Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

ДИПЛОМНИЙ ПРОЕКТ

Розробка мобільного додатку з використанням багатоплатформних фреймворків з нативним виводом
Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр ДПІПЗ. 190157.19.09.ПЗ

Виконав студент III курсу група ІПЗс-19-1 _____ А.А. Починок
Підпис Ініціали, прізвище

Керівник к. т. н., доцент _____ І.В. Гурман
Науковий ступінь, звання Підпис Ініціали, прізвище

Нормоконтролер к. т. н., доцент _____ Ю.В. Форкун
Підпис Ініціали, прізвище

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Л. П. Бедратюк
Підпис Ініціали, прізвище

13 червня 2022 р.

Хмельницький 2022

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри _____
Л. П. Бедратюк
01 03 2022 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ)

Починку Артуру Андрійовичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Розробка мобільного додатку з використанням багатоплатформних фреймворків з нативним виводом

Керівник проекту (роботи) Гурман Іван Васильович, канд. техн. наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 01.03.2022 р. № 18
2. Строк подання студентом проекту (роботи) на кафедру 01.06.2022 р.
3. Вихідні дані до проекту (роботи) Матеріали переддипломної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження предметної області та постановка задачі, проектування програмного забезпечення, програмна реалізація, тестування програмного забезпечення

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди, 14 шт.)

6. Консультанти розділів дипломного проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдан прийн
Нормоконтроль	Форкун Ю.В., доцент кафедри ІПЗ		
Антиплагіат	Гурман І.В., доцент кафедри ІПЗ		

7. Дата видачі завдання « 01 » березня 2022р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Прим
1 Ознайомлення з тематикою дипломного проектування (ДП), визначення та узгодження індивідуальних там ДП	01.12 – 30.12.2021	
2 Дослідження предметної області, в якій планується використання програмного засобу (ПЗ), визначення задач та вимог, розробка технічного завдання	02.01 – 31.01.2022	
3 Проектування програмного забезпечення	01.02 – 28.02.2022	
4 Програмна реалізація	01.03 – 10.04.2022	
5 Тестування програмного забезпечення	11.04 – 30.04.2022	
6 Написання вступу, загальних висновків, оформлення джерел посилання та додатків. Оформлення пояснювальної записки ДП згідно вимог стандартів	01.05 – 25.05.2022	
7 Попередній захист ДП	Травень 2022 (згідно графіка)	
8 Перевірка ДП на плагіат, нормконтроль, отримання відгуків та рецензій. Брошурування (зшиття) пояснювальної записки	26.05 – 30.05.2022	
9 Підготовка до захисту та захист ДП	з 01.06.2022	

Студент _____ А.А. Починок
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ І.В. Гурман
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема дипломного проекту: Розробка мобільного додатку з використанням багатоплатформних фреймворків з нативним виводом.

Автор проекту: Починок Артур Андрійович.

Керівник проекту: Гурман Іван Васильович.

Пояснювальна записка: 59 с., 17 рис., 4 табл., 3 дод., 21 джерело.

Графічна частина: 14 слайдів.

MULTIPLATFORM APPLICATION, REACT NATIVE, ANDROID, IOS, JAVASCRIPT, XAMARIN, NATIVESCRIPT, NATIVE APPLICATION.

Метою цієї бакалаврської роботи є дослідження поточних доступних варіантів розробки мультиплатформних додатків, оцінка їх якостей і недоліків і представлення React Native як найкращого доступного на даний момент рішення.

Під час виконання дипломного проекту складено специфікацію вимог та розроблено зручний для користувача інтерфейс.

Даний програмний продукт буде надавати можливість користувачам *виконувати оплату паркувальних місць з автоматичним розрахунком вартості,* у залежності від місця паркування, та з врахуванням місця проживання та роботи користувача.

Під час виконання проекту було проведено огляд та порівняння кросплатформенних фреймворків для розробки додатків.

10.06.22р
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	ДПІПЗ. 190157.19.09.ПЗ	Пояснювальна записка	83		
2	А4		Завдання на дипломний	1		
			проект			
3	А4		Анотація	1		
			Графічні документи			
4	А4		Презентаційні матеріали	14		

Змн.	Арк.	№ докум.	Підпис	Дата	ДПІПЗ. 190157.19.09.ВД		
Розроб.		Починок А.А.		9.06	Розробка мобільного додатку з використанням багатоплатформних фреймворків з нативним виводом Відомість документів	Літ.	Арк. 1 / Акрушів 1
Перевір.		Гурман І.В.		10.06			
Реценз.						ХНУ, ІПЗс-19-1	
Н. Контр.		Форкун Ю.В.		10.06			
Затверд.		Бедратюк Л.П.		13.06			

ЗМІСТ

ВСТУП

У своїй першій частині робота зосереджується на визначенні мобільного додатка та його основних вимог, а також порівнянні різних підходів до розробки мобільних додатків, включаючи їх переваги та недоліки.

У другій частині робота спрямована на встановлення визначення розробки багатоплатформних мобільних додатків. Крім того, у розглядаються багатоплатформні типи розробки та багатоплатформні типи фреймворків. Наприкінці проводиться глибоке порівняння кількох мультиплатформних фреймворків, щоб зробити висновок, яка з них найкраще підходить для сучасної мультиплатформної розробки.

У наступній частині фокус дипломного проекту полягає в тому, щоб надати поглиблений погляд на React Native та технології, які використовуються з React Native. Далі в дипломному проекті детально розглядаються їх функціональні можливості, переваги, недоліки та аргументи щодо вибору технологій для практичного застосування.

Нарешті, у дипломному проекті представлено демонстраційний додаток, написаний на React Native, з мотивацією та визначенням проблеми, аналізом із дизайном структури та демонстрацією реалізації з фотографіями та варіантами використання програми. Проводиться оцінка процесу розробки з обговоренням можливих напрямків, у які може розвиватися отримана програма.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

Цей розділ зосереджений на визначенні терміну мобільного додатка та визначенні основних потреб для розробки мобільного додатка. Він також представить різні підходи до розробки мобільних додатків, а також зручності та недосконаліть кожного підходу до розробки.

1.1 Визначення мобільного додатку

Під мобільним додатком розуміють тип виконуваного програмного забезпечення, призначеного для роботи на мобільному пристрої, наприклад, смартфоні чи планшеті. Мобільні програми, як правило, вважалися невеликими окремими програмними блоками з обмеженою функціональністю. Сьогодні мобільні додатки функціонально порівняні з додатками для персональних комп’ютерів. На відміну від додатків для персональних комп’ютерів, мобільні додатки зазвичай отримують із відповідного магазину програм.

1.2 Основні вимоги до мобільного додатка

Залежно від випадку використання, вимоги до програми можуть сильно відрізнятися, різниця в основному полягає в необхідних системних ресурсах і спеціальних функціях пристрою.

Основною вимогою до мобільного додатка є його легкість у використанні. Звичайним ПК зазвичай керують за допомогою клавіатури та миші, у порівнянні зі звичайним розумним пристроєм, яким зазвичай керують виключно за

допомогою сенсорного екрана, що охоплює лише кілька дюймів. Переважна більшість цих пристроїв керується за допомогою пальця, причому раніше популярний стилус тепер є неприйнятним варіантом.

Обмежений розмір екрана смарт-пристрою є основною причиною вимог до користувальницького інтерфейсу – елементи керування повинні мати достатній розмір, забезпечувати зручне використання на пристрої малого діаметру та бути розумно розміщеними. Це означає, що користувач програми

Інтерфейс має відповідати рекомендаціям щодо людського інтерфейсу для конкретної платформи*c*1,використання рідних компонентів, де це можливо, - наприклад, вбудованої сенсорної клавіатури, рідного набору піктограм або рідного меню. Причина цієї вимоги полягає в тому, що користувачі, як правило, вже звикли використовувати ці компоненти, тому якщо вони ідентичні в кількох програмах, використання програми буде менш заплутаним.

Розробник мобільного додатка також повинен враховувати економічну сторону розробки мобільного додатка. У фінансовому плані розробка мобільних додатків має кілька характеристик, що відрізняються від розробки веб-сайтів або персональних комп’ютерів.

Вигідно для розробника, кілька платформ обмежують шляхи поширення програми – iOS або Windows Phone зазвичай дозволяють встановлювати програму лише через офіційний магазин виробника платформи. Фактично, отримати нелегальну копію програми набагато важче, ніж на персональному комп’ютері.

З іншого боку, за програму для персонального комп’ютера зазвичай платять значно більше, ніж за мобільний. Для цього розробник повинен мати можливість продати велику кількість копій, або скористайтеся іншим способом монетизації.

На рисунку 1.1 показано середні ціни програми в Apple App Store за червень 2021 року. Джерело[1]зазначає, що середня ціна становить приблизно 1,02 дол.

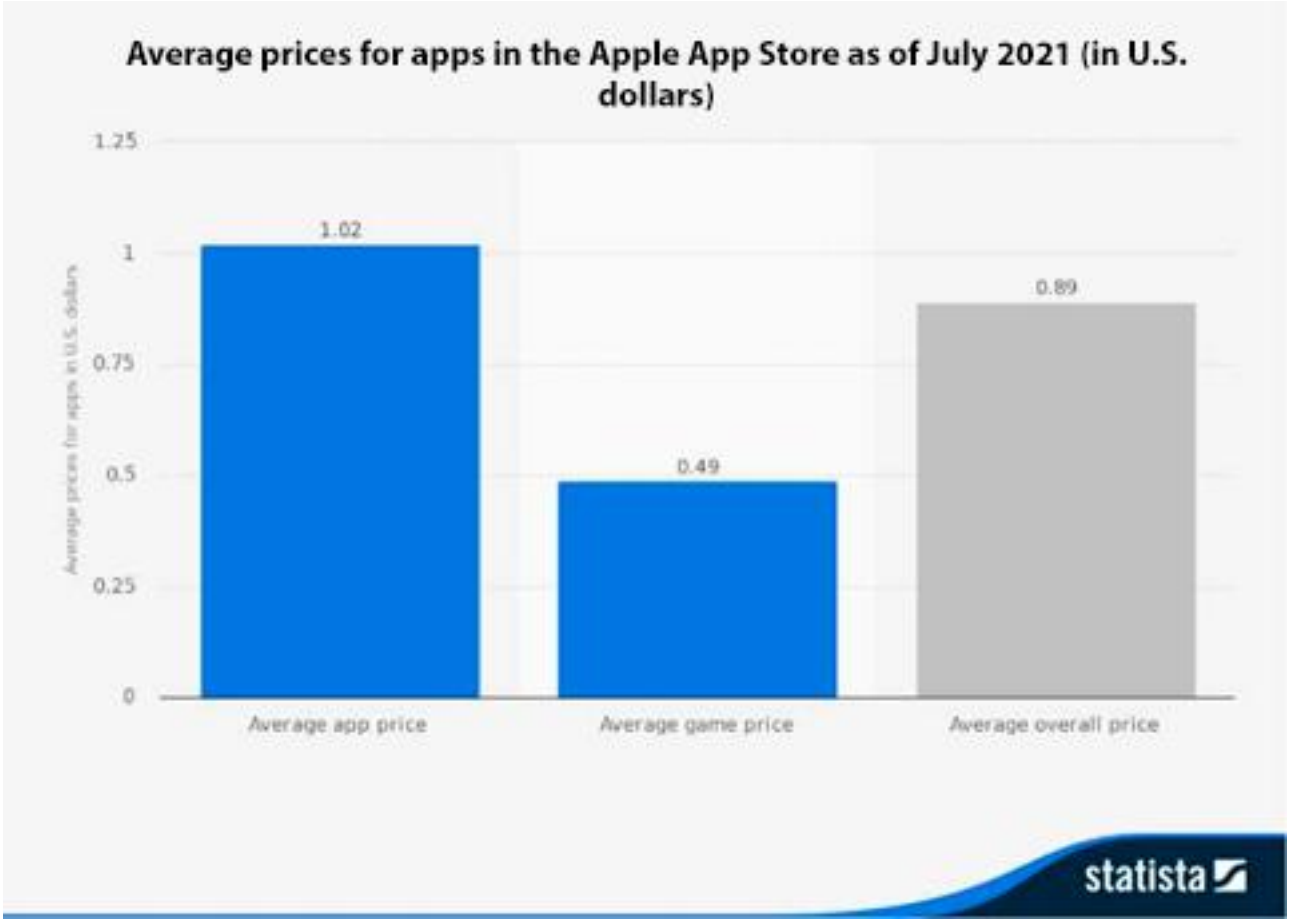


Рисунок 1.1 – Середня ціна програми в магазині iOS, липень 2021 року[3]

Мобільні програми також впливають на погляди користувачів на фінансування програмного забезпечення. У той час як модель одноразової покупки програми все ще переважає на персональних комп'ютерах, мобільні додатки, як правило, або дуже дешеві, або безкоштовні, а фінансування зазвичай вирішується за допомогою моделі freemium. Це включає покупки в програмі або модель підписки. Ця тенденція показана на обох рисунках – 1.1 і 1.2. Перший показує дешеві середні ціни на мобільні додатки, а другий показує тенденцію випуску розробниками або безкоштовних, або дуже дешевих програм.

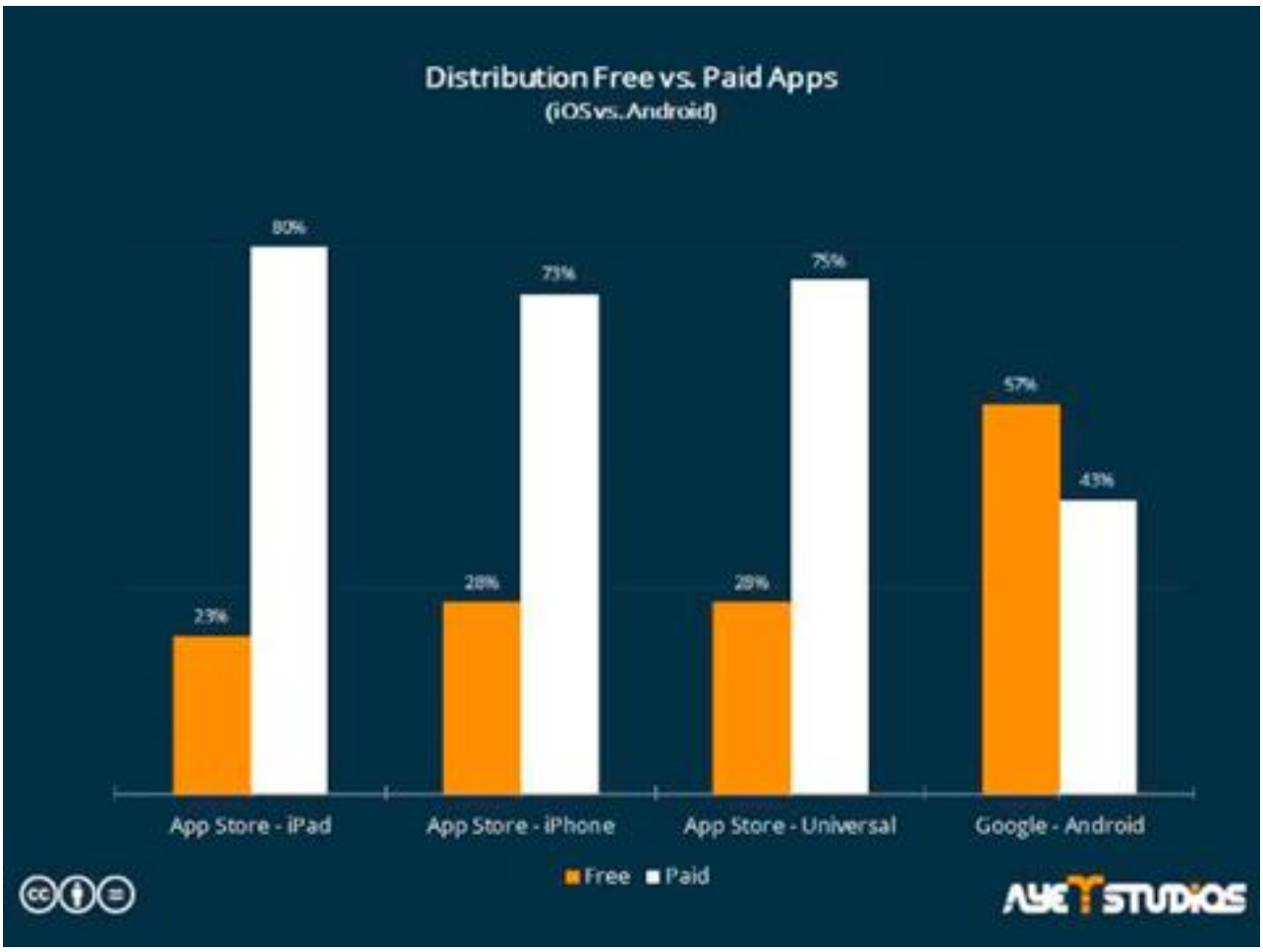


Рисунок 1.2 – Розповсюдження безкоштовного та платного додатків на iOS та Android, липень 2021 [2]

Важливо, щоб мобільний додаток був плавним і плавним, щоб користувацький досвід був комфортним. На персональних комп'ютерах користувачі, як правило, звикли до часу відповіді програм, тоді як на мобільних пристроях більшість користувачів очікує плавного, швидкого та стабільного використання. Таким чином, мобільний додаток повинен негайно реагувати на введення користувача і жодним чином не може вплинути на інші функції пристрою (наприклад, програма блокує користувача від прийняття телефонного дзвінка).

Мобільний пристрій зазвичай живиться від акумулятора з обмеженою ємністю. Тому програма, яка працює на пристрої, має прагнути використовувати якнайменшу можливу кількість акумулятора – наприклад, використовуючи

системні ресурси лише тоді, коли це абсолютно необхідно, уникаючи непотрібних операцій та обчислень або підтримуючи мережеве з'єднання довше, ніж необхідно.

Програма повинна розумно реагувати на безліч зовнішніх ефектів, таких як розрядка акумулятора, вимкнення пристрою, примусове припинення, нестабільне з'єднання даних або неточні дані з датчика GPS. Це означало б – у випадку з першими трьома згаданими – що користувач не втрачає жодних даних або не отримує відмову у вході після пізнішого повторного запуску програми.

2.3 Типи розробки мобільних додатків

Під час розробки мобільного додатка доцільно розділити отриману програму на менші розділи. Спочатку два основні розділи:

– Back-End (серверна частина) – зазвичай реалізує API (інтерфейс програмного програмування) для полегшення доступу до серверних служб, наприклад, редагування та доступ до даних, аутентифікація або складні операції, які не підходять для виконання на пристрої клієнта.

– Front-End (клієнтська частина) - зазвичай представляє набір інтерфейсів користувача, які реалізують послуги, що надаються сервером, через його API. Ця частина зазвичай специфічна для кожного конкретного типу клієнтського пристрою, на відміну від серверної частини.

Важливо зазначити, що мобільний додаток не обов'язково повинен мати внутрішню службу, наприклад, у простий ігровій програмі не буде використовуватися сервер.

Основна увага в дипломному проекті зосереджена на Front-End частині програми, зокрема на тому, як її розробити за допомогою багатоплатформної розробки. Сегмент Front-End розглядається або як веб-інтерфейс, або як

мобільний додаток для Android та iOS, тоді як сегмент BackEnd сприймається як вже готовий продукт, що надає API для сегмента Front-End для використання.

Під час розробки веб-інтерфейсу перед початком розробки мобільного додатка рекомендується вибрати метод розробки. Наразі існує три різні підходи до розробки мобільних додатків – веб-розробка, нативна та гібридна розробка.

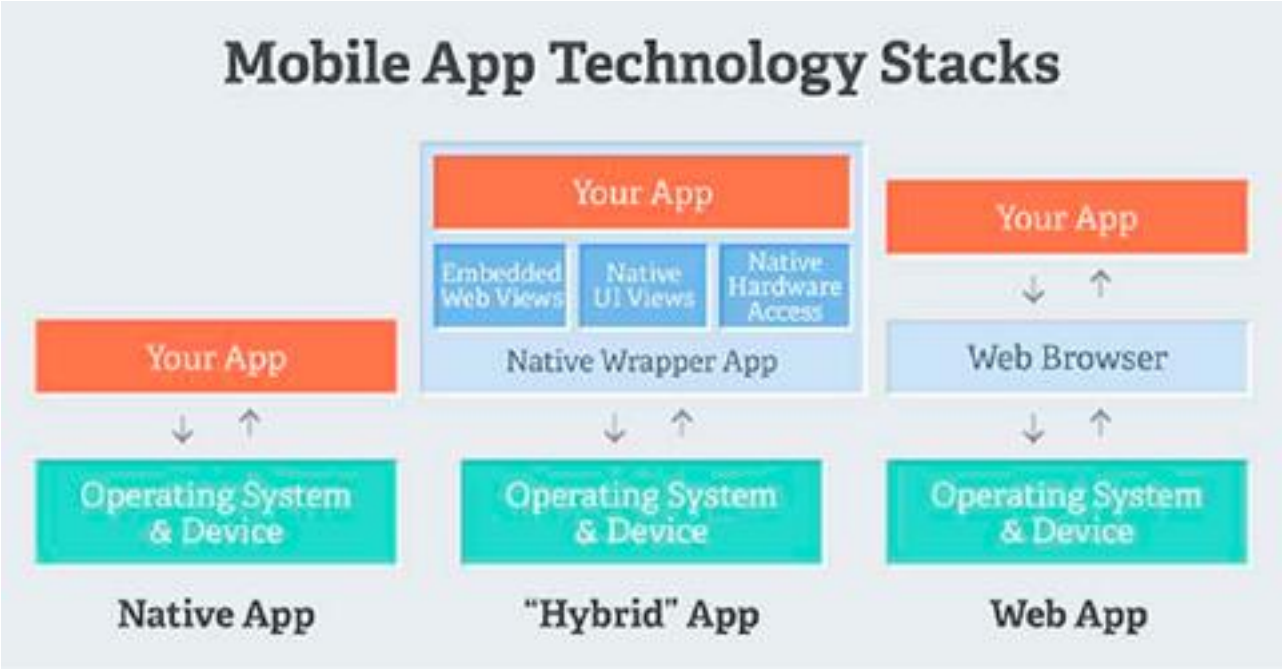


Рисунок 1.3 – Архітектури нативних, гібридних і веб-додатків[3]

Розробка додатка в оригінальному вигляді вважається розробкою окремої програми для кожної платформи незалежно. Під час розробки програми розробник змушений використовувати інструменти, необхідні для певної платформи, наприклад, певну мову програмування (Java, Kotlin, Swift,...) або конкретне середовище розробки (наприклад, Xcode).

Переваги:

– Оптимізований користувальницький інтерфейс – користувальницький інтерфейс оптимізовано для кожної платформи відповідно до її конкретних вказівок щодо дизайну.

– Висока плавність і швидкість реагування. Додаток є значно більш плавним і чуйним, ніж веб-додаток, що позитивно впливає на роботу користувачів.

– З'єднання з Інтернетом не потрібне. Застосовуючи лише деякі програми, власні програми за замовчуванням не забезпечують необхідний доступ до Інтернету під час використання.

Недоліки

– Окрема розробка. Нативні програми вимагають окремої розробки для кожної платформи.

– Дорога розробка. Через те, що необхідно підтримувати кілька реалізацій на всіх підтримуваних платформах, страждають витрати на розробку та обслуговування.

– Складне обслуговування. Розробник, як правило, розгортається в магазині додатків, не може довіряти користувачеві оновлення після кожного оновлення, що робить підтримку старих версій необхідністю. Хоча розробник може змусити користувача оновлюватися, відмовляючи в доступі до програми до завершення оновлення, цей примусовий підхід не дуже подобається користувачам.

Веб-розробка

Мобільний додаток складається з веб-сторінки, оптимізованої для використання як на мобільних пристроях із сенсорною підтримкою, так і на настільних комп’ютерах. Програма адаптується відповідно до пристрою, з якого до нього здійснюється доступ, і доступ до нього здійснюється виключно через веб-браузер пристрою. Цей підхід до розробки зазвичай використовує HTML, CSS і JavaScript, а також позначається як адаптивний веб-дизайн.

Переваги:

– Єдина програма для всіх платформ. Розробка єдиної веб-додатки для виконання всіх необхідних умов замість кількох подібних версій для всіх платформ.

– Нижчі витрати на розробку. Оскільки немає необхідності розробляти конкретну програму для кожної платформи, вартість розробки значно знижується.

– Просте розгортання та обслуговування. Необхідно підтримувати лише один вихідний код. Оновлення програми здійснюється шляхом розгортання нової програми на сервері, що, можливо, набагато простіше та економніше, ніж розповсюдження програми на кількох платформах. Оновлення відбувається миттєво і без взаємодії з користувачем.

Недоліки:

– Інтерфейс користувача не оптимізований. Інтерфейс користувача суттєво відрізняється на iOS та Android, що зазвичай не враховується у веб-додатках.

– Нижча плинність та швидкість. Веб-додаток не може повністю використати потенціал пристрою, що, можливо, призведе до посереднього користувацького досвіду з проблемами плавності та продуктивності.

– Необхідне підключення до Інтернету. Щоб підключитися до веб-програми, потрібне підключення до Інтернету.

Гібридний розвиток

Гібридні додатки розглядаються як точка зустрічі між нативними підходами та підходами до веб-розробки. Вони являють собою комбінацію цих двох методів, орієнтованих на усунення недоліків обох вищезгаданих методик.

З точки зору кінцевого користувача, програма діє так само, як і рідна програма, тобто вона доступна в магазині додатків, сама встановлюється на пристрої та оновлюється через магазин програм.

Дивлячись на програму з точки зору розробника, програма розроблена з використанням веб-технологій - HTML, CSS і JavaScript, при цьому міститься у нативній обгортці. Під час запуску програми обгортка створює WebView - мінімальний повноекранний веб-браузер і завантажує веб-програму всередині браузера. Такий підхід дозволяє повторно використовувати вихідний код у різних програмах і платформах, зменшуючи витрати на розробку.

При гібридній розробці все одно необхідно розробляти додаток окремо для кожної цільової платформи. Завдяки повторному використанню більшості кодової бази на всіх платформах програма все одно потребує підтримки на кількох платформах одночасно. Щоб допомогти з одночасною розробкою на кількох платформах одночасно, кілька інструментів, таких як Marmalade або Phonegap, надають допомогу в розробці та поширенні отриманої програми.

Переваги:

– Інтерфейс користувача для певної платформи – простий інтерфейс користувача можна адаптувати залежно від платформи, на якій розгорнуто додаток (через рідну оболонку)

– Повторне використання коду. Повторне використання коду сприяє більш плавному та швидшому процесу розробки

– Нижчі витрати на розробку. Завдяки повторному використанню більшої частини кодової бази витрати на розробку нижчі в порівнянні з розробкою на власному пристрої.

Недоліки:

– Нижча плавність і швидкість – гібридна програма не може повністю використати потенціал пристрою, що зазвичай призводить до погіршення роботи користувачів із проблемами плавності та продуктивності.

– Складне обслуговування та розгортання. Програму необхідно розгортати в сховищах додатків для певної платформи, що ускладнює процеси оновлення та розгортання.

– Складне налагодження. Оскільки для кожної платформи потрібно використовувати різні WebView, налагодження програми може виявитися проблематичним. Їхні реалізації сильно відрізняються від платформи до платформи та від версії до версії.

– Потенційно вище споживання енергії. Неможливість мікрооптимізації програми на низькому рівні потенційно може призвести до небажаного розрядження акумулятора.

1.4 Порівняння різних методів

Існує кілька основних рис, спільних для всіх методів розробки додатків. Таблиця 1.1 надає стисле порівняння того, наскільки успішні методи в кожній конкретній точці.

Таблиця 1.1 – Порівняння методів розробки додатків

Особливості	Рідний	Веб	Гібридний
Інтерфейс користувача	Чудово	Середній	добре
Досвід користувача	Чудово	Середній	Середній
Час розробки	Високий	Низька	Середній
Витрати	Високий	Низька	Середній
Розгортання	Складний	Просто	Складний
Підтримка	Варіюється	Варіюється	Варіюється
Автономний режим	Так	Ні	Так*

Перш ніж підсумувати результати таблиці, необхідно пояснити, чому кажуть, що підтримка відрізняється на кожній окремій платформі. Дуже важко визначити складність обслуговування додатків виключно на основі підходу до розробки; добре розроблену рідну програму може бути значно легше підтримувати, ніж веб- або гібридну програму. З іншого боку, погано продумана рідна програма буде страждати від складної процедури обслуговування в порівнянні з гібридною або веб-додатком.

Підводячи підсумок таблиці, очевидно, що основна перевага власного підходу полягає в чудовому користувальницькому інтерфейсі та досвіді користувача за ціною вищих витрат на розробку. Кожен підхід до розвитку служить іншій меті та затьмарює інші в конкретних ситуаціях.

2 БАГАТОПЛАТФОРМНИЙ ПІДХІД

У цьому розділі зосереджено на обґрунтуванні багатоплатформного підходу до розробки мобільних додатків. По-перше, розрізняють подібні багатоплатформні підходи. Далі в цій главі пропонується кілька популярних фреймворків для цього типу розробки мобільних додатків. Нарешті, проводиться глибоке порівняння, завершується рішенням, яка структура буде використана для практичного розділу дипломної роботи.

Термін багатоплатформний додаток розуміється як додаток, що містить кілька точок входу на різних платформах. Платформою може бути, наприклад, конкретна операційна система мобільного чи настільного пристрою або веб-інтерфейс. Іншими словами, багатоплатформний додаток – це програма, доступна з усіх цільових платформ лише з незначними модифікаціями[4].

2.1 Багатоплатформні види розробки

По суті розрізняється два багатоплатформних підходи до розробки; один із них є гібридним багатоплатформним підходом, а інший є власним багатоплатформним підходом.

Гібридний багатоплатформний підхід – це в основному гібридний підхід, згаданий у попередньому розділі. Цей підхід використовується, наприклад, Apache Cordova/Phonegap.

З іншого боку, нативний мультиплатформний підхід відрізняється від гібридного тим, що отримана програма є повністю рідною програмою, яка використовує як виклики нативного API до функціональних можливостей пристрою, так і рідні компоненти. Важливо зазначити, що результати різних

нативних мультиплатформних фреймворків сильно розходяться за своєю «рідністю». Цей підхід використовується, наприклад, у Xamarin або React Native.

2.2 Нативний багатоплатформний додаток

Нативний багатоплатформний додаток – це програма, яка не реалізована рідною мовою платформи, а використовує іншу мову програмування, яка зазвичай не підтримується платформою.[5]. Зазвичай середовище розробки, що обробляє власний пакет програм, надається третьою стороною. Отриманий продукт являє собою програму, що використовує рідний API, з майже рідною продуктивністю.

Нативний мультиплатформний підхід розробки має більшість переваг із гібридним підходом розробки, але також розширює ці переваги, додаючи деякі свої власні. Насамперед, нативний мультиплатформний підхід вирішує найбільшу проблему гібридних додатків, знижену плинність і чутливість. він також вирішує проблему WebView, пов’язану з вимушеним покладатися на хорошу реалізацію WebView, або використовувати плагін на зразок Cordova Crosswalk для окремої реалізації – збільшення розміру натомість. Нативні багатоплатформні програми взагалі уникають WebView, натомість покладаючись на використання лише рідних компонентів.

2.3 Найвідоміші мультиплатформні фреймворки

Подальші фреймворки були обрані в основному за їх загальну популярність і за їх добре виконані підходи.

Xamarin – набір інструментів для розробки додатків, створений компанією Xamarin. Він заснований на Mono, проекті з відкритим кодом, що реалізує кілька

інструментів, що використовуються в .NET для багатоплатформної розробки. Через свої моно-коріння Xamarin використовує C# як свою основну мову. Хоча набір інструментів Xamarin містить інструменти, орієнтовані на певні платформи, як-от Xamarin.iOS або Xamarin.Android, основна увага буде спрямована на Xamarin.forms. У той час як інструменти для певних платформ, таких як Xamarin.iOS, зазвичай досягають близько 70% спільного використання коду, Xamarin.forms, як кажуть, здатний досягти до 100% спільної бази коду[6].

Просте порівняння різних інструментів Xamarin показано на рисунку 2.1.

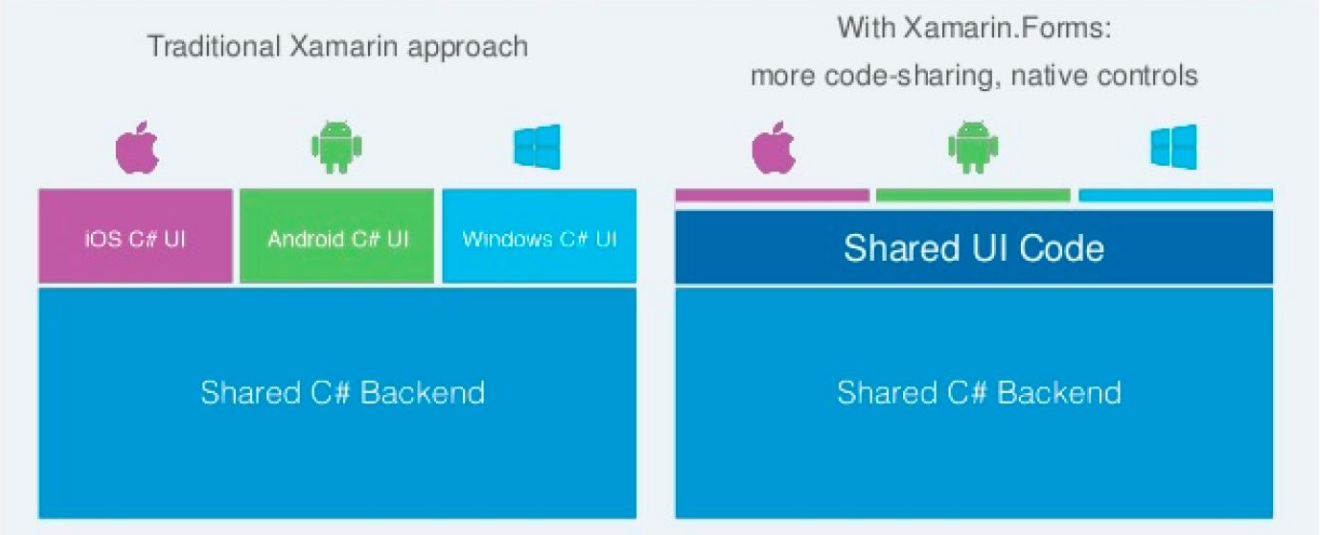


Рисунок 2.1 – Порівняння традиційних Xamarin і Xamarin.forms

У минулому Xamarin використовував власний IDE Xamarin Studio для розробки та розгортання додатків. Він також мав обмежену початкову версію, яка дозволяла повноцінно використовувати до певного розміру скомпільованого коду, що обмежувало можливості розробника. Початкова версія також не дозволяла використовувати Xamarin.Forms.

Нещодавно Xamarin була придбана Microsoft Corp., що призвело до кількох серйозних змін. Найбільшими змінами буде повна інтеграція інструментів Xamarin з Visual Studio та зняття деяких обмежень початкової версії, замість того щоб зробити Xamarin відкритим вихідним кодом.[7].

Серед небагатьох сервісів, які все ще доступні лише для преміум-учасників, є Xamarin Test Cloud, хмарна служба, яка дозволяє розробнику виконувати автоматизовані тести на тисячах різних пристроїв.[8]. Цінова модель Xamarin Test Cloud починається від 99 доларів США на місяць з одночасним використанням одного пристрою та однієї години пристрою на день, із зростанням ціни обох.

Додаток Xamarin компілюється у рідний двійковий файл, а не інтерпретується. На Android Xamarin постачає середу виконання Mono в комплекті з програмою, що призводить до JIT компіляція типова для Android додатків. На iOS знову використовується Mono, але замість цього компілює програму AOT у рідний виконуваний файл[9].

Ionic 2 – це фреймворк з відкритим вихідним кодом, створений компанією Drifty Co. для розробки мобільних додатків із використанням лише HTML5, CSS та JavaScript. З понад 32 000 зірок на GitHub[10], яскрава, активна спільнота та добре виготовлена документація з наочними прикладами[11], Ionic називають найпопулярнішою гібридною мобільною розробкою. Потужність Ionic походить від використання фреймворку Angular, фреймворку Apache Cordova та SASS.

Angular – це фреймворк JavaScript з відкритим кодом, спонсорований і підтримуваний Google LLC, який використовує MVC архітектурна модель. Метою Angular було спростити розробку та тестування веб-додатків, використовуючи архітектурну модель MVC. Angular розширює основний синтаксис HTML, додаючи спеціальні атрибути, класи та елементи. Ці налаштовані маркери, які називаються директивами, призводять до того, що компілятор Angular додає користувацькі поведінки до відповідного DOM вузли.

Apache Cordova – це платформа для розробки мобільних додатків JavaScript з відкритим вихідним кодом, заснована на оригінальному PhoneGap. Після придбання Adobe оригінального PhoneGap[12], вони пожертвували PhoneGap до The Apache Software Foundation під назвою Cordova. Зараз Adobe розробляє свою екосистему PhoneGap, яка базується на движку Cordova, але реалізує додаткову функціональність. Cordova реалізує набір API, які створюють міст від JavaScript

до рідного коду, що використовується для безпосереднього доступу до рідних ресурсів смартфона. За замовчуванням Cordova має набір готових плагінів для доступу, наприклад, до мікрофона чи акселерометра. Наявні готові API узгоджені на кількох платформах, включаючи Android, iOS, Windows Phone і навіть Blackberry.

SASS (Syntacically Awesome Style Sheets) – це попередній процесор CSS, який розширює можливості CSS. SASS, популярний завдяки своєму простому синтаксису та потужним функціям, компілює SASS (.scss) у файли CSS. Серед його головних нововведень – імпорт і розширення інших файлів .scss, вкладення селекторів CSS, змінних, міксів, подібних до HTML, і математичні оператори.

Нещодавно компанія Drifty Co. випустила нову версію Ionic під назвою Ionic 2, щоб відзначити перехід від фреймворку Angular до його новітньої ітерації Angular 2+. Перехід призвів до значного підвищення продуктивності та зниження складності коду.

На завершення, Ionic реалізує фреймворк Angular для свого потужного підходу до створення веб-додатків, фреймворк Cordova для своїх потужних рідних API і препроцесор SASS для багатьох доповнень якості життя до базового CSS.

NativeScript

NativeScript є нативною багатоплатформною структурою з відкритим вихідним кодом. Він зосереджений на використанні JavaScript, CSS і XML, з можливістю використання вже згаданої вище рамки Angular. Він був створений і наразі підтримується Telerik AD. NativeScript дозволяє розробнику створювати власні програми для Android та iOS, планується підтримка Windows UWP. Отримана програма має нативну продуктивність і користувацький досвід[13].

Основною особливістю NativeScript є відображення рідного API. Замість того, щоб використовувати окремий проміжний шар між NativeScript і певними мобільними платформами, NativeScript використовує відображення для створення відображення всієї екосистеми.[14]. Використовуючи це відображення,

NativeScript може легко викликати рідний код – наприклад, створюючи екземпляр об’єкта Java та викликаючи його методи (див. у лістингу)[15].

```
1 let time = new android . text . format . Time ();
2 time .set( 1, 0, 2015 );
3 console .log( time . format ( "%D" ) );
```

Декларація інтерфейсу користувача в NativeScript зазвичай виконується за допомогою файлів XML, які потім компілятор NativeScript використовує для виклику нативних елементів інтерфейсу користувача для кожної конкретної платформи. NativeScript підтримує використання чистого JavaScript або використання TypeScript з платформою Angular.

TypeScript є доповненням із відкритим кодом мови JavaScript, розробленої компанією Microsoft Corp., що надає багато додаткових функцій базовій мові. Серед цих функцій є інтерфейси, перевірка типів під час компіляції (як показано на рисунку 2.2), загальні типи та анотації типів – JavaScript за замовчуванням вводиться нечітко. TypeScript здобув прихильність серед розробників, які звикли працювати з об'єктно-орієнтованими мовами високого рівня, в основному завдяки сильному застосуванню типів, інтерфейсам та багатьом іншим об'єктно-орієнтованим доповненням до якості життя.

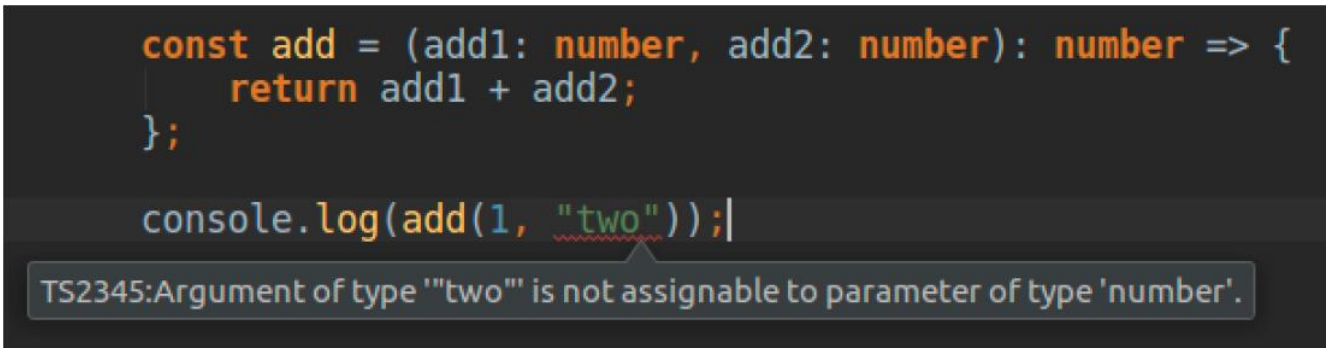


Рисунок 2.2 – Приклад застосування типів TypeScript

React Native є багатоплатформенним фреймворком з відкритим вихідним кодом для створення рідних додатків за допомогою JavaScript і React. Він був створений і підтримується компанією Facebook, Inc. React Native дозволяє розробнику створювати нативні програми для Android та iOS з розширенням, що дозволяє розробляти для Windows UWP React Native використовує свою власну мову - JSX - розширення синтаксису, подібне до XML, для ECMAScript без будь-якої визначеної семантики[16]. Зазвичай це транспіляція в JavaScript за допомогою Babel. Разом з JSX і React Native розробник використовує декларативні компоненти інтерфейсу користувача, які компілюються в власні елементи для кожної конкретної платформи.

Однією з визначних функцій React Native є гаряче перезавантаження - спосіб швидкого перезавантаження програми зберігаючи стан програми. Очевидно, що це лише вигідно розробникам, дозволяючи їм набагато швидше взаємодіяти з розробленим додатком, що, у свою чергу, призводить до швидшого виробничого циклу. Раніше Live Reload використовувався в React Native, що дозволяло так само швидко перезавантажувати програми, але не зберігало стан програми. Це може викликати роздратування під час розробки, наприклад, функції, яка містить лише кілька екранів від точки входу програми, або під час налаштування незначних деталей у дизайні інтерфейсу користувача. Натомість Hot Reloading вводить нові версії відредагованих файлів у запущену програму. Проста схема концепції Hot Reloading показана на рисунку 2.3.

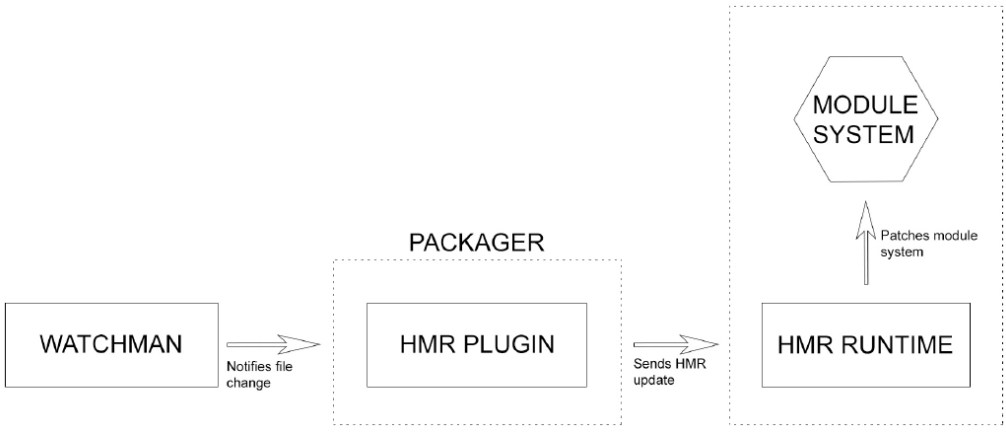


Рисунок 2.3 – Схема перезавантаження гарячого модуля в React Native

З функцією гарячого перезавантаження безпосередньо пов’язана проста обробка помилок і налагодження React Native. У разі нестандартних дій у програмі розробнику негайно (завдяки функції гарячого перезавантаження) надається екран помилки, що відображається як повноекранний звіт із повідомленням про помилку та відповідним трасуванням стека. Доступ до цих журналів JavaScript зазвичай можливий за допомогою react-native log-android, або react-native log-ios команд, відповідно на Android та iOS. Іншим варіантом буде використання веб-браузера, який підтримує інструменти розробника Chrome. По-перше, розробник вмикає віддалену налагодження JavaScript у програмі, що, у свою чергу, дозволяє підключатися до програми через веб-браузер. У браузері ми можемо переглядати журнали програми та використовувати всі стандартні функції налагодження, властиві веб-розробці. Ці функції включають перегляд поточних значень змінних, використання точок зупинки для призупинення програми в критичних точках або покрокове виконання методу.

Також важливо зазначити, що React Native може похвалитися дуже великою, постійно зростаючою спільнотою в порівнянні з іншими згаданими фреймворками. Це призводить до збільшення кількості учасників, більш релевантних питань із правильними відповідями на Stack Overflow, і більше бібліотек з відкритим кодом для спрощення процесу розробки.

2.4 Порівняння

Щоб краще розрізняти окремі особливості вищезгаданих фреймворків, необхідно розділити ці функції на кілька розділів. Атрибути розділені на основі точок зору, щоб допомогти розрізнити їх конкретні якості. Такими точками зору є, а саме, точка зору розробника, точка зору щодо ціни та точка зору користувача.

Перш ніж представити стисле порівняння точки зору розробника у вигляді таблиці, важливо спочатку встановити оцінювані точки та їх значення.

– Простота використання передає накладні витрати, необхідні до і після фактичної розробки. Це включає такі речі, як налаштування IDE, налаштування проекту, створення додатків і (повторне) розгортання додатків.

– Якість життя означає покращення та переваги, які має конкретна платформа для розробників у порівнянні з іншими фреймворками. Це може означати специфічний підхід до програмування, складність інтеграції з нативними компонентами або додатками або різні вдосконалення, які скорочують час простою та прискорюють роботу розробника, як-от Hot Reload в React Native або Live Reload в Ionic 2.

– Спільнота вказує на загальний розмір спільноти навколо рамки. Розмір спільноти буде прямо пропорційним до таких речей, як обсяг сторонніх бібліотек, кількість пов'язаних питань (і рішень) на таких веб-сайтах, як Stack Overflow, або імпульсу фреймворку.

– Додаткові функції представляють будь-які додаткові функції, які має фреймворк порівняно з іншими, будь то функція Test Cloud від Xamarin або будь-яка інша функція, яка полегшує цикл розробки.

Таблиця 2.1 – Порівняння фреймворків з точки зору розробника

Framework	Ease-Of- Use	Quality-of-Life	Community	Extras
Xamarin	Difficult	Medium	Large	Test Cloud
Ionic 2	Easy	Medium	Medium	Codepusha
NativeScript	Medium	Low	Medium	Sidekickb
React Native	Easy	High	Large	Codepush

Цінова перспектива

Перш ніж представити стисле порівняння перспективи ціноутворення у вигляді таблиці, важливо спочатку встановити оцінювані точки та їх значення.

– Витрати на розробку позначають відносні витрати на розробку, на які впливає специфічний підхід фреймворку до багатоплатформної розробки або повторне використання бази коду на кількох платформах.

– Ціни на фреймворк натякають на те, платний чи безкоштовний фреймворк.

– Витрати на розгортання символізують витрати, пов'язані з розгортанням програми на кількох платформах, оновленням програми та обслуговуванням програми. Хоча витрати на розгортання, як правило, мають залежати від фреймворку, різниця здебільшого походить від витрат, спричинених оновленням та підтримкою програми.

Таблиця 2.2 – Порівняння фреймворків з точки зору ціноутворення

Фреймворк	Витрати на розробку	Ціни на фреймворк	Витрати на розгортання
Xamarin	Medium	Free*	High
Ionic 2	Low	Free	Low
NativeScript	Low	Free*	High
React Native	Medium	Free	Low

Точка зору користувача

Перш ніж представити стисле порівняння точки зору користувача у вигляді таблиці, важливо спочатку встановити оцінювані точки та їх значення.

– Продуктивність показує, наскільки добре фреймворк виконує важкі обчислення або анімаційні операції. Основною метрикою тут буде частота кадрів отриманої програми під час виконання складних операцій.

– Рідність вказує, наскільки добре вихідна програма фреймворка відповідає набору функцій і підходів до проектування, характерних для кожної платформи.

– Розмір вказує відносний розмір програми порівняно з рідною програмою. Найважливішим фактором тут буде те, наскільки малим є необхідний код, специфічний для фреймворку.

Таблиця 2. 3 – Порівняння фреймворків з точки зору користувача

Фреймворк	Продуктивність	Рідність	Розмір
Xamarin	High	High	Big
Ionic 2	Low	Low	Medium
NativeScript	Medium	High	Big
React Native	High	High	Medium

Висновок

Що стосується точки зору розробника, React Native виходить переможцем, головним чином завдяки своїй простоті, добре зроблені покращення, (наприклад, Hot Reload) і, можливо, найбільша спільнота з представлених фреймворків. Важливо зазначити, що хоча функція Test Cloud від Xamarin дійсно допомагає при серйозному тестуванні програми, функція Codepush, яку використовують як Ionic 2, так і React Native, спрощує розгортання оновлень. Обидві ці функції допомагають розробнику вирішувати дуже різні проблеми, тому їх важко порівнювати.

Що стосується цінової точки зору фреймворку, то Ionic 2 виходить вперед з його низькими витратами на розробку (викликані практично на 100% спільною базою коду), безкоштовними цінами на фреймворк і низькими витратами на розгортання (в основному це впливає на Codepush). Наслідком буде як React Native, так і NativeScript, причому перший має менше спільного коду, але простіший у розгортанні, а другий – більшу спільну базу коду, але дорожче розгортає оновлення. Важливо зазначити, що обговорення спільної кодової бази є специфічним на основі від програми до програми. Усі ці фреймворки здатні досягти майже 100% спільної кодової бази в ідеальній ситуації.

Що стосується точки зору користувача на фреймворк, React Native виходить переможцем, з дуже хорошою продуктивністю, чудовою нативністю та відносно невеликим розміром (порівняно з іншими фреймворками). Хоча модель компіляції Xamarin можна вважати кращою, React Native досягає принаймні ідентичної продуктивності за допомогою асинхронної потокової обробки.

Підводячи підсумок, зрозуміло, що React Native виходить як чудовий фреймворк, з чудовим досвідом розробників і користувачів, і займає друге місце з точки зору цін. Хоча Ionic 2 виходить як потенційно дешевший фреймворк для розробки, інші його недоліки ускладнюють його використання у високоякісних програмах.

3 REACT NATIVE – ПОГЛИБЛЕНИЙ ОГЛЯД

Після результатів порівняння, зробленого у фіналі попередньої глави, цей розділ зосереджено на представленні поглибленого погляду на React Native. Спочатку в цій главі представлений огляд JavaScript як мови з оглядом на його історію, поточний стан, переваги та недоліки. Після цього у цій главі розповідається про технологію, яка забезпечує живлення React Native - React. Зрештою, у цій главі обговорюються особливості React Native.

3.1 JavaScript

JavaScript це мова високого рівня, багатопарадигмальна з динамічною системою типів. JavaScript вважається однією з трьох основних технологій веб-розробки, присутній на більш ніж 94% усіх веб-сайтів.[17]. JavaScript заснований на стандарті ECMA Script, його поточна версія ECMA-262[18].

Важливою ознакою мови програмування є те, що вона компільована або інтерпретована. Ця характеристика була цілеспрямовано опущена в попередньому розділі; JavaScript - це мова, яку можна як компілювати, так і інтерпретувати. Механізми JavaScript, такі як V8 або Rhino, використовують концепцію, відому як JIT Compilation. Коли фрагмент коду JavaScript інтерпретується кілька разів, двигун позначає цей код як теплий. Оскільки код стає теплішим (це означає, що код інтерпретується багаторазово), двигун компілює код і використовує цей скомпільований код замість того, щоб повторно інтерпретувати той самий код багаторазово. Як пояснюється в лістингу. Основна мета компіляції JIT - підвищити продуктивність.

```
1 for ( i = 0; i < 1000; i++ ){
2 sum += i;
3 }
4 // A JIT Compiler notices that the addition
     code will be run repeatedly , so instead of
     interpreting , the code gets compiled
     directly into machine code ; removing the
     necessity to interpret the code 1000 times
```

Як багатопарадигмальна мова, JavaScript підтримує кілька стилів програмування і навіть поєднує їх; будь то парадигма, керована подіями, функціональна чи імперативна парадигма. Хоча спочатку він мав на меті бути лише клієнтською мовою для веб-браузерів, зараз JavaScript також використовується на веб-серверах.[19], бази даних[20], а також у деяких невеб-програмах, як-от програмне забезпечення PDF[21].

Ще в 1993 році Університет Іллінойсу випустив перший популярний графічний веб-браузер під назвою NCSA Mosaic. Після цього в Каліфорнії в Mountain View кількома авторами оригінального браузера NCSA Mosaic була заснована компанія під назвою Mosaic Communications. Основною метою компанії було створення браузера, схожого на NCSA Mosaic, але кращого в усіх відношеннях. Внутрішньо браузер називався «Mozilla» і був випущений під назвою Mosaic Netscape у третьому кварталі 1994 року. Після набуття популярності в Інтернеті, компанія вирішила перейменувати браузер на Netscape Navigator, а себе в Netscape Communications, щоб уникнути потенційного позову з боку оригінальних творців.

У наступні роки Netscape почав вірити в те, що Інтернет має стати більш динамічним, використовуючи просту мову, яку веб-дизайнери могли б легко використовувати для зручного збирання та спілкування з компонентами. Зокрема, Netscape надав таке пояснення:

«Ми прагнули створити «мову склеювання» для веб-дизайнерів і програмісти на неповний робочий день, які створювали веб-контент із таких компонентів, як зображення, плагіни та аплети Java. Ми побачили Яву як «мову

компонентів», яку використовують програмісти з вищими цінами, де програмісти склеюють – дизайнери веб-сторінок – збирайте компоненти та автоматизуйте їх взаємодії з використанням[мови сценаріїв].»

Після цього компанія вирішила найняти Брендана Айха, метою якого було вбудовування мови схеми в Netscape Navigator.[27]. Незабаром після його придбання Netscape почав переговори з Sun Microsystems про використання мови Java в Netscape Navigator. Отже, компанія вирішила розробити власну мову, яка мала «виглядати як Java», виключивши такі мови, як Python, Perl або Scheme. Компанії швидко потрібен був робочий прототип, щоб захистити походження цієї нової мови. Айх зміг написати прототип у травні 1995 року всього за 10 днів.

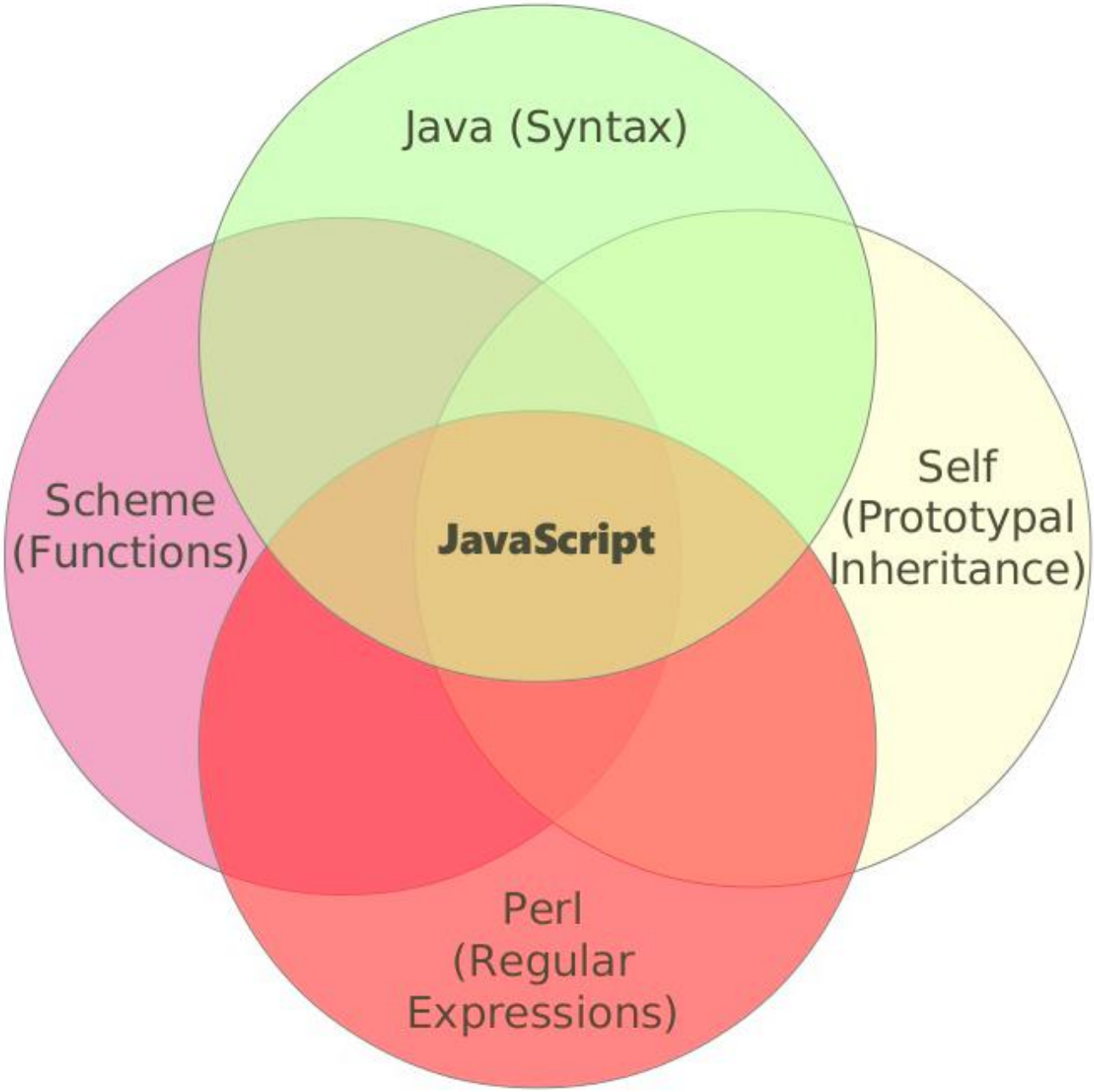


Рисунок 3.1 – Схема, що зображує джерела функцій JavaScript

Хоча спочатку мова була розроблена під назвою Mocha, мова була перейменована в LiveScript, коли вона була випущена з бета-версією Netscape Navigator 2.0 у вересні 1995 року, після чого кілька місяців по тому було перейменовано в JavaScript. Майже рік потому Netscape почав співпрацювати з Ecma International для створення стандартизованої специфікації на основі роботи

Netscape[29]. Ця співпраця привела до випуску специфікації мови ECMAScript, опублікованої в першій версії стандарту ECMA-262 у червні 1997 року.[30]- з яких JavaScript є найпопулярнішою реалізацією.

JavaScript, як і будь-яка інша мова програмування, має власний набір переваг і недоліків, які важливо згадати

Переваги:

– Динамічний тип — JavaScript використовує перевірку типів під час виконання та неявне перетворення типів, де це необхідно. Це можна вважати як перевагою, так і недоліком; перший у невеликих та середніх проектах, другий у більших та корпоративних проектах.

– » Просте налагодження. Налагодження програми JavaScript зазвичай спрощується за допомогою добре обладнаного браузера, такого як Google Chrome, який уже реалізує інструменти для налагодження – Chrome DevTools – із коробки.

– » Незалежно від платформи. Враховуючи, що JavaScript зазвичай обробляється (використовуючи механізм JavaScript, наприклад V8 або SpiderMonkey), отриманий в результаті інтерпретований (або скомпільований) код зазвичай не залежить від платформи.

– » Багатопарадигмальний підхід – JavaScript підтримує використання кількох парадигм, що дозволяє розробнику використовувати будь-який підхід, який він вважає найбільш підходящим у своїй конкретній ситуації. Приклади підходів різних парадигм зображені в лістингах.

```
1 let text = " Hello World !";
2 console .log( text );
3 // > Hello World !
```

```
1 let text = () => console .log(" Hello World !");
2 text ();
3 // > Hello World !
```

```
1 function Text () {
2 console .log(this. text );
3 }
4
5 Text . prototype = {
6 text : " Hello World !"
7 }
8 let foo = new Text ();
9 // > Hello World !
```

Недоліки:

– Динамічний тип — як згадувалося вище, перевірка типів під час виконання та неявне перетворення типів можна вважати перевагами в меншому проекті, але стати перешкодою в проектах більшого розміру.

– Потенційні проблеми з безпекою – сценарії JavaScript виконуються відразу після того, як користувач заходить на веб-сторінку, що відкриває двері для потенційних шкідливих подвигів користувача. Хоча сучасні веб-браузери зазвичай обмежують ці експлойти, все ж існують способи зловживання користувачами.

– Сумісність. Різні механізми браузера можуть виконувати ідентичні фрагменти JavaScript по-різному, що спричиняє невідповідність, яку важко відстежити. На щастя, ця проблема зазвичай вирішується за допомогою транспілера.

Транспілятори – або транскомпілятори – є важливою частиною екосистеми JavaScript. Транспілятор — це інструмент, який зчитує вихідний код, написаний однією мовою програмування, і виводить еквівалентний код іншою мовою..

Використання транспілерів двояке. З одного боку, вони використовуються з такими мовами, як TypeScript або CoffeeScript, щоб допомогти реалізувати різні специфічні концепції, які відсутні в JavaScript. Інше використання полягає у використанні функцій з новіших редакцій специфікацій ECMAScript, які не обов’язково підтримуються всіма браузерами. Кожен браузер використовує інший механізм JavaScript, з різними характеристиками продуктивності та різними

реалізаціями стандарту ECMAScript, що змушує розробника або писати старий код JavaScript, або використовувати транспілятор для вирішення проблеми.

Транспілятори також відіграють важливу роль у прийнятті рішень комітетом TC39[32]- комітет, відповідальний за розробку стандарту ECMAScript. Комітет розглядає реалізацію функції в транспілері, щоб визначити, чи має бути реалізована функція в стандарті ECMAScript.

Мабуть, найпопулярнішим транспілятором JavaScript є Babel, який підтримує транспіляцію як стандартизованої мови ECMAScript, так і кількох найсучасніших версій ECMAScript, позначених як етапи.

3.2 React

React – бібліотека JavaScript з відкритим кодом, яка використовується для створення інтерфейсів користувача. Він був створений і підтримується компанією Facebook, Inc. Основною мотивацією React було створення бібліотеки з великою швидкістю та плавністю користувацьких інтерфейсів, яка миттєво реагує на будь-які зміни, але оновлює лише компоненти, які потребують оновлення. React використовує так звані компоненти як основний будівельний блок.

Кожен компонент є або функцією JavaScript (компонент без стану), або класом JavaScript (компонент із збереженням стану). Компонент без стану повертає HTML-розмітку безпосередньо, тоді як компонент із збереженням стану розширює клас React.Component і реалізує метод render(), який повертає або інший компонент, або HTML-розмітку. React зазвичай вимагає кореневого компонента, який відображається в тегу з ідентифікатором, встановленим на «root».

React використовує односпрямований потік даних. Іншими словами, в vanilla React дані, отримані компонентом, не можуть бути безпосередньо змінені, але

можуть використовуватися як змінні в його методах або атрибутах. З іншого боку, компонент може отримати функцію зворотного виклику, щоб змінити ці дані в компоненті вище, а потім отримати оновлені дані. Це гарантує, що дані рухаються лише вниз, тоді як зворотні виклики функцій рухаються лише вгору по дереву компонентів.

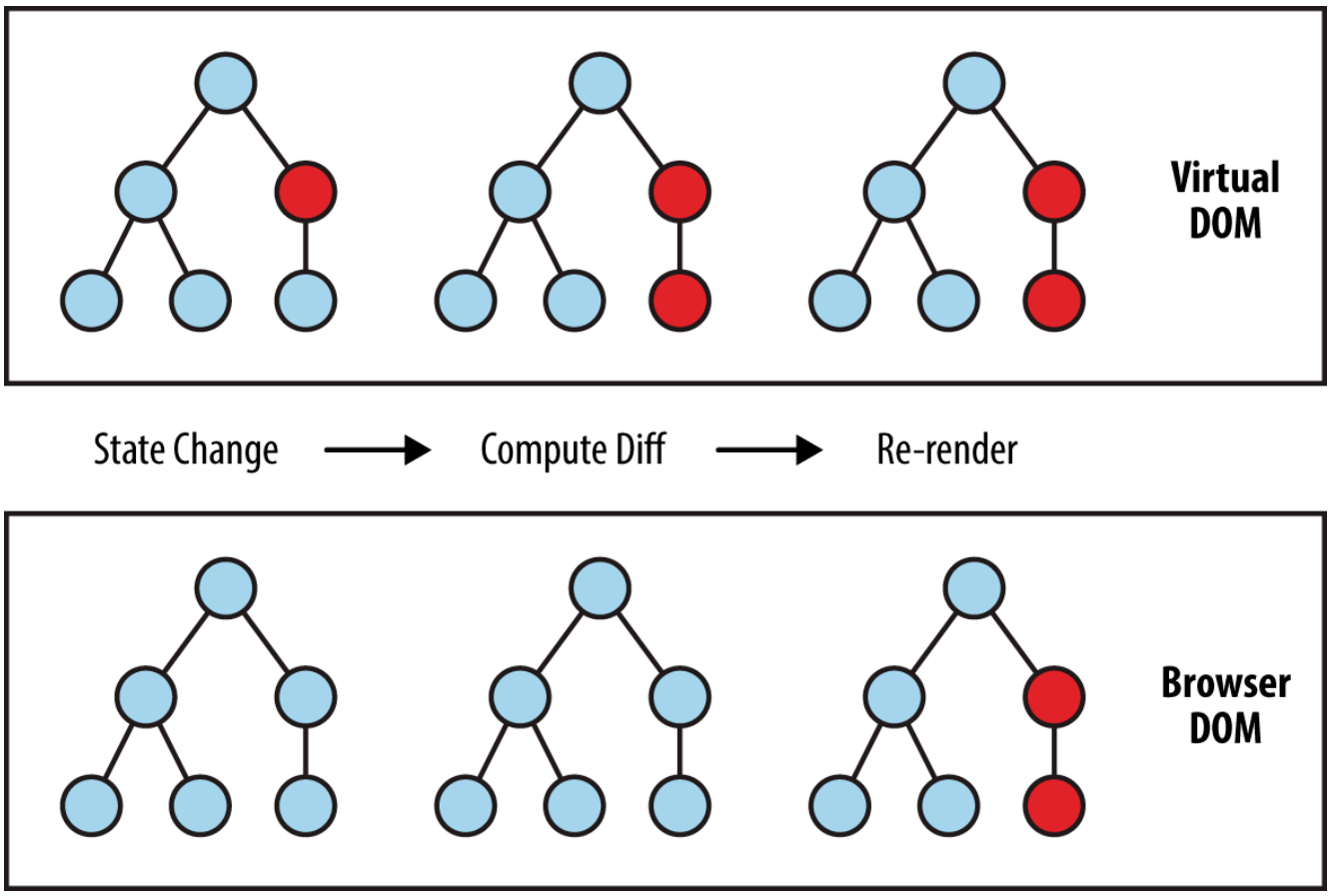


Рисунок 3.2 – Спрощена схема узгодження DOM React

Ще одна важлива функція, яку надає React, - це віртуальна DOM. DOM це інтерфейс програмування, що представляє HTML-документ у вигляді деревоподібної структури вузлів та об'єктів. Це дозволяє розробнику програмно редагувати вузли. Основним недоліком традиційних маніпуляцій з DOM є їх обчислення з великими ресурсами. Натомість React використовує віртуальну DOM; абстрактне представлення DOM, яке синхронізується з справжньою DOM, коли це необхідно. Цей процес синхронізації називається Reconciliation і детально пояснюється в документації React[35]. Щоразу, коли розробник змінює DOM за

допомогою React, замість цього змінюється віртуальна DOM. Потім React запускає узгодження, яке порівнює DOM, знаходить відмінності та перетворює оригінальний DOM найефективнішим способом. Це забезпечує як швидшу реактивність, так і кращий досвід розробника.

Важливо вказати, що React — це не фреймворк, а бібліотека Javascript. Відмінність стосується моделі архітектури програмного забезпечення MVC, де «Модель» підтримує дані та логіку програми, «Перегляд» представляє візуалізовані дані та інтерфейс користувача, а «Контролер» приймає введення та передає його двом іншим частинам. У цій архітектурі React виконує роль компонента «Перегляд» моделі MVC, оскільки, наприклад, робота з базою даних вимагає використання зовсім іншої технології.

JSX

JSX – це спеціальне для React розширення синтаксису, яке покращує стандарт ECMAScript за допомогою тегів, подібних до XML. JSX складається з елементів і компонентів. Елементи є JSX еквівалентами звичайних тегів HTML (наприклад, або <h1>), а компоненти є посиланнями на компоненти React.

Лістинг демонструє декларативний характер React, просту компонування компонентів і масштабованість, зберігаючи високу читабельність, відносну простоту та хорошу можливість повторного використання компонентів і коду.

Приклад використання JSX в React

```
1    // Initial class declaration
2 class Warning extends React . Component {
3    render () {
4         return <div >Be careful , {this. props . name }! </
          div >;
5    }
6 }
7 // Rendering the class at the " root " node
8 ReactDOM . render (< Warning name =" Freddie " />, root
);
9 // Resulting HTML
10 <div >Be careful , Freddie !</div >
```

Наступний лістинг демонструє, як babel транспілює JSX у стандартизований код JavaScript.

```
let nav = (
   <ul id="nav">
        <li className =" item " ><a href ="#">Home </a
             ></li >
   </ul >
);
// Using Babel , the JSX gets transformed into
   regular JavaScript .
var nav = React . createElement ("ul",
   { id: "nav" },
   React . createElement ("li",
        { classname : " item "} ,
        React . createElement ("a",
             { href : "#" },
             " Home "
             )
        )
   )
```

3.3 Під капотом React Native

React Native реалізує концепцію під назвою React Native bridge. По суті, міст є зв'язком між кількома потоками, які React Native використовує в отриманій програмі. Ці потоки, як правило, є потоками JavaScript і рідними потоками. Потужність React Native походить від ефективного поділу програми між потоками JavaScript і рідними потоками.

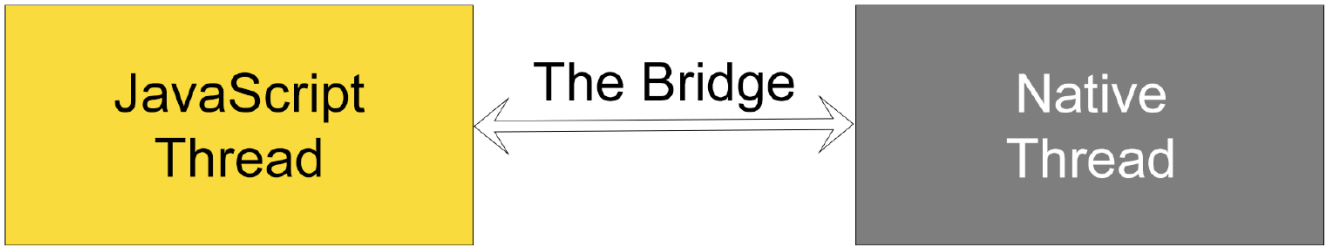


Рисунок 3.3 – Схема потокової системи, яку реалізує React Native.

Потік JavaScript зазвичай містить бізнес-логіку програми. Нативні потоки складаються з основного потоку інтерфейсу користувача, який піклується про рідні компоненти, які використовує React native, і кількох можливих фонових потоків, які за потреби піклуються про фонові завдання. Щоб уточнити, рідні модулі зазвичай вводяться в потік JavaScript як глобальні змінні, що дозволяє їх просте та швидке створення. Через це логічне розділення зависання програми рідко трапляється з React Native, оскільки інтерфейс користувача малюється рідним потоком.

Щоб додатково пояснити ідею мосту React Native, важливо розуміти, що використовується вищезгадана техніка Reconciliation, хоча і з модифікаціями, щоб відповідати мобільному додатку. Виклики, що проходять через міст, зазвичай є асинхронними та пакетними, щоб обмежити виклики мосту до мінімуму. Міст можна вважати основним вузьким місцем програми React Native, і мінімізувати його використання необхідно, щоб мати програму, схожу на native.

Прикладом неправильно використаної функціональності моста може бути реалізація JavaScript модуля Swipable. Припустимо компонент JavaScript, який дозволяє проводити пальцем убік своїх дочірніх елементів, змінюючи при цьому його непрозорість і зміщення залежно від довжини пальця. Реалізація в JavaScript призведе до наступного:

– Власна подія дотику генерується в власному потоці.

– Власна подія дотику передається через міст до потоку JavaScript.

– Потік JavaScript отримує подію і використовує наявну інформацію для обчислення правильного зміщення та непрозорості компонента, оновлюючи його стан.

– Відтепер ці обчислення відправляються назад через міст до рідного потоку.

– Нативний потік обробляє дані, отримані через міст, і виконує необхідні зміни в інтерфейсі користувача.

Цей підхід на перший погляд здається нешкідливим, але може легко закрити міст непотрібними обчисленнями, які замість цього можна було б обробити в рідному потоці. Зрештою, анімація, ймовірно, не буде плавною при частоті оновлення 60 кадрів в секунду.

Щоб вирішити цю проблему, розробнику слід створити власний рідний модуль, який приймає функцію стилізації для подій і дочірніх елементів, які будуть намальовані як властивості. Таким чином, немає зайвої комунікації, яка закриває міст, оскільки рідний потік вже знає, як зміщувати й стилізувати дочірні компоненти. Подібний підхід дозволяє створити плавну анімацію та адаптивну програму.

Підсумовуючи, головна сила React Native полягає у можливості використовувати нативні модулі взаємозамінно зі своїми власними, де це необхідно, для підтримки якісної програми.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ

У цьому розділі відображено пояснення основного функціоналу та рішень, які реалізовані в програмі. Після цього формулюється аналіз запропонованого рішення з проектуванням конструкції. Нижче наводиться реалізація програми, яка закінчується підведенням результатів і визначенням наступних кроків розвитку програми.

4.1 Визначення функціоналу програмної системи

Місто Брно планує запустити версію системи паркування для мешканців протягом кількох наступних років. З досить складними моделями ціноутворення на основі зони проживання, зони роботи та зон відвідування, додаток для максимального спрощення системи паркування допоможе жителям Брно якомога плавніше перейти на систему паркування для мешканців.

Житлова система паркування планує розділити громадян на три категорії:

– Резидент – особа, місце постійного проживання якої знаходиться в одній із зазначених зон. У всіх інших зонах людина вважається відвідувачем.

– Абонент – фізична або юридична особа, місце роботи якої знаходиться в одній із зазначених зон. У всіх інших зонах відвідувачем вважається особа або юридична особа.

– Відвідувач – Особа, яка не має права ні на житлове, ні на позачергове паркування.

Оскільки житлові та позачергові системи паркування не повністю розроблені, рішення має бути в основному орієнтоване на людей, яких вважають відвідувачами. Оскільки модель ціноутворення для відвідувачів і зони паркування вже були запропоновані містом Брно, рішення мало напівтверду основу.

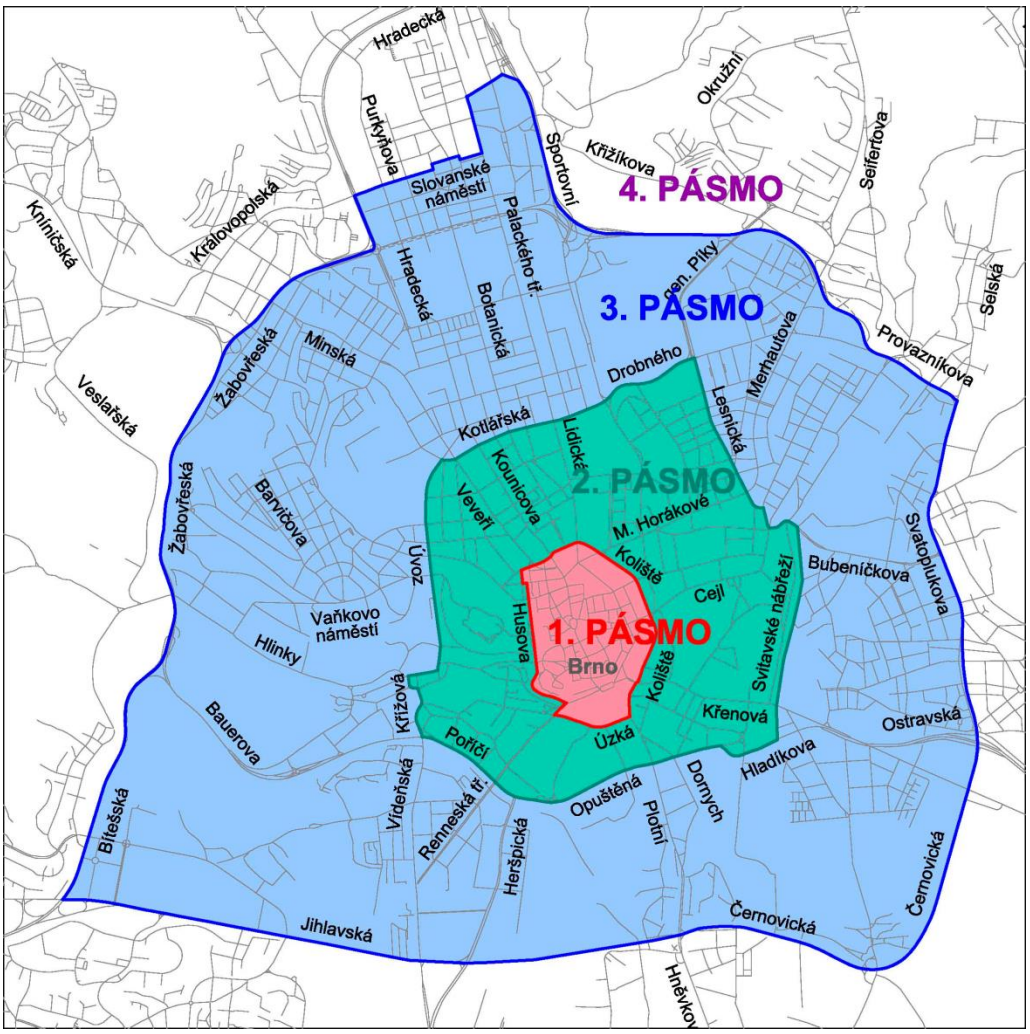


Рисунок 4.1 – Схема пропонованих регіонів ціноутворення для паркування для відвідувачів у Брно

Основною метою рішення має бути спрощення процесу купівлі квитків з деякими доповненнями до якості життя, як-от квитки, які розраховуються по хвилинах, спосіб перегляду минулих квитків і транзакцій, а також спосіб придбання квитків для кількох окремих автомобілів. Рішення повинно мати можливість визначати позицію користувача та залежно від регіону, в якому він перебуває, використовувати іншу модель ціноутворення для квитка.

4.2 Аналіз і проектування структури програмної системи

React Native Раніше вважалося, що він є найкращою мультиплатформенною платформою для використання на даний момент, тому, природно, додаток буде написане на ньому. На основі описаного вище визначення проблеми, проста діаграма варіантів використання, показана на малюнку5.2був створений для допомоги в розробці структури програми.

Враховуючи вимоги рішення, я оцінив варіанти і визначив, що використання служби завантаження React Native здається оптимальним підходом. Expo буде використовуватися для спрощення робочого процесу та піклування про компоненти, для яких натомість буде потрібно рідний код.

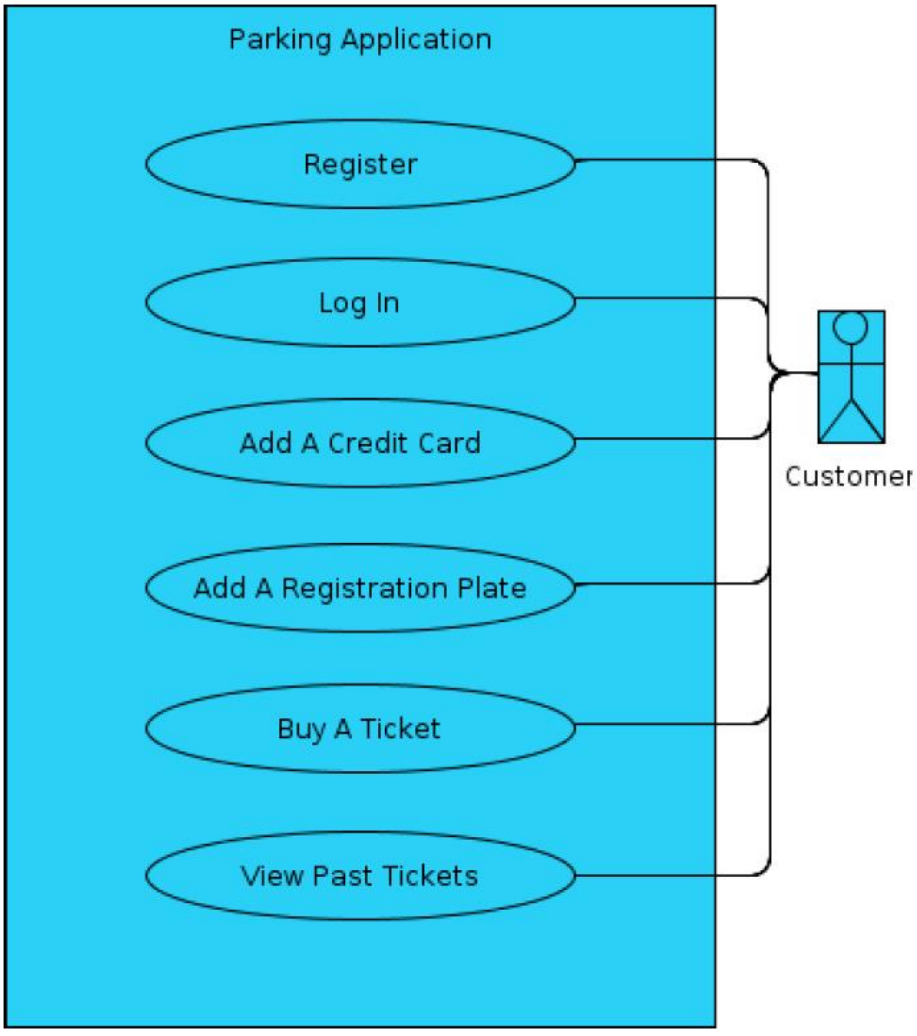


Рисунок 4.2 – Діаграма варіантів використання програми

Expo – це безкоштовний набір інструментів із відкритим вихідним кодом, створений на основі React Native, щоб допомогти створювати власні програми без будь-яких знань. Експо інтенсивно оновлюється та підтримується; щомісяця випускається нова версія SDK, з більшою кількістю готових функцій і меншою потребою вдаватися до рідного коду. Спрощуючи робочий процес розробника, надаючи безліч вбудованих функцій, Expo також допомагає розробнику в розгортанні та створенні додатків. Він надає послугу, подібну до згаданої раніше CodePush, і дозволяє розробнику створювати додаток для iOS, не використовуючи Xcode, або взагалі мати Mac. Спільно з Expo, CRNA буде використовуватися для запуску проекту. CRNA вже включає в себе готову платформу Expo.

Іншою проблемою, яку потрібно було вирішити, було те, як буде вирішено аутентифікацію та зберігання даних. Я вирішив використовувати Firebase – платформу для розробки мобільних і веб-додатків від Google LLC. Платформа, серед іншого, забезпечує підтримку кількох внутрішніх служб, таких як аутентифікація користувача, два параметри бази даних або параметр зберігання.

Основна мета Firebase — надати базові функціональні можливості сервера без створення спеціального серверного сервісу. Firebase надає власну бібліотеку JavaScript для зв'язку з серверними службами, хоча можливість спілкування через REST API доступна за допомогою допоміжної бібліотеки, наприклад node-firebase-rest[39]. Варто зазначити, що API RESTful, наданий вищезгаданою бібліотекою, не надає можливості для практичного, постійного підключення. Хоча це все ще технічно досяжно, замість нього буде використовуватися офіційна бібліотека Firebase, головним чином через те, що вона добре перевірена та оптимізована для подібних випадків використання.

На основі цих попередніх рішень і підтверджень складається діаграма розгортання. Рисунок 4.3 показує основні зв'язки та методи комунікації між використовуваними технологіями. Він також відображає модулі, які використовуються сервісом Firebase.

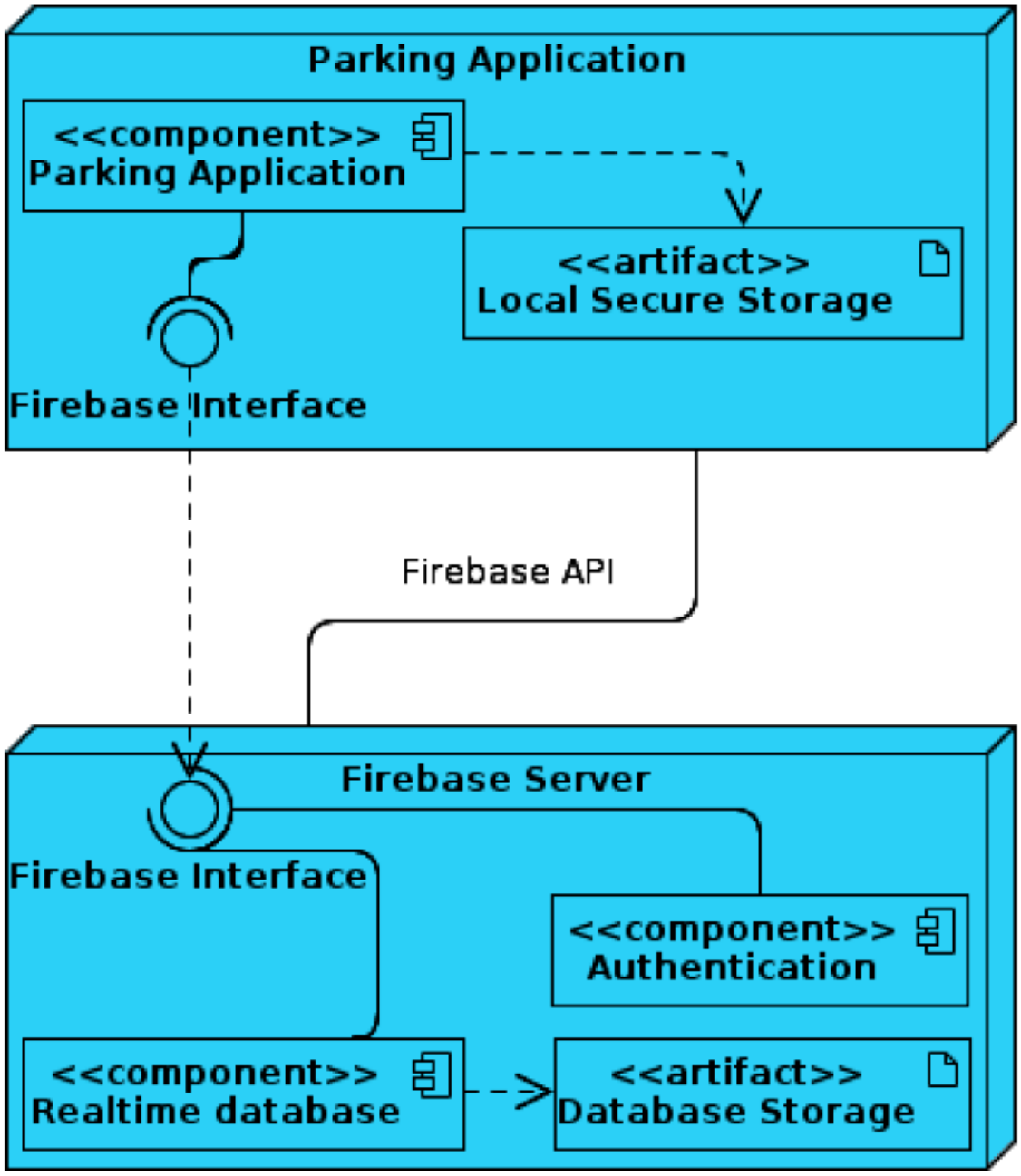


Рисунок 4.3 – Діаграма розгортання розробленої програми

4.3 Реалізація

Враховуючи, що це був мій перший досвід використання React Native, я зіткнувся з кількома проблемами як з реалізацією специфіки рішення, так і з технологіями, згаданими вище. Спочатку утиліта CRNA у поєднанні з Expo допомогла мені запустити проект за лічені хвилини.

Перше питання, яке потрібно було вирішити, – це визначити, як буде оброблятися маршрутизація. Офіційна документація React Native представила три бібліотеки для маршрутизації: native-navigation react-native-navigation і React navigation. Я вирішив подивитися офіційну документацію Expo, де остання бібліотека згадана як рекомендована. Дотримуючись цієї рекомендації, я вибрав бібліотеку React Navigation, щоб подбати про навігацію додатком. Вищезгадана бібліотека містить кілька навігаторів – компонентів React Native, які визначають структуру навігації програми. Структуру навігації отриманої програми можна побачити на рисунку 4.4. Прямокутні вузли представляють різні уявлення, які містить додаток, де жовтий вузол завжди представляє екран за замовчуванням, до якого веде навігатор, а сині вузли представляють інші параметри, до яких навігатор дозволяє користувачеві переходити.

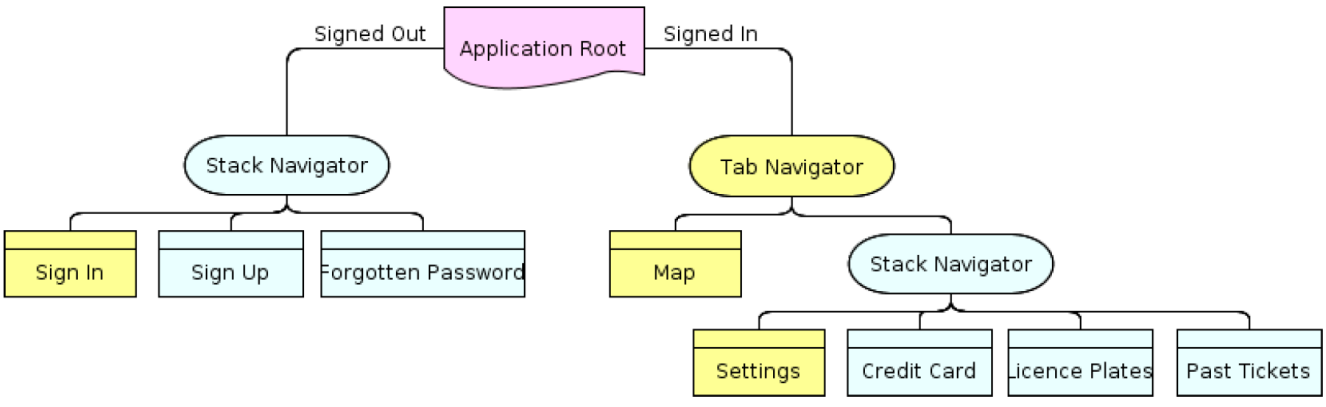


Рисунок 4.4 – Діаграма навігації

Після цього мені довелося визначити, яка карта буде використана для отриманої програми. Оскільки Expo вже реалізує react-native-maps звичайне рішення було легко прийняти. Після цього потрібно було прийняти рішення про підхід до того, як накладення регіону буде відображатися на карті. Ще важливіше те, як буде зберігатися та оброблятися накладення регіону. Маючи для початку лише зображення накладання карти, я спочатку перевірив загальнодоступні набори даних міста Брно .На жаль, жоден із наборів даних не виявився корисним, тому я вирішив використовувати утиліту Snap to Roads API Google Maps Roads.

На жаль, ця послуга в основному спрямована на відстеження маршруту транспортного засобу, тому в отриманий JSO довелося внести кілька ручних налаштувань об'єкта, щоб дотримуватись початкових меж регіону.

Треба було прийняти ще одне рішення щодо параметрів бази даних платформи Firebase. Наразі Firebase надає два різні варіанти: базу даних реального часу та бета-версію Cloud Firestore. Після вивчення обох варіантів Cloud Firestore здався кращим варіантом для мого випадку використання. Тому я вирішив використовувати Firestore як свою базу даних, але досить швидко зіткнувся з проблемами. На жаль, реалізація JavaScript Firestore погано працює з React Native. Оскільки конкретна помилка трапляється лише на Android, Firestore не може зв’язатися зі службою Firebase. Хоча офіційна команда Firebase знає про цю проблему і активно працює над її вирішенням, на сьогоднішній день помилка все ще виникає. Виправлено помилку, показано в списку5.1,мені здалося непрактичним обхідним шляхом. Через це (та інші проблеми з налаштуванням Firestore у моїй програмі React Native) я вирішив використовувати замість цього базу даних реального часу. Ця версія модуля бази даних Firebase, хоча і має менше функцій, уже добре перевірена на більшості платформ, у тому числі в React Native.

Рішення помилки Firestore в React Native

```
1 const originalSend = XMLHttpRequest . prototype .
    send ;
2 XMLHttpRequest . prototype . send = function ( body ) {
```

```
3     if ( body === '') {
4          originalSend . call (this);
5     } else {
6          originalSend . call (this , body );
7     }
8 };
```

Що стосується Firebase, я також скористався його модулем аутентифікації. Вивчивши можливості інструментарію Expo, я реалізував вхід за допомогою типової комбінації електронної пошти та пароля, соціального входу у Facebook та соціального входу Google. Однак для використання цих служб соціального входу у розгорнутому додатку необхідно виконати кілька вимог, наприклад, наявність дійсної політики конфіденційності, яка має бути підтверджена Facebook. Зрозуміло, я вирішив пропустити це, залишивши отриману програму в режимі «розробки», тобто лише авторизовані облікові записи Facebook можуть використовувати логін із соціальної мережі.

Структура впровадження

Реалізація програми логічно розділена на кілька папок. Всю структуру папок можна побачити на рисунку 4.5. Хоча я частково зрозумів сам за себе, я спробую пояснити використання деяких неінтуїтивно зрозумілих папок.

Папка .expo містить необхідні конфігураційні файли для забезпечення функціональних можливостей Expo. Папка node_modules містить встановлені залежності, і про неї мова піде пізніше. Папка Auth містить усі функції автентифікації користувачів, включаючи вхід, вихід із системи, скидання пароля та вхід. Ця логіка реалізована за допомогою функцій, наданих Expo, що істотно спрощує процес. Папки карт і налаштувань містять відповідні реалізації екрана, а також логіку, необхідну для їх функціонування. Папка входу містить екрани, доступні користувачеві під час виходу, тоді як більшість логіки міститься в вищезгаданій папці auth. Маршрутизатор містить логіку маршрутизації, необхідну для того, щоб зробити обхід додатку можливим.

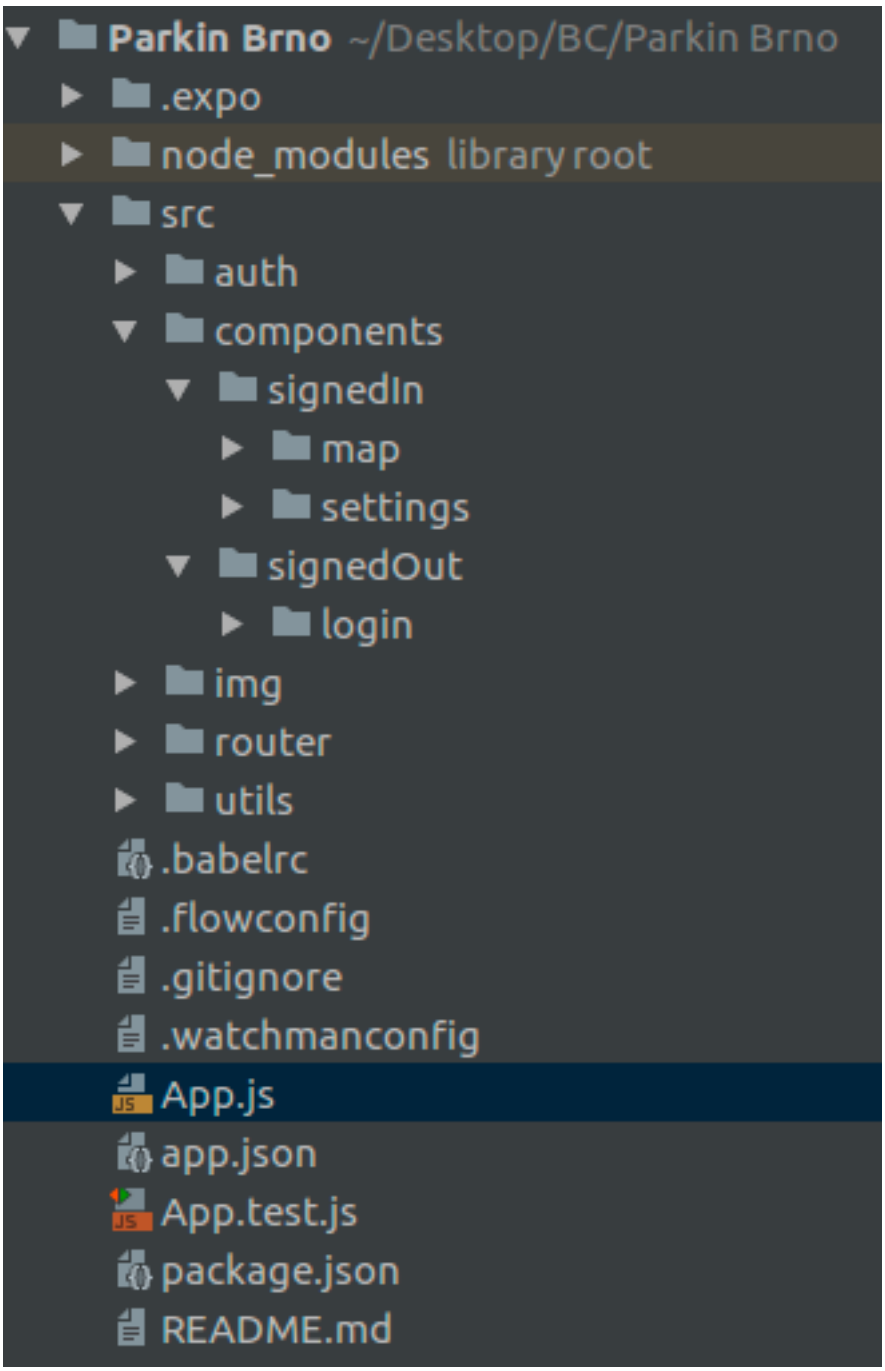


Рисунок 4.5 – Структура проекту програми

App.js є точкою входу програми; Це корінь системи маршрутизації та керує обліковими даними користувача. Package.json містить декілька відомостей про програму та список залежностей, які необхідні для правильної функціональності програми. Ці залежності зазвичай встановлюються за допомогою npm або yarn за допомогою простої команди, наприклад npm or yarn install. Вони встановлюються у вищезгадану папку node_modules.

Як правило, після першого запуску програми користувачеві з'являється екран «Увійти». Екран «Увійти» містить увійти, щоб увійти за допомогою методів, згаданих вище, і дві кнопки переспрямування; один, що веде до екрана «Забули пароль», і другий, який веде до екрана «реєстрація». Обидва ці екрани мають схожий стиль дизайну до екрана «Вхід», і в іншому випадку вони зрозумілі.

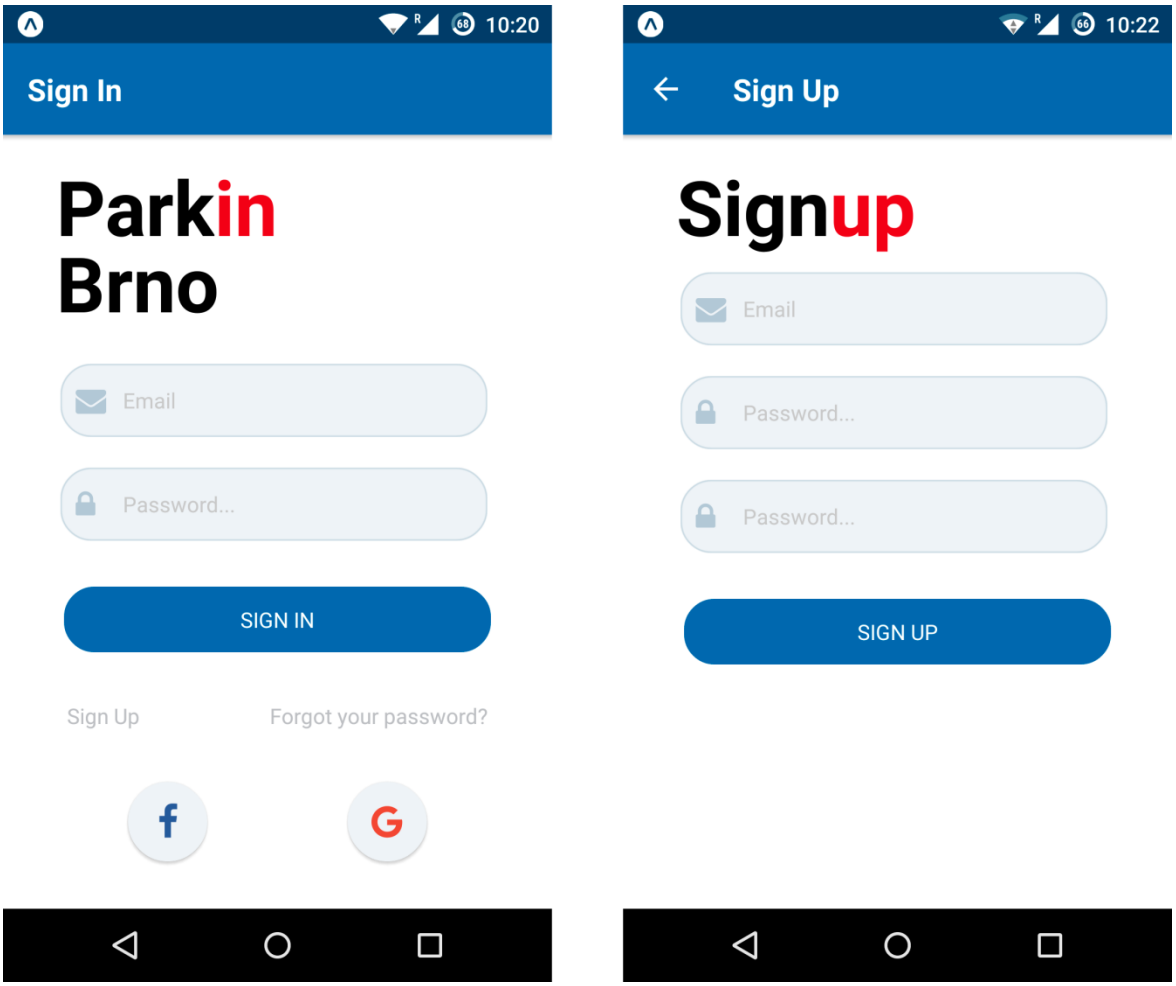


Рисунок 4.6 – Екрани входу та реєстрації

Якщо користувачу вдалося ввійти, з'явиться екран «Карта» з повзунком меню внизу. Це меню дозволяє користувачеві переходити між двома основними екранами програми, екранами «Карта» та «Налаштування». Екран «карта» спочатку пропонує користувачеві активувати служби визначення місцезнаходження (якщо вони вимкнені), а потім відображає поточне положення

користувача, а також відображає накладення регіону на карту Брно. Користувачеві надається кнопка в нижній частині карти, яка дозволяє йому почати свій паркувальний талон. Повторне натискання цієї кнопки відкриває для користувача модальне вікно, яке інформує його про вартість квитка та відображає кнопку для продовження платежу та для скасування платежу (тому квитка продовжує працювати).

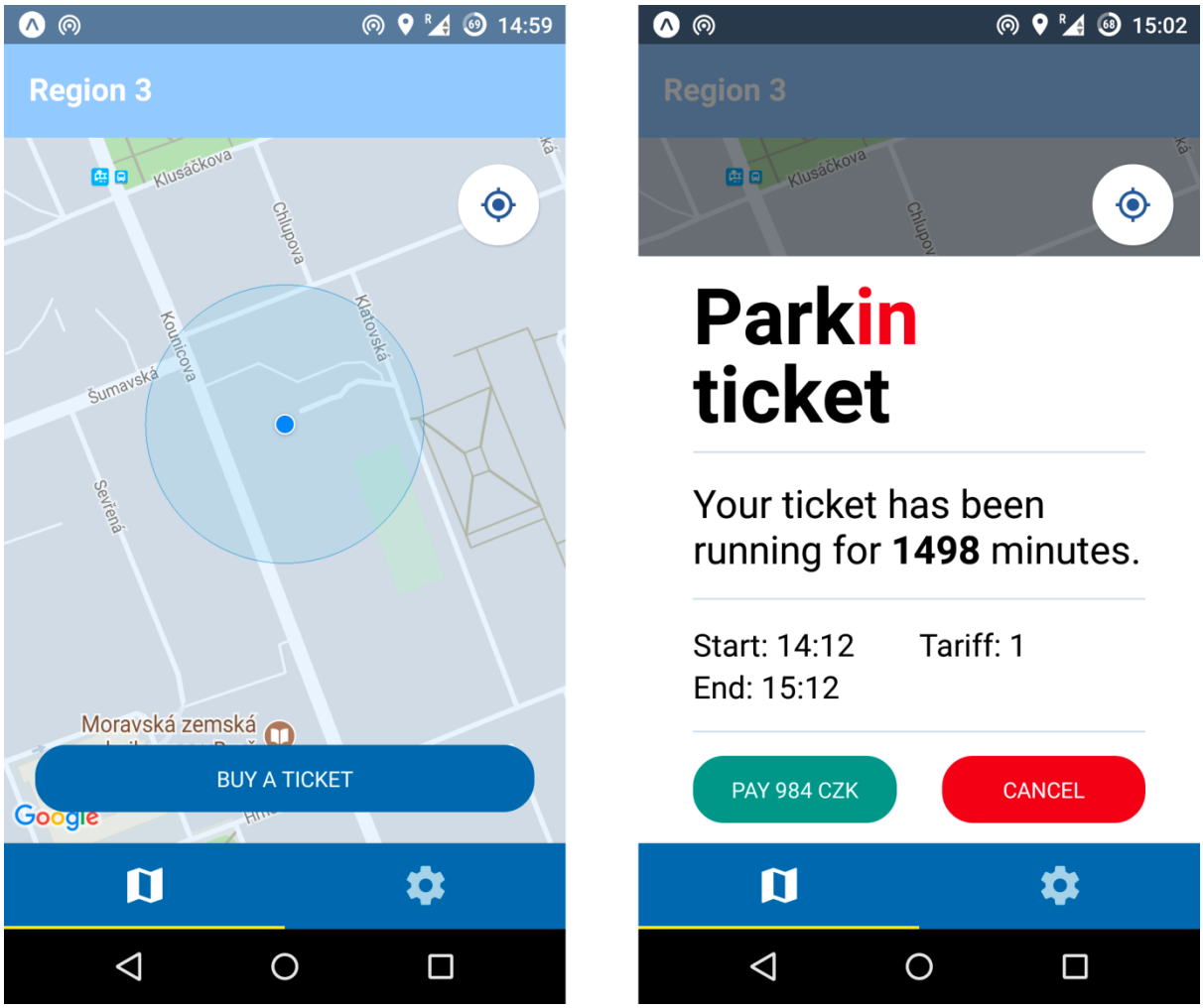


Рисунок 4.7 – Екран карти та підтвердження покупки квитка

Після переходу на екран «Налаштування» користувачеві відкривається меню, яке дозволяє йому переміщатися в різні розділи налаштувань. Меню розділене на дві частини; розділ «Паркування» та розділ «Користувач».

Розділ «Парковка» складається з трьох окремих екранів: екрана «Рахунок», екрана «Реєстраційні номери» та екрану «Квитки». Перший екран

використовується для зберігання інформації про кредитну картку користувача для обробки квитка. Другий екран дозволяє користувачеві переглядати, додавати, редагувати та видаляти свої номерні знаки. Вони прив'язані безпосередньо до квитка. Третій екран дозволяє користувачеві переглянути свої минулі квитки з такою інформацією, як загальна ціна або початок і кінець сеансу паркування.

Розділ «Користувач» містить екран «Профіль» і кнопку «Вийти». Екран «Профіль» дозволяє користувачеві змінити свій пароль, тоді як кнопка «Вийти» забезпечує просту функцію виходу користувача.

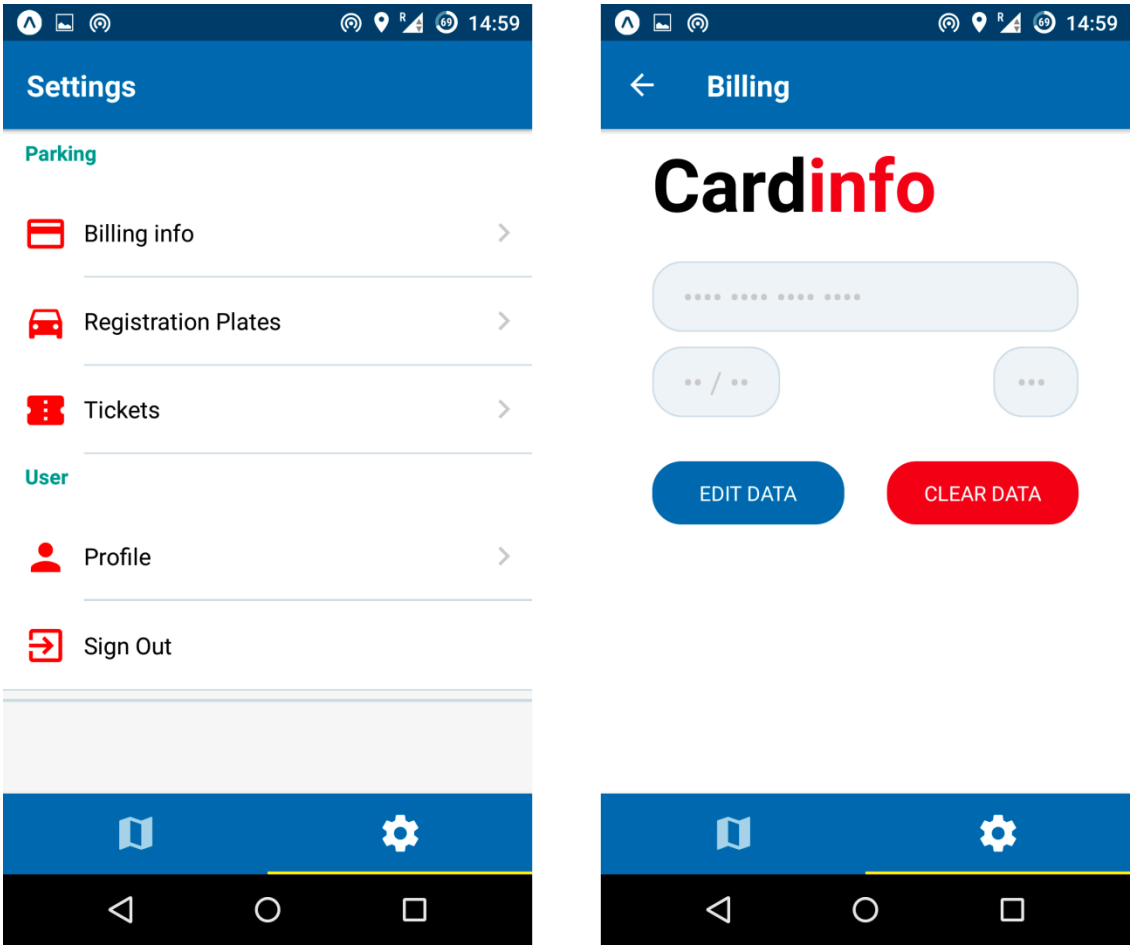


Рисунок 5.8 – Екран налаштувань та екран рахунку

Незважаючи на те, що під час впровадження програми виникли деякі складності, вони врешті-решт були задовільно вирішені. Додаток надає прототип того, як можна спростити систему паркування будинків Брно для відвідувачів і

людей без постійного проживання. Це є доказом концепції рішення для розумного міста, яке в кінцевому підсумку може бути інтегровано в Брно.

Я вважаю, що розробка програми була дуже ефективною, що дозволило мені розробити робочий прототип за кілька днів. Лише з незначними доповненнями цей прототип можна було дуже швидко адаптувати до платформи iOS.

Додаток потенційно може бути використаний для реальної покупки квитків, з деякими змінами. Насамперед, активне спілкування з містом Брно необхідне для правильного аналізу проблеми та важливих рішень, які необхідно прийняти щодо всіх частин процесу покупки та перевірки квитків.

Іншою необхідністю є реалізація належного серверного сервісу. Хоча Firebase дуже зручно використовувати для невеликих проектів, для подібного масштабу проекту виділений серверний сервер може бути кращим і безпечнішим вибором.

ВИСНОВКИ

Перший розділ дипломної роботи зосереджений на дослідженні основних вимог до розробки мобільних додатків, представленні різних можливих підходів до мобільної розробки та порівнянні їх переваг та недоліків.

Згодом робота була спрямована на обґрунтування нативного мультиплатформного підходу. Були представлені найвідоміші мультиплатформні фреймворки з поясненням технологій, які вони використовують, і як вони функціонують. Було проведено порівняння між вищезгаданими структурами, використовуючи кілька точок зору, щоб встановити об'єктивну точку зору на якості кожної рамки. Розділ було завершено остаточним рішенням щодо того, яка є поточною провідною структурою.

Наступний розділ має на меті детально ознайомитися з технологіями, що стоять за React Native – переможцем порівняння фреймворків у попередньому розділі. JavaScript і React представлені як основні будівельні блоки фреймворку React Native, а також зроблено короткий опис технологій, які використовуються движком React Native.

Нарешті, на основі визначення проблеми, сформульовано демонстраційне завдання. Зроблено аналіз та початковий дизайн структури, після чого перераховується кілька проблем, які потребували вирішення під час впровадження, і підсумовуються можливі сценарії користувача кінцевої програми. Розділ завершується короткою оцінкою процесу розробки та деякими пропозиціями щодо можливих застосувань прототипу демонстративної програми.

Загалом, я вважаю, що React Native зарекомендував себе як досить зріла нативна мультиплатформенна структура розробки з добре налагодженою екосистемою та відносно простим життєвим циклом розробки додатків. React Native елегантно заповнює порожній простір між нативним і гібридним підходами, використовуючи найкраще з обох світів, намагаючись звести недоліки

до мінімуму. Моя розробка в React Native була загалом дуже приємним досвідом, хоча, як і з будь-якою новою технологією, я мав деякі незначні проблеми, які потребували оперативного вирішення проблем. Мені вдалося створити життєздатний прототип програми, подібний до нативної системи за лічені дні. Додаток можливо демонструє а

Висновок, можливий напрямок, в якому могла б рухатися система паркування Брно. Я б зробив висновок, що React Native – це чудовий варіант розробки для невеликих і середніх проектів, а також для невеликих компаній без великих, відданих команд розробників нативної мови для кожної платформи. React Native не має на меті зробити нативну розробку застарілою; натомість він використовує сильні сторони React, щоб забезпечити високу можливість повторного використання та компонування рідних компонентів, при цьому дозволяючи розробнику розробляти рідний модуль, коли це необхідно. У цьому конкретному підрозділі мобільної розробки React Native досягає успіху.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. POCKETGAMER.BIZ. Середні ціни на програми в Apple App Store станом на липень 2021 року (у доларах США) [онлайн]. режим доступу: https://www.statista.com/statistics/267346/ середня-apple-app-store-price-app/.

2. AYET-STUDIOS. Free and paid app distribution for iOS in comparison to Android © ayeT-Studios [онлайн]. режим доступу: https://www.ayetstudios.com/blog/app-promotion/mobile-app-promotion/promote-ios-app.

3. GLOBETROTTER. Hybrid Applications And Android Native Browser [онлайн]. режим доступу: https://myshadesofgray.wordpress.com/2021/04/15/hybrid - applications-and-android-native-browser/.

4. Christensson P. Multiplatform Definition [онлайн]. режим доступу: https://techterms.com/definition/multiplatform.

5. POTOTSKA, I. Hidden Advantages of Cross-Platform Development with React Native [онлайн]. режим доступу: https://yalantis.com/blog/native-vs-cross-platform-app-development-shouldnt-work-cross-platform/.

6. XAMARIN. The Xamarin platform [онлайн]. режим доступу: https://www.xamarin.com/platform.

7. FRIEDMAN, Nat. Xamarin for all [онлайн]. режим доступу: https://blog.xamarin.com/xamarin-for-all/.

8. XAMARIN. Mobile app testing made easy [онлайн]. режим доступу: https://www.xamarin.com/test-cloud.

9. XAMARIN. Xamarin FAQ [онлайн]. режим доступу: https://www.xamarin.com/faq#q3.

10. TEAM, Ionic. Ionic GitHub page [онлайн]. режим доступу: https://github.com/ionic-team/ionic.

11. TEAM, Ionic. Ionic documentation [онлайн]. режим доступу: http://ionicframework.com/docs/.

12. Rao Leena. Adobe Acquires Developer Of HTML5 Mobile App Framework PhoneGap Nitobi [онлайн]. режим доступу: https://techcrunch.com/2018/10/03/adobe-acquires-developer-of-html5-mobile-app-framework-phonegap-nitobi/.

13. TELERIK, Progress. About NativeScript [онлайн]. режим доступу: https://www.nativescript.org/about.

14. TELERIK, Progress. NativeScript Under the Hood [онлайн]. режим доступу: https://developer.telerik.com/products/nativescript-inside-the-black-box/.

15. TELERIK, Progress. How NativeScript Works [онлайн]. режим доступу: https://developer.telerik.com/featured/nativescript-works/.

16. FACEBOOK, Inc. Draft: JSX Specification [онлайн]. режим доступу: https://facebook.github.io/jsx/.

17. W3TECHS. Usage Statistics for Websites, December 2020 [онлайн]. режим доступу: https://w3techs.com/technologies/details/cp-javascript/all/all.

18. ECMA. Standard ECMA-262 [онлайн]. режим доступу: https://www.ecma-international.org/publications/standards/Ecma-262.htm.

19. FOUNDATION, Node.js. Node. [онлайн]. режим доступу: https://nodejs.org/en/.

20. FOUNDATION, The Apache Software. Apache CouchDB [онлайн]. режим доступу: https://couchdb.apache.org/.

21. INCORPORATED, Adobe Systems. Javascript for Acrobat [онлайн]. режим доступу: http://www.adobe.com/devnet/acrobat/javascript.html.

ДОДАТОК А

(обов’язковий)

ДІАГРАМИ

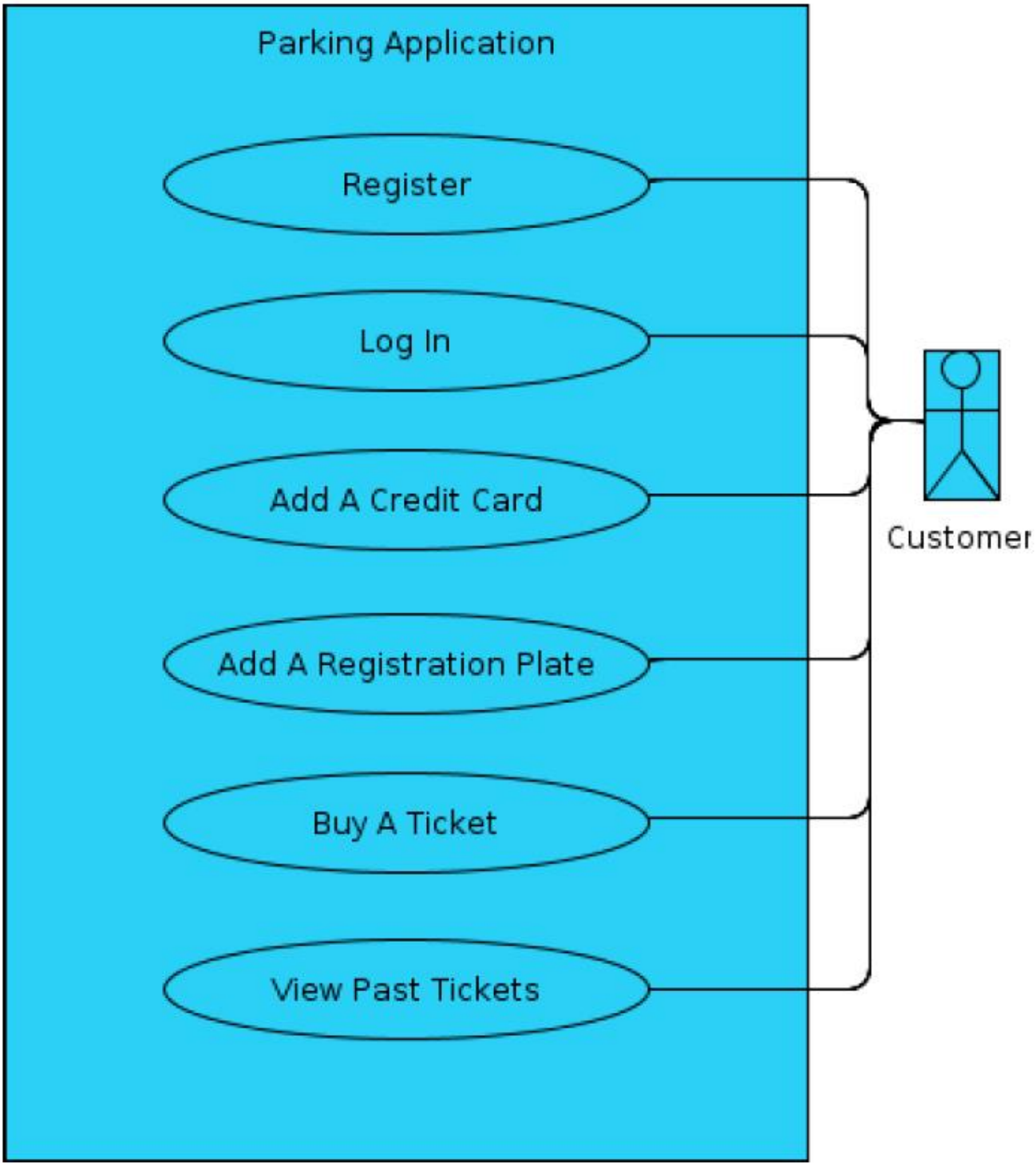


Рисунок А.1 – діаграма варіантів використання програми

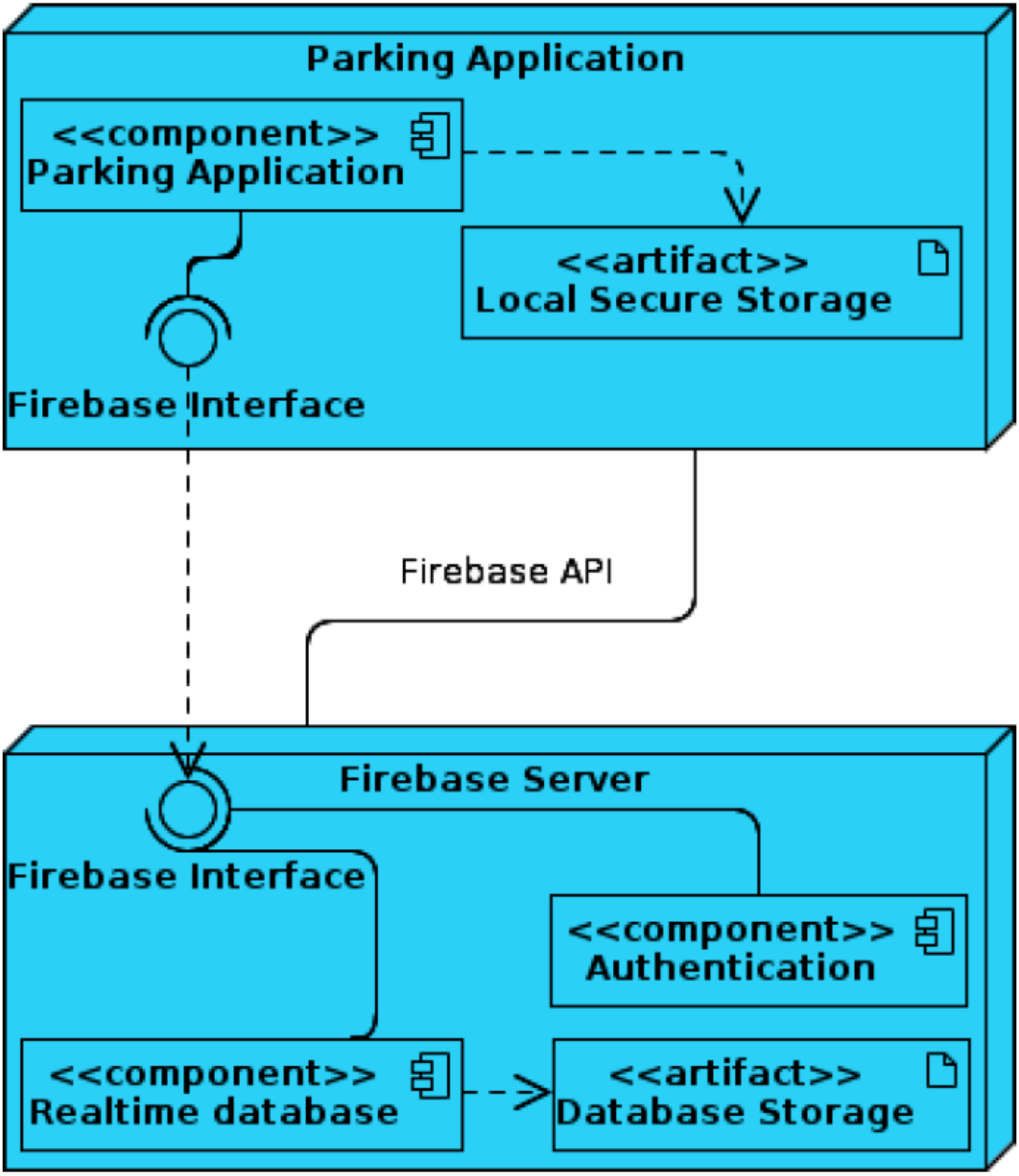


Рисунок А.2 – Діаграма розгортання розробленої програми

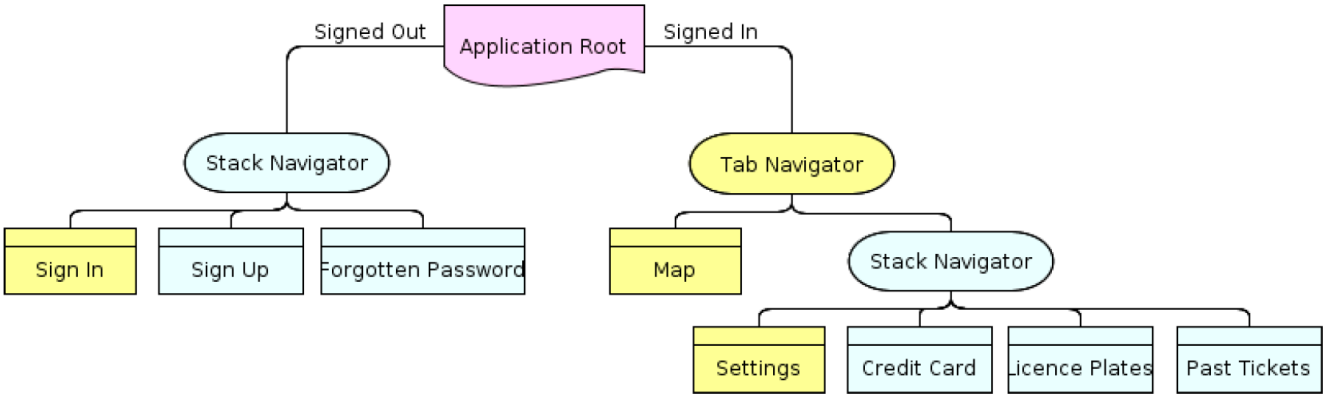


Рисунок А.3 – Діаграма навігації

ДОДАТОК Б

(обов’язковий)

КОД (ЛІСТИНГ) ПРОГРАМИ

```
import React, { memo, useMemo, useState, useRef, useCallback } from 'react';
import { View, TextInput, TouchableOpacity, Keyboard } from 'react-native';
import Text from '../../commons/Text';
import SvgParkingSpot from '../../../assets/images/SmartParking/parkingSpot.svg';
import { t } from 'i18n-js';
import { useBoolean } from '../../hooks/Common';
import Routes from '../../utils/Route';
import { Colors, API } from '../../configs';
import { axiosGet } from '../../utils/Apis/axios';
import ParkingSpotInput from './components/ParkingSpotInput';
import { IconOutline } from '@ant-design/icons-react-native';
import { CircleButton } from '../../commons';
import { useNavigation } from '@react-navigation/native';
import { ButtonPopup } from '../../commons';
import WrapHeaderScrollable from '../../commons/Sharing/WrapHeaderScrollable';
import Modal from 'react-native-modal';
import { SPOT_STATUS_CHECK_CAR, TESTID } from '../../configs/Constants';
import { styles } from './styles';

const ParkingInputManually = memo(() => {
const [parkingSpot, setParkingSpot] = useState('');
const [parkingInfo, setParkingInfo] = useState({});
const [isTextFocus, setTextFocus] = useState(false);
const inputRef = useRef(null);
const enterTimeout = useRef(null);
const { navigate } = useNavigation();
const [isNotify, setShowModal, setHideModal] = useBoolean(false);
const [resultCheckCar, setResultCheckCar] = useState('');
const [showInfo, setShowInfo, setHideInfo] = useBoolean(false);
```

```
const onTextInputFocus = useCallback(() => {
inputRef.current.focus(1);
}, [inputRef]);

const onChangeText = useCallback(
(text) => {
const _parkingSpot = text.substring(0, 3);
setParkingSpot(_parkingSpot);
if (text.length === 3) {
onSpotNumberEntered(_parkingSpot);
} else {
setParkingInfo({ ...parkingInfo, address: '' });
}
},
[onSpotNumberEntered, parkingInfo]
);
const onSpotNumberEntered = useCallback((_parkingSpot) => {
inputRef.current.blur();
clearTimeout(enterTimeout.current);
enterTimeout.current = setTimeout(async () => {
const { success, data } = await axiosGet(API.PARKING.PARKING_INFO(), {
params: {
spot_name: _parkingSpot,
},
});
success && setParkingInfo(data);
}, 100);
}, []);

const onPressConfirmSpot = useCallback(async () => {
```

```
Keyboard.dismiss();
if (parkingInfo.booking_id) {
navigate(Routes.SmartParkingBookingDetails, {
id: parkingInfo.booking_id,
});
return;
}
const { success, data } = await axiosGet(API.PARKING.CHECK_CAR_PARKED(), {
params: {
spot_name: parkingSpot,
},
});
if (success && data && data.can_park) {
navigate(Routes.ParkingAreaDetail, {
id: parkingInfo.parking_id,
spot_status_check_car_parked:
data.status === SPOT_STATUS_CHECK_CAR.THERE_IS_CAR_PARKED,
booking_id: parkingInfo.booking_id,
spot_id: parkingInfo.id,
spot_name: parkingSpot,
unLock: true,
});
return;
}
if (data && data.status) {
switch (data.status) {
case SPOT_STATUS_CHECK_CAR.MOVE_CAR_TO_SPOT:
setResultCheckCar(t('notify_no_car_parked'));
break;
case SPOT_STATUS_CHECK_CAR.THIS_SPOT_HAVE_BOOKED:
```

```
setResultCheckCar(t('notify_spot_has_been_booked'));
break;
default:
break;
}
setShowModal();
return;
}
setResultCheckCar(t('notify_spot_not_exist'));
setShowModal();
}, [navigate, parkingInfo, parkingSpot, setShowModal]);

const onFocus = useCallback(() => {
setTextFocus(true);
}, []);
const onBlur = useCallback(() => {
setTextFocus(false);
}, []);

const onPressScan = useCallback(() => {
navigate(Routes.SmartParkingScanQR);
}, [navigate]);

const [elementHeight, setElementHeight] = useState(0);

const onLayout = useCallback(
(event) => {
const layout = event.nativeEvent.layout;
setElementHeight(layout.height);
},
```

```
[setElementHeight]
);

const inlineModelInfoStyle = useMemo(() => {
return {
top: elementHeight + 36,
};
}, [elementHeight]);

return (
<View style={styles.container}>
<WrapHeaderScrollable
title={t('parking_spot')}
contentContainerStyle={styles.contentContainerStyle}
headerAniStyle={styles.scrollView}
styleScrollView={styles.scrollView}
testID={TESTID.PARKING_INPUT_MANUALLY_PARKING_SPOT}
>
<View style={styles.content}>
<SvgParkingSpot style={styles.svg} />
<Text
type="H3"
semibold
color={Colors.Gray9}
center
style={styles.wrap}
onLayout={onLayout}
testID={TESTID.PARKING_INPUT_MANUALLY_ENTER_PARKING_SPOT}
>
{t('enter_parking_spot_number')}
```

```
<TouchableOpacity
onPress={setShowInfo}
testID={TESTID.PARKING_SPOT_INFO_BUTTON}
>
<IconOutline
name={'question-circle'}
size={20}
color={Colors.Gray8}
style={styles.iconInfo}
testID={TESTID.PARKING_INPUT_MANUALLY_QUESTION_ICON}
/>
</TouchableOpacity>
</Text>
<ParkingSpotInput
onTextInputFocus={onTextInputFocus}
input={parkingSpot}
style={styles.parkingInput}
isFocus={isTextFocus}
/>
<View style={styles.parkingArea}>
<IconOutline
name={'environment'}
color={Colors.Gray8}
size={20}
style={styles.icon}
/>
<Text
testID={TESTID.PARKING_INPUT_MANUALLY_PARKING_AREA_TEXT}
type="H4"
color={Colors.Gray8}
```

```
>
{t('parking_area')}
</Text>
</View>
<Text
testID={TESTID.PARKING_INPUT_MANUALLY_PARKING_ADDRESS}
type={'Body'}
color={Colors.Primary}
numberOfLines={1}
>
{parkingInfo.address}
</Text>
</View>
<View style={styles.confirmView}>
<CircleButton
size={56}
backgroundColor={!parkingSpot ? Colors.Gray6 : Colors.Primary}
style={parkingSpot && styles.buttonShadow}
onPress={onPressConfirmSpot}
disabled={!parkingSpot}
testID={TESTID.PARKING_SPOT_CONFIRM_SPOT}
>
<IconOutline
testID={TESTID.PARKING_INPUT_MANUALLY_GO_NEXT_ICON}
name={'arrow-right'}
size={24}
color={Colors.White}
/>
</CircleButton>
</View>
```

```
<Text
testID={TESTID.PARKING_INPUT_MANUALLY_SCAN_QR_CODE_NOTE}
center
type="Body"
color={Colors.Gray9}
>
{t('or_scan_qr_code_to_confirm')}
</Text>
<TouchableOpacity
testID={TESTID.PARKING_INPUT_MANUALLY_SCAN_QR_CODE_BUTTON}
style={styles.scanButton}
onPress={onPressScan}
>
<Text
testID={TESTID.PARKING_INPUT_MANUALLY_SCAN_QR_CODE_TEXT}
type="H4"
color={Colors.Orange}
bold
>
{t('scan_qr_code')}
</Text>
</TouchableOpacity>

<TextInput
value={parkingSpot}
onChangeText={onChangeText}
ref={inputRef}
style={styles.inputText}
onFocus={onFocus}
onBlur={onBlur}
```

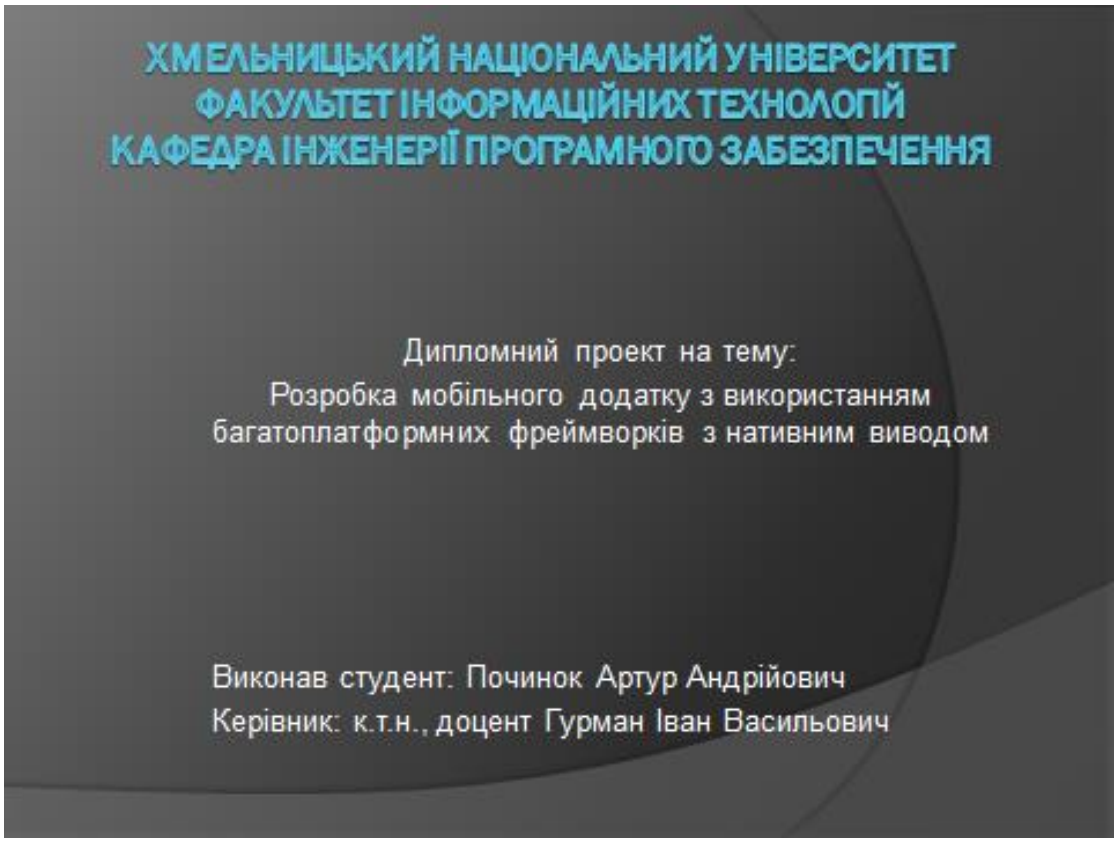
```
testID={TESTID.PARKING_SPOT_INPUT}
/>
</WrapHeaderScrollable>

<ButtonPopup
visible={isNotify}
secondaryTitle={'OK'}
onPressSecondary={setHideModal}
onClose={setHideModal}
testID={TESTID.PARKING_SPOT_BUTTON_POPUP}
>
<Text
type="H4"
style={styles.title}
testID={TESTID.PARKING_SPOT_TEXT_RESULT}
>
{resultCheckCar}
</Text>
</ButtonPopup>

<Modal
isVisible={showInfo}
onBackButtonPress={setHideInfo}
onBackdropPress={setHideInfo}
backdropColor={Colors.Transparent}
animationIn={'zoomIn'}
animationOut={'zoomOut'}
style={inlineModelInfoStyle}
testID={TESTID.PARKING_SPOT_MODAL_INFO}
>
```

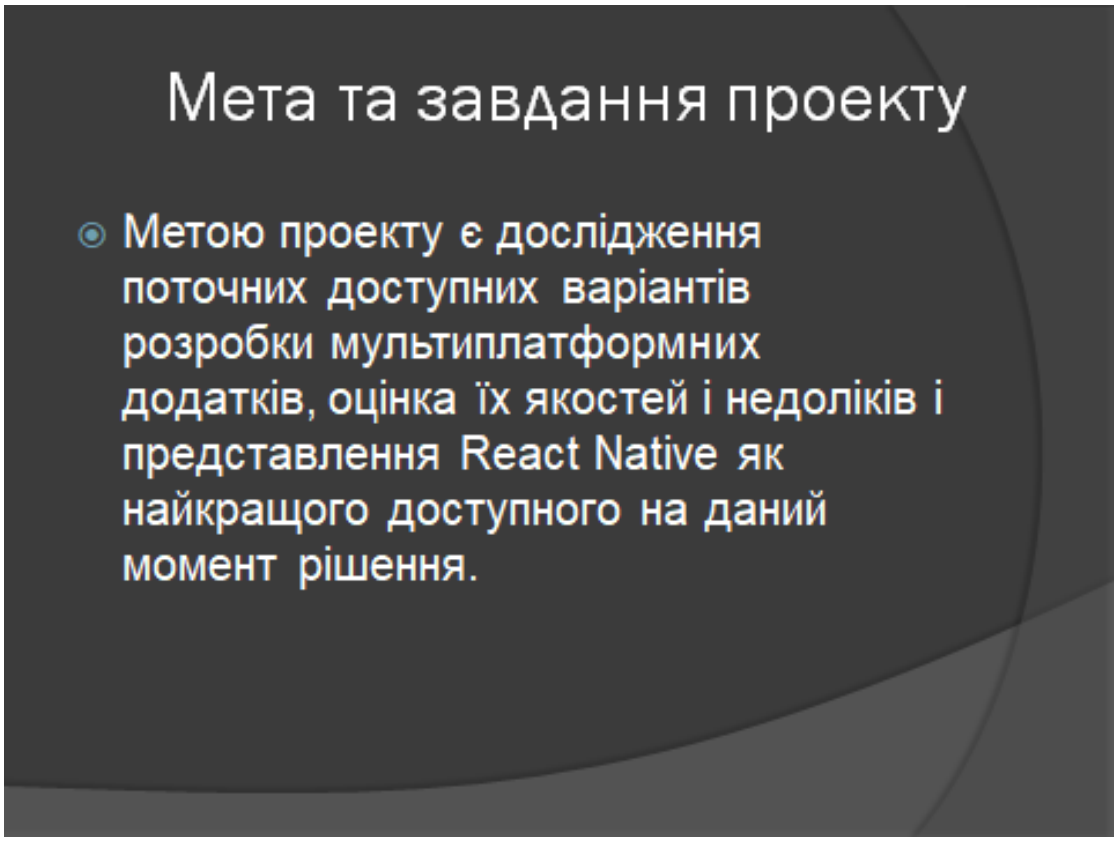
```
<View style={[styles.popoverStyle, styles.buttonShadow]}>
<Text
testID={TESTID.PARKING_INPUT_MANUALLY_QUESTION_TEXT}
type="Label"
style={styles.textDescription}
color={Colors.Gray8}
center
>
{t('enter_parking_spot_number_description')}
</Text>
</View>
</Modal>
</View>
);
});

export default ParkingInputManually;
```

ДОДАТОК В

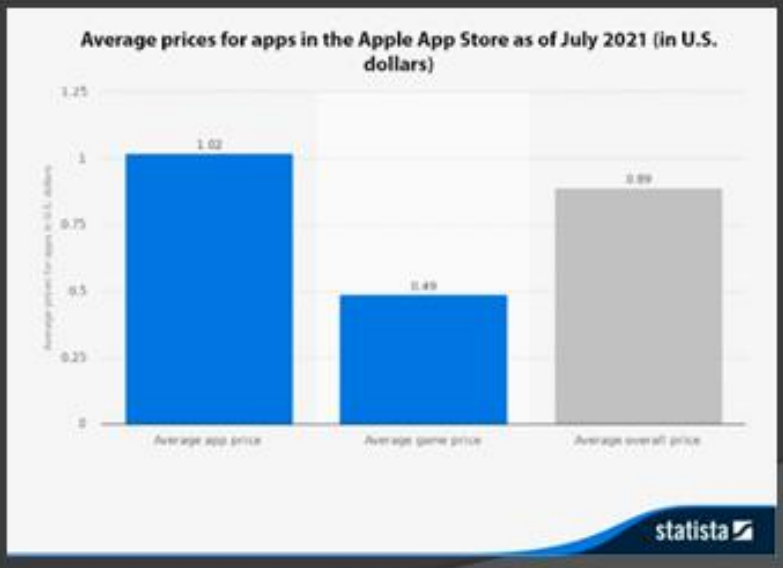
(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

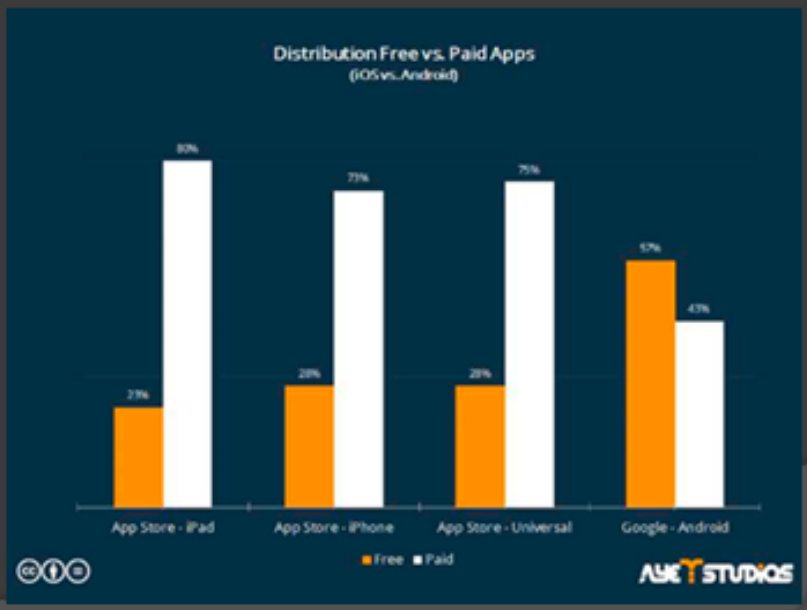

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
Дипломний проект на тему:
Розробка мобільного додатку з використанням багатоплатформних фреймворків з нативним виводом
Виконав студент: Починок Артур Андрійович
Керівник: к.т.н., доцент Гурман Іван Васильович


Мета та завдання проекту
Метою проекту є дослідження поточних доступних варіантів розробки мультиплатформних додатків, оцінка їх якостей і недоліків і представлення React Native як найкращого доступного на даний момент рішення.

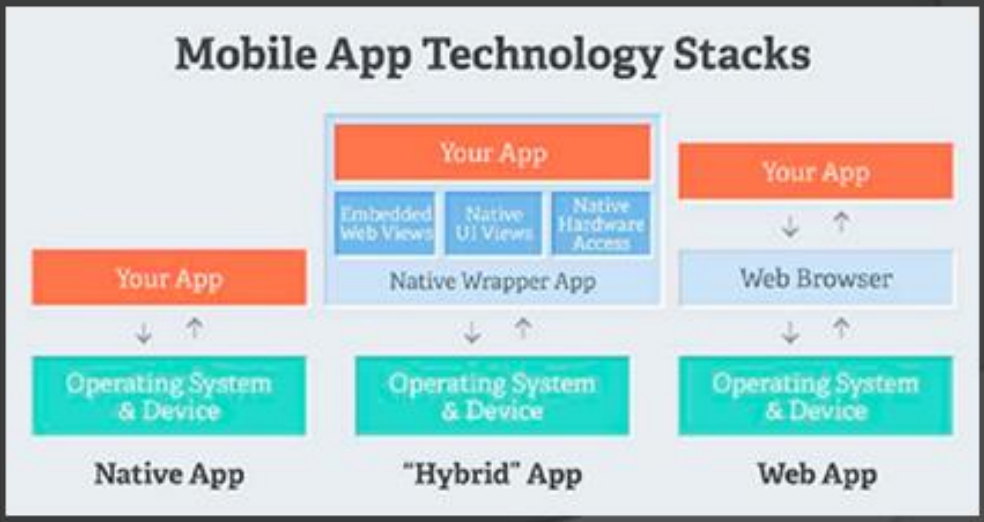
Середня ціна додатків у Apple App Store



Розподіл безкоштовних та платних додатків



Стек технологій мобільних додатків



Порівняння традиційних Xamarin і Xamarin.forms

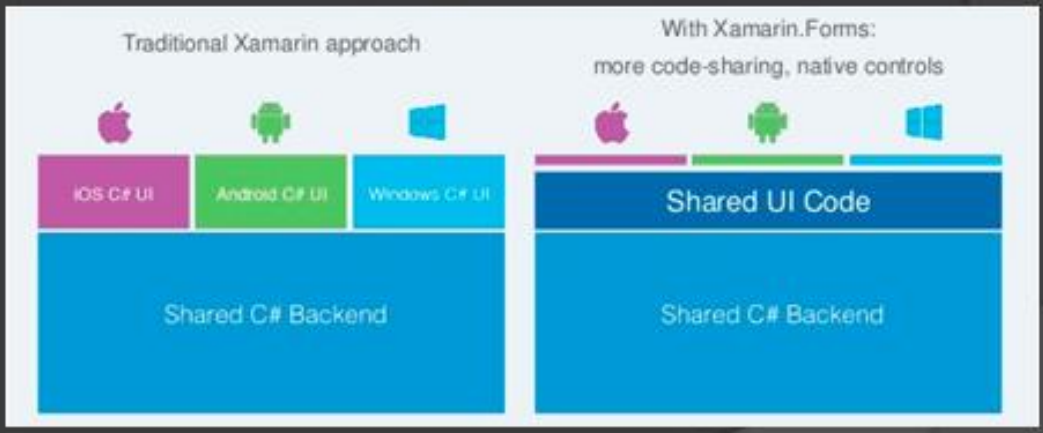
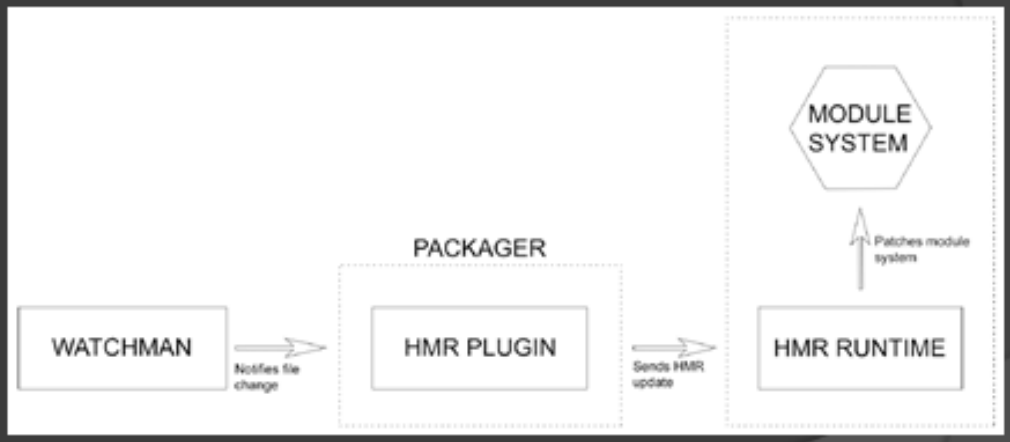
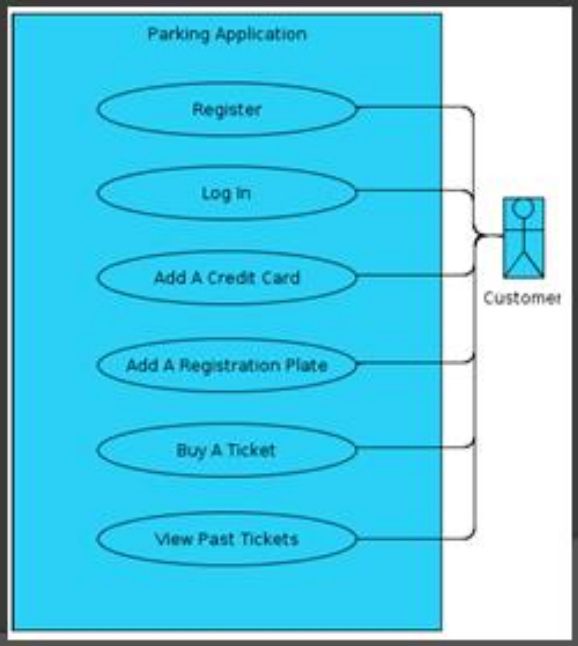


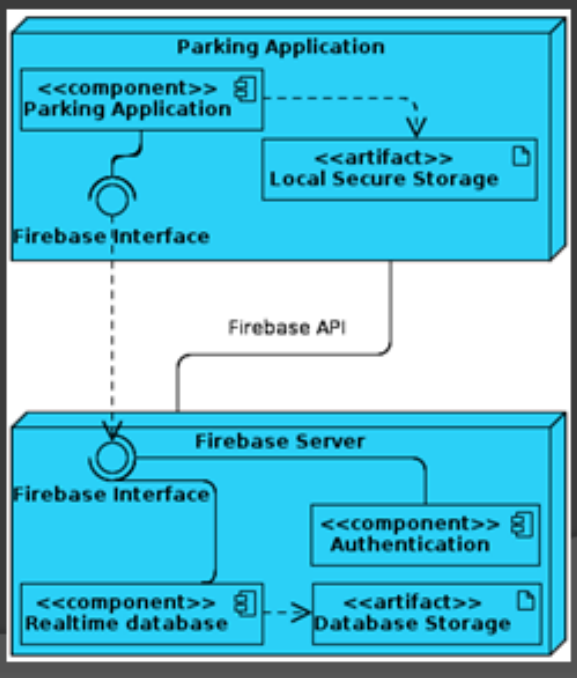
Схема перезавантаження гарячого модуля в React Native



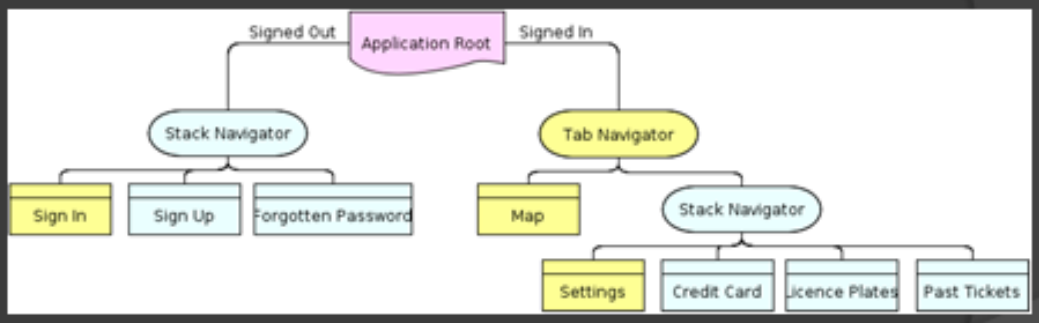
Діаграма варіантів використання програми



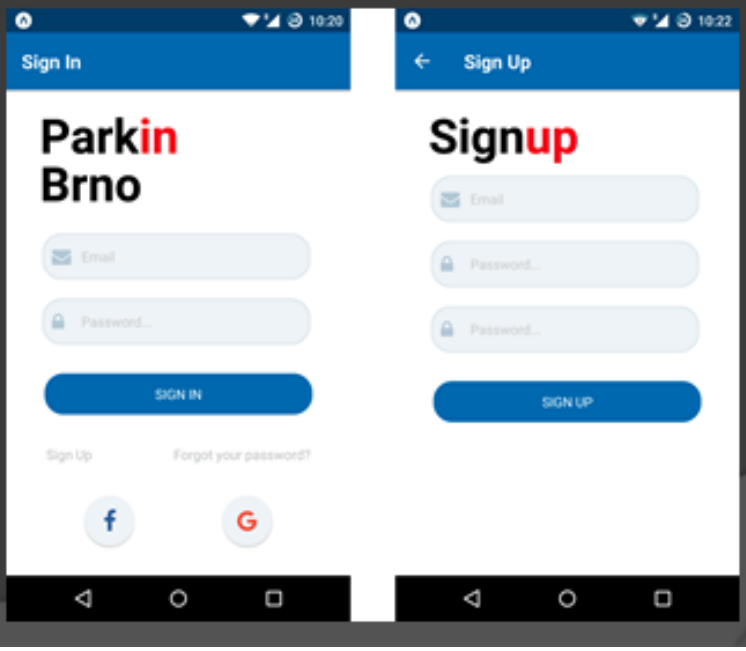
Діаграма розгортання розробленої програми



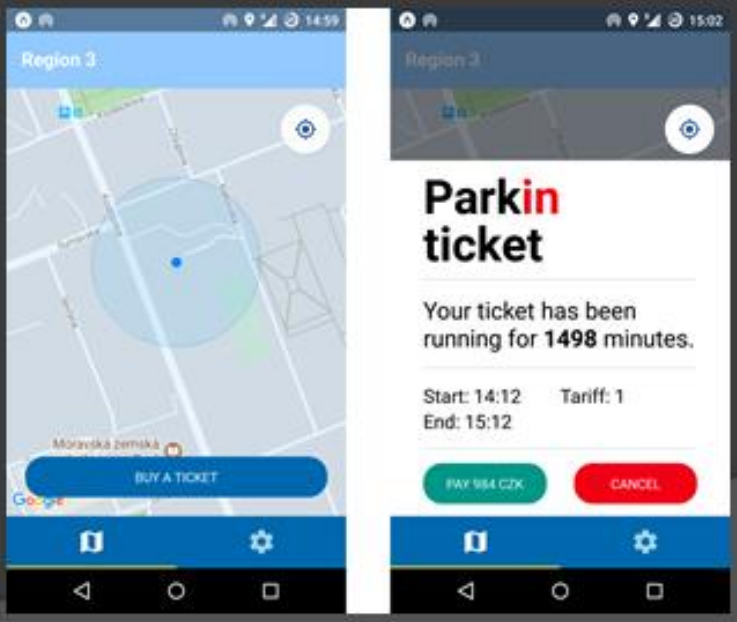
Діаграма навігації



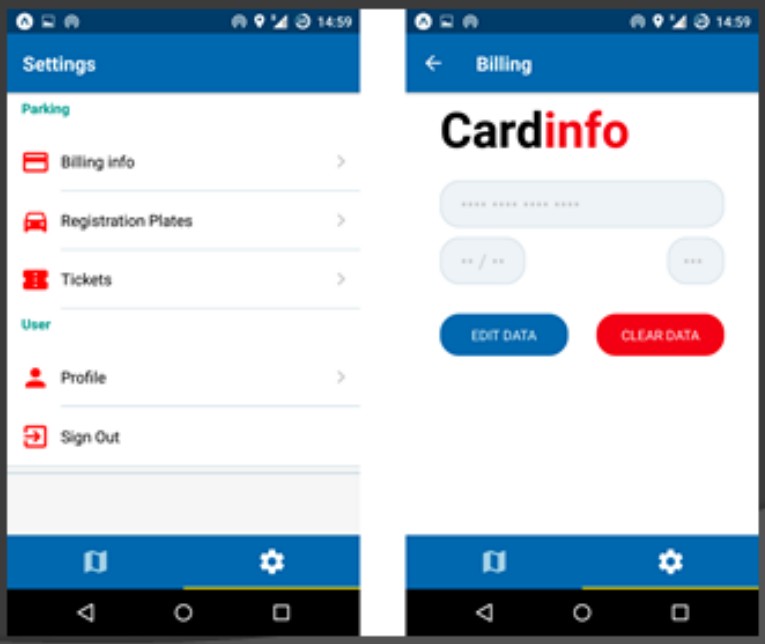
Екрани входу та реєстрації



Екран карти та підтвердження покупки квитка



Екран налаштувань та екран рахунку



Висновки

- У дипломному проекті проаналізовано можливості, переваги, недоліки декількох мультиплатформних фреймворків.
- На основі проведеного аналізу для реалізації проекту була віддана перевага фреймворку React Native.
- Був розроблений додаток для здійснення платежів за паркування.
- Завдяки мультиплатформному фреймворку, який застосовувався для розробки, значно спрощується портування проекту на декілька платформ.

Завідувачу кафедри інженерії програмного
забезпечення проф. Бедратюку Л. П.
здобувача вищої освіти
Починка А.А.
факультет ІТ, 3 курс, група ІПЗс-19-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

10.06.2022р.
дата

підпис

Tue Jun 14 10:05:37 EEST 2022, Хіврич Володимир Русланович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 4.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 12%**

ID: 105152 Назва: Розробка мобільного додатку з використанням багатоплатформних фреймворків з нативним виводом Додано в БД: 2022-06-14 Автора: А.А. Починок Керівники: І.В. Гурман Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	65840	559	2639 (4%)	38 (7%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

ID перевірки:
1011570985

Дата перевірки:
14.06.2022 09:43:04 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
14.06.2022 09:43:24 EEST

ID користувача:
100005589

Назва документа: **Диплом Починок_1_без дод**

Кількість сторінок: **61** Кількість слів: **9550** Кількість символів: **76719** Розмір файлу: **3.02 MB** ID файлу: **1011441800**

5.78%
Схожість

Найбільша схожість: **4.75%** з джерелом з Бібліотеки (ID файлу: **1011347703**)

1.38% Джерела з Інтернету	138	Сторінка 63
5.19% Джерела з Бібліотеки	117	Сторінка 64

0% Цитат

Включення цитат вимкнене

Включення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	7

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНИЙ ПРОЕКТ
освітнього ступеня «Бакалавр»

Дипломник Починок Артур Андрійович

Тема Розробка мобільного додатка з використанням багатоплатформних фреймворків з нативним виводом

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг дипломного проекту:

Кількість листів креслень _____; кількість сторінок записки 59

1.Короткий зміст пояснювальної записки та прийнятих рішень У ході роботи над дипломним проектом було здійснено аналіз предметної області та постановку задачі. У другому розділі проаналізовано особливості багатоплатформної розробки. У третьому розділі проведено поглиблений огляд фреймворку React Native як програмного середовища для розробки нативних мобільних додатків. У четвертому розділі наведено програмний додаток, що надає можливість користувачам здійснювати оплату паркувальних місць з автоматичним розрахунком вартості у залежності від місця паркування та з врахуванням місця проживання та роботи користувача.

2. Висновок про відповідність проекту поставленому завданню Дипломний проект виконаний відповідно до поставленого завдання та з дотриманням вимог.

3. Характеристика виконання кожного розділу проекту, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі викладено короткий зміст дипломного проекту. У першому розділі визначено основні вимоги до мобільного додатку, наведено та порівняно відомі методи розроблення додатків. У другому розділі проаналізовано багатоплатформний підхід до розроблення мобільних додатків, зокрема нативних, а також існуючі мультиплатформні фреймворки. У третьому розділі зроблено поглиблений огляд можливостей кросплатформного фреймворку для розробки нативних мобільних та настільних додатків React Native. У четвертому розділі представлено програмну реалізацію нативного мобільного додатку з використанням React Native, зокрема, визначено функціонал додатку, проаналізовано та спроектовано його структуру та описано реалізацію.

4. Позитивні сторони проекту Тематика дипломного проекту є актуальною, оскільки вирішує актуальну задачу розробки мобільного додатку із застосуванням сучасних технологій програмування нативних мобільних додатків та фреймворку React Native.

5. Негативні сторони проекту _____

6. Оцінка графічного оформлення та пояснювальної записки проекту Графічне оформлення виконане відповідно до теми дипломного проекту та подане у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про дипломний проект в цілому Дипломний проект заслуговує оцінки задовільно. Матеріал пояснювальної записки структурований та послідовний, що дозволяє зрозуміти викладений матеріал у рамках тематики дипломного проекту. Графічний матеріал надає можливість наочно побачити деталі проектування програмного додатку.

8. Інші зауваження _____

9. Оцінка дипломного проекту Дипломний проект виконаний у достатньому обсязі, відповідає поставленій задачі та заслуговує на оцінку «задовільно».

РЕЦЕНЗЕНТ Бобровнікова Кіра Юліївна, к.т.н., доц. кафедри комп'ютерної інженерії та інформаційних систем

“13” червня 2022 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:
Назва: «Розробка мобільного додатку з використанням багатоплатформних фреймворків з нативним виводом»
Автор: Починок Артур Андрійович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: освітньо-професійна програма Інженерія програмного забезпечення
Науковий керівник: Гурман Іван Васильович, к.т.н, доцент.
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних). Робота приймається до захисту.	Відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Сумарні співпадіння документа складають 5.78%

Серед співпадінь з документами у глобальній мережі максимальне співпадіння з одним документом становить 1,38% та стосуються переліку джерел посилань.

Серед співпадінь з документами з бібліотеки максимальне співпадіння з одним документом становить 5,19% та стосуються стандартних сторінок поясновальних записок тобто – титульний аркуш, бланк завдання, тощо.

Керівник _____ І. В. Гурман

Гарант ОП _____ Л. П. Бедратюк

Завідувач кафедри _____ Л. П. Бедратюк