Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження підходів до інтеграції картографічних систем у React Native застосунки
(тема)

Виконав:
студент (ка) 2 курсу, групи ІПЗм-22-5

Коновалов Б. В.
(прізвище, ініціали)

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

Керівник проф. каф. ПІ, Руткас А.Г.
(посада, прізвище, ініціали)

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

2024 р.

Харківський національний університет радіоелектроніки

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

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Коновалову Богдану Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження підходів до інтеграції картографічних систем у React Native застосунки»

Затверджена наказом по університету від 29.03. 2024р. № 250 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 21.06.2024

3. Вихідні дані до роботи Дослідження підходів до інтеграції картографічних систем у React Native застосунки

4. Перелік питань, що потрібно опрацювати в роботі

Метою роботи – є дослідити та порівняти для знаходження оптимального або більш підходящого до певної функціональної задачі підходу до інтеграції картографічних систем у мобільних додатках на базі React Native. В результаті необхідно знайти оптимальний або більш підходящий до певної функціональної задачі підхід до інтеграції картографічних систем у React Native середовищі.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Огляд літератури, аналіз проблеми та постановка задачі дослідження	22.01.2024 – 12.02.2024	виконано
2	Дослідження підходів до інтеграції картографічних систем	12.02.2024 – 25.03.2024	виконано
3	Програмна реалізація застосунку	25.02.2024 – 18.05.2024	виконано
4	Проведення експериментальних досліджень та аналіз результатів	01.04.2024 – 29.04.2024	виконано
5	Підготовка пояснювальної записки	22.04.2024 – 20.05.2024	виконано
6	Перевірка на плагіат та нормоконтроль	12.06.2024 – 15.06.2024	виконано
7	Підготовка презентації та доповіді	12.06.2024 – 19.06.2024	виконано
8	Рецензування	17.06.2024 – 19.06.2024	виконано
9	Попередній захист	19.06.2024	виконано
10	Занесення диплома в електронний архів	20.06.2024	виконано
11	Допуск до захисту у зав. кафедри	20.06.2024	виконано

Дата видачі завдання 22.01.2024 р.

Студент (ка) _____ Коновалов Б. В.
(підпис)

Керівник роботи _____ проф. каф. ПІ, Руткас А.Г.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Кваліфікаційна робота магістра містить: 69 с., 18 рис., 2 таб., 16 джерел.

АНАЛІТИКА, ВЕБ-В'Ю, ПАТЕРН, ANDROID, IOS, REACT NATIVE, SDK.

Об’єктом дослідження – є підходи до інтеграції картографічних систем у мобільних додатках на базі React Native.

Метою роботи – є дослідити та порівняти для знаходження оптимального або більш підходящого до певної функціональної задачі підходу до інтеграції картографічних систем у React Native середовищі.

В результаті буде знайдено оптимальний або більш підходящий до певної функціональної задачі підхід до інтеграції картографічних систем у React Native додатках.

ANALYTICS, SDK, PATTERN, WEB VIEW, IOS, ANDROID, REACT NATIVE.

The object of the research is the approaches of integration of map systems into React Native mobile applications.

The purpose of the work is to investigate and compare approaches of integration of map user interfaces (UI) on React Native in order to find the optimal or more suitable for a certain functional task.

As a result, the optimal or more suitable approaches for displaying cartographic systems on React Native will be found for a certain functional task.

Я, Коновалов Богдан Володимирович студент групи ІПЗм-22-5 здобувач вищої освіти на другому (магістерському) рівні кафедра програмної інженерії, заявляю: моя кваліфікаційна робота на тему Дослідження підходів до інтеграції картографічних систем у React Native застосунки, що буде представлена в

екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

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

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface

FPS – Frames Per Second

CPU – Central Processing Unit

RAM – Random Access Memory

UI/UX – User Interface and User Experience

QA – Quality Assurance

ВСТУП

React Native – це відкрита платформа для розробки мобільних додатків, що дозволяє розробникам створювати кросплатформові додатки для Android і iOS, використовуючи відомий та популярний фреймворк React. Основним перевагою React Native є можливість розробляти мобільні додатки з високою швидкістю та якістю, використовуючи знання JavaScript та React.

Він дозволяє ефективно використовувати один код для обох платформ, зменшуючи час та витрати на розробку.

Однією з ключових функцій сучасних мобільних додатків є можливість відображати та взаємодіяти з географічними даними, такими як карти, маршрути, місця та інші локаційні дані. Тут картографічний функціонал стає важливою частиною додатків для різних сфер, таких як туризм, транспорт, соціальні мережі, логістика та багато інших.

Тож ефективна імплементація картографічних SDK у React Native важлива для забезпечення оптимальної продуктивності, покращення користувацького досвіду, зменшення витрат ресурсів, масштабованості додатку та підтримки.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної області картографічних систем

Сьогодні картографічні системи представляють собою складні та інтегровані геоінформаційні системи, які поєднують в собі сучасні технології збору, обробки та візуалізації географічних даних. Вони включають в себе набір інструментів та ресурсів для створення, аналізу та використання географічної інформації.

Вони виконують безліч важливих функцій та знаходять застосування в різних галузях.

Основна функція карт – це створення візуального представлення географічної інформації. Карти дозволяють нам бачити географічні об'єкти, рельєф, водні дороги і багато інших аспектів нашого світу.

Карти використовуються для навігації та орієнтування в просторі. Вони допомагають нам знаходити шлях до пункту призначення, планувати подорожі та вибирати оптимальні маршрути.

У галузі містобудування, геології, лісництва та інших галузях картографія використовується для планування розвитку і використання територій. Вона допомагає вирішувати питання щодо розміщення об'єктів і інфраструктури.

Картографія використовується для проведення досліджень і аналізу географічних даних. Вона дозволяє науковцям вивчати зміни в природних та соціальних процесах, а також відстежувати тенденції[1].

Картографія допомагає в спостереженні за природними явищами, такими як зміни в кліматі, рухи льодовиків та інші явища. Вона також використовується для моніторингу забруднення навколишнього середовища.

Сучасні картографічні системи використовують високотехнологічні інструменти, такі як ГІС, супутникова зйомка та комп'ютерна обробка даних. Однією з найважливіших ролей таких систем є забезпечення точного та актуального доступу до географічної інформації для широкого кола користувачів.

Подібні системи грають важливу роль у глобальній навігації, забезпечуючи GPS-навігацію для автомобілів, смартфонів і навігаційних систем. Вони

допомагають екологам вивчати та моніторити зміни в середовищі, а лікарям - визначати поширення хвороби та епідемій[2].

Картографічні системи також використовуються у бізнесі для реклами та маркетингу, допомагаючи підприємствам аналізувати ринки та споживачів. Картографічні дані також стають важливим ресурсом для державних установ та організацій у вирішенні питань національної безпеки, плануванні інфраструктури та екологічних заходів.

Сучасні картографічні системи наприклад, Google Maps, Mapbox, Here Maps надають користувачам можливість швидко та легко отримувати доступ до географічної інформації через інтернет. Вони використовуються для навігації, пошуку місць та інших цілей.

1.2 Предметна область інтеграції різноманітних бібліотек та SDK у React Native застосунки

Інтеграція різноманітних бібліотек та SDK (Software Development Kit) в React Native застосунки є важливою для розробників мобільних додатків. React Native – це популярний фреймворк для розробки мобільних додатків з використанням JavaScript та React. Інтеграція різних бібліотек та SDK дозволяє розробникам розширити функціональність своїх додатків та використовувати сторонні сервіси та функції.

У світі React Native існує безліч бібліотек і SDK для різних цілей, включаючи роботу з графікою[3], мережевими запитами, аутентифікацією, базами даних, мапами, інтеграцією соціальних мереж, аналітикою і багато іншого. Деякі з популярних бібліотек і SDK для React Native включають react-navigation (навігація), Redux (управління станом додатка), Firebase (хмарні сервіси), Mapbox та react-native-maps (карти), Facebook SDK (інтеграція з Facebook API) та багато інших.

Інтеграція бібліотек та SDK в React Native додаток зазвичай включає в себе кілька кроків. Перш за все, розробник повинен встановити бібліотеку або SDK в свій проект за допомогою пакетного менеджера, такого як npm чи yarn. Після

цього необхідно змінити конфігураційні файли проекту та внести необхідні зміни в код. Кожна бібліотека може мати свою документацію та рекомендації щодо інтеграції, які розробник повинен слідувати.

Після встановлення бібліотеки або SDK необхідно налаштувати його для потрібного функціоналу. Це може включати в себе налаштування ключів API, дозволів або параметрів конфігурації. Розробники також можуть створювати власні компоненти та обгортки (wrappers), щоб зробити інтеграцію з React Native більш зручною.

Після інтеграції бібліотеки чи SDK важливо провести тестування, щоб переконатися, що вони працюють правильно та не викликають конфліктів з іншими частинами додатка. Також, необхідно враховувати можливі проблеми з сумісністю і забезпечити належний облік помилок (error handling)[4].

Розробники повинні відслідковувати оновлення бібліотек та SDK, оскільки це може вплинути на функціональність та безпеку додатка.

При інтеграції сторонніх бібліотек і SDK важливо враховувати питання безпеки та конфіденційності даних. Розробники повинні переконатися, що дані користувачів залишаються захищеними та що інтегровані компоненти відповідають стандартам безпеки.

Інтеграція різних бібліотек та SDK є важливою частиною розробки React Native додатків, оскільки вона дозволяє розширювати функціональність та використовувати сторонні сервіси для поліпшення додатків. Однак це також потребує уважності, тестування та підтримки для забезпечення безпеки та стабільності додатків.

1.3 Актуальність проблеми

Розбираючи варіанти інтеграції різноманітних картографічних систем до React Native застосунків ми розуміємо що поки що є можливість повноцінно працювати лише із Google Maps SDK, та не існує готових рішень для імплементації іншого SDK.

Google Maps SDK є однією з найпопулярніших та найрозповсюдженіших картографічних платформ у світі. Багато розробників вибирають його через його високу якість, доступність та обширний функціонал.

Проте ця популярність призводить до монополії, де інші картографічні сервіси мають обмежену можливість конкурувати та надавати свої послуги на платформі React Native.

Розробники можуть бажати використовувати інші картографічні сервіси, такі як Mapbox, Here Maps, TomTom та інші, які можуть краще відповідати їхнім потребам або бізнес-вимогам.

Проте відсутність бібліотек для інтеграції цих сервісів у React Native призводить до того, що розробники обмежені в виборі та не мають можливості легко інтегрувати інші картографічні рішення[5].

Саме тому перед нами постає проблема вирішення питання най ефективнішої імплементації картографічних SDK у додаток React Native.

1.4 Постановка задачі

У ході даної роботи нами буде досліджено предметну область картографічних систем, імплементації сторонніх бібліотек та SDK до застосунків на React Native.

Ми проведемо порівняльний аналіз та аналіз призначення кожного архітектурного елементу обраного підходу, надалі планується створити загальну архітектурну реалізацію обох підходів для майбутнього порівняння.

Ця робота спрямована на створення дорожньої карти для розробників щодо імплементації їх картографічної системи у свій застосунок.

Реалізація наукового дослідження складається з наступних етапів:

– аналіз предметної області;

– аналіз архітектурних особливостей сучасних SDK;

– огляд та аналіз існуючих підходів;

– розробка рекомендацій та класифікація кожного підходу до певних потреб проекту.

2 АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ТА МЕТОДІВ

2.1 Інтеграція картографічних систем через веб-в'ю підхід

React Native – це крос-платформний фреймворк для розробки мобільних додатків. У 2013 році, коли команда Facebook розпочала його створення, основною метою було надати користувачам можливість взаємодіяти з їх сервісом не через мобільний браузер, а через окремий додаток на їх смартфоні. Це мало вирішити проблему недостатньо розвинених мобільних браузерів, на яких соціальна мережа не могла показати ті самі UI/UX показники, що і на десктопній веб-версії. Коли знайшлась можливість за допомогою мови JavaScript генерувати елементи нативного інтерфейсу, команда зрозуміла, що зможе використати їх флагманську бібліотеку React і в світі мобільної розробки.

Таке поєднання давало можливість використовувати один і той самий код для iOS та Android платформ, що мало б пришвидшити розробку, зробити ідентичний дизайн і наповнення додатку. При цьому розробники сайту могли б тепер реалізовувати фічі і в мобільному світі також, завдяки ідентичній мові програмування і схожим парадигмам. Так з'явився React Native.

Як ми бачимо, нічого не згадувалося про взаємодію з картографічними системами. Акцент був зроблений на прості елементи користувацького інтерфейсу, такі як кнопки, списки, блоки, текст і т.д. Чи щось змінилося зараз?

Згідно з останньою публічною активністю представників команди Meta (раніше Facebook), ми спостерігаємо доволі часте наголошення на тому, що React Native сповідує концепцію «Lean Core»[6]. Це означає, що фреймворк має імплементувати лише самі необхідні речі, які будуть використовуватися розробниками будь-якого проекту у 90% випадків, а все, що має менш розповсюджене використання, буде винесено в окремі бібліотеки, які будуть підтримуватися ком'юніті.

Отже, з вищезазначеного можемо зробити висновок, що картографія в React Native не була і не буде вбудованою. Так, як же такий популярний фреймворк на протязі 11 років надає можливість розробникам інтегрувати мапи у свої додатки? Як було сказано вище – за допомогою окремих JavaScript бібліотек.

Наразі в світі React Native існують наступні більш-менш стабільні картографічні бібліотеки: react-native-maps, @rnmapbox/maps. Вибір є досить обмеженим, враховуючи той факт, що всі вони надають різний набір функцій, мають різні архітектури й обмеження.

Також не всі популярні провайдери картографічних систем представлені у нашому списку. І ось ми підходимо до головної проблеми: як бути, якщо бібліотека відсутня або не підходить з тих чи інших причин?

Для того, щоб зрозуміти відповідь на це питання і розглянути підходи до його вирішення, потрібно зрозуміти, з чого складається картографічна бібліотека для React Native. Вона складається з двох шарів: шару візуалізації та шару контролю. Підходи будуються в залежності від візуалізації, методи - в залежності від того, як ми плануємо здійснювати контроль[7].

Для того, щоб побачити підходи, нам треба спочатку з’ясувати, які типи картографічних систем в цілому можуть існувати (або, простіше кажучи, бути відображеними) у мобільному середовищі.

В мобільному світі можуть існувати 2 типи картографічних систем:

– побудовані для нативного (ios/android) середовища;
– побудовані для браузерного (web) середовища.

Виходячи з цього, ми можемо зробити логічний висновок. Якщо картографічна система не призначена для нативного або браузерного середовища, тобто не працює в браузері або на мобільних телефонах iOS/Android, то вона не може бути інтегрована у React Native.

Тепер давайте розглянемо наші підходи. Почнемо з інтеграції картографічної системи через нативне середовище.

Як ми пам’ятаємо з першого абзацу, React Native поєднує два світи: світ JavaScript та світ нативний (iOS/Android). Наразі у React Native є можливість будувати програмне забезпечення і для Windows, і для watchOS, і для tvOS, і навіть для visionOS, але в рамках цієї роботи ми розглядаємо лише ті операційні системи, що працюють на мобільних телефонах – це iOS і Android. Отже, якщо React Native спілкується з нативним світом, щоб відтворювати прості речі, чи

може він також дозволити розробникам взаємодіяти з чимось більш складним, наприклад, картографічними системами[8].

Дійсно, React Native надає можливість розробникам здійснювати комунікацію з нативним світом самостійно. Реалізація такої взаємодії можлива через дві сутності: Native Module та Native Component. Різниця між цими двома полягає в тому, чи потрібно нам візуально щось показувати користувачу, чи достатньо отримати дані з нативного світу. У випадку з картою нам потрібно показувати карту, тому нас цікавить саме друга сутність.

Реалізація Native Component для картографічної системи передбачає наявність SDK (Software Development Kit) для iOS та Android. Саме SDK в нативному середовищі ми будемо контролювати через нашу сутність. І саме SDK буде відображати нашу карту на екрані смартфону. Такий підхід має наступні переваги:

- канонічність, тобто саме так рекомендує інтегрувати карти документація та RN ком'юніті;
- більшу кількість функцій для мобільних телефонів надають саме mobile sdk;
- краща продуктивність, оскільки mobile sdk може використовувати всі ресурси телефона і не обмежена лише браузерними api (application programming interface);
- rn надає всі необхідні методи для ефективного і послідовного контролю над mobile sdk (props, синхронні та асинхронні команди, механізм узгодження ui елементів з його оптимізацією тощо)[9];
- гнучкість у реалізації функцій додатку, будуванні архітектури картографічної інтеграції;
- дизайн картографічної системи адаптований до мобільних стандартів.

З недоліків можна зазначити наступні – розробники мають вміти програмувати під iOS та Android.

Перейдемо до другого підходу. Як ми знаємо, браузери досить широко використовуються на смартфонах. У кожному браузері сьогодні є вбудований

механізм – canvas, що дозволяє працювати з різноманітною 2D графікою. Саме завдяки цьому інструменту провайдери картографічних систем створюють свої бібліотеки для браузерів. Для того щоб використати браузер у мобільному додатку, потрібно скористатися WebView компонентом. Він, до речі, також поставляється у вигляді RN бібліотеки з недавніх часів відповідно до «Lean Core». Для здійснення програмної взаємодії з картою (контролю) в рамках цього підходу в нас є три методи: 1) injectedJavaScript property або injectJavaScript method, 2) postMessage, 3) onMessage. Такий підхід матиме наступні переваги:

- не має потреби у навичках програмувати під iOS та Android (для пересічного RN розробника програмування на JavaScript у браузері є більш зрозумілим);
- браузер регулює використання ресурсів смартфону і розробники матимуть менше оптимізаційних проблем;
- швидший час первинної інтеграції, для запуску нативного проекту потрібно більше часу і ресурсів.
- Наявні наступні недоліки:
- обмежена кількість функцій, доступних через браузерне API;
- гірша продуктивність;
- обмежений набір інструментів контролю, що не дає можливості повноцінно будувати проекти з великим навантаженням на картографічну систему;
- дизайн не адаптований до мобільних стандартів.

Веб-в'ю підхід у React Native для інтеграції картографічних систем використовує вбудований веб-переглядач, що дозволяє інтегрувати повнофункціональні веб-базовані картографічні сервіси безпосередньо в мобільний додаток.

Використання HTML, CSS, та JavaScript у веб-в'ю забезпечує розробникам гнучкість та зручність у створенні користувацького інтерфейсу, одночасно зберігаючи переваги нативного додатку.

Цей підхід дозволяє швидко впроваджувати та оновлювати картографічні функції, мінімізуючи потребу в складній нативній розробці та спрощуючи підтримку додатку на різних платформах.

Використання веб-в'ю в React Native для інтеграції картографічних систем є привабливим рішенням для розробників, оскільки це дає можливість швидко інтегрувати багатий веб-контент у мобільні додатки.

Завдяки вбудовуванню веб-переглядача всередині нативного додатку, цей підхід дозволяє використовувати вже існуючі веб-базовані інструменти та фреймворки для відображення карт, забезпечуючи велику гнучкість у дизайні та функціональності.

Однак, важливо враховувати можливі обмеження веб-в'ю, такі як знижена продуктивність та потенційні обмеження у використанні деяких нативних можливостей пристрою.

Підводячи підсумок, важливо зауважити, що існуючі картографічні бібліотеки у React Native реалізовані за допомогою першого (нативного) підходу. Другий підхід може бути використаний як тимчасова альтернатива на початку проекту або на стадії MVP (Minimum viable product). Також браузерний підхід може бути прийнятним у разі відсутності мобільної реалізації для картографічної системи або в разі відсутності навичок програмування під iOS/Android у команди[10].

У всіх інших випадках нативний підхід має бути взятий за основу, оскільки саме він здатен опанувати картографічні інтеграції максимальної складності, надаючи всі необхідні інструменти контролю розробникам, що є запорукою довготривалих рішень.

Першим кроком у розробці додатку, який вимагає інтеграції карт, є вибір відповідного картографічного SDK. На ринку існує багато варіантів, які можна використовувати для React Native додатків. Варто розглянути кожен із них.

Mapbox SDK – Mapbox є однією з популярних платформ для картографії, яка пропонує SDK для різних мобільних платформ, включаючи React Native. Вони надають потужні можливості для створення інтерактивних карт та маршрутів.

Google Maps SDK – Google Maps - це інший відомий гравець у сфері картографії. Вони також надають SDK для React Native, яке дозволяє використовувати всі можливості Google Maps у ваших додатках.

Leaflet – це легкий і простий використанні JavaScript-фреймворк для картографії. Хоча він не є SDK, він може бути легко інтегрований у React Native за допомогою веб-в'ю підходу.

OpenLayers – це ще один потужний JavaScript-фреймворк для картографії. Він надає багато можливостей для роботи з картами та геоданими на веб-сторінках та додатках React Native.

HERE – це глобальна компанія, яка спеціалізується на геоданих і місцевих послугах. Вони також пропонують SDK для React Native, яке дозволяє використовувати їхні карти та сервіси.

Інтеграція картографічного SDK у React Native за допомогою веб-в'ю підходу може бути розділена на кілька етапів(див. рис. 2.1).

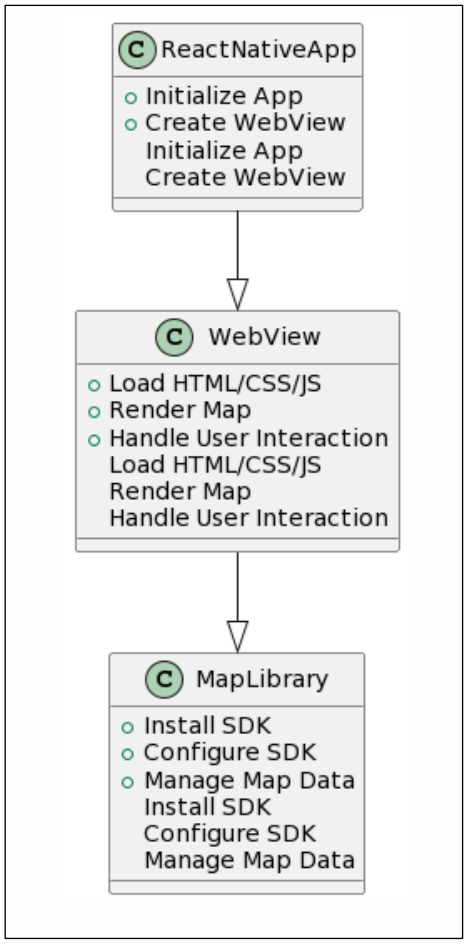


Рисунок 2.1 – Інтеграція карти по веб-в’ю підходу

Етапи інтеграції картографічного SDK в React Native без коду можна розглядати так:

- вибір картографічного sdk: на першому етапі необхідно вибрати відповідне картографічне sdk для вашого проекту. Оцініть потреби додатку та виберіть sdk, яке найкраще відповідає вашим вимогам. Розгляньте функціональність, ліцензійні умови, вартість, підтримку та спільноту користувачів;
- встановлення sdk: після вибору sdk, встановіть його у вашому React Native проекті. Використовуйте менеджер пакетів, такий як npm або yarn, для завантаження та встановлення sdk. цей крок зазвичай включає в себе виконання команди у терміналі для завантаження sdk та його залежностей;
- ініціалізація sdk: після встановлення sdk, ініціалізуйте його у вашому проекті. Це може включати в себе налаштування ключів api (якщо необхідно), налаштування параметрів sdk та інших конфігураційних параметрів. На цьому етапі також зазвичай створюється початковий об'єкт картографічного sdk;
- створення веб-в'ю для карти: для відображення карти використовується веб-в'ю. Створіть веб-в'ю компонент у вашому React Native додатку, який буде відповідати за відображення картографічної інформації. В цей компонент буде вбудовано веб-сторінку з відображенням карти;
- завантаження html/css/js для карти: веб-в'ю компонент потребує html, css та javascript для відображення карти. Завантажте необхідні файли для карти та включіть їх у вашому проекті. Це може бути файлами, розміщеними локально, або завантаженими з віддаленого джерела;
- налаштування та відображення карти: встановіть необхідні налаштування та параметри для картографічного sdk через javascript api. Це може включати в себе встановлення початкового масштабу, центру карти, створення маркерів та інші налаштування для відображення карти відповідно до ваших потреб;

- керування картографією: додайте функціональність для керування картографією у вашому react native додатку. Це може включати в себе можливість додавати маркери, створювати маршрути, відслідковувати події на карті та інші функції, які дозволяють користувачам взаємодіяти з картографією;
- оптимізація та взаємодія: останній етап включає в себе оптимізацію та взаємодію вашого додатку з відображеною картою. Розгляньте можливості кешування даних для покращення продуктивності, обробку помилок та вдосконалення інтерфейсу взаємодії з користувачами.

Ці етапи необхідні щоб послідовно інтегрувати картографічне SDK у React Native додатку та створити ефективну систему відображення та керування картою.

Для того щоб краще зрозуміти як даний підхід працює та для подальшого аналізу цього підходу та його особливостей – розглянемо коротко процес відображення картографічної системи через веб-в'ю.

У React Native додатку спершу потрібно імпортувати необхідний модуль або SDK згідно з документацією обраної SDK. Далі необхідно виконати ініціалізацію SDK та налаштувати його параметри відповідно до потреб. Цей крок може включати встановлення ключа API, налаштування масштабу, стартового місця, типів мапи та інших параметрів.

Для реалізації конкретної функціональності, як-от додавання маркерів, створення маршрутів або іншої взаємодії з картою, використовуються методи та інтерфейси, які надає SDK[11], наприклад: додавання маркера або створення маршруту.

Далі йде механізм обробники подій та з'єднання їх із відповідними функціями додатку, які будуть викликатися при взаємодії користувача з картою.

Також варто зазначити що веб-в'ю підхід дозволяє використовувати один і той же код для інтеграції карт на різних мобільних платформах, що зменшує зусилля та витрати часу на розробку.

Розглядаючи очевидні переваги – розробники можуть вибирати серед різних картографічних SDK в залежності від їхніх потреб та вимог проекту. За допомогою веб-в'ю, розробники можуть легко налаштувати та керувати виглядом та поведінкою карти за допомогою стандартних веб-технологій, таких як HTML, CSS і JavaScript.

Також ще є багато картографічних SDK постійно оновлюються і підтримуються, забезпечуючи актуальність та безпеку для вашого додатку.

У підсумку, веб-в'ю підхід для інтеграції картографічне SDK на React Native є потужним та ефективним способом створення додатків з інтерактивними картами. Він надає розробникам гнучкість та можливість вибору серед багатьох варіантів SDK, щоб задовольнити всі потреби проекту та дозволяє невеликим командам розробникам задовольняти функціональні потреби проекту відносно просто та без великих затрат по часу.

2.2 Кастомна інтеграція картографічних систем у React Native

"Кастомна Інтеграція Карти в React Native" - це підхід, який передбачає створення власного компонента карти для використання в React Native додатку замість використання готових бібліотек, таких як react-native-maps. Цей підхід дає розробникам більший контроль над функціональністю та виглядом карти, а також можливість інтеграції з різними картографічними сервісами, такими як OpenStreetMap, Here Maps, TomTom і інші.

Ключовою ідеєю цього підходу є створення React Native компонентів, які інкапсулюють функціональність інтерфейсу користувача для відображення карти та взаємодії з нею. Для цього використовується комбінація нативного коду для Android і iOS, а також коду JavaScript для React Native[12].

Спочатку розробники повинні вивчити документацію та API обраного картографічного сервісу, щоб зрозуміти, як взаємодіяти з ним. Потім вони створюють новий пакет React Native, який буде відповідати за інтеграцію з цим сервісом. Нативний код, написаний для обох платформ (Android і iOS), відповідає за ініціалізацію та взаємодію з API картографічного сервісу.

У контексті цього підходу, розробники створюють React Native компоненти, які дозволяють відображати карту на екрані, додавати маркери, створювати маршрути, налаштовувати вигляд та поведінку карти та обробляти події користувача. Ці компоненти забезпечують зручний інтерфейс для розробників React Native і можуть бути використані для створення додатків з картою за власними потребами.

Кастомна Інтеграція Карти в React Native вимагає ретельного тестування та відлагодження, а також документування для інших розробників, які можуть використовувати таку бібліотеку. Важливо також враховувати умови використання обраного картографічного сервісу та підтримувати проект після публікації.

Отже, "Кастомна Інтеграція Карти в React Native" дозволяє розробникам створювати власні компоненти карти з обраною функціональністю та виглядом, що відповідають їхнім унікальним потребам та вимогам. Цей підхід вимагає технічних знань, але дозволяє досягти більшого контролю над інтеграцією картографії в додаток.

Далі розглянемо етапи інтеграції. Створення аналогу react-native-maps для іншого картографічного сервісу може бути складним завданням, яке передбачає кілька етапів. Ось загальна послідовність дій для реалізації такого проекту:

- почати треба з детального вивчення документації та API обраного картографічного сервісу. Ретельно ознайомитись з усіма функціями, методами та ресурсами, які надає цей сервіс, і познайомтеся з умовами його використання;
- створити новий проект React Native або додати функціональність до існуючого проекту. Для реалізації інтеграції з обраним картографічним сервісом створіть окремий пакет або модуль;
- написати нативний код для платформ Android і iOS, який буде відповідати за взаємодію з API обраного сервісу та відображення картографії;

– створити компоненти React Native, які будуть відповідати за відображення карти та функціональність інтерфейсу користувача. Після – необхідне впровадження можливості керування цими компонентами через React Native API;
– реалізувати можливість відображення карти на екрані, забезпечуючи налаштування вигляду, позиції, масштабу та інших параметрів відображення;
– реалізувати функціональність для додавання маркерів, ліній, полігонів та інших графічних елементів на карту. Потім необхідно забезпечити можливість налаштовувати їх вигляд і взаємодію з користувачем.
– інтегрувати специфічний для обраного сервісу функціонал, такий як пошук місць, відображення інформації про локації, відображення трафіку та інші функції, які цей сервіс пропонує[13];
– наприкінці необхідно реалізувати можливість обробки подій на карті та взаємодії з компонентами JavaScript через React Native;
– завершити процес потрібно протестувавши результат реалізації підходу.

На рисунку 2.2 відображено візуально етапи інтеграції картографічних систем у додаток на React Native.

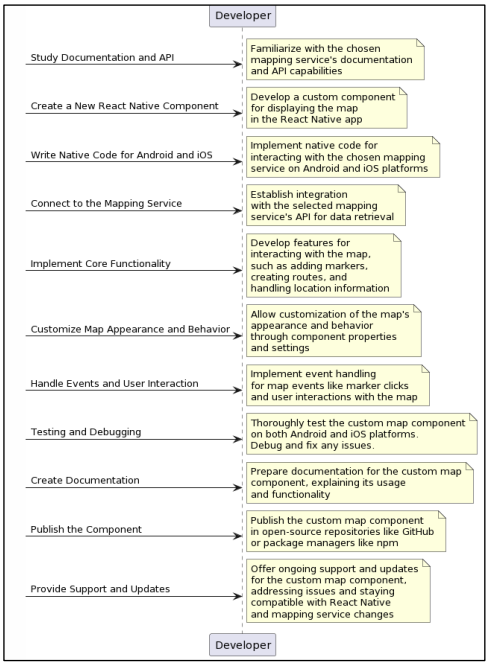


Рисунок 2.2 – Інтеграція карти в React Native через SDK

2.3 REACT NATIVE MAPS

Для більшого розуміння підходу до розробки та імплементації кастомного SDK – нам потрібно розглянути аналогічну систему яка вже існує, а це саме – react-native-maps.

React native maps – це популярна бібліотека для React Native, яка надає інструменти для інтеграції та використання картографічних функцій у мобільних додатках, розроблених з використанням React Native. Ця бібліотека надає зручний спосіб відображення і взаємодії з інтерактивною картою в додатках для платформ Android і iOS.

Бібліотека дозволяє легко відображати інтерактивну карту у вашому мобільному додатку. Ви можете вибрати тип карти (наприклад, карту Google або Apple Maps) та налаштувати відображення карти.

Також можна легко додавати маркери на карту, щоб позначити конкретні точки або об'єкти на ній. Кожен маркер може бути налаштований з власними значками та додатковою інформацією. Є можливість малювати лінії та полігони на карті. Це корисно для відображення маршрутів, кордонів об'єктів або інших географічних областей.

За допомогою цієї системи користувачі можуть збільшувати та зменшувати масштаб карти (зум) або переміщатися по ній. "react-native-maps" дозволяє керувати цими операціями.

"react-native-maps" також підтримує можливість пошуку місць і створення маршрутів між ними за допомогою відповідних сервісів.

Далі варто розглянути її складові елементи та їх взаємодію. Основні архітектурні елементи цієї бібліотеки та їх взаємодія між собою виглядають наступним чином:

- MapView: Клас MapView є центральним компонентом бібліотеки "react-native-maps"[14]. Він представляє собою контейнер, в якому відображається карта. Карту можна відобразити за допомогою цього компонента, вказавши його розміщення, розмір і інші параметри.

MapView також надає можливість взаємодіяти з картою та обробляти події;

- Marker: Компонент Marker використовується для додавання маркерів (позначок) на карту. Ви можете вказати координати маркера, іконку, заголовок та інші властивості. Вони використовуються для позначення конкретних точок на карті;
- Polyline та Polygon: Компоненти Polyline та Polygon використовуються для малювання ліній і полігонів на карті відповідно. Вони призначені для створення візуальних елементів для відображення маршрутів, кордонів та інших об'єктів;
- Circle: Компонент Circle дозволяє створювати кола на карті з заданим радіусом та координатами центру. Вони часто використовуються для відображення областей покриття;
- MapView.Callout: Callout - це компонент, який можна вкладати в Marker і використовувати для відображення додаткової інформації або взаємодії з маркером після натискання на нього;
- MapView.Polygon, MapView.Polyline і MapView.Circle: Класи Polygon, Polyline і Circle є розширеними версіями компонентів, що відповідають за малювання полігонів, ліній та кол;
- Map Events: "react-native-maps" також надає можливість обробляти події на карті, такі як натискання на маркери, перетягування карти і т. д. Ви можете підписатися на ці події та виконувати необхідні дії відповідно до подій;
- Map Styling: Бібліотека дозволяє налаштовувати стиль карт, включаючи колір фону, стиль маркерів та інші властивості.

Взаємодія між цими компонентами відбувається шляхом вкладання їх один в одного. Наприклад, Marker може бути доданий в MapView, а Callout - до Marker, щоб відобразити інформацію про маркер після натискання на нього. Події можна обробляти на рівні компонентів, таких як Marker, або на рівні MapView для

обробки подій на рівні карти. Архітектурний вигляд системи зображено на рисунку 2.3.

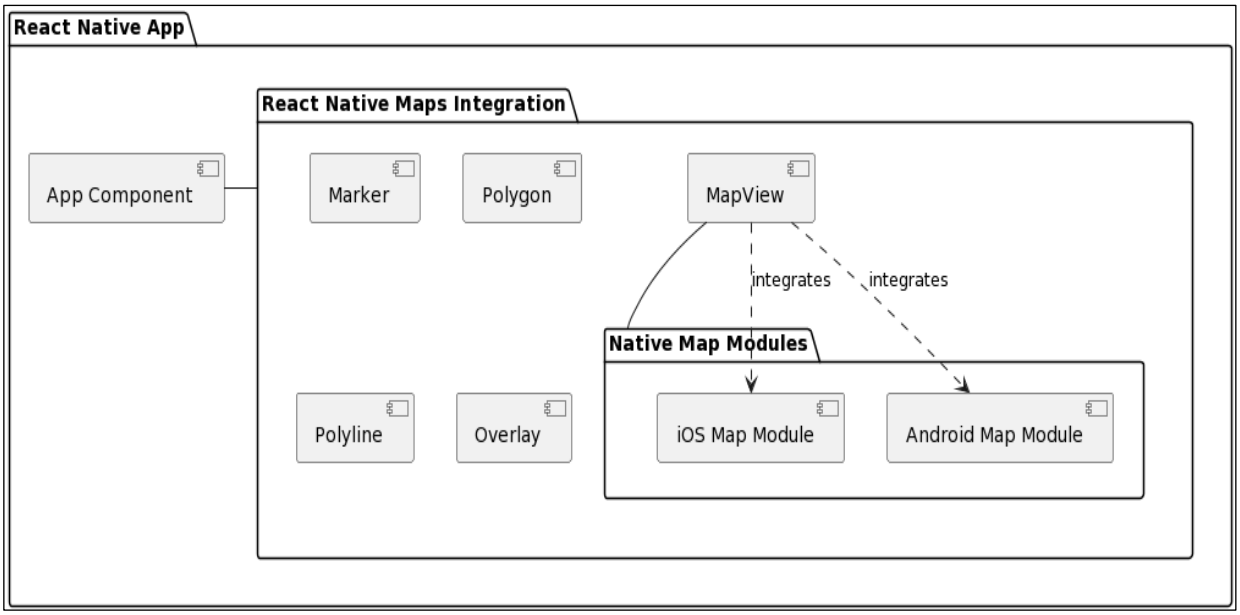


Рисунок 2.3 – Архітектура React-native-maps

Загалом, ця архітектура дозволяє легко створювати і взаємодіяти з картографічними елементами у React Native додатках.

3 ДОСЛІДЖЕННЯ ОПТИМАЛЬНОГО ПІДХОДУ ІНТЕГРАЦІЇ

Для більшого розуміння роботи обох підходів – нам необхідно більш детально розібрати їх, разом із цим порівнюючи паралельно ці підходи. Нам необхідно встановити симптоматичні та функціональні характеристики, патерни роботи та специфічні нюанси кожного з варіантів інтеграції картографічного функціоналу.

Варто розуміти що кожен з цих підходів має свої специфічні особливості, які впливають на вибір певного підходу, у певній ситуації.

У рамках порівняння архітектурних підходів, варто зосередитись на виявленні їх сильних та слабких сторін. Це включає оцінку таких аспектів, як продуктивність, масштабованість, легкість інтеграції та загальна гнучкість розробки та модифікації.

По закінченню аналізу буде виявлено, які архітектурні рішення найкраще відповідають конкретним потребам проекту, за певних умов та сценаріїв.

Перед цим потрібно проаналізувати – наскільки кожен з підходів може задовольнити технічним вимогам проекту. Це включає оцінку сумісності з різними архітектурними сценаріями, можливість інтеграції з іншими компонентами системи, гнучкість для подальшої модифікації, складності розробки та функціональність кінцевого результату.

3.1 Аналіз інтеграційних та архітектурних особливостей

Гнучкість розробки в мобільних додатках, використовуючи наші два підходи, може бути розглянута через три основні аспекти: модифікація, масштабованість та інтеграція. Кожен з цих аспектів має свої особливості у кожному з двох підходів.

Модифікація Веб-В'ю підходу:

- легкість оновлення: модифікація вмісту або дизайну через веб-в'ю може відбуватися швидко, оскільки зміни вносяться безпосередньо на веб-

сервері. Це забезпечує миттєве оновлення без потреби в оновленні самого мобільного додатку;

– гнучкість дизайну: використання html/css для інтерфейсу у веб-в'ю дозволяє легко адаптувати дизайн, реагуючи на змінювані тренди або вимоги користувачів.

Модифікація SDK підходу:

– комплексність оновлення: модифікація через sdk вимагає внесення змін у нативний код, що може бути більш об’ємним по часу та складним процесом. Оновлення часто вимагають перекомпіляції та розгортання нової версії додатку;

– обмеження у дизайні: зміни дизайну можуть вимагати глибоких технічних знань та роботи з нативними інструментами, що знижує швидкість внесення змін.

Далі розглянемо процес модифікації застосунку обома способами інтеграції. Змоделюємо сценарій модифікації на розробницькому та архітектурному рівні.

На рисунку 3.1. – зображено діаграму сценарію процесу модифікації певного функціоналу використовуючи підхід розробки та інтеграції SDK.

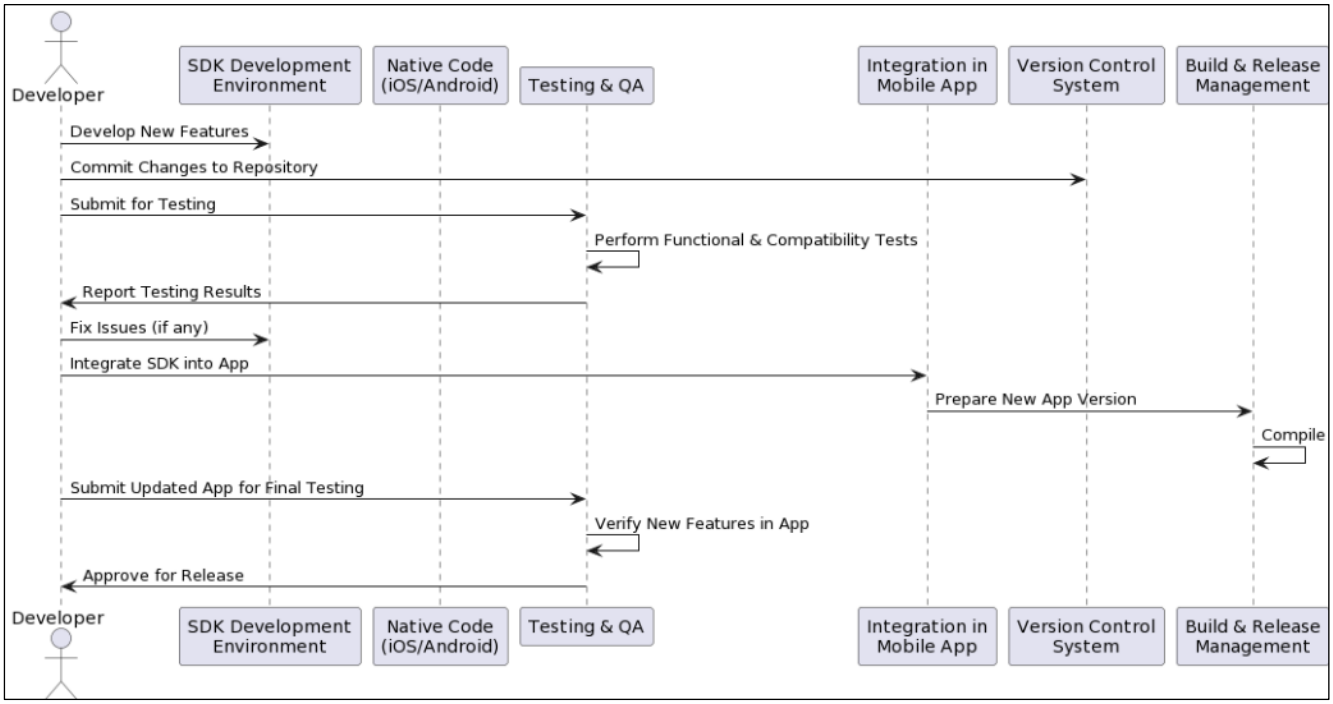


Рисунок 3.1. – Діаграма додавання функціоналу з використанням SDK

Цей процес включає наступні етапи:

- розробка нових функцій у sdk: розробник працює в середовищі розробки sdk, створюючи або оновлюючи функції;
- внесення змін у систему контролю версій: всі зміни комітять до репозиторію;
- тестування та qa: sdk проходить ряд тестів, включаючи функціональні та тести сумісності;
- виправлення проблем: будь-які знайдені під час тестування проблеми потребують виправлення у коді sdk;
- інтеграція sdk у мобільний додаток: sdk інтегрується з кодом мобільного додатку;
- підготовка нової версії додатку: виконується збірка та компіляція нової версії додатку;
- фінальне тестування оновленого додатку: оновлений додаток проходить фінальне тестування;
- схвалення для випуску: після успішного тестування додаток готовий до випуску.

Загалом можна сказати що розробка та інтеграція SDK вимагає глибоких знань нативних мов програмування для кожної платформи (наприклад, Swift для iOS, Kotlin/Java для Android) та потребує більше часу на реалізацію та дублювання функціоналу, що є більш складним у порівнянні з веб-технологіями.

Також SDK потребує ретельного тестування на різних пристроях і версіях операційних систем, що збільшує складність і час, необхідний для розробки.

У випадку використання веб-в'ю підходу, більшість цих складнощів відсутня. Оновлення вмісту відбувається на сервері, і ці зміни автоматично відображаються в мобільному додатку через веб-в'ю без необхідності роботи з нативним кодом або компіляції додатку. Це значно спрощує процес розробки та скорочує час від ідеї до реалізації функціоналу.

Далі розглянемо процес модифікації через веб-в’ю метод. Змоделюємо сценарій модифікації на розробницькому та архітектурному рівні.

На рисунку 3.2 – зображено діаграму сценарію процесу модифікації певного функціоналу використовуючи підхід веб-в’ю.

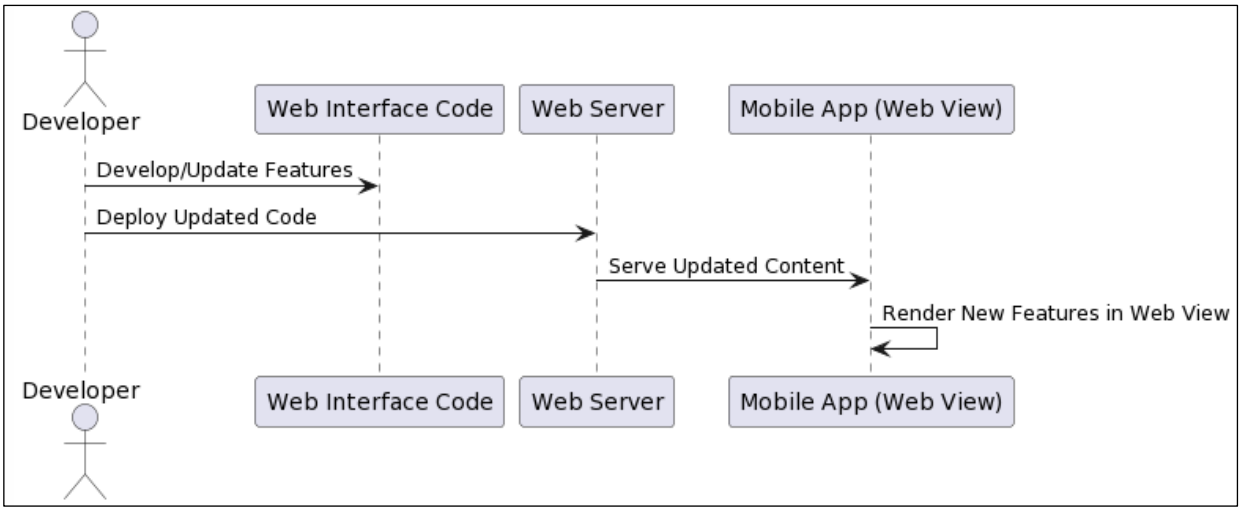


Рисунок 3.2 – Діаграма додавання функціоналу з використанням веб-в’ю

Цей процес включає наступні кроки:

- розробка та оновлення веб-інтерфейсу: розробник працює з веб-кодом (html/css/javascript), додаючи або модифікуючи функції;
- розгортання оновленого коду: розробник розміщує оновлений код на веб-сервері;
- сервер подає оновлений вміст: коли мобільний додаток завантажує вміст через веб-в'ю, сервер надає оновлену версію;
- відображення нових функцій у веб-в'ю: мобільний додаток відображає нові функції, які були додані або оновлені на веб-сторінці.

Таким чином, розробка та інтеграція власного SDK вимагає значно більше ресурсів, часу та технічної експертизи, у порівнянні з використанням веб-в'ю підходу. Хоча власний SDK пропонує більше можливостей для кастомізації та оптимізації під конкретні потреби, веб-в'ю є більш простим і швидким способом для впровадження та оновлення функціоналу мобільного додатку.

Далі необхідно проаналізувати та порівняти масштабованість двох підходів.

Масштабованість веб-в'ю підходу:

- обробка великих даних: використовуючи серверні ресурси, веб-в'ю дозволяє ефективно обробляти великі обсяги даних, не перевантажуючи мобільний додаток;
- адаптація до зростання користувачів: зі збільшенням кількості користувачів, сервер може обробляти підвищене навантаження, тоді як мобільний додаток продовжує функціонувати стабільно.

Масштабованість sdk підходу:

- складність масштабування: розширення функціоналу через sdk може бути більш складним, оскільки вимагає розширення нативного коду та забезпечення його сумісності з різними пристроями та платформами;
- масштабування через sdk вимагає додаткової оптимізації та тестування для забезпечення стабільної роботи на різних пристроях і під різними умовами.

Далі розглянемо аспект інтеграції системи по аналізу інтеграційних сценаріїв які були досліджені раніше.

Якщо не брати до уваги продуктивність в перспективі – можемо зробити висновок що метод інтеграції через веб-в’ю буде набагато простішим, адже використання веб-технологій (HTML, CSS, JavaScript) для створення картографічного інтерфейсу є простішим і швидшим. Це знижує поріг входження для розробників, які не обов'язково мають глибокі знання в нативному програмуванні для мобільних платформ.

Веб-в'ю підхід дозволяє створювати однаковий користувацький інтерфейс для різних мобільних платформ. Це забезпечує консистентність вигляду та функціональності незалежно від платформи.

Також оновлення та зміни можуть бути швидко впроваджені на сервері, що означає, що новий вміст або функції можуть бути доступні користувачам без необхідності оновлення самого додатку. Веб-в'ю підхід дозволяє масштабувати ресурси більш ефективно, оскільки обробка даних здійснюється на сервері, зменшуючи навантаження на мобільний пристрій.

У цілому, веб-в'ю підхід забезпечує велику гнучкість, простоту та ефективність при інтеграції картографічних систем та не тільки, роблячи його доволі пріоритетним у порівнянні з іншими.

3.2 Виявлення функціональних обмежень підходів

Після проведеного аналізу архітектури та інтеграції веб-в’ю підходу, нами було виявлено наступні, потенційні функціональні та експлуатаційні проблеми:

- веб-в'ю, використовуване в React Native для інтеграції SDK картографічного сервісу, може бути менш продуктивним порівняно з нативними компонентами. Це пов'язано з тим, що веб-в'ю запускає веб-сторінку всередині мобільного додатка, що вимагає додаткових ресурсів для рендеринга та виконання JavaScript;
- веб-в'ю може споживати більше пам'яті, оскільки воно створює додатковий веб-переглядач в контексті додатка;
- час відповіді в системі може бути повільнішим через необхідність завантаження веб-контенту та обробки JavaScript;
- може мати обмежений доступ до нативних функцій пристрою, таких як GPS, акселерометр, або жестове керування;
- існує потенційна затримка у відповіді при спробі доступу до цих функцій через веб-в'ю, що може вплинути на користувацький досвід;
- кастомізація інтерфейсу користувача в межах веб-в'ю може бути обмеженою, особливо якщо порівнювати з нативними компонентами, які пропонують ширші можливості кастомізації;
- інтерфейс, створений у веб-в'ю, може не ідеально відповідати дизайну та стилістиці мобільного додатку, що може вплинути на загальну естетику;
- якщо система використовується для доступу до онлайн-сервісів, можуть виникати труднощі з синхронізацією даних у режимі офлайн та їх оновленням при відновленні з'єднання з інтернетом[15];
- підтримка збереження поточного стану додатку при переході в офлайн та назад може бути обмежена або вимагати додаткових розробок;

- використання веб-в'ю часто вимагає більше ресурсів процесора для рендеринга веб-сторінок, що може впливати на загальну продуктивність додатку;
- графічні операції, такі як відображення карт і анімацій, можуть бути менш ефективними у веб-в'ю порівняно з нативними компонентами.

Обмеження підходу через розробку та інтеграцію SDK – не має особливого сенсу, адже очевидно що порівняно із веб-в’ю підходом, він має тільки переваги. Зважаючи на це – є сенс оцінити обмеження підходу розглядаючи його з точки зору альтернативи для невеликої команди розробників.

Отже, у такому випадку ми маємо наступні обмеження:

- технічна складність: розробка і налаштування компонента може бути технічно складною, особливо для команди з обмеженим досвідом у роботі з нативними компонентами React Native;
- обмежені ресурси: маленька команда може мати обмежені ресурси для підтримки та оновлення компонента, особливо якщо потрібно адаптувати його під різні платформи та версії ос;
- залежність від зовнішніх бібліотек: "react-native-maps" залежить від зовнішніх бібліотек і сервісів, які можуть вимагати додаткового вивчення та інтеграції;
- підтримка та оновлення: підтримка та оновлення бібліотеки можуть вимагати значних зусиль від маленької команди, особливо при виході нових версій React Native або зміні в API картографічних сервісів.

Далі варто класифікувати умовні функціональні потреби проектів з точки зору функціональних вимог, та порівняти підходи до них.

Якщо основна вимога – це просте відображення статичної карти без складних функцій, то "Веб-в'ю підхід" може бути зручним. Наприклад, додавання статичного місця розташування на карті або простого відображення маркерів та стислої інформації, також якщо потрібно швидко створити MVP і показати його користувачам, веб-в'ю підхід дозволить реалізувати карту на стартовому етапі проекту і зосередитися на інших аспектах розробки.

Якщо є потреба протестувати концепцію додатку та перевірити, чи користувачі зацікавлені у використанні карти, веб-в'ю підхід дозволить швидко імплементувати карту без додаткового налаштування.

Далі розглянемо функціональність для інтеграції SDK – якщо наш проект потребує інтерактивної карти з багатьма функціями, такими як додавання маркерів, створення маршрутів, відображення додаткових даних на карті, зміна кутів огляду, складні 3d маніпуляції.

Також варто відзначити повний контроль над виглядом та поведінкою карти, якщо ми хочемо максимально налаштувати вигляд та поведінку карти, змінюючи стилі, додавати анімації та впроваджувати складні функції, то кастомна інтеграція дає більший контроль.

4 ПРОВЕДЕННЯ ТЕСТУВАННЯ

4.1 Мета тестування

У нашому дослідженні ми застосовуємо комплексний підхід для оцінки продуктивності інтеграції картографічних систем у мобільні застосунки React Native, використовуючи два різні методи: веб-в'ю та нативний SDK. Наш вибір зосереджується на ключових метриках: розмір пакета, використання CPU та пам'яті, кадри на секунду (FPS), час відгуку та час запуску додатку.

Ці параметри важливі для оцінки як ефективності ресурсів, так і загальної продуктивності, що є критично важливими аспектами для мобільних застосунків, які покладаються на швидке та ефективне відображення картографічних даних.

Розмір пакета впливає на час завантаження та оновлення застосунку, особливо в умовах обмеженого доступу до мережі або повільних інтернет-з'єднань. Використання CPU та пам'яті відображає загальне навантаження на пристрій, що може вплинути на загальну продуктивність додатку та його здатність працювати паралельно з іншими процесами.

Фрейм-рейт (FPS) важливий для забезпечення плавності анімації та взаємодії, що є особливо важливим для застосунків, які включають інтерактивні карти.

Час відгуку та час запуску безпосередньо впливають на досвід користувача, адже швидкі та чутливі додатки створюють краще враження та підтримують високий рівень користувацької залученості.

Оцінюючи ці метрики на обох платформах – Android та iOS – ми можемо зробити узагальнені висновки щодо того, який підхід краще підходить для різних сценаріїв використання. Це забезпечить розробникам цінні дані для прийняття обґрунтованих рішень щодо технічної реалізації та дозволить оптимізувати продуктивність та якість майбутніх проектів.

Використовуючи ретельно підібрані метрики для оцінки двох підходів, ми сподіваємося надати глибокий аналіз їх ефективності, допомагаючи розробникам вибирати найкраще рішення для своїх потреб.

Наше дослідження приділяє особливу увагу збору даних для оцінки коефіцієнта, що дозволить оцінити рівень комфорту користувачів мобільних додатків з інтегрованими картографічними системами. Цей коефіцієнт важливий для розуміння, наскільки добре карта відповідає потребам користувачів і чи здатна вона забезпечити позитивний користувацький досвід у контексті специфічних вимог проекту.

Коефіцієнт головним чином буде базуватися на існуючих дослідженнях відносно зацікавленості та ступеню комфорту користувачів від характеристик кожного підходу, метрично ми це приведемо до відсотку користувачів який втратить гірший підхід[16].

Після вимірювання ключових метрик, наш наступний крок полягатиме в аналізі суб'єктивних сценаріїв використання систем, в яких інтегровані карти. Ми розглянемо різні варіанти використання карт у мобільних застосунках за обома підходами – веб-в'ю та нативний SDK – для встановлення, наскільки добре кожен підхід відповідає вимогам цих систем з точки зору швидкості та продуктивності. Цей аналіз допоможе визначити оптимальні умови для інтеграції карт, забезпечуючи високий рівень взаємодії з користувачем і загальну ефективність системи.

4.2 Вхідні дані

4.2.1 Засоби тестування

Bundle size – production build info: Ця метрика вимірює загальний розмір файлів додатку, коли вони готові до розгортання у продакшн. Важлива для розуміння впливу на час завантаження та використання простору на пристрої.

TTI (Час до інтерактивності) - Збірка в продакшні: TTI вимірює час, який потрібен додатку, щоб стати повністю інтерактивним. Зазвичай тестується на реальних пристроях чи емуляторах, як iOS або Android, за допомогою інструментів типу MacBook screen recorder для фіксації часу.

CPU, Пам'ять, FPS, RAM, UI/UX:

- React Native Performance Monitor: Цей інструмент використовується для моніторингу продуктивності застосунків React Native в реальному часі, дозволяє відстежувати CPU, пам'ять, FPS, та інші параметри;
- Xcode Profiler: Використовується для профілювання застосунків на iOS, дає можливість аналізувати використання CPU, пам'яті, а також оцінювати продуктивність інтерфейсу;
- Android Studio Profiler: Цей інструмент дозволяє детально проаналізувати продуктивність Android додатків, зокрема використання CPU, пам'яті, FPS, та відкликання UI.

У нашому дослідженні ми використаємо зазначені інструменти для оцінювання та порівняння продуктивності двох підходів до інтеграції картографічних систем у React Native додатки: веб-підходу та нативного підходу.

React Native Performance Monitor та Profiler tools з Xcode та Android Studio будуть застосовані для вимірювання ключових метрик продуктивності таких як використання CPU, споживання пам'яті (RAM), кількість кадрів на секунду (FPS), що дозволить нам оцінити відповідь UI та загальну плавність додатків.

MacBook screen recorder використовуватиметься для визначення показника TTI (час до інтерактивності) на емуляторах iOS та Android, що дозволить виявити, як швидко застосунки стають доступними для взаємодії з користувачем після запуску.

Ці інструменти допоможуть нам цілеспрямовано зібрати дані про ефективність веб та нативних підходів, що є критично важливим для вибору оптимального рішення для нашого проекту.

4.2.2 Метрики вимірювань

Нами буде визначено наступні метрики:

- Bundle size;
- FPS;
- CPU;
- RAM.

Далі на їх основі буде побудовано невелику метематичну модель для оцінки ефективності обох підходів до інтеграції карт.

Bundle Size – це метрика, яка вимірює розмір кінцевого пакета (bundle) мобільного додатка, який завантажується на пристрій користувача. Вона включає всі файли, необхідні для роботи додатка, такі як вихідний код, ресурси (зображення, шрифти), бібліотеки та інші залежності.

Ми використовуємо метрику bundle size з кількох причин:

- витрати на сервери: менший розмір додатка знижує витрати на серверні ресурси та передачу даних, оскільки менше даних потрібно передавати користувачам. це особливо важливо для хмарних сервісів, де вартість залежить від обсягу переданих даних;
- конверсійна ставка: збільшення розміру додатка може призвести до зниження кількості завантажень, оскільки користувачі можуть відмовитися від встановлення великих додатків. За дослідженням google, за кожне збільшення розміру додатка на 6 мб, конверсійна ставка знижується на 1%.

При кожному завантаженні додатка, дані передаються від сервера до пристрою користувача. Більший розмір додатка означає більше переданих даних, що призводить до вищих витрат на сервери. Ці витрати включають:

- пропускна здатність: плата за обсяг переданих даних;
- зберігання даних: плата за зберігання більших файлів на сервері.

Менші додатки зазвичай мають вищу конверсійну ставку через такі причини:

- час завантаження: користувачі віддають перевагу швидкому завантаженню додатків;
- місце на пристрої: додатки, які займають менше місця, більш привабливі для користувачів з обмеженою пам'яттю на пристроях.

Таким чином, аналізуючи bundle size, ми можемо оптимізувати витрати на сервери та покращити конверсійну ставку, що веде до більшого залучення користувачів та ефективного використання ресурсів.

FPS (Frames Per Second): FPS – це метрика, яка вимірює кількість кадрів, які відображаються на екрані за секунду. Високий FPS забезпечує плавність анімації та взаємодії користувача з додатком.

Для мобільних додатків оптимальним значенням вважається 60 FPS, але значення, яке нижче 30 FPS, може призвести до помітних затримок та погіршення користувацького досвіду.

CPU (Central Processing Unit): CPU – це метрика, яка вимірює завантаження центрального процесора додатком. Високе завантаження CPU може призвести до зниження продуктивності пристрою, швидшого розряду акумулятора та перегріву. Оптимізація використання CPU важлива для забезпечення ефективної роботи додатка та зменшення впливу на інші процеси.

RAM (Random Access Memory): RAM – це метрика, яка вимірює обсяг оперативної пам'яті, що використовується додатком. Використання RAM впливає на швидкість доступу до даних і загальну продуктивність додатка. Надмірне використання RAM може призвести до завершення роботи додатка або інших додатків, що працюють одночасно. Оптимізація використання RAM дозволяє забезпечити стабільну роботу додатка навіть на пристроях з обмеженою пам'яттю.

Час відкриття застосунку – це метрика, яка вимірює час, необхідний для запуску додатку від моменту натискання на іконку додатку до моменту, коли користувач бачить перший інтерактивний екран. Ця метрика є критично важливою для користувацького досвіду, оскільки довгий час відкриття може призвести до розчарування користувачів і потенційної відмови від використання додатку.

Ці метрики є критично важливими для аналізу продуктивності мобільних додатків і визначення областей, які потребують оптимізації для покращення користувацького досвіду.

4.3 Розробка математичної моделі метрик

Розрахунок ефективності Bundle Size, вплив розміру додатка на завантаження: згідно з дослідженнями, збільшення розміру додатка негативно

впливає на кількість завантажень. Наприклад, дослідження Twilio Segment показало, що збільшення розміру додатка з 3 МБ до 150 МБ знижує кількість завантажень на 66% (Segment). Інші дослідження підтверджують, що для кожного збільшення розміру додатка на 6 МБ конверсійна ставка знижується на 1% (Hashbrown).

Для розрахунку зниження конверсійної ставки (CR) через збільшення розміру додатка використовується наступна формула 4.1:

$$\triangle CK = \left(\frac{\triangle S}{6}\right) \times 1\% \tag{4.1}$$

де $\triangle CK$ – зміна конверсійної ставки;

$\triangle S$ – збільшення розміру додатка (в МБ).

Розрахунок ефективності FPS, дослідження та залежність: дослідження показують, що висока кількість кадрів в секунду (FPS) є критичною для забезпечення позитивного користувацького досвіду. Високий FPS (приблизно 60 кадрів на секунду) забезпечує гладку анімацію та взаємодію з додатком, що значно підвищує задоволеність користувачів. Низький FPS, особливо нижче 30 кадрів на секунду, може призвести до значного погіршення досвіду і відмови користувачів від використання додатку.

В дослідженнях встановлено, що є лінійна залежність між FPS і задоволеністю користувачів. Для кожного зниження FPS на 10 кадрів, рівень задоволеності користувачів знижується на 20-30%.

Для визначення впливу FPS на задоволеність користувачів можна використовувати наступну формулу 4.2:

$$\triangle S = \left(\frac{\triangle FPS}{10}\right) \times 20\% \tag{4.2}$$

де $\triangle S$ – зміна рівня задоволеності користувачів;

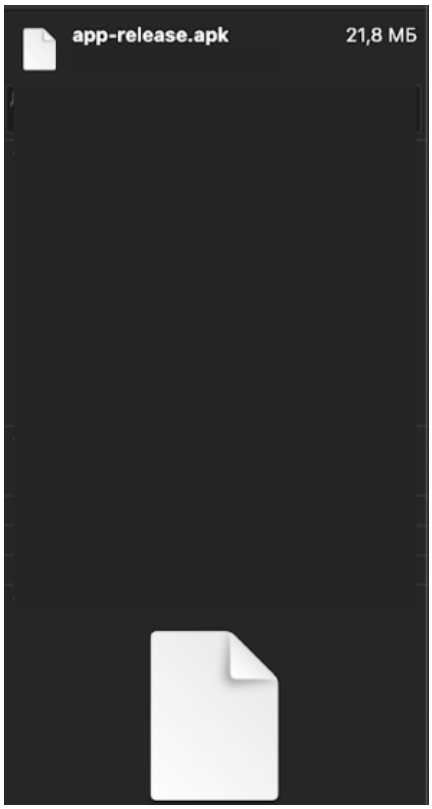
$\triangle FPS$ – зміна кількості кадрів в секунду.

4.4 Результати дослідження

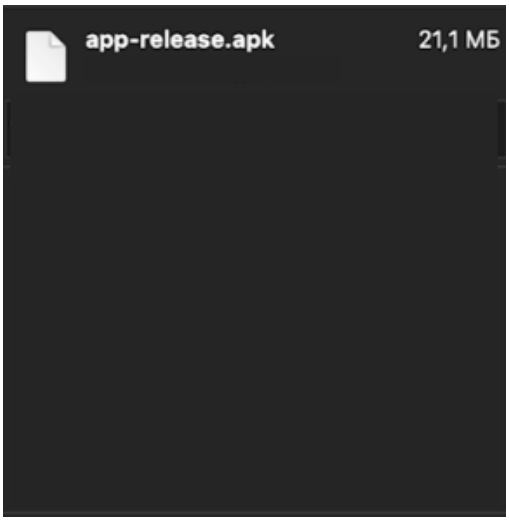
4.4.1 Результати Android

Далі проведемо експеримент із реалізацією web підходу на платформі Android та IOS, так само було реалізовано на нативному підходу.

Тож на рисунку 4.1, 4.2 наведено приклади bundle size замірів застосунку.

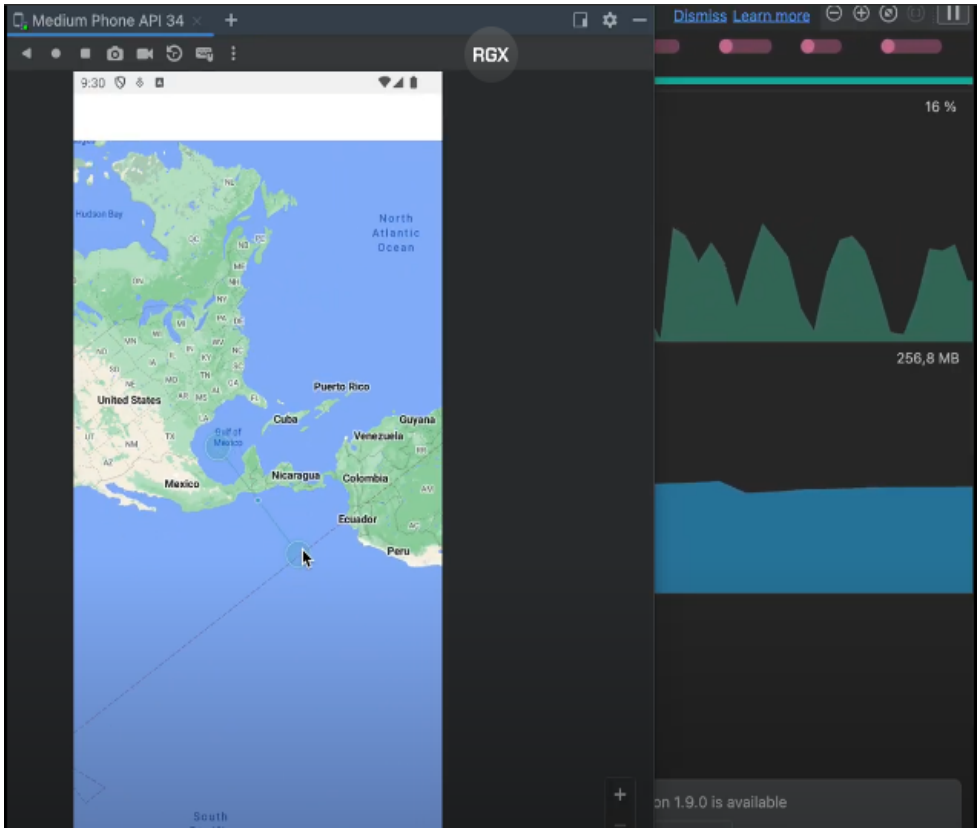


Рисунки 4.1 – bundle size у web

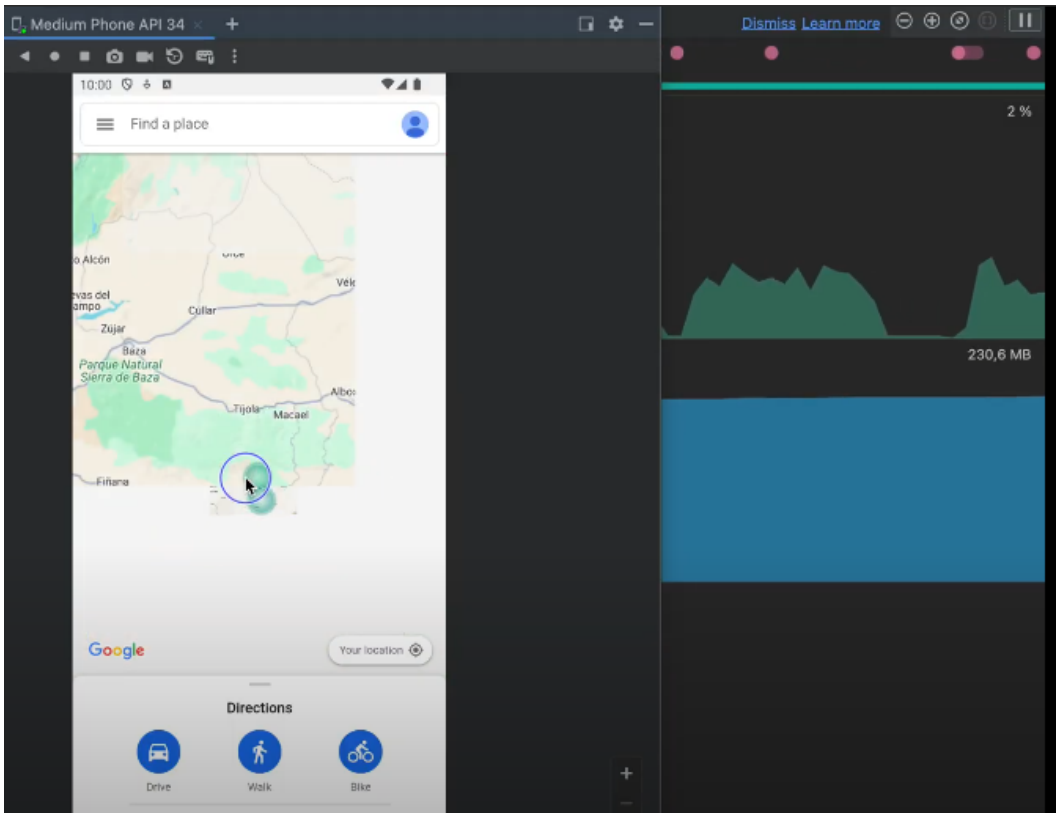


Рисунки 4.2 – bundle size у native

На рисунку 4.3, 4.4 наведено приклади CPU замірів застосунку.

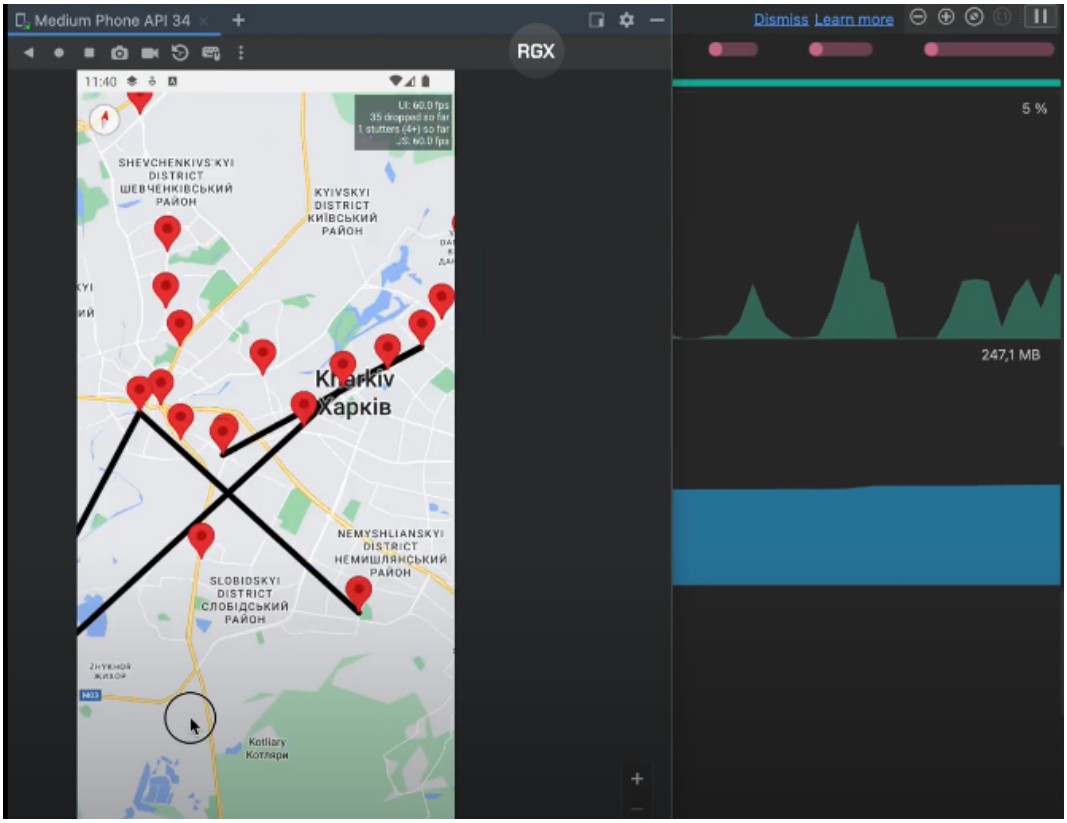


Рисунки 4.3 – CPU у web

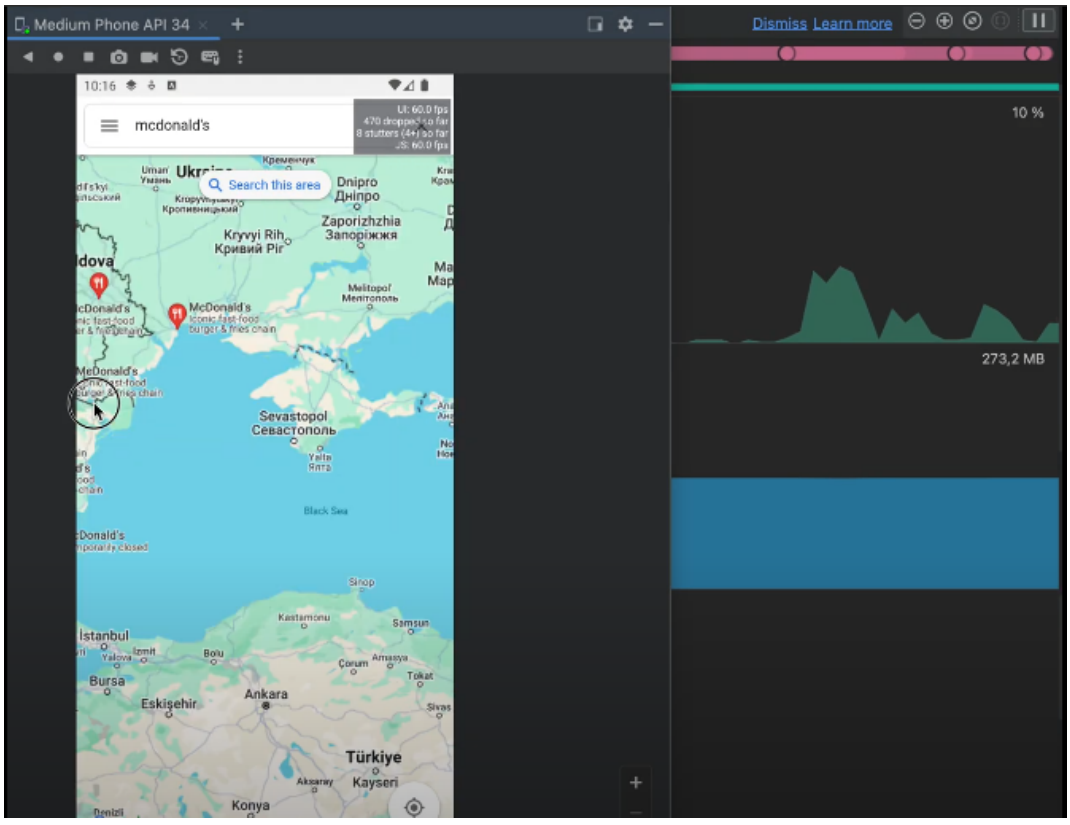


Рисунки 4.4 – CPU у native

На рисунку 4.5, 4.6 наведено приклади FPS замірів застосунку.



Рисунки 4.5 – FPS у web

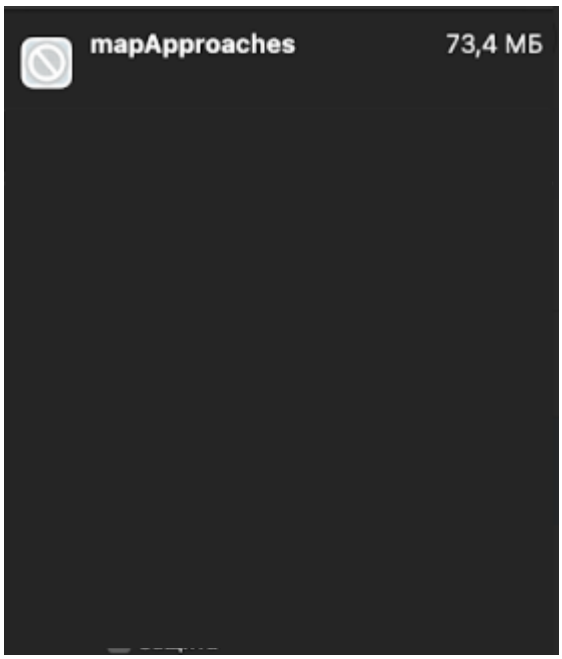


Рисунки 4.6 – FPS у native

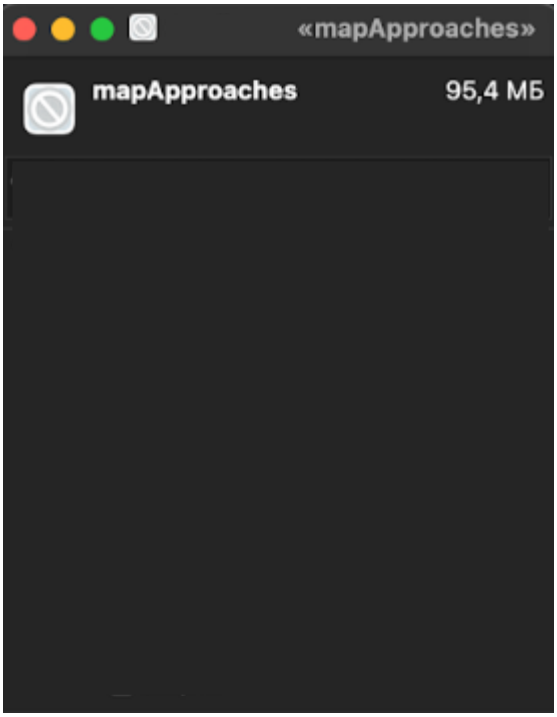
4.4.2 Результати IOS

Далі проведемо експеримент із реалізацією web підходу на платформі Android та IOS, так само було реалізовано на нативному підході.

Тож на рисунку 4.7, 4.8 наведено приклади bundle size замірів застосунку.

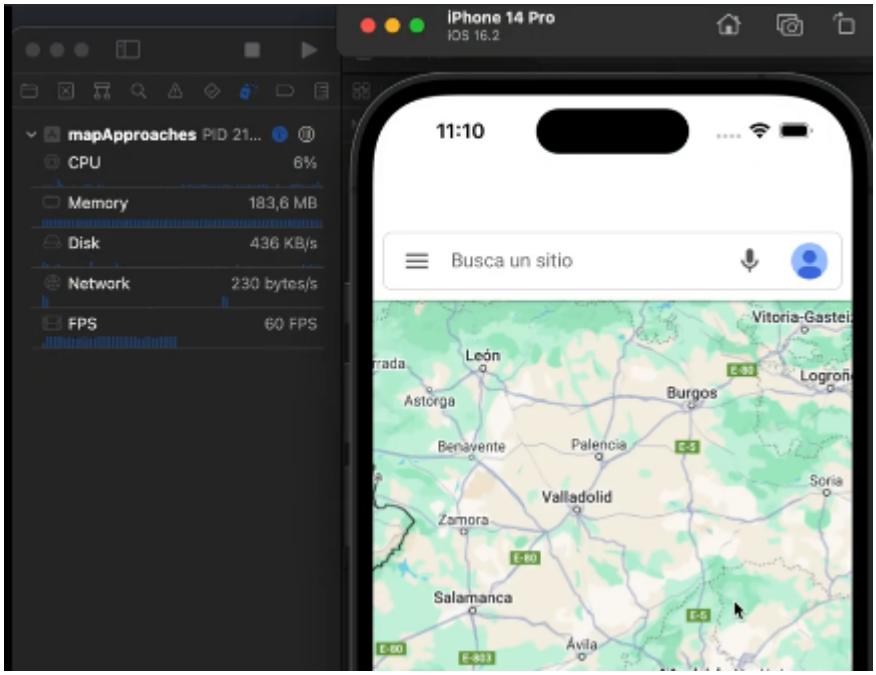


Рисунки 4.7 – bundle size у web

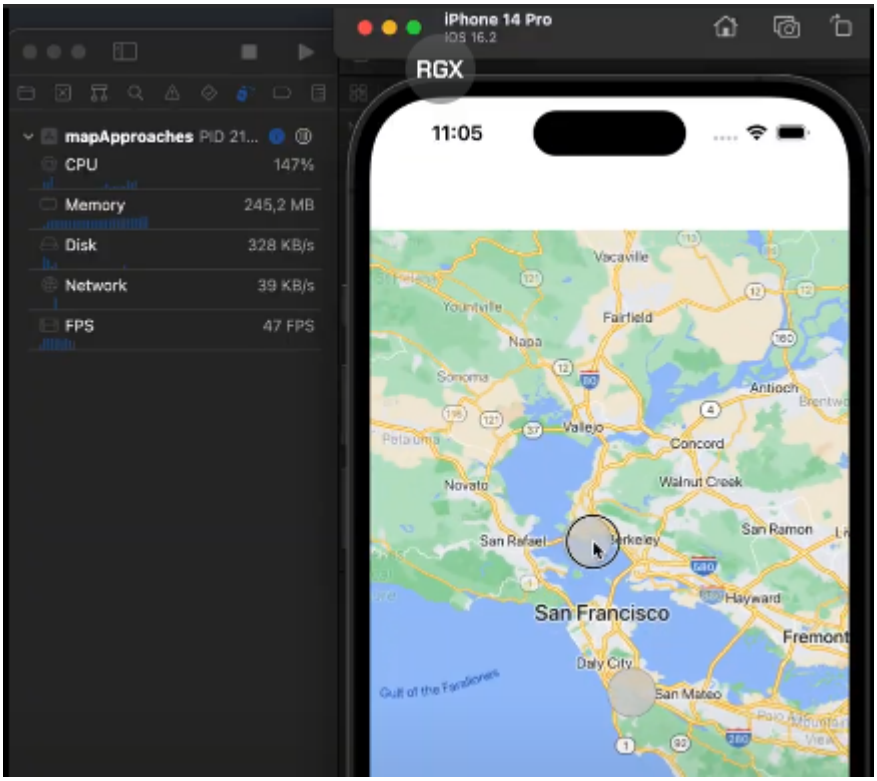


Рисунки 4.8 – bundle size у native

На рисунку 4.9, 4.10 наведено приклади CPU замірів застосунку.

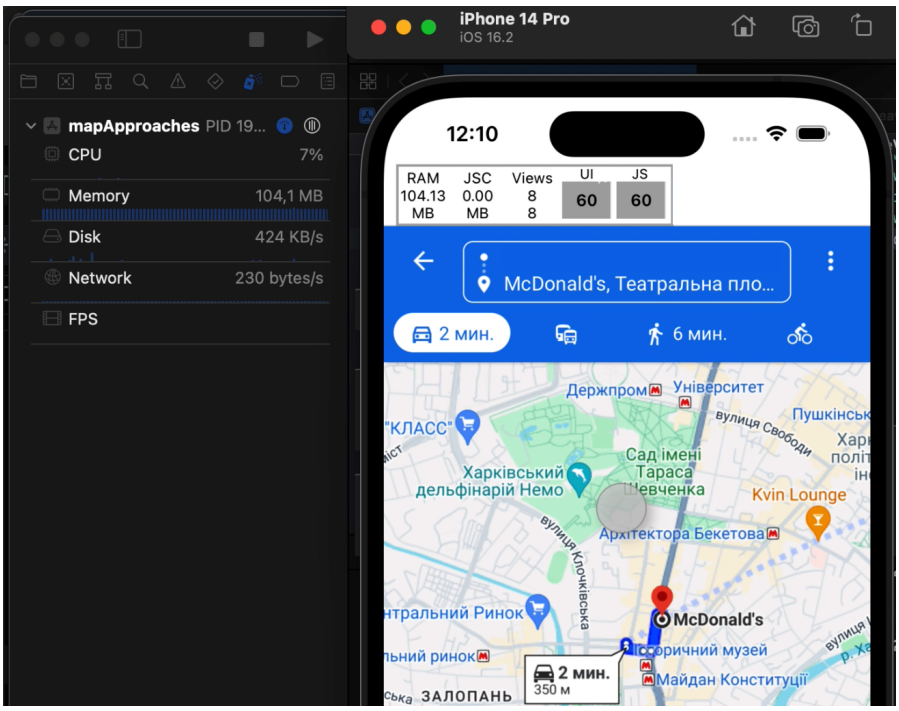


Рисунки 4.9 – CPU у web

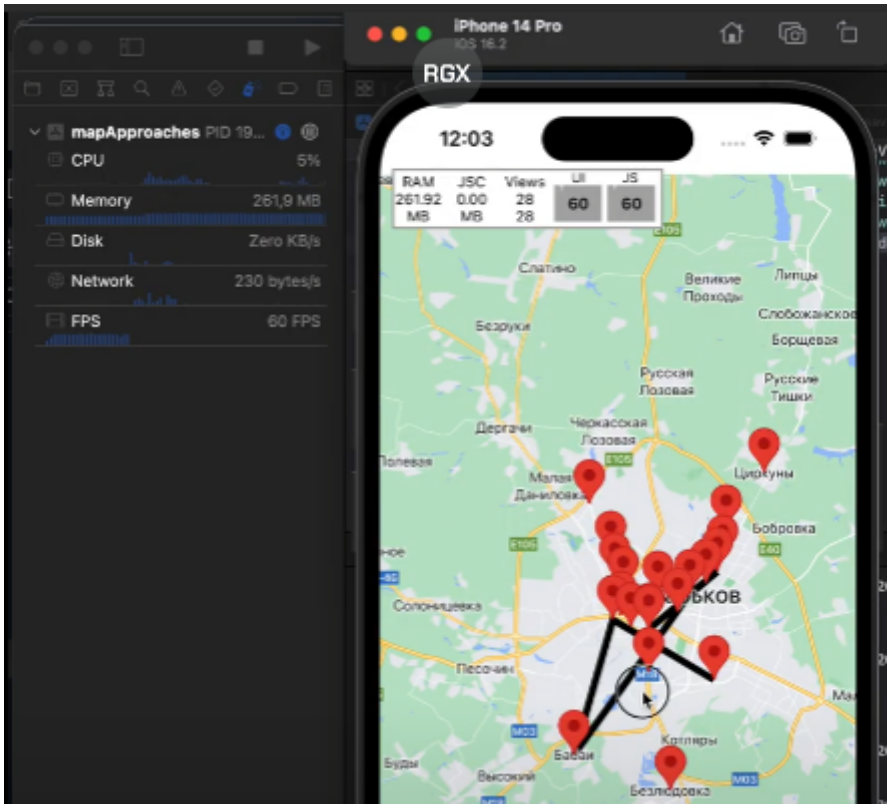


Рисунки 4.10 – CPU у native

На рисунку 4.11, 4.12 наведено приклади FPS замірів застосунку.



Рисунки 4.11 – FPS у web



Рисунки 4.12 – FPS у native

5 АНАЛІЗ РЕЗУЛЬТАТІВ

5.1 Аналіз даних

Далі наведемо таблицю 5.1, в яку зберемо усі отримані раніше дані із проведеного експерименту. Варто зазначити що метрики лінійного характеру були пораховані та зведені до медіанного значення імперичним методом.

Таблиця 5.1 – Дані метрик

Платформа	Метрика	Веб підхід	Нативний підхід
Android	Bandle size	21.1 МБ	21.8 МБ
	FPS	59 FPS	55 FPS
	CPU	18 % завантаженості	40 % завантаженості
	Холодний старт	4.5 с.	4.2 с.
IOS	Bandle size	72 МБ	98 МБ
	FPS	60 FPS	45 FPS
	CPU	10 % завантаженості	35 % завантаженості
	Холодний старт	3.8 с.	1.9 с.

Веб-підхід:

- переваги: менший розмір додатку, кращий fps, менше навантаження на cpu;
- недоліки: більший час холодного старту.

Нативний підхід:

- переваги: швидший холодний старт;
- недоліки: більший розмір додатку, нижчий fps, більше навантаження на cpu.

Загальні висновки експерименту:

- нативний підхід трохи швидше завантажується через більш тісну інтеграцію з операційною системою та менше залежність від завантаження зовнішніх ресурсів;
- веб-підхід має значно менший розмір, що пов'язано з тим, що багато ресурсів завантажуються динамічно з веб-серверів, а не включаються в інсталяційний пакет;
- веб-підхід забезпечує кращий fps, що може свідчити про кращу оптимізацію для рендерингу на веб-платформах, однак нативні додатки можуть мати більше процесорне навантаження через складніші анімації або більше фонових задач, також можливо що інструменти для замірів не можуть правильно обрахувати метрики через блокування web-view;
- веб-підхід значно менш навантажує CPU, оскільки використовує оптимізований браузерний рендеринг, тоді як нативні додатки можуть виконувати більше обчислень локально.

Також варто відмітити що платформа iOS виявилася оптимізованішою, ніж Android, з меншим часом холодного старту та меншою завантаженістю CPU у нативному підході, що може свідчити про кращу оптимізацію операційної системи та середовища розробки для нативних додатків на iOS.

5.2 Розрахунок метрик

Далі проведено розрахунки використовуючи наші формули по зниженню конверсії від ваги додатку та за формулою для залежності від fps. Результати розрахунків наведено у таблиці 5.2.

Результати:

- Android, збільшення ваги додатку на 0.7 МБ може знизити конверсійну ставку приблизно на 0.12%;
- IOS, збільшення ваги додатку на 26 МБ може знизити конверсійну ставку приблизно на 4.33%;
- Android, зниження FPS на 4 кадри може знизити рівень задоволеності користувачів приблизно на 8%;

– IOS, зниження FPS на 15 кадрів може знизити рівень задоволеності користувачів приблизно на 30%.

Таблиця 5.2 Результати переваги нативного підходу

Платформа	Метрики	Результат втрати користувачів
Android	конверсія	0.12%
	FPS	8%
IOS	конверсія	4.33%
	FPS	30%

За результатами IOS варто зазначити що обравши метод нативного підходу – потенційних проблем із задоволеністю користувачів – буде більше, а саме порівняно із вед підходом потенційна втрата буде дорівнювати приблизно 34% користувачів, у порівнянні із Android це більше різниця адже на Android втрата буде приблизно 9%.

Загалом по нативному підходу за обома платформами він буде перемагати у порівнянні із веб підходом, але на Android це не буде особливо помітною різницею.

Підводячи загальний підсумок можна зазначити що для не великих проектів – дуже ефективно буде інтегрувати веб підхід, адже на його розробку буде затрачено менше людино-годин та він має меншу вагу та витрачає менше ресурсів мобільного пристрою що робить його гарним вибором для не менш продуктивних систем, натомість хоч нативний підхід і вимагає значно більших ресурсів розробки та витрачає більше ресурсів системи – він має більшу візуальну плавність інтерфейсу та більшу швидкість запуску, також він буде ефективним та більш гнучким у підтримці та супроводженні, натомість на потужних системах – продуктивність буде майже така ж сама як на веб підході.

ВИСНОВКИ

У цій дослідницькій роботі ми зосередили увагу на аналізі інтеграції картографічних систем у React Native застосунки з метою визначити рекомендації що до використання кожного з підходів. Основна мета дослідження полягала в детальному аналізі кожного з підходів та виявлення ключових аспектів які будуть впливати на вибір що до методів інтеграції.

Нами було проаналізовано предметну область, та варіанти застосування картографічних систем, потім нами була поставлена задача та розроблено стратегію для вибору того чи іншого підходу в залежності від потреб та ситуацій.

По результатах, можна зробити висновок що веб-в'ю підхід є більш підходящим для швидкої розробки та простих застосунків, особливо коли команда обмежена у ресурсах. Він дозволяє легко оновлювати контент, має гнучкість у дизайні та спрощує процес розробки. Проте, він може бути менш продуктивним і мати обмеження у використанні нативних функцій пристрою.

Натомість SDK підхід вимагає більше часу та технічної експертизи для розробки, але пропонує кращу продуктивність, більші можливості кастомізації та повну інтеграцію з навивними можливостями пристрою. Цей підхід краще підходить для складних додатків та команд з досвідом у роботі з нативними платформами.

Підводячи загальний підсумок можна зазначити що для не великих проектів – дуже ефективно буде інтегрувати веб підхід, адже на його розробку буде затрачено менше людино-годин та він має меншу вагу та витрачає менше ресурсів мобільного пристрою що робить його гарним вибором для не менш продуктивних систем, натомість хоч нативний підхід і вимагає значно більших ресурсів розробки та витрачає більше ресурсів системи – він має більшу візуальну плавність інтерфейсу та більшу швидкість запуску, також він буде ефективним та більш гнучким у підтримці та супроводженні, натомість на потужних системах – продуктивність буде майже така ж сама як на веб підході.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Аналіз предметної області SDK картографічних систем, URL: https://developers.kameleoon.com/feature-management-and-experimentation/web-sdks/react-js-sdk/ (дата звернення: 01.04.2024).

2. Аналіз існуючих рішень, URL:https://docs.mapbox.com/android/maps/guides/ (дата звернення: 01.04.2024).

3. Scientific forum: theory and practice of research, REVIEW OF APPROACHES TO INTEGRATING CARTOGRAPHIC USER INTERFACES INTO REACT NATIVE MOBILE APPLICATIONS., Bohdan K., Research supervisor., c.51, URL: https://previous.scientia.report/index.php/archive/issue/view/05.04.2024 (дата звернення: 01.04.2024)

4. Аналіз Веб-в'ю підходу, URL: https://reactnative.dev/docs/custom-webview-android (дата звернення: 01.04.2024).

5. Occhino T. React Native: Bringing modern web techniques to mobile. Engineering at Meta. URL: https://engineering.fb.com/2015/03/26/android/react-native-bringing-modern-web-techniques-to-mobile/ (дата звернення: 01.04.2024).

6. Varshneya K. The History of React Native: Facebook's Open Source App Development Framework. Techahead., URL:https://www.techaheadcorp.com/knowledge-center/history-of-react-native/ (date of access: 01.04.2024).

7. Sciandra L. The New React Native Architecture Explained: Part Four. Nearform Commerce. URL:https://commerce.nearform.com/blog/2019/lean-core-part-4/ (дата звернення: 01.04.2024).

8. Native Modules Intro. React Native Official Documentation. URL: https://reactnative.dev/docs/native-modulesintro (дата звернення: 01.04.2024).

9. iOS Native UI Components. React Native Official Documentation. URL: https://reactnative.dev/docs/nativecomponents-ios (дата звернення: 01.04.2024).

10. Аналіз імплементації SDK, URL: https://medium.com/@michael.avoyan/a-comprehensive-guide-of-mobile-sdk-implementation-ca0543ba3d23 (дата звернення: 01.04.2024).

11. Аналіз архітектурної імплементації SDK, URL: https://developers.kameleoon.com/feature-management-and-experimentation/web-sdks/react-js-sdk/ (дата звернення: 01.04.2024).

12. Дослідження аналізу впливу ваги застосунків // Why Does App Size Matter?, Vikram Thakur, 2023 р. , URL: https://hashbrown.com/blog/product-design-application/why-does-app-size-matter (дата звернення: 01.04.2024).

13. Дослідження аналізу впливу фрейм-рейту на задоволеність користувачів, URL: https://www.emerald.com/insight/ (дата звернення: 05.05.2024).

14. Falatiuk H., Shirokopetleva M., Dudar Z. Investigation of Architecture and Technology Stack for e-Archive System. 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), Kyiv,Ukraine,8-11October2019.2019.,URL: https://doi.org/10.1109/picst47496.2019.9061407 (дата звернення: 05.05.2024).

15. DATA EXCHANGE MODEL IN THE INTERNET OF THINGS CONCEPT / I. Afanasieva et al. Telecommunications and Radio Engineering. 2019. Vol. 78,no. 10.P. 869–878.
URL: https://doi.org/10.1615/telecomradeng.v78.i10.30 (date of access: 05.05.2024).

16. Smelyakov S. V., Stoyan Y. G. Modelling of the space of paths in problems of constructing optimal trajectories. *USSR Computational Mathematics and Mathematical Physics*. 1983. Vol. 23, no. 1. P. 50–55. URL: https://doi.org/10.1016/s0041-5553(83)80009-3 (дата звернення: 05.05.2024).