export const Favorites = () => {
  const [movies, setMovies] = useState<IShortMovieInfo[]>([]);
  const { navigate }      = useNavigation<NavigationProp<INavigationStackPages>>();

  const { getMovies, checkIsFavorite } = useFavoriteMovies();

  useEffect(
    () => {
      const init = async () => {
        const data = await getMovies();

        setMovies(data);
      };

      init();
    },
    [getMovies],
  );

  const handleOpenDetails = useCallback(
    async (id: number) => {
      const isFavorite = await checkIsFavorite(id);

      navigate('Details', { id, isFavorite });
    },
    [checkIsFavorite, navigate],
  );

  return (
    <Favorites.Wrapper>
      <FlatList<IShortMovieInfo>
        data         = {movies}
        keyExtractor  = {({ id }) => String(id)}
```

```
      renderItem     = {({ item }) => (
        <MovieCard
          {...item}
          onOpenDetails={handleOpenDetails}
        />
      )}
    />
  </Favorites.Wrapper>
 );
};

Favorites.Wrapper = styled.View`
 min-height      : 100%;
 padding-top     : 20px;
 background-color: ${({ theme: { colors } }) => colors.primary.darkNavy};
`;
```

Файл Search.tsx:

```
import {
 useCallback,
 useState
}                          from 'react';
import {
 ActivityIndicator,
 FlatList,
 Text,
}                          from 'react-native';
import styled                  from 'styled-components/native';
import { useDebounce }         from 'use-debounce';
import { useSearchMovieQuery }   from '@queries/movies';
import { MovieCard }           from '@components/MovieCard';
import {
 NavigationProp,
 useNavigation
}                          from '@react-navigation/native';
import { INavigationStackPages } from '@typings/navigation';
import { IShortMovieInfo }       from '@typings/movies';
import { theme }               from '@styles/theme';
import { useFavoriteMovies }     from '@utils/useFavoriteMovies';

export const Search = () => {
 const { navigate }       = useNavigation<NavigationProp<INavigationStackPages>>();
 const [search, setSearch] = useState<string>('');
 const [debouncedSearch]   = useDebounce(search, 500);
 const { checkIsFavorite } = useFavoriteMovies();

 const {
   data,
   isFetching,
 } = useSearchMovieQuery({ search: debouncedSearch }, { skip: debouncedSearch.length < 3 })

 const handleOpenDetails = useCallback(
   async (id: number) => {
     const isFavorite = await checkIsFavorite(id);
```

```
    navigate('Details', { id, isFavorite });
  },
  [],
 );

 return (
  <Search.Wrapper>
   <Search.Input
    value       = {search}
    onChangeText = {(value) => setSearch(value)}
    placeholder  = "Enter movie name"
   />

   <Search.ListContainer>
    {isFetching ? (
     <ActivityIndicator
      size  = "large"
      color = {theme.colors.type.accent}
     />
    ) : (
     <FlatList<IShortMovieInfo>
      data        = {data?.results ?? []}
      keyExtractor = {({ id }) => String(id)}
      renderItem   = {({ item }) => (
       <MovieCard
        {...item}
        onOpenDetails={handleOpenDetails}
       />
      )}
      ListEmptyComponent = {() => (
       <Search.ListEmptyComponent>
        <Text>No movies to display</Text>
       </Search.ListEmptyComponent>
      )}
     />
    )}
   </Search.ListContainer>
  </Search.Wrapper>
 );
};

Search.Wrapper = styled.View`
 padding        : 20px 15px;
 height         : 100%;
 background-color: ${({ theme: { colors } }) => colors.primary.darkNavy};
`;

Search.Input = styled.TextInput`
 border-bottom-width: 1px;
 border-bottom-color: ${({ theme: { colors } }) => colors.type.accent};
`;

Search.ListContainer = styled.View`
```

```
  margin-top: 20px;
`;

Search.ListEmptyComponent = styled.View`
  align-items: center;
`;
```

Файл MovieCard.tsx:

```
import { useCallback }       from 'react';
import {
  TouchableOpacity,
  View,
}                            from 'react-native';
import styled                from 'styled-components/native';
import { format }            from 'date-fns';
import { IShortMovieInfo }   from '@typings/movies';
import { POSTER_URL }        from '@env';
import { useGetGenresQuery } from '@queries/movies';
import ViewDetails           from '@images/icons/ViewDetails.svg';

interface IMovieCardProps extends IShortMovieInfo {
  onOpenDetails: (id: number) => void,
}

const MAX_WORDS = 35;

export const MovieCard = ({
  title,
  poster_path,
  overview,
  genre_ids,
  vote_average,
  release_date,
  onOpenDetails,
  id,
}: IMovieCardProps) => {
  const { data }        = useGetGenresQuery();
  const needCutOverview = overview.split(' ').length > MAX_WORDS;
  const cutOverview     = needCutOverview
    ? overview.split(' ').slice(0, MAX_WORDS).join(' ')
    : overview;

  const handleOpenDetails = useCallback(
    () => onOpenDetails(id),
    [],
  );

  return (
    <MovieCard.Wrapper>
      <MovieCard.Poster
        source = {{ uri: `${POSTER_URL}${poster_path}` }}
        style  = {{ resizeMode: 'cover' }}
```

```
    />

    <MovieCard.InfoContainer>
      <MovieCard.TitleContainer>
        <View>
          <TouchableOpacity onPress={handleOpenDetails}>
            <MovieCard.Title>{title}</MovieCard.Title>
          </TouchableOpacity>

          {vote_average > 0 && (
            <MovieCard.Rating>
              <MovieCard.RatingMark>{vote_average}</MovieCard.RatingMark> / 10
            </MovieCard.Rating>
          )}
        </View>

        <TouchableOpacity onPress={handleOpenDetails}>
          <ViewDetails height={32} width={32} />
        </TouchableOpacity>
      </MovieCard.TitleContainer>

      <MovieCard.Date>
        {format(new Date(release_date), 'dd MMM u')}
      </MovieCard.Date>

      <MovieCard.GenreWrapper>
        {genre_ids.map((genreId) => data?.genres?.find((genre) => genre.id === genreId)?.name).join(' / ')}
      </MovieCard.GenreWrapper>

      <MovieCard.Description>
        {cutOverview}
      </MovieCard.Description>

      {needCutOverview ? (
        <TouchableOpacity onPress={handleOpenDetails}>
          <MovieCard.ReadMore>Read More</MovieCard.ReadMore>
        </TouchableOpacity>
      ) : ''}
    </MovieCard.InfoContainer>
  </MovieCard.Wrapper>
 );
};

MovieCard.Wrapper = styled.View`
 flex-direction: row;
 margin-bottom : 20px;
`;

MovieCard.InfoContainer = styled.View`
 margin-horizontal: 15px;
 width          : 40%;
`;

MovieCard.GenreWrapper = styled.Text`
```

```
  margin-vertical: 10px;
`;

MovieCard.Title = styled.Text`
  color    : ${({ theme: { colors } }) => colors.type.white};
  font-size: ${({ theme: { fontSizes } }) => fontSizes.xl};
`;

MovieCard.Description = styled.Text``;

MovieCard.ReadMore = styled.Text`
  color: ${({ theme: { colors } }) => colors.type.accent}
`;

MovieCard.Poster = styled.Image`
  width : 50%;
  height: 400px;
`;

MovieCard.Rating = styled.Text`
  font-size: ${({ theme: { fontSizes } }) => fontSizes.l};
`;

MovieCard.RatingMark = styled.Text`
  color: ${({ theme: { colors } }) => colors.type.accent}
`;

MovieCard.Date = styled.Text`
  margin-top: 10px;
  font-size : ${({ theme: { fontSizes } }) => fontSizes.l};
  color     : ${({ theme: { colors } }) => colors.type.light}
`;

MovieCard.TitleContainer = styled.View`
  flex-direction : row;
  justify-content: space-between;
  align-items    : center;
`;
```

Файл FavoritePanel.tsx:

```
import { useCallback }          from 'react';
import {
  TouchableOpacity,
  View,
}                               from 'react-native';
import {
  NavigationProp,
  RouteProp,
  useNavigation,
  useRoute,
}                               from '@react-navigation/native';
import { INavigationStackPages }   from '@typings/navigation';
import HeartOutlinedIcon          from '@images/icons/HeartOutlined.svg';
import HeartFilledIcon            from '@images/icons/HeartFilled.svg';
```

```
import { useTheme }               from 'styled-components/native';
import { useGetMovieDetailsQuery } from '@queries/movies';
import { useFavoriteMovies }      from '@utils/useFavoriteMovies';
import { FShortMovie }            from '@typings/movies';

export const FavoritePanel = () => {
  const { colors }    = useTheme();
  // eslint-disable-next-line @typescript-eslint/ban-ts-comment
  // @ts-ignore
  const { params }    = useRoute<RouteProp<INavigationStackPages, 'Details'>>();
  const { setParams } = useNavigation<NavigationProp<INavigationStackPages>>();

  const { data } = useGetMovieDetailsQuery({ movie_id: params?.id ?? 0 }, { skip: !params?.id ||
params?.isFavorite });

  const {
    addToFavorite,
    removeFromFavorites,
  } = useFavoriteMovies();

  const handleAddToFavoritePress = useCallback(
    async () => {
      if (params?.isFavorite) {
        await removeFromFavorites(params?.id);

        setParams({ ...params, isFavorite: false });
      } else if (data) {
        const movieData = FShortMovie({
          genre_ids : data.genres.map(({ id }) => id),
          ...data,
        });

        await addToFavorite(movieData);

        setParams({ ...params, isFavorite: true });
      }
    },
    [
      addToFavorite,
      data,
      params,
      removeFromFavorites,
      setParams,
    ],
  );

  return (
    <TouchableOpacity onPress={handleAddToFavoritePress}>
      <View>
        {params?.isFavorite ? (
          <HeartFilledIcon fill={colors.type.accent} width={32} height={32} />
        ) : (
          <HeartOutlinedIcon fill={colors.type.accent} width={32} height={32} />
        )}
```

```
    </View>
   </TouchableOpacity>
  );
};
```

Файл BackPanel.tsx:

```
import {
  TouchableOpacity,
  View,
}                    from 'react-native';
import ViewDetails       from '@images/icons/ViewDetails.svg';
import { useNavigation } from '@react-navigation/native';
import { useTheme }      from 'styled-components/native';

export const BackPanel = () => {
  const { goBack } = useNavigation();
  const { colors } = useTheme();

  return (
    <TouchableOpacity onPress={goBack}>
      <View>
        <ViewDetails
          width  = {40}
          height = {40}
          fill   = {colors.type.accent}
          style  = {{
            transform : [{ rotateZ: '180deg' }],
          }}
        />
      </View>
    </TouchableOpacity>
  );
};
```

Файл MoviesHeader.tsx:

```
import {
  TouchableOpacity,
  View,
}                          from 'react-native';
import styled                  from 'styled-components/native';
import {
  NavigationProp,
  useNavigation,
}                          from '@react-navigation/native';
import { INavigationStackPages } from '@typings/navigation';
import SearchIcon              from '@images/icons/Search2.svg';

export const MoviesHeader = () => {
  const { navigate } = useNavigation<NavigationProp<INavigationStackPages>>();

  return (
    <MoviesHeader.Wrapper>
```

```
    <MoviesHeader.Title>Movies</MoviesHeader.Title>

    <MoviesHeader.IconsWrapper>
      <MoviesHeader.IconContainer style={{ marginTop: 4 }}>
        <TouchableOpacity onPress={() => navigate('Favorites')}>
          <MoviesHeader.ListIcon>
            &#9776;
          </MoviesHeader.ListIcon>
        </TouchableOpacity>
      </MoviesHeader.IconContainer>

      <MoviesHeader.IconContainer>
        <TouchableOpacity onPress={() => navigate('Search')}>
          <SearchIcon />
        </TouchableOpacity>
      </MoviesHeader.IconContainer>
    </MoviesHeader.IconsWrapper>
  </MoviesHeader.Wrapper>
 );
};

MoviesHeader.Wrapper = styled.View`
 padding        : 10px 15px;
 height         : 60px;
 background-color: ${({ theme: { colors } }) => colors.primary.darkNavy};
 flex-direction  : row;
 align-items     : center;
 justify-content : space-between;
`;

MoviesHeader.Title = styled.Text`
 font-size  : ${({ theme: { fontSizes } }) => fontSizes.xxl};
 color      : ${({ theme: { colors } }) => colors.type.light};
 font-weight: bold;
`;

MoviesHeader.IconsWrapper = styled.View`
 flex-direction: row;
`;

MoviesHeader.ListIcon = styled.Text`
 color      : ${({ theme: { colors } }) => colors.type.accent};
 font-size   : 30px;
 line-height : 30px;
 margin-right: 15px;
`;

MoviesHeader.IconContainer = styled.View`
 margin-right: 15px;
 height      : 30px;
`;
```

Файл SearchPanel.tsx:

```
import {
  TouchableOpacity,
  View,
}                          from 'react-native';
import styled                  from 'styled-components/native';
import {
  NavigationProp,
  useNavigation,
}                          from '@react-navigation/native';
import { INavigationStackPages } from '@typings/navigation';
import ListIcon                from '@images/icons/List.svg';
import SearchIcon              from '@images/icons/Search2.svg';

export const SearchPanel = () => {
  const { navigate } = useNavigation<NavigationProp<INavigationStackPages>>();

  return (
    <SearchPanel.IconsContainer>
      <TouchableOpacity onPress={() => navigate('Favorites')}>
        <View>
          <ListIcon />
        </View>
      </TouchableOpacity>

      <TouchableOpacity onPress={() => navigate('Search')}>
        <View>
          <SearchIcon />
        </View>
      </TouchableOpacity>
    </SearchPanel.IconsContainer>
  );
};

SearchPanel.IconsContainer = styled.View`
  flex-direction: row;
`;
```

Файл movies.ts:

```
export interface IPaginatedResponse<T> {
  page        : number;
  results     : T[];
  total_pages  : number;
  total_results: number;
}

export interface IShortMovieInfo {
  adult: boolean;
  backdrop_path: string | null;
  genre_ids: number[];
  id: number;
```

```
  original_language: string;
  original_title: string;
  overview: string;
  popularity: number;
  poster_path: string | null;
  release_date: string;
  title: string;
  video: boolean;
  vote_average: number;
  vote_count: number;
}

export interface IGenre {
  id: number;
  name: string;
}

export interface IMovieInfo extends Omit<IShortMovieInfo, 'genre_ids'> {
  budget : number;
  genres: IGenre[];
  production_companies: { name: string; }[];
  production_countries: { name: string; }[];
  revenue: number;
  vote_average: number;
  vote_count: number;
}

export interface IConfiguration {
  images: {
    backdrop_sizes: string[];
    base_url: string;
    logo_sizes: string[];
    poster_sizes: string[];
    profile_sizes: string[];
    secure_base_url: string;
  };
}

export const FShortMovie = (data: Partial<IShortMovieInfo> | null): IShortMovieInfo => ({
  id               : data?.id || Date.now(),
  adult            : data?.adult ?? false,
  backdrop_path    : data?.backdrop_path || null,
  genre_ids        : data?.genre_ids || [],
  original_language : data?.original_language ?? '',
  original_title   : data?.original_title ?? '',
  overview         : data?.overview ?? '',
  popularity       : data?.popularity ?? 0,
  poster_path      : data?.poster_path ?? null,
  release_date     : data?.release_date ?? '',
  title            : data?.title ?? '',
  video            : data?.video ?? false,
  vote_average     : data?.vote_average ?? 0,
  vote_count       : data?.vote_count ?? 0,
});
```

Файл global.d.ts:

```
declare module '@env' {
  export const MOVIES_DB_API_KEY: string;
  export const MOVIES_API_URL   : string;
  export const POSTER_URL       : string;
}

declare module '*.svg' {
  import React from 'react';
  import { SvgProps } from 'react-native-svg';

  const content: React.FC<SvgProps>;
  export default content;
}

declare module '*.png';
declare module '*.jpeg';
declare module '*.jpg';
```

Файл navigation.ts:

```
export interface INavigationStackPages {
  Details: {
    id       : number;
    isFavorite: boolean;
  };
  Favorites: undefined;
  Movies   : undefined;
  Search   : undefined;
}
```

Файл styles.d.ts:

```
import 'styled-components';
import { ITheme } from '@styles/theme';

declare module 'styled-components' {
  // eslint-disable-next-line @typescript-eslint/no-empty-interface
  export interface DefaultTheme extends ITheme {}
}
```

Файл theme.ts:

```
export interface ITheme {
  colors: {
    primary: {
      black      : string;
      darkNavy   : string;
      navy       : string;
      plum       : string;
      purple     : string;
      rubineDark : string;
      rubineLight : string;
```

```
    white      : string;
   },
   type: {
    accent : string;
    dark   : string;
    light  : string;
    medium : string;
    white  : string;
   }
  },
  fontSizes: {
   l  : string;
   m  : string;
   s  : string;
   xl : string;
   xs : string;
   xxl: string;
  },
  fontSizesNumeric: {
   l  : number;
   m  : number;
   s  : number;
   xl : number;
   xxl: number;
  }
}

export const theme: ITheme = {
  colors : {
   primary : {
    black      : '#000000',
    white      : '#FFFFFF',
    rubineLight : '#ED0D6C',   // filter: invert(22%) sepia(52%) saturate(6783%) hue-rotate(323deg)
brightness(91%) contrast(106%);
    rubineDark  : '#D10056',
    plum       : '#8F1165',
    purple     : '#5A2B6B',
    navy       : '#122E52',
    darkNavy    : '#1E1B26',
   },
   type : {
    white  : '#FFFFFF',
    light  : '#B2B2B2',
    medium : '#77787A',
    dark   : '#24272A',
    accent : '#FE4141',
   },
  },
  fontSizes : {
   xxl : '20px',
   xl  : '18px',
   l   : '16px',
   m   : '14px',
   s   : '12px',
```

```
    xs  : '10px',
  },
  fontSizesNumeric : {
    xxl : 20,
    xl  : 18,
    l   : 16,
    m   : 14,
    s   : 12,
  },
};
```

Файл ConfigurationProvider.tsx:

```
import {
  createContext,
  PropsWithChildren,
  useEffect,
  useState,
}                       from 'react';
import axios            from 'axios';
import AsyncStorage     from '@react-native-async-storage/async-storage';
import { IConfiguration } from '@typings/movies';
import {
  MOVIES_API_URL,
  MOVIES_DB_API_KEY,
}                       from '@env';

export const ConfigurationContext = createContext<IConfiguration | null>(null);

const CONFIG_KEY = '@configuration';

export const ConfigurationProvider = ({ children }: PropsWithChildren) => {
  const [config, setConfig] = useState<IConfiguration | null>(null);

  useEffect(
    () => {
      const init = async () => {
        const rawData = await AsyncStorage.getItem(CONFIG_KEY);

        if (rawData) {
          setConfig(JSON.parse(rawData));
        } else {
          const response = await axios.get<IConfiguration>(`${MOVIES_API_URL}/configuration`, {
            params : { api_key: MOVIES_DB_API_KEY },
          });

          await AsyncStorage.setItem(CONFIG_KEY, JSON.stringify(response.data));

          setConfig(response.data);
        }
      };

      init();
    },
    [],
```

```
  );

  return (
    <ConfigurationContext.Provider value={config}>
      {children}
    </ConfigurationContext.Provider>
  );
};
```

Файл useConfiguration.ts:

```
import { useContext }          from 'react';
import { ConfigurationContext } from '@utils/configuration/ConfigurationProvider';
import { IConfiguration }      from '@typings/movies';

export const useConfig = () => useContext<IConfiguration | null>(ConfigurationContext);
```

Файл movies.ts:

```
import { createApi } from '@reduxjs/toolkit/query/react';
import { apiCaller } from '@utils/apiCaller';
import {
  IShortMovieInfo,
  IPaginatedResponse,
  IMovieInfo,
  IGenre,
}                   from '@typings/movies';

/* eslint-disable camelcase */
export const moviesApi = createApi({
  reducerPath : 'movies',
  baseQuery   : apiCaller,
  tagTypes    : [
    'MoviesDiscover',
    'MovieDetails',
    'Genres',
    'Search',
  ],
  endpoints : (builder) => ({
    // GET Discover Movies
    getDiscoverMovies : builder.query<IPaginatedResponse<IShortMovieInfo>, { page: number; }>({
      query : ({ page }) => ({
        url    : '/movie/popular',
        params : { page },
      }),
      providesTags : ['MoviesDiscover'],
    }),

    // GET Movie Details
    getMovieDetails : builder.query<IMovieInfo, { movie_id: number; }>({
      query : ({ movie_id }) => ({
        url    : `/movie/${movie_id}`,
        params : { movie_id },
      }),
      providesTags : ['MovieDetails'],
```

```
    }),

    // GET List of all genres
    getGenres : builder.query<{ genres: IGenre[] }, void>({
      query : () => ({
        url : '/genre/movie/list',
      }),
      keepUnusedDataFor : Infinity,
      providesTags      : ['Genres'],
    }),

    // GET Search Movie
    searchMovie : builder.query<IPaginatedResponse<IShortMovieInfo>, { search: string; }>({
      query : ({ search }) => ({
        url    : '/search/movie',
        params : { query: search },
      }),
      providesTags : ['Search'],
    }),
  }),
});

export const {
  useGetDiscoverMoviesQuery,
  useGetMovieDetailsQuery,
  useGetGenresQuery,
  useSearchMovieQuery,
} = moviesApi;
```

Файл favoriteMovies.ts:

```
import { createSlice } from '@reduxjs/toolkit';

const favouriteMovies = createSlice({
  name         : 'favourite',
  initialState : [],
  reducers     : {},
});

export const { reducer: favouriteMoviesReducer } = favouriteMovies;
```

Файл store.ts:

```
import { configureStore }         from '@reduxjs/toolkit';
import { moviesApi }              from '@queries/movies';
import { favouriteMoviesReducer } from './reducers/favouriteMovies';

export const store = configureStore({
  reducer : {
    favourites             : favouriteMoviesReducer,
    [moviesApi.reducerPath] : moviesApi.reducer,
  },
  middleware : (getDefaultMiddleware) => getDefaultMiddleware().concat(moviesApi.middleware),
});
```

Файл apiCaller.ts:

```
import axios, {
  AxiosError,
  AxiosRequestConfig,
} from 'axios';
import {
  MOVIES_API_URL,
  MOVIES_DB_API_KEY,
} from '@env';

interface IAxiosBaseQuery {
  body?          : AxiosRequestConfig['data'];
  headers?       : AxiosRequestConfig['headers'];
  method?        : AxiosRequestConfig['method'];
  params?        : AxiosRequestConfig['params'] & { requestWithoutAuth?: boolean };
  successMessage?: string;
  url            : string
}

export const apiCaller = async ({
  url,
  method = 'GET',
  headers,
  body,
  params,
}: IAxiosBaseQuery) => {
  const axiosConfig: AxiosRequestConfig = {
    url    : `${MOVIES_API_URL}${url}`,
    params : {
      api_key : MOVIES_DB_API_KEY,
      ...params,
    },
    method,
    headers,
  };

  if (body) {
    if (method === 'GET') {
      axiosConfig.params = {
        ...params?.params,
        ...body,
      };
    } else {
      axiosConfig.data = body;
    }
  }

  try {
    const result = await axios(axiosConfig);

    return { data: result.data };
  } catch (axiosError) {
    const err = axiosError as AxiosError;
```

```
    return {
      error : {
        status : err.response?.status,
        data   : err.response?.data || err?.message,
      },
    };
  }
};
```

Файл useFavoriteMovies.ts:

```
import { useCallback }     from 'react';
import AsyncStorage        from '@react-native-async-storage/async-storage';
import { IShortMovieInfo } from '@typings/movies';

const STORAGE_KEY = 'FavoriteMovies';

interface IUseFavoriteMoviesReturn {
  addToFavorite      : (movie: IShortMovieInfo) => Promise<void>;
  checkIsFavorite    : (id: number) => Promise<boolean>;
  getMovies          : () => Promise<IShortMovieInfo[]>;
  removeFromFavorites: (id: number) => Promise<void>;
}

export const useFavoriteMovies = (): IUseFavoriteMoviesReturn => {
  const getMovies = useCallback<(
  ) => Promise<IShortMovieInfo[]>>(
    async () => {
      try {
        const rawData = await AsyncStorage.getItem(STORAGE_KEY);

        return rawData ? JSON.parse(rawData) : [];
      } catch (error) {
        console.log(error);
      }

      return [];
    },
    [],
    );

  const checkIsFavorite = useCallback(
    async (movieId: number) => {
      const currentData = await getMovies();

      return !!currentData.find(({ id }) => id === movieId);
    },
    [getMovies],
  );

  const addToFavorite = useCallback(
    async (movie: IShortMovieInfo) => {
      const currentData = await getMovies();
```

```
    const isAlreadyAdded = currentData.find(({ id }) => id === movie.id);

    try {
      if (!isAlreadyAdded) {
        const newData = [...currentData, movie];

        await AsyncStorage.setItem(STORAGE_KEY, JSON.stringify(newData));
      }
    } catch (error) {
      console.log(error);
    }
  },
  [getMovies],
 );

 const removeFromFavorites = useCallback(
  async (itemId: number) => {
    const currentData = await getMovies();

    const newData = currentData.filter(({ id }) => id !== itemId);

    try {
      await AsyncStorage.setItem(STORAGE_KEY, JSON.stringify(newData));
    } catch (error) {
      console.log(error);
    }
  },
  [getMovies],
 );

 return {
  getMovies,
  addToFavorite,
  removeFromFavorites,
  checkIsFavorite,
 };
};
```

Файл usePerfMonitor.ts:

```
import {
  useEffect,
  useRef,
  useState,
}                       from 'react';
import {
  getUsedMemory,
  getTotalMemory,
}                       from 'react-native-device-info';
import { useInterpolation }  from '@modules/perfomance/useInterpolation';
import { useAnimationFrame } from '@modules/perfomance/useAnimationFrame';

interface IUsePerfMonitorParams {
  period: number;
}
```

```
interface IUsePerfMonitorReturn {
  averageFps   : number;
  fps          : number;
  memoryUsed   : number;
  memoryUsedMbs: number;
  time         : number;
}

const INITIAL: IUsePerfMonitorReturn = {
  fps          : 0,
  averageFps   : 0,
  time         : 0,
  memoryUsed   : 0,
  memoryUsedMbs : 0,
};

export const usePerfMonitor = ({ period }: IUsePerfMonitorParams): IUsePerfMonitorReturn => {
  const [fps, setFps] = useInterpolation(1000);
  const timeRef       = useRef<number>(0);
  const totalRef      = useRef<{ count: number; fps: number; }>({ fps: 0, count: 0 });

  const [fpsResult, setFpsResult] = useState<IUsePerfMonitorReturn>(INITIAL);

  useAnimationFrame(
    ({ time, delta }) => {
      const currentFps = 1 / delta;
      const prevTotal = { ...totalRef.current };

      totalRef.current = {
        fps   : prevTotal.fps + currentFps,
        count : prevTotal.count + 1,
      };

      setFps(currentFps);

      timeRef.current = time;
    },
    [],
  );

  useEffect(
    () => {
      const interval = setInterval(
        () => {
          Promise.all([getUsedMemory(), getTotalMemory()])
            .then(([used, total]) => {
              const memoryUsed = used / total;

              setFpsResult({
                time         : Number(timeRef.current.toFixed(1)),
                memoryUsed   : Number(memoryUsed.toFixed(3)) * 100,
                averageFps   : Math.round(totalRef.current.fps / totalRef.current.count),
                fps          : Math.round(fps.value),
```

```
            memoryUsedMbs : Math.round(used / 1024 / 1024),
          });
        });
      },
      period,
    );

    return () => clearInterval(interval);
  },
  [],
 );

 return fpsResult;
};
```

Файл useInterpolation.ts:

```
import {
 useCallback,
 useRef,
}        from 'react';
import Ola from 'ola';

type TUseInterpolationReturn = [
 { value: number },
 (value: number) => void,
];

export const useInterpolation = (time: number): TUseInterpolationReturn => {
 const ref = useRef<{ value: number }>({ value: 0 });

 const setValue = useCallback(
  (value: number) => {
   if (ref.current) {
    ref.current.value = value;
   } else {
    // eslint-disable-next-line @typescript-eslint/ban-ts-comment
    // @ts-ignore
    ref.current = new Ola(value, time);
   }
  },
  [time],
 );

 return [ref.current, setValue];
};
```

Файл useAnimationFrame.ts:

```
import { useEffect, useRef } from 'react';
import performance          from 'react-native-performance';

export const useAnimationFrame = (
```

```
  cb  : (arg: { delta: number; time: number; }) => void,
  deps: Array<unknown>,
) => {
  const frame = useRef<number>();
  const last  = useRef(performance.now());
  const init  = useRef(performance.now());

  const animate = () => {
    const now   = performance.now();
    const time  = (now - init.current) / 1000;
    const delta = (now - last.current) / 1000;

    cb({ time, delta });

    last.current = now;

    frame.current = requestAnimationFrame(animate);
  };

  useEffect(() => {
    frame.current = requestAnimationFrame(animate);

    return () => cancelAnimationFrame(frame.current as number);
  }, deps); // Make sure to change it if the deps change
};
```

Файл PerformanceTools.tsx:

```
import { useEffect }      from 'react';
import styled             from 'styled-components/native';
import { usePerfMonitor } from '@modules/perfomance/usePerfMonitor';

interface IPerformanceToolsProps {
  enableConsoleOutput?: boolean;
  enableUiOutput?     : boolean;
  enabled             : boolean;
  updatePeriod?       : number;
}

export const PerformanceTools = ({
  enabled,
  enableConsoleOutput = false,
  enableUiOutput = true,
  updatePeriod = 1000,
}: IPerformanceToolsProps) => {
  const {
    time,
    fps,
    averageFps,
    memoryUsed,
    memoryUsedMbs,
  } = usePerfMonitor({ period: updatePeriod });

  useEffect(
```

```
    () => {
      if (enabled && enableConsoleOutput) {
        console.log(`FPS: ${fps}`);
        console.log(`Av. FPS: ${averageFps}`);
        console.log(`Runtime: ${time}s`);
        console.log(`Memory: ${memoryUsedMbs}Mb (${memoryUsed.toFixed(1)}%)`);
        console.log('---------');
      }
    },
    [
      averageFps,
      enableConsoleOutput,
      enabled,
      fps,
      memoryUsed,
      memoryUsedMbs,
      time,
    ],
  );

  if (!enabled || !enableUiOutput) {
    return null;
  }

  return (
    <PerformanceTools.MonitorWrapper>
      <PerformanceTools.Metric>
        FPS: {fps}
      </PerformanceTools.Metric>

      <PerformanceTools.Metric>
        Av. FPS: {averageFps}
      </PerformanceTools.Metric>

      <PerformanceTools.Metric>
        Memory: {memoryUsedMbs}Mb ({memoryUsed.toFixed(1)}%)
      </PerformanceTools.Metric>

      <PerformanceTools.Metric>
        Runtime: {time}s
      </PerformanceTools.Metric>
    </PerformanceTools.MonitorWrapper>
  );
};

PerformanceTools.MonitorWrapper = styled.View`
  position        : absolute;
  padding         : 10px;
  bottom          : 0;
  right           : 0;
  background-color: rgba(255, 255, 255, 0.5);
`;

PerformanceTools.Metric = styled.Text`
```

```
  color: #000;
  font-weight: 500;
`;
```

Файл .eslintr.js:

```
module.exports = {
  parser        : '@typescript-eslint/parser',
  extends       : ['plugin:@typescript-eslint/recommended', 'airbnb', 'airbnb/hooks'],
  parserOptions : {
    ecmaFeatures : {
      jsx : true,
    },
  },
  plugins : ['typescript-sort-keys'],
  globals : {
    React            : true,
    JSX              : true,
    intlTelInputUtils : true,
    RequireAtLeastOne : true,
    AxiosResponseData : true,
  },
  env : {
    browser : true,
    node    : true,
  },
  rules : {
    'react/react-in-jsx-scope'            : 'off',
    'react/require-default-props'         : 'off',
    'react/function-component-definition' : [
      2,
      {
        namedComponents   : 'arrow-function',
        unnamedComponents : 'arrow-function',
      },
    ],
    'react/no-unstable-nested-components' : 'off',
    'react/jsx-props-no-spreading'        : 'off',
    'react/jsx-one-expression-per-line'   : 'off',
    'react-hooks/exhaustive-deps'         : 'warn',
    'react/jsx-equals-spacing'            : 'off',
    'react/jsx-filename-extension'        : 'off',
    'react/jsx-sort-props'                : [
      2,
      {
        callbacksLast        : true,
        shorthandFirst       : true,
        shorthandLast        : false,
        multiline            : 'last',
        ignoreCase           : true,
        noSortAlphabetically : true,
        reservedFirst        : true,
        locale               : 'auto',
      },
    ],
```

```
    '@typescript-eslint/no-unnecessary-type-constraint' : 'off',
    '@typescript-eslint/camelcase'                      : 'off',
    '@typescript-eslint/no-unused-vars'                 : 'error',
    '@typescript-eslint/no-shadow'                      : 'error',
    'typescript-sort-keys/interface'                    : 'error',
    'typescript-sort-keys/string-enum'                  : 'error',

    'import/extensions'           : 'off',
    'import/no-unresolved'        : 'off',
    'import/prefer-default-export' : 'off',

    camelcase            : 'off',
    'linebreak-style'    : 'off',
    'no-else-return'     : ['error', { allowElseIf: true }],
    'no-shadow'          : 'off',
    'no-unused-vars'     : 'off',
    'no-multi-spaces'    : 'off',
    'no-trailing-spaces' : 'error',
    'key-spacing'        : [
      2,
      {
        multiLine : {
          afterColon  : true,
          align       : 'colon',
          beforeColon : true,
        },
        singleLine : {
          afterColon  : true,
          beforeColon : false,
        },
      },
    ],
    'max-len' : [
      'error',
      120,
      2,
      {
        ignoreUrls            : true,
        ignoreComments        : false,
        ignoreRegExpLiterals  : true,
        ignoreStrings         : true,
        ignoreTemplateLiterals : true,
      },
    ],
  },
};
```

Файл babel.config.js:

```
module.exports = {
  presets : ['module:metro-react-native-babel-preset'],
  plugins : [
    ['module-resolver', {
      root  : ['./src'],
```

```
      alias : {
        '@'           : './src',
        '@modules'    : './src/modules',
        '@components' : './src/components',
        '@styles'     : './src/styles',
        '@images'     : './src/images',
        '@typings'    : './src/typings',
        '@utils'      : './src/utils',
        '@store'      : './src/store',
        '@queries'    : './src/queries',
      },
    }],
    ['module:react-native-dotenv', {
      envName       : 'APP_ENV',
      moduleName    : '@env',
      path          : '.env',
      blocklist     : null,
      allowlist     : null,
      blacklist     : null, // DEPRECATED
      whitelist     : null, // DEPRECATED
      safe          : false,
      allowUndefined : false,
      verbose       : false,
    }],
  ],
};
```

Файл index.js:

```
/**
 * @format
 */

import { AppRegistry }     from 'react-native';
import App                 from './src/App';
import { name as appName } from './app.json';

AppRegistry.registerComponent(appName, () => App);
```

Файл tsconfig.json:

```
// prettier-ignore
{
  "compilerOptions": {
    "skipLibCheck"               : true,
    "allowJs"                    : true,
    "allowSyntheticDefaultImports": true,
    "esModuleInterop"            : true,
    "isolatedModules"            : true,
    "jsx"                        : "react-jsx",
    "lib"                        : ["es2017"],
    "types"                      : ["react-native", "@types/styled-components-react-native"],
    "moduleResolution"           : "node",
    "noEmit"                     : true,
    "strict"                     : true,
```

```
    "target"                    : "esnext",
    "baseUrl"                   : "src",
    "plugins"                   : [
     {
      "name": "typescript-styled-plugin",
      "lint": {
       "validProperties": [
        "shadow-color",
        "shadow-opacity",
        "shadow-offset",
        "shadow-radius",
        "padding-horizontal",
        "padding-vertical",
        "margin-vertical",
        "margin-horizontal",
        "tint-color",
        "aspect-ratio",
        "elevation"
       ]
      }
     }
    ],
    "paths": {
     "@modules/*"   : ["modules/*"],
     "@utils/*"     : ["utils/*"],
     "@store/*"     : ["store/*"],
     "@models/*"    : ["models/*"],
     "@images/*"    : ["images/*"],
     "@styles/*"    : ["styles/*"],
     "@typings/*"   : ["typings/*"],
     "@components/*": ["components/*"],
     "@queries/*"   : ["queries/*"],
     "@env/*"       : ["../.env"],
     "@/*"          : ["./*"]
    },
  },
  "include": [
    "src/**/*"
  ],
  "exclude": [
    "babe.js",
    "metro.config.js",
  ]
}
```

Файл package.json:

```
{
  "name": "moviesgallery",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "android": "react-native run-android",
    "ios": "react-native run-ios",
    "start": "react-native start",
```

```
  "test": "jest",
  "lint": "eslint . --ext .js,.jsx,.ts,.tsx"
 },
 "dependencies": {
  "@react-native-async-storage/async-storage": "^1.17.11",
  "@react-navigation/native": "^6.0.13",
  "@react-navigation/native-stack": "^6.9.1",
  "@reduxjs/toolkit": "^1.9.0",
  "@types/styled-components": "^5.1.26",
  "@types/styled-components-react-native": "^5.2.0",
  "axios": "0.27.2",
  "date-fns": "^2.29.3",
  "ola": "^1.2.1",
  "react": "18.1.0",
  "react-native": "0.70.5",
  "react-native-device-info": "^10.3.0",
  "react-native-dotenv": "^3.4.2",
  "react-native-performance": "^4.0.0",
  "react-native-safe-area-context": "^4.4.1",
  "react-native-screens": "^3.18.2",
  "react-native-svg": "^13.6.0",
  "react-redux": "^8.0.5",
  "styled-components": "^5.3.6",
  "use-debounce": "^8.0.4"
 },
 "devDependencies": {
  "@babel/core": "^7.12.9",
  "@babel/runtime": "^7.12.5",
  "@react-native-community/eslint-config": "^2.0.0",
  "@tsconfig/react-native": "^2.0.2",
  "@types/jest": "^26.0.23",
  "@types/ola": "^1.2.0",
  "@types/react": "^18.0.21",
  "@types/react-native": "^0.70.6",
  "@types/react-redux": "^7.1.24",
  "@types/react-test-renderer": "^18.0.0",
  "@typescript-eslint/eslint-plugin": "^5.37.0",
  "@typescript-eslint/parser": "^5.37.0",
  "babel-jest": "^26.6.3",
  "babel-plugin-module-resolver": "^4.1.0",
  "eslint": "^7.32.0",
  "eslint-config-airbnb": "^19.0.4",
  "eslint-plugin-import": "^2.26.0",
  "eslint-plugin-jsx-a11y": "^6.6.1",
  "eslint-plugin-react": "^7.31.10",
  "eslint-plugin-react-hooks": "^4.6.0",
  "eslint-plugin-typescript-sort-keys": "^2.1.0",
  "jest": "^26.6.3",
  "metro-config": "^0.73.3",
  "metro-react-native-babel-preset": "0.72.3",
  "react-native-svg-transformer": "^1.0.0",
  "react-test-renderer": "18.1.0",
  "typescript": "^4.8.3"
```

Додаток Б

ВІДГУК КЕРІВНИКА

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ «ДНІПРОВСЬКА ПОЛІТЕХНІКА»

Факультет інформаційних технологій
Кафедра програмного забезпечення комп'ютерних систем

ВІДГУК

Наукового керівника Мещерякова Леоніда Івановича, д.т.н., проф. каф. ПЗКС
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання, посада, місце роботи)

на магістерську роботу

студента Кунденка Павла Руслановича
(прізвище, ім'я, по батькові)

курсу II групи 122м-21-1
спеціальності 122 Комп'ютерні науки
освітньої програми
на тему Дослідження ефективності фреймворку React Native при розробці мобільних додатків.

Актуальність теми Представлена магістерська кваліфікаційна робота присвячена дослідженню ефективності фреймворку для розробки мобільних додатків React Native. Даний фреймворк є одним з найпоширеніших при розробці додатків під операційні системи Android та iOS. На сьогоднішній день ці системи є провідними на стрімко зростаючому ринку мобільних пристроїв, тому з огляду на це, робота характеризується актуальністю та своєчасністю.

Мета досліджень Полягає у дослідженні та вдосконаленні процесу розробки мобільних додатків під найпоширеніші операційні системи з використанням фреймворку React Native.

Коротка характеристика розділів роботи Перший розділ роботи містить

аналітичний огляд літературних джерел за темою магістерської роботи. Другий розділ присвячено детальному аналізу фреймворку React Native як засобу для створення мобільних додатків. В третьому розділі розглянута задача реалізації програмного пакету для оцінки ефективності мобільного додатку, створеного за допомогою React Native, а також розглянуто створений демонстраційний додаток.

Практичне значення роботи Отримані результати роботи є застосовними при розробці мобільних додатків під операційні системи Android та iOS. Також ці результати є підставою для більш детального майбутнього дослідження підходів та методів створення мобільних додатків.

Зауваження та недоліки В роботі відсутній більш детальний огляд проблеми кросплатформенності, а саме огляд роботи розроблених рішень на різних платформах, акцент покладений на конкретну платформу.

Висновки та оцінка Магістром було проведено аналіз та порівняння можливих методів розв'язання поставленої задачі та обрано оптимальний варіант. Під час виконання магістерської кваліфікаційної роботи студент Кунденко П.Р. проявив себе грамотним, кваліфікованим спеціалістом, здатним приймати самостійно складні технічні рішення. Вважаю, що магістерська кваліфікаційна робота заслуговує оцінку «відмінно», а Кунденко П.Р. – присвоєння кваліфікації «магістра» з комп'ютерних наук.

Науковий керівник Мещеряков Л.І., док. техн. наук, проф., проф. каф. ПЗКС

(прізвище, ім'я, по батькові, посада, місце роботи)

«___»_____20__ р. _____

(підпис)

Додаток В

РЕЦЕНЗІЯ

на магістерську роботу

студента Кунденка Павла Руслановича
(прізвище, ім'я, по батькові)

курсу II групи 122м-21-1
кафедри програмного забезпечення комп'ютерних систем
спеціальності 122 Комп'ютерні науки
освітньої програми
Тема роботи Дослідження ефективності фреймворка React Native
При розробці мобільних додатків

Стисла характеристика розділів роботи Другий розділ присвячено детальному аналізу фреймворку React Native як засобу для створення мобільних додатків. В третьому розділі розглянута задача реалізації програмного пакету для оцінки ефективності мобільного додатку.

Пропозиції, внесені студентом, рівень їх наукового обґрунтування В даній Кваліфікаційній роботі студентом надано декілька пропозицій щодо вирішення поставлених задач. Кожна з пропозицій була об обґрунтована та підкріплена науковими даними.

Практичне значення роботи Результати роботи можуть бути застосовані для подальших наукових досліджень в даній сфері, а також вони можуть бути корисними для практичного використання.

Якість оформлення роботи Магістерська кваліфікаційна робота, яку подано на рецензію, виконана у повному обсязі у встановлений термін. Робота є добре структурованою та достатньо проілюстрованою. Викладена основна суть проблеми, що вирішується в ході виконання роботи, і шляхів її вирішення.

Недоліки в роботі відсутність більш детального аналізу проблеми кросплатформенності. Проте вказаний недолік не впливає на позитивне враження від роботи.

Загальний висновок Магістерська кваліфікаційна робота виконана у
(підготовленість студента до самостійної роботи як спеціаліста)

відповідності з завданням із дотриманням всіх вимог.

Оцінка магістерської роботи Робота заслуговує оцінки «відмінно», а студент Кунденко П.Р. – присвоєння кваліфікації «магістра» з комп'ютерних наук.

Рецензент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання, посада, місце роботи)

«___»_____20__ р. _____
(підпис)

Додаток Г

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	Опис
Пояснювальні документи	
Кунденко_диплом.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кунденко_диплом.pdf	Пояснювальна записка до кваліфікаційної роботи у форматі PDF
Програма	
moviesgalery.rar	Архів. Містить вихідні коди програми, файли ресурсів і компільовану програму
Презентація	
Кунденко_презентація	Презентація кваліфікаційної роботи