МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему **Особливості застосування фреймворків Flutter і React Native для створення мобільних застосунків**

Виконав: студент 2 курсу, групи 8.1212-пзс-2-дн
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
Макушин Д. В.
(ініціали та прізвище)
Керівник доцент, к.ф.-м.н. Коломоєць Г. П.
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра __електроніки, інформаційних систем та програмного__ забезпечення
Рівень вищої освіти _____другий (магістерський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Інженерія програмного забезпечення
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри __Т.В. Критська_____
"_01_" __жовтня_____2023_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Макушину Дмитру Володимировичу
(прізвище, ім'я, по батькові)

1.Тема роботи __Особливості застосування фреймворків Flutter і React Native для створення мобільних застосунків_____

керівник роботи ___доцент, к.ф.-м.н. Коломоєць Геннадій Павлович_____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від _09.10. 2023 р. №1577-с__
2. Строк подання студентом кваліфікаційної роботи_____01.03.2024_____
3. Вихідні дані магістерської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження є порівняння можливостей, які надають фреймворки Flutter та React Native для кросплатформної розробки мобільних застосунків;
- створення програмного продукту та його опис;
- перелік вимог для роботи програми;
- дослідження поставленої проблеми та розробка висновків та пропозицій.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
_____слайдів презентації

6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.10.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Переваги та недоліки нативного та кросплатформного підходу до розробки мобільних застосунків. Порівняння фреймворків кросплатформної розробки.	01.12.2022	виконано
2	Порівняльний огляд мов програмування. Засоби фреймворків для розробки кросплатформних мобільних застосунків.	01.02.2023	виконано
3	Опис функціональних вимог застосунку Опис розгортання середовищ розробки та створеня проектів та приклади коду реалізації. Опис розгортання застосунку.	20.06.2023	виконано
4	Реалізація повної функціональності застосунку засобами Flutter та React Native. Аналіз процесів розробки, формулювання результатів порівняльного дослідження.	01.12.2023	виконано
5	Оформлення дипломної роботи	15.01.24	виконано
6	Попередній захист	26.02.24	виконано
7	Захист дипломної роботи	12.03.2024	виконано

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ Коломоєць Г. П.
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ Скрипник І.А,
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок: 106

Рисунків: 23

Таблиць: 1

Джерел: 31

Макушин Д.В. Особливості застосування фреймворків Flutter і React Native для створення мобільних застосунків кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник Г. П. Коломоєць. Запоріжжя : ЗНУ, 2024. 106 с.

Метою дослідження є порівняння можливостей та дослідження особливостей використання фреймворків Flutter та React Native, а також визначення рекомендацій щодо використання фреймворків для розробки мобільних застосунків залежно від вимог до їх функціональності.

Під час дослідження розглянув проблему вибору кросплатформної розробки мобільних застосунків. Для цього створив застосунки для двох популярних операційних систем, iOS та Android, на базі фреймворків Flutter та React Native. Провів порівняльний аналіз роботи базових компонентів, використання датчиків, навігації між екранами, обробки введення користувача (включаючи жести), роботи з графікою та анімацією, доступу до нативних UI компонентів, використання сторонніх бібліотек і таке інше. Також здійснив порівняльний огляд мов програмування Dart та JavaScript (TypeScript).

На основі результатів дослідження розробив рекомендації щодо використання фреймворків для розробки мобільних застосунків, враховуючи вимоги до їх функціональності.

Ключові слова: кросплатформна розробка, React Native, Flutter, фреймворки мобільних застосунків, нативні компоненти, особливості розробки мобільних застосунків, мови програмування (Dart, JavaScript), навігація у застосунках.

SUMMARY

Pages: 106

Figures: 23

Tables: 1

Sources: 31

Makushyn D.V. Features of using Flutter and React Native frameworks for mobile applications creating: qualification work of the master of specialty 121 "Software Engineering" / Science head H.P. Kolomoiets. Zaporizhzhia: ZNU, 2024. 106p

The aim of the research is to compare the capabilities and explore the features of using Flutter and React Native frameworks, as well as to determine recommendations for using frameworks for mobile application development based on their functionality requirements.

During the research, I examined the problem of choosing cross-platform mobile application development. For this purpose, I created applications for two popular operating systems, iOS and Android, based on the Flutter and React Native frameworks. I conducted a comparative analysis of the performance of basic components, sensor usage, screen navigation, user input handling (including gestures), graphics and animation handling, access to native UI components, and the use of third-party libraries, among others. Additionally, I conducted a comparative review of the Dart and JavaScript (TypeScript) programming languages.

Based on the research findings, I developed recommendations for using frameworks for mobile application development, taking into account their functionality requirements.

Keywords: cross-platform development, React Native, Flutter, mobile application frameworks, native components, mobile application development features, programming languages (Dart, JavaScript), navigation in applications.

ЗМІСТ

ВСТУП

Актуальність теми

Кількість мобільних застосунків з року в рік зростає із зростанням кількості користувачів смартфонів та планшетів. У зв'язку з цим, з'являється необхідність в пошуку швидких та ефективних способів розробки мобільних застосунків.

Однак розробка мобільних застосунків для смартфонів та планшетів до 2010 року була можлива лише на різних мовах програмування. Наприклад, для створення застосунку для Android треба було використовувати мову Java, а для iOS використовували Objective-C або Swift.

Перший кросплатформний фреймворк з'явився у 2009 році це була Apache Cordova [5], а потім з'явилися інші.

Apache Cordova — це відкрите програмне забезпечення, яке надає засоби для розробки мобільних застосунків та використанням веб-технологій, таких як HTML, CSS і JavaScript. Основним принципом роботи Cordova є упакування веб-застосунку в нативні обгортки для різних мобільних платформ, що дозволяє використовувати єдиний код для розробки на різних пристроях [5].

Завдяки Apache Cordova розробники можуть створювати кросплатформні мобільні застосунки, які можуть працювати як на Android, так і на iOS, використовуючи звичайні веб-технології. Це спрощує розробку, оскільки забезпечує використання єдиного коду для різних платформ, зменшуючи зусилля та час, потрібний для створення та підтримки мобільних застосунків.

Flutter (офіційний реліз 2017) [1] та React Native (офіційний реліз 2015) [2] — це два з найбільш популярних фреймворків, що використовуються для розробки мобільних застосунків. Вони дозволяють створювати високоякісні мобільні застосунки з використанням однієї кодової бази, що зменшує час та затрати на розробку. В той же час, вони мають свої особливості та різницю у функціональності, що може вплинути на вибір одного з них для конкретного проекту.

Фреймворк Flutter розроблений Google [4], а фреймворк React Native — Facebook [3], тому ці технологічні гіганти мають дуже великі команди, які займаються усіма напрямками — від розробки SDK платформи до його документування, підтримки тощо.

Переваги кросплатформної розробки мобільних застосунків:

- одноразовий код: можливість використовувати єдиний кодовий базис для створення застосунків для різних платформ (iOS, Android, тощо);
- ефективність витрат: зменшення витрат часу і ресурсів, оскільки необхідно розробляти лише один код для всіх платформ;
- одноразове тестування: можна провести тестування одного кодової бази, що значно зменшує обсяг ручного тестування і сприяє виявленню помилок;
- швидкість виходу на ринок: швидше випускання продукту завдяки відсутності необхідності окремої розробки для кожної платформи;
- загальна інтеграція: зручна можливість інтеграції з різними платформами та сервісами завдяки використанню єдиної кодової бази;
- спрощена підтримка: зменшення зусиль, пов'язаних з підтримкою та виправленням помилок, оскільки необхідно обслуговувати лише один код;
- рефакторинг і розширення: зручність рефакторингу та розширення функціональності для всіх платформ одночасно;
- спільні інструменти розробки: використання спільних інструментів розробки, таких як фреймворки та IDE, для уніфікованого процесу розробки;
- доступ до платформенних функцій: зручна інтеграція з платформенними функціями за допомогою кросплатформенних фреймворків та плагинів;
- різноманітність бібліотек та плагинів: велика кількість готових рішень, бібліотек та плагинів, які полегшують розробку та розширення функціональності;
- рівномірний UX: забезпечення рівномірного користувацького досвіду для користувачів на різних платформах;

– більша аудиторія: здатність залучити більше користувачів, оскільки застосунок доступний на різних платформах.

Мета дослідження

Метою дослідження є порівняння можливостей та дослідження особливостей використання фреймворків Flutter та React Native, а також визначення рекомендацій щодо використання фреймворків для розробки мобільних застосунків залежно від вимог до їх функціональності.

Для досягнення цієї мети необхідно було вирішити наступні завдання:

– розгортання середовища для Flutter та налаштування необхідних компонентів;
– розгортання середовища для React Native та налаштування необхідних компонентів;
– створення застосунків: розробка мобільних застосунків на базі фреймворків Flutter та React Native для платформ iOS та Android;
– вивчення основних компонентів: аналіз роботи базових компонентів (кнопок, текстових полів, списків тощо) фреймворків та їх порівняння;
– використання датчиків: тестування та порівняння можливостей використання датчиків пристрою (геолокації, акселерометру тощо);
– навігація між екранами: аналіз систем навігації та порівняння методів переміщення між екранами;
– обробка введення користувача: порівняння обробки введення користувача, включаючи жести та клавіатуру;
– робота з графікою та анімацією: дослідження можливостей роботи з графікою та створення анімацій;
– доступ до нативних компонентів: порівняння способів взаємодії та доступу до нативних компонентів пристрою;
– використання сторонніх бібліотек: оцінка можливостей використання сторонніх бібліотек для розширення функціональності застосунків;

- огляд мов програмування: порівняльний аналіз мов програмування, які використовуються у Flutter (Dart) та React Native (JavaScript/TypeScript);
- надання рекомендацій: заключні висновки та рекомендації щодо використання фреймворків у залежності від вимог до функціональності мобільних застосунків.

Об'єктом дослідження є процес кросплатформної розробки мобільних застосунків.

Предметом дослідження є фреймворки Flutter та React Native.

Методи дослідження

Методами дослідження можуть використовуватися наступні:

- розробка застосунків: створення мобільних застосунків на обох фреймворках для порівняльного дослідження їхніх можливостей та особливостей;
- аналіз створених застосунків: детальний розгляд та порівняння функціональних компонентів, виявлення сильних та слабких сторін;
- оцінка роботи зі сторонніми бібліотеками: вивчення можливостей використання та інтеграції сторонніх бібліотек для розширення функціональності;
- аналіз взаємодії із системними компонентами: дослідження взаємодії мобільних застосунків із системними компонентами пристроїв (камера, датчики, геолокація);
- аналіз програмного коду: детальний аналіз коду створених застосунків для вивчення структури, якості та зручності написання коду;
- огляд документації та спільноти: огляд документації фреймворків та активності їхніх спільнот для визначення рівня підтримки та розвитку;
- порівняльний аналіз мов програмування: вивчення та порівняння мов програмування, що використовуються у фреймворках;

- висновки та рекомендації: формулювання висновків та рекомендацій щодо використання фреймворків для конкретних випадків розробки мобільних застосунків.

Наукова новизна одержаних результатів

Наукова новизна одержаних в даному дослідженні результатів характеризується такими основними аспектами:

- аналіз вбудованих компонентів та нативних компонентів операційних систем Android та iOS;
- дослідження особливостей інтеграції Flutter та React Native з мобільними платформами (Java/Kotlin, Swift/Obj-C) та нативним кодом. Порівняння підходів і можливостей;
- аналіз та порівняння підтримки наприклад: картографічні сервіси, жести, біометрична автентифікація тощо;
- оцінка ресурсів і часу на розробку та підтримку однакових за функціоналом застосунків на Flutter і React Native для iOS та Android;
- розробка рекомендацій щодо використання фреймворків для конкретних випадків розробки мобільних застосунків.

Практичне значення одержаних результатів

На основі отриманих результатів надані обґрунтовані рекомендації щодо вибору між Flutter і React Native для конкретних сценаріїв розробки мобільних застосунків.

Для бізнесу результати дослідження будуть корисні для розуміння переваг та недоліків обраних фреймворків та обґрунтування вибору фреймворку для свого майбутнього проекту. Для розробників дослідження буде корисним прикладами використання програмних бібліотек та підходів при вирішенні типових задач кросплатформної розробки мобільних застосунків.

Апробація результатів

Результати дослідження були представлені на XVI науково-практичній конференції студентів, аспірантів, докторантів і молодих вчених Запорізького національного університету «Молода наука-2023» [13], а також на III Всеукраїнській науково-практичної конференції за участю молодих науковців «Актуальні питання сталого науково-технічного та соціально-економічного розвитку регіонів України»[18].

Глосарій

Фреймворк (Framework) — абстрактна платформа, що надає узагальнену структуру і набір основних функцій для спрощення розробки програмного забезпечення, зменшуючи необхідність написання повторюваного коду та надаючи інструменти для автоматизації рутинних завдань

UI, або "User Interface" (інтерфейс користувача) — все те, через що користувач взаємодіє з продуктом або службою. В комп'ютерних системах і програмах це може включати елементи, такі як кнопки, іконки, слайдери, поля вводу та все інше, що користувач може клацати, торкатися або взаємодіяти інакше. Дизайнери UI зосереджуються на естетиці та відчуттях від взаємодії з продуктом.

UX, або "User Experience" (досвід користувача) — описує всі аспекти взаємодії користувача з продуктом, службою або компанією. UX може включати не тільки цифровий інтерфейс, але і ширший досвід взаємодії користувача з продуктом або брендом. Дизайнери UX зосереджуються на зручності використання, логіці навігації, ефективності виконання завдань та загальному задоволенні користувача.

JavaScript — високорівнева, інтерпретована мова програмування, яка використовується для створення інтерактивних веб-сайтів. Вона може використовуватися як на стороні клієнта (браузер), так і на стороні сервера (за допомогою Node.js).

TypeScript — розширення мови JavaScript, яке додає статичні типи та інші функції до мови, що полегшує розробку великих та складних програм.

Функція (Function) — блок коду, який може виконуватися при виклику. Вона може приймати аргументи та повертати значення.

Інтерфейс програмування застосунків API (Application Programming Interface) — набір правил та інструкцій, які дозволяють різним програмам взаємодіяти одна з одною. У веб-розробці, API визначає спосіб комунікації між різними компонентами програмного забезпечення, такими як фронтенд та бекенд.

Stripe — компанія та платіжна платформа, яка дозволяє компаніям та розробникам приймати онлайн-платежі та керувати фінансами.

Інтерфейс (Interface) — точка взаємодії між користувачем та системою, яка може бути графічною (GUI) або командною (CLI). У контексті програмування, інтерфейс може також вказувати на визначені конструкції для спілкування між компонентами програми.

Модуль (Module) — самостійна частина програмного забезпечення, яка має певний функціонал та може бути використана незалежно від інших частин системи.

Бібліотека (Library) — набір функцій або ресурсів, які можуть бути використані для розробки програм або веб-сайтів. Бібліотеки полегшують розробку, оскільки реалізують певні функціональності, які можна використовувати безпосередньо.

Dart — об'єктно-орієнтована мова програмування, яка розроблена компанією Google. Вона використовується для розробки високопродуктивних веб-застосунків, а також є основною мовою програмування для фреймворку Flutter, який використовується для побудови кросплатформених мобільних та веб-застосунків. Dart має сучасний синтаксис, підтримує шаблони програмування зі змінними типами даних і включає в себе механізми управління пам'яттю та асинхронне програмування.

Watchman — інструмент для службового моніторингу файлів у реальному часі. Він надає можливість відстежувати зміни в файлах і каталогах, сповіщаючи про них розробників. Watchman зазвичай використовується в розробці

програмного забезпечення для автоматизації процесу перекомпіляції або перезапуску застосунків під час розробки у React Native.

Node.js — середовище виконання для JavaScript на сервері, яке дозволяє виконувати код JavaScript поза браузером.

Firebase — платформа для розробки мобільних та веб-застосунків, яка надає різноманітні сервіси та інструменти для роботи з базами даних, аутентифікацією, хостингом, збереженням файлів, аналітикою та іншими функціональностями. Вона володіє облачною архітектурою і дозволяє розробникам зосередитися на функціональності своїх застосунків, перекладаючи на Firebase деякі аспекти інфраструктури. Firebase належить до компанії Google та широко використовується для розробки застосунків в реальному часі та покращення їхньої продуктивності.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРОБЛЕМИ ВИБОРУ КРОСПЛАТФОРМНОГО ФРЕЙМВОРКУ ДЛЯ СВОРЕННЯ МОБІЛЬНИХ ЗАСТОСУНКІВ

1.1 Визначення проблеми вибору фреймворку

Вибір кросплатформного фреймворка для розробки мобільних застосунків - це важливе рішення, яке може суттєво вплинути на продуктивність розробки, вартість проекту, а також на якість і функціональність кінцевого продукту. З одного боку, використання кросплатформних фреймворків може привести до швидшої розробки та зниження витрат, оскільки пишеться код один раз і використовується для різних платформ (Android та iOS). З іншого боку, вибір неправильного фреймворка може призвести до поганих показників продуктивності, важкості в усуненні помилок і витрати більше часу на вирішення специфічних проблем [10].

При виборі фреймворка розробники повинні враховувати ряд важливих факторів, включаючи:

- продуктивність - як добре фреймворк може виконувати розроблені застосунки, особливо при високих навантаженнях;
- сумісність з нативним API - як легко фреймворк може інтегруватися з нативними функціями операційної системи;
- швидкість розробки - як швидко можна розробляти застосунки з використанням цього фреймворка;
- підтримка спільноти - чи активна спільнота розробників, яка може допомогти вирішувати проблеми і навчитися нових речей;
- доступність та якість документації - чи є достатньо інформації та ресурсів для навчання та вирішення проблем.

Виклики, пов'язані з вибором кросплатформного фреймворка, включають в себе необхідність вивчити новий фреймворк та його особливості, потенційні

проблеми з продуктивністю та сумісністю, а також ризик обрати фреймворк, який може стати застарілим та більше не підтримуватиметься розробниками.

1.2 Переваги та недоліки кросплатформної розробки

Переваги кросплатформної розробки мобільних застосунків [10]:

- економія часу та ресурсів - розробники можуть використовувати один кодовий базис для створення застосунків для різних платформ (наприклад: Android та iOS). Це може скоротити час розробки та зменшити витрати;
- узгодженість UI/UX - з використанням одного коду для різних платформ розробники можуть створити однорідний користувацький інтерфейс та досвід користування на різних платформах;
- швидше впровадження - кросплатформні фреймворки зазвичай пропонують швидку розробку, що дозволяє продуктам швидше вийти на ринок.

Недоліки кросплатформної розробки мобільних застосунків [10]:

- Обмежена доступність до нативних API - деякі фреймворки можуть не мати повноцінного доступу до нативних API, що може обмежити функціональність застосунку;
- Можливі проблеми з продуктивністю - хоча багато сучасних кросплатформенних фреймворків дуже ефективні, нативні застосунки можуть бути більш ефективними в термінах продуктивності, особливо для ресурсоємних операцій;
- Затримка доступі до нових функцій платформ - коли нові функції або API для конкретних платформ, кросплатформенним застосункам може знадобитися більше часу, щоб отримати доступ до цих нових функцій;
- Підтримка - хоча багато кросплатформених фреймворків мають велику спільноту розробників, нативні застосунки, зазвичай, мають кращу підтримку від платформ, на яких вони працюють.

1.3 Огляд кросплатформних фреймворків

1.3.1 Загальні відомості про фреймворк Apache Cordova

Apache Cordova [5] (раніше відомий як PhoneGap) — це відкритий фреймворк для розробки кросплатформних мобільних застосунків. Він був створений в 2009 році компанією Nitobi, яку пізніше купила Adobe Systems. Після здобуттям проекту статусу верхнього рівня в рамках Apache Software Foundation в 2012 році, Cordova є незалежною технологією. Він дозволяє розробникам створювати застосунки, які можуть працювати на різних мобільних платформах, використовуючи технології веб-розробки, такі як HTML5, CSS3 і JavaScript, замість написання нативного коду для кожної окремої платформи.

Ось декілька ключових характеристик Apache Cordova:

- кросплатформність: Cordova дозволяє розробникам створювати застосунки, які працюють на різних мобільних платформах (таких як iOS, Android, Windows Phone та інші) з єдиної бази коду;
- використання веб-технологій: Cordova використовує веб-технології (HTML5, CSS3 та JavaScript) для розробки застосунків. Це дозволяє розробникам використовувати існуючі технології веб-розробки для створення мобільних застосунків;
- підтримка нативних API: Cordova надає JavaScript API, який відповідає нативним API мобільних платформ. Це дозволяє застосункам Cordova використовувати нативні функції пристрою, такі як камера, геолокація, акселерометр;
- плагіни: Cordova має систему плагинів, яка дозволяє розширювати функціональність застосунків. Розробники можуть використовувати готові плагини або створювати свої власні;
- спільнота: Apache Cordova має велику та активну спільноту розробників, яка постійно вносить внески та покращує фреймворк;

Варто відзначити, що Apache Cordova є основою для декількох інших фреймворків кросплатформної розробки, таких як Ionic [19], які розширюють функціональність і надають додаткові можливості.

1.3.2 Загальні відомості про фреймворк Ionic

Ionic — це відкритий фреймворк для розробки кросплатформних мобільних застосунків [19] з використанням веб-технологій (HTML, CSS, JavaScript). Він був створений в 2013 році компанією Drifty Co. Ionic дозволяє розробникам створювати застосунки з єдиною базою коду, які можна запускати на різних платформах, таких як iOS, Android, Windows та інших.

Ionic використовує веб-технології, такі як HTML5, CSS і JavaScript або TypeScript для написання та виконання застосунків, а також використовує Apache Cordova (раніше відомий як PhoneGap) або Capacitor для доступу до нативних функцій пристрою [19].

Можна визначити такі ключові характеристики Ionic:

- містить власні UI компоненти та інструменти для полегшення розробки дизайну і логіки застосунків;
- використовує Cordova як базовий шар для доступу до нативних API мобільних пристроїв. Тобто Ionic - це обгортка навколо Apache Cordova;
- містить власні служби та інструменти - Ionic Pro, Ionic Appflow, Capacitor, Stencil для полегшення розробки та збірки застосунків;
- використання веб-технологій: може використовувати фреймворк Angular або бібліотеки React.js або VUE.js
- співтовариство: Ionic має велике та активне співтовариство розробників, що надає велику кількість ресурсів для навчання та підтримки.

1.3.3 Загальні відомості про фреймворк React Native

React Native [3] — це фреймворк для розробки мобільних застосунків, який був створений Facebook. Вперше він був анонсований на конференції Facebook's React.js в 2015 році.

React Native дозволяє розробникам створювати мобільні застосунки на JavaScript або TypeScript, які виконуються на нативних платформах. Застосунки, розроблені за допомогою React Native, мають вигляд і відчуття нативних застосунків і використовують теж нативні компоненти інтерфейсу, що й застосунки, написані на Swift, Objective-C (iOS) або Kotlin, Java (Android).

Можна визначити такі ключові характеристики React Native [6]:

- використовує мову програмування JavaScript - одну з популярніших мов програмування - що робить його доступним для великої кількості розробників;
- використовує нативні елементи UI платформи [3];
- потребує окремої компіляції коду для iOS та Android;
- має велику та активну спільноту розробників, яка постійно розробляє нові бібліотеки та інструменти для покращення екосистеми React Native [6].

Незважаючи на багато переваг, React Native також має деякі недоліки [2]. Зокрема, він може бути більш складним для вивчення, ніж нативні мови програмування, і деякі складні або налаштовуванні нативні функції можуть бути важкими для імплементації або можуть взагалі не підтримуватися.

1.3.4 Загальні відомості про фреймворк Flutter

Flutter [4] — це відкритий кросплатформний фреймворк для розробки мобільних, веб- та десктопних застосунків, який був створений компанією Google. Він був офіційно представлений у 2017 році.

Основою Flutter є мова Dart, яку також розробила Google [1, 4]. Це статично типізована мова, яка має синтаксис, схожий на JavaScript, що робить її доступною для великої кількості розробників.

Можна визначити такі ключові характеристики Flutter:

- включає функцію "гарячого перезавантаження" (Hot Reload), яка дозволяє розробникам бачити зміни в коді миттєво без перезапуску застосунку, що покращує продуктивність розробки.
- вимальовує все UI через свої віджети (Skia графічна бібліотека);

- компілює код в нативні бібліотеки для кожної платформи;
- UI у Flutter вважається кращою ніж у React Native через використання Skia та компіляції в машинний код;
- має активну спільноту розробників та постійно отримує підтримку від Google, що допомагає у покращенні екосистеми фреймворку.

Хоча Flutter має багато переваг, є деякі обмеження, які варто враховувати. Це включає в себе те, що деякі нативні функції можуть бути важкими для імплементації, і те, що мова Dart не настільки поширена, як JavaScript, що може обмежувати кількість доступних розробників.

1.3.5 Загальні відомості про фреймворк Xamarin

Xamarin — це кросплатформний фреймворк для розробки мобільних застосунків, який надає розробникам можливість використовувати мову програмування C# або F# та платформу .NET для створення застосунків для різних мобільних платформ, таких як iOS та Android.

Можна визначити такі ключові характеристики Xamarin:

- дозволяє писати нативний код для iOS та Android на мовах C# або F# та компілюються нативно в .ipa та .apk файлів для розміщення на маркетах;
- дозволяє максимально використовувати нативні інтерфейси та API мобільних платформ;
- розробка ведеться в інтегрованому середовищі розробки (IDE) Visual Studio, що полегшує роботу з кодом та надає широкі можливості від лагодження;
- можливо використовувати спільний код для бізнес-логіки та інших складових, зберігаючи при цьому можливість нативної розробки для конкретних платформ. Можлива повна інтеграція з нативним кодом Objective-C, Swift, Java або Kotlin;
- має активну спільноту розробників.

Xamarin - потужний інструмент “нативної” розробки під iOS та Android з використанням мов C#. Але потребує знання специфічних мов та платформ.

1.4 Критерії вибору фреймворків

При виборі кросплатформного фреймворка для створення мобільних застосунків варто враховувати декілька ключових критеріїв [6]:

- підтримка платформ: перш за все потрібно переконатися, що фреймворк підтримує всі цільові платформи для вашого застосунка;
- продуктивність: важливо оцінити продуктивність фреймворку. Деякі кросплатформні фреймворки можуть працювати повільніше, ніж нативні;
- Спільнота та підтримка: чим більше спільнота, тим краще. Велика активна спільнота може полегшити розв'язання проблем, забезпечити широкий вибір готових до використання пакетів та бібліотек.
- мова програмування: мова, на якій базується фреймворк, також має значення. Наприклад, якщо ви вже знайомі з JavaScript, вам може бути легше розробляти на React Native, ніж на Flutter, який використовує Dart, а якщо ви знаєте мову програмування C# то вам краще Xamarin.
- можливості персоналізації UI: рівень підтримки персоналізації UI може відрізнятися від фреймворка до фреймворка. Деякі фреймворки, наприклад, Flutter, надають велику кількість віджетів для побудови UI.
- підтримка пакетів: оцініть наскільки легко встановлювати та керувати зовнішніми пакетами та бібліотеками в конкретному фреймворку.
- тестування та налагодження: деякі фреймворки надають більш передові інструменти для тестування і налагодження, ніж інші.
- вартість: хоча багато фреймворків є безкоштовними та відкритими, вони можуть потребувати певних витрат на підтримку, навчання, інтеграцію або персоналізацію.

Кожен проект унікальний, тому важливо враховувати всі ці фактори в контексті конкретних потреб та обмежень проекту.

На сьогоднішній день фреймворк Flutter випереджає свого найближчого конкурента, React Native. В той самий час, інші фреймворки значно відстають це

видно на двох рисунках 1.1 та 1.2, де на першому відображається статистика популярності [24] та на іншому кількість запитів на місяць розробників [12].

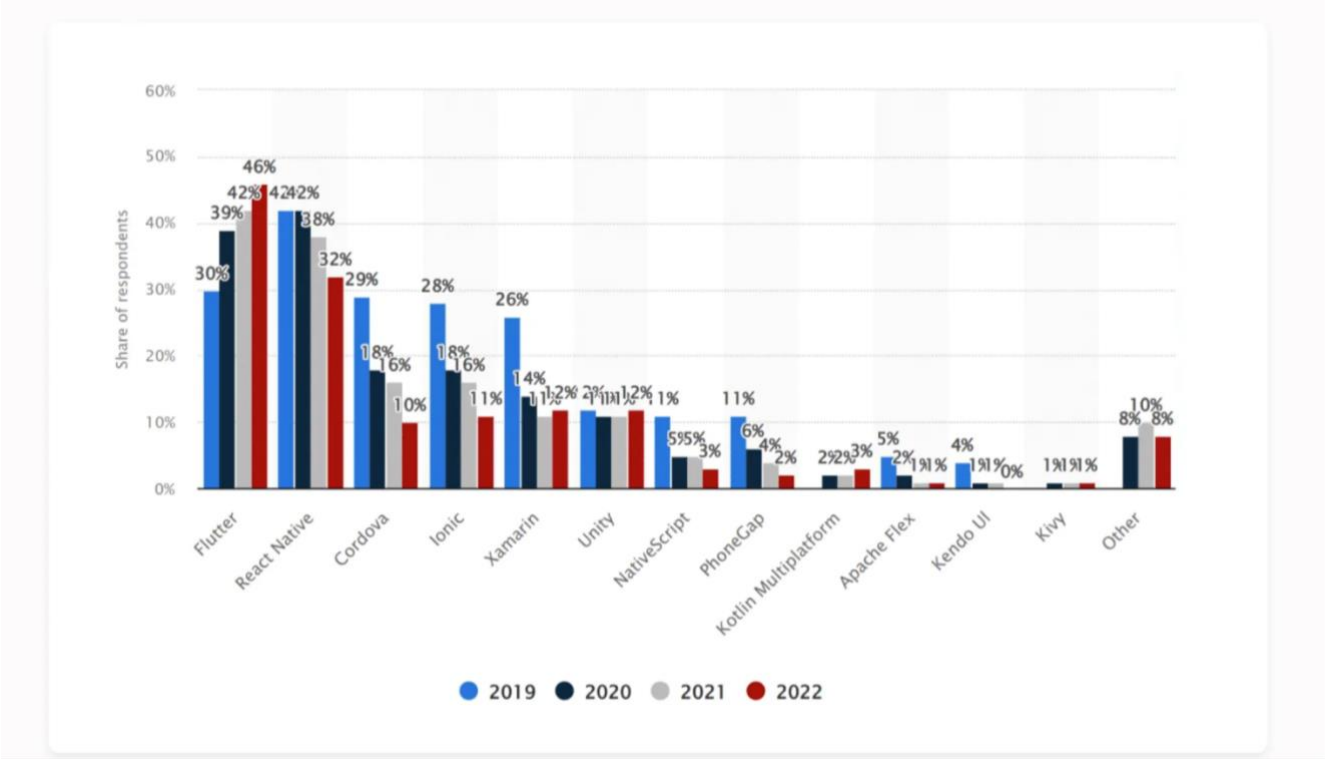

Рисунок 1.1 — Статистика за 2019 - 2022 рік популярності фреймворків

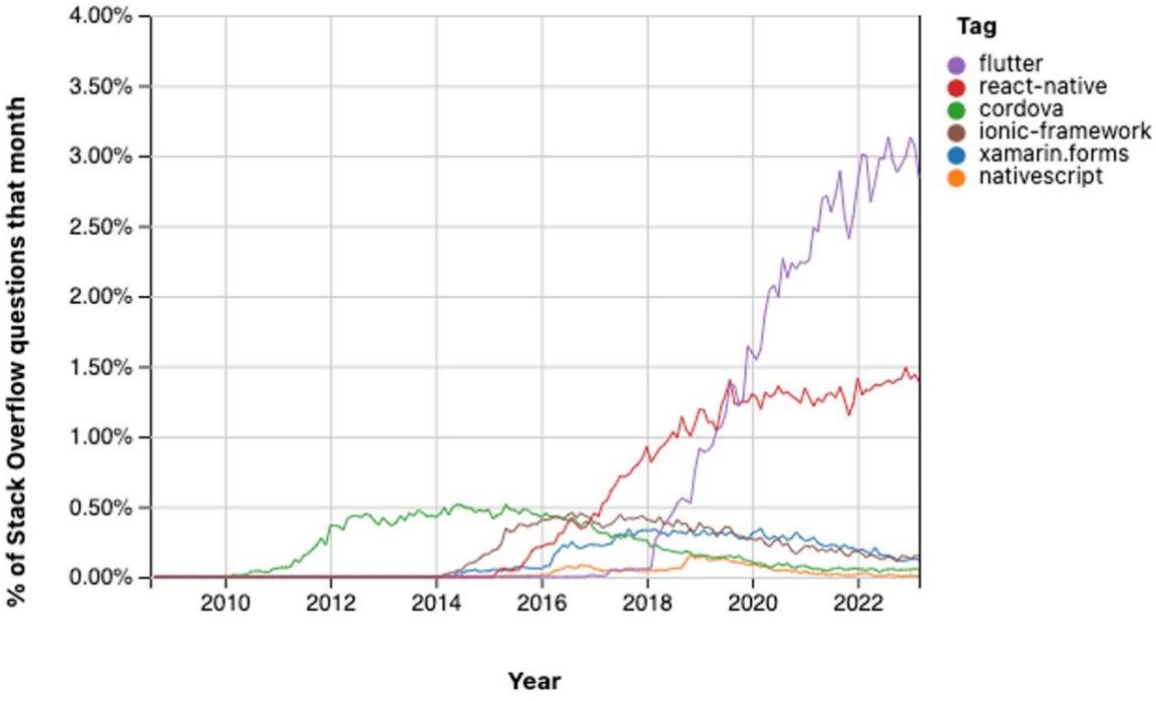

Рисунок 1.2 — Кількість питань у відсотках протягом місяця на StackOverflow

Після аналізу усіх цих даних для цієї роботи було вибрано дослідження фреймворків Flutter та React Native.

1.5 Порівняльний аналіз мов програмування

Мови програмування JavaScript/TypeScript та Dart є популярними мовами програмування, які використовуються для розробки веб-застосунків, мобільних застосунків і серверних програм [6]. Вони мають переваги та недоліки, і вибір між ними залежить від конкретних потреб вашого проекту.

У контексті теми магістерської роботи на тему “Особливості застосування фреймворків Flutter та React Native для створення мобільних застосунків” порівняльний аналіз мов програмування JavaScript/TypeScript та Dart є важливим, оскільки ці мови використовуються як основою для цих фреймворків [6].

JavaScript - інтерпретована мова програмування, яка виконується у браузері або на сервері за допомогою відповідного середовища виконання, такого як Node.js. Він інтерпретується браузером або середовищем виконання JavaScript на льоту, без необхідності попередньої компіляції.

TypeScript - це розширення JavaScript [22], яке додає статичну типізацію та інші можливості, такі як інтерфейси та перерахування. TypeScript код в кінцевому підсумку компілюється в звичайний JavaScript, який потім може бути інтерпретований браузером або середовищем виконання JavaScript.

Отже, JavaScript та TypeScript можуть бути інтерпретовані, але TypeScript зазвичай спочатку компілюється в JavaScript перед тим, як його можна буде інтерпретувати.

Мови є об'єктно-орієнтованими та підтримують асинхронне програмування [6].

Відмінності:

– JavaScript та TypeScript — це інтерпретовані мови, а Dart - це компільована мова [21]. Це означає, що JavaScript та TypeScript код виконується

безпосередньо в браузері або на пристрої [22, 23], а Dart немає проміжної віртуальної машини. Код для Dart виконується безпосередньо процесором пристрою. Це дозволяє Dart конкурувати по швидкодії з нативними мовами[6];

– JavaScript та TypeScript — це динамічні типізовані мови, а Dart — це статично типізована мова [21]. Це означає, що типи змінних в JavaScript/TypeScript визначаються автоматично, а в Dart вони повинні бути явно вказані [6];
– JavaScript/TypeScript має більшу кількість бібліотек і інструментів, ніж Dart. Це пов'язано з тим, що JavaScript/TypeScript використовується значно довше, ніж Dart [6].

JavaScript та TypeScript

Переваги:

– широка підтримка: JavaScript та TypeScript підтримується більшістю браузерів і серверних платформ [6];
– співтовариство: JavaScript/TypeScript має велике і активне співтовариство розробників, які створили безліч бібліотек і інструментів;
– простота: JavaScript/TypeScript відносно простий у вивченні та використанні [6].

Недоліки:

– Продуктивність: JavaScript/TypeScript код, як правило, виконується повільніше, ніж Dart код [6];
– Безпека: JavaScript/TypeScript код може бути складним для перевірки на наявність помилок типу [6].

Dart

Переваги:

– Швидкість: Dart код, як правило, виконується швидше, ніж JavaScript/TypeScript код [6].
– Безпека: Dart код можна перевірити на наявність помилок типу під час компіляції [6];

– Продуктивність: Dart має ряд функцій, які можуть допомогти розробникам бути більш продуктивними, наприклад, функції асинхронного програмування та аналітик типу [6].

Недоліки:

– Залежність від Google: Dart розробляється компанією Google, що може бути проблемою для деяких розробників [6, 10];
– Менша підтримка: Dart має меншу бібліотеку бібліотек і інструментів, ніж JavaScript/TypeScript [6].

JavaScript/TypeScript і Dart — це обидві потужні мови програмування, які можна використовувати для створення різних типів застосунків. Вибір між ними залежить від ваших конкретних потреб і переваг.

1.6 Практичні випадки використання фреймворків React Native та Flutter

Приклади реалізації для React Native:

Walmart [26], масштабна міжнародна роздрібна корпорація із США, також вибрала React Native для розробки свого мобільного застосунку. Керуючи бізнесом в Канаді, Аргентині, Південній Африці та Чилі, Walmart постійно експонує, зокрема завдяки онлайн-платформам. Вибір React Native був обумовлений стремлінням покращити досвід користувачів своїх додатків усією планетою. Після низки тестів із використанням інтегрованих web view, лише React Native зміг задовольнити всі вимоги. Walmart Labs підтверджує, що використання React Native сприяло підвищенню графічної продуктивності, прискоренню роботи додатку та спрощенню виправлення помилок.

Instagram [27] та Skype [28] — це два відомих мобільних застосунки, які мають велику популярність серед користувачів усього світу. Instagram - це соціальна мережа для обміну фотографіями та відео, де користувачі можуть ділитися своїми моментами життя, спостереженнями та враженнями з іншими користувачами. Skype, з іншого боку, - це популярний месенджер, який надає

можливість здійснювати голосові та відео дзвінки, а також обмінюватися повідомленнями з друзями, родиною та колегами з усього світу.

Як і Walmart, ці компанії вибрали React Native для розробки своїх мобільних застосунків. Інтеграція цієї технології дозволила покращити досвід користувачів, незалежно від їх місця знаходження. Instagram та Skype постійно розвиваються та розширюють свій функціонал, надаючи користувачам нові можливості та інструменти для спілкування.

Використання React Native дозволило підвищити продуктивність, прискорити роботу застосунків та спростити процес виправлення помилок. Ці компанії переконалися, що вибір цієї технології був вдалим і допоміг їм покращити якість своїх мобільних застосунків. Приклади відображення інтерфейсів застосунків Instagram та Skype зображені на рисунку 1.3.

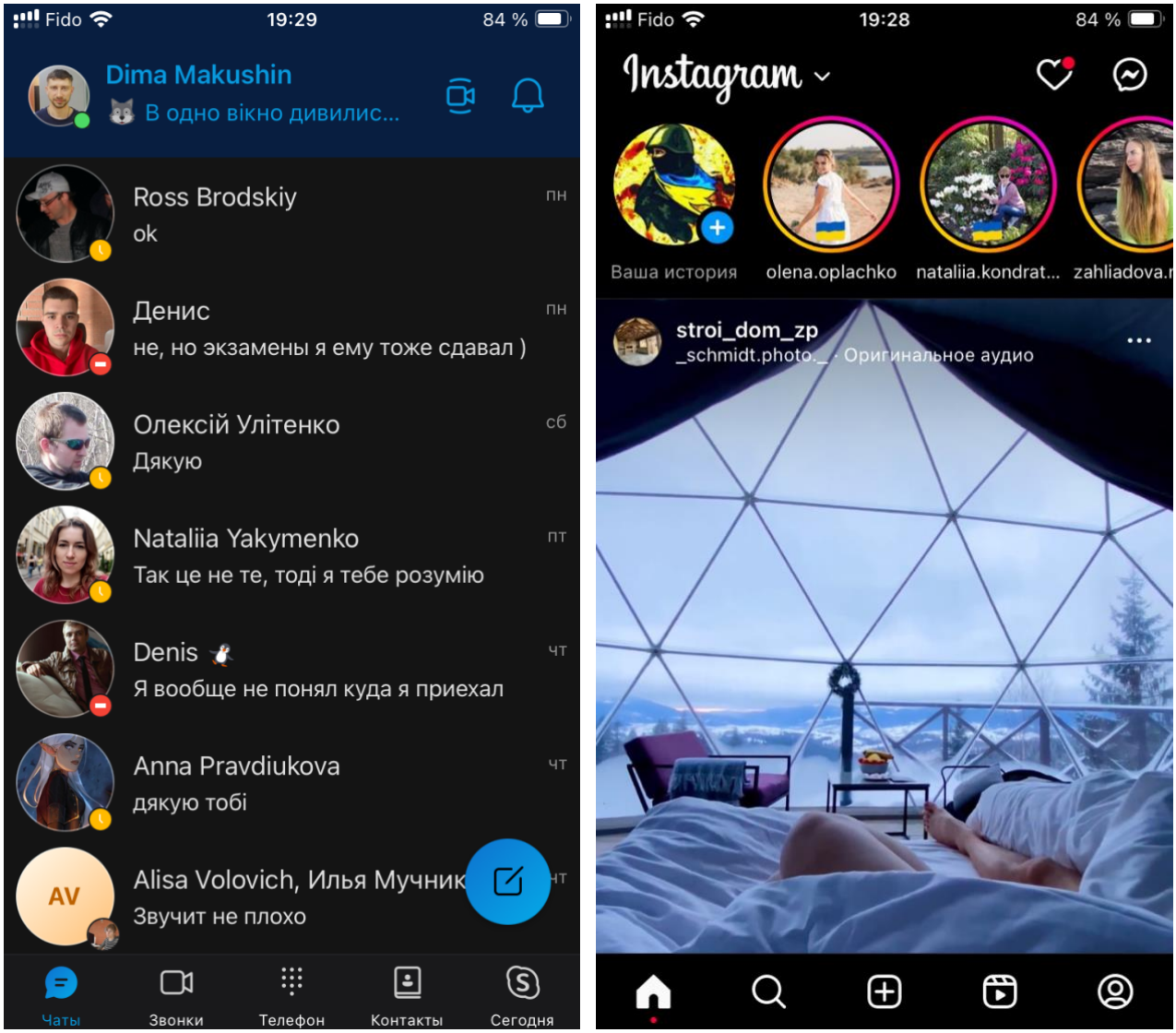

Рисунок 1.3 — Приклади відображення інтерфейсів Instagram та Skype

Приклади реалізації для Flutter:

Hamilton [29] — це мобільна програма, що дозволяє користувачам зручно слухати аудіо версію популярного мюзиклу "Гамільтон". Користувачі можуть відтворювати музику та текст пісень, дивитися відео та отримувати інші додаткові матеріали, пов'язані з цим видатним твором мистецтва.

“The New York Times” [29] — це мобільний застосунок, який надає доступ до останніх новин, статей та репортажів від видання "The New York Times". Користувачі можуть отримувати інформацію про події у всьому світі, читати аналітичні матеріали та переглядати мультимедійний контент, представлений у різних форматах, таких як тексти, фотографії та відео.

Приклади відображення інтерфейсів застосунків Hamilton та The New York Times зображені на рисунку 1.4.

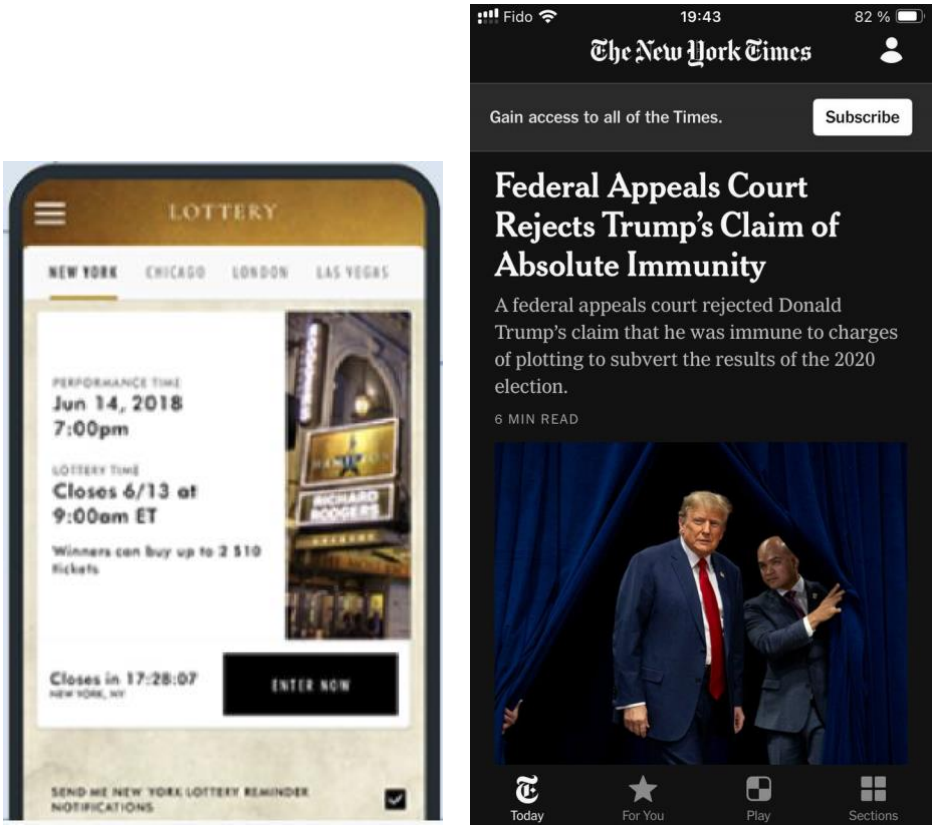

Рисунок 1.4 — Приклади відображення інтерфейсів Hamilton та The New York Times

Висновки до розділу 1

У цьому розділі ми зосередилися на аналізі викликів, пов'язаних з вибором кросплатформного фреймворку. Було враховано переваги та недоліки кросплатформної розробки, а також проведено огляд розповсюджених фреймворків. Окремо були розроблені критерії, що є важливими при виборі фреймворку.

РОЗДІЛ 2 ДОСЛІДЖЕННЯ ЗАСОБІВ СТВОРЕННЯ МОБІЛЬНИХ ЗАСТОСУНКІВ

Мобільні застосунки за технологією кросплатформи можна розробляти на усіх популярних операційних системах Windows, Linux та MacOS. Для розробки мобільних застосунків для операційних систем iOS та Android у своєму дослідженні я використовував комп’ютер на операційній системі MacOS version 14.2 (Sonoma). Додатково встановлюється та налагоджується Xcode та Xcode Command Line Tools та Android Studio.

2.1 Розгортання оточення для розробки на React Native

React Native використовує два різних середовища для створення мобільних додатків, а саме Expo Go та React Native CLI [3]. Для навчання програмуванню на React Native самі розробники фреймворку рекомендують використовувати Expo Go, але для професійної розробки краще використовувати React Native CLI [3].

Expo Go має обмеження для використання сторонніх сервісів та бібліотек, а також обмежує можливість безпосереднього доступу до мобільного коду. З іншої сторони, React Native CLI дозволяє використовувати будь-які бібліотеки без обмежень [3].

Для вирішення задачі було обрано React Native CLI на момент написання роботи актуальна версія була 0.73.2.

Для коректної роботи React Native CLI на комп'ютері було встановлено додаткове ПЗ: Node.js версії v18.15.0, Watchman.

Watchman та Node.js потрібні для використання React Native CLI з метою розробки мобільних застосунків. Node.js використовується для виконання JavaScript-коду та для управління залежностями проекту. Watchman використовується для відслідковування змін у файлах проекту та автоматичної перекомпіляції або перезавантаження застосунку під час розробки. Вони є

важливими інструментами для розробників React Native, які дозволяють ефективно створювати та тестувати мобільні застосунки.

Для написання коду використовується потужне інтегроване середовище розробки (IDE) WebStorm, створене компанією JetBrains.

2.2 Розгортання оточення для розробки на Flutter

Спочатку треба скачати архів з офіційного сайту (в моєму випадку для MacOS https://docs.flutter.dev/get-started/install/macos) та розгорнути його локально на комп'ютері. Потім написати налаштування для середовища командою:

$ export PATH="$PATH:`pwd`/flutter/bin"

Щоб перевірити що все було успішно встановлено потрібно в ввести команду:

$ flutter doctor

Якщо все було правильно встановлено то на екрані буде відображатися як показано на рисунку 2.1.

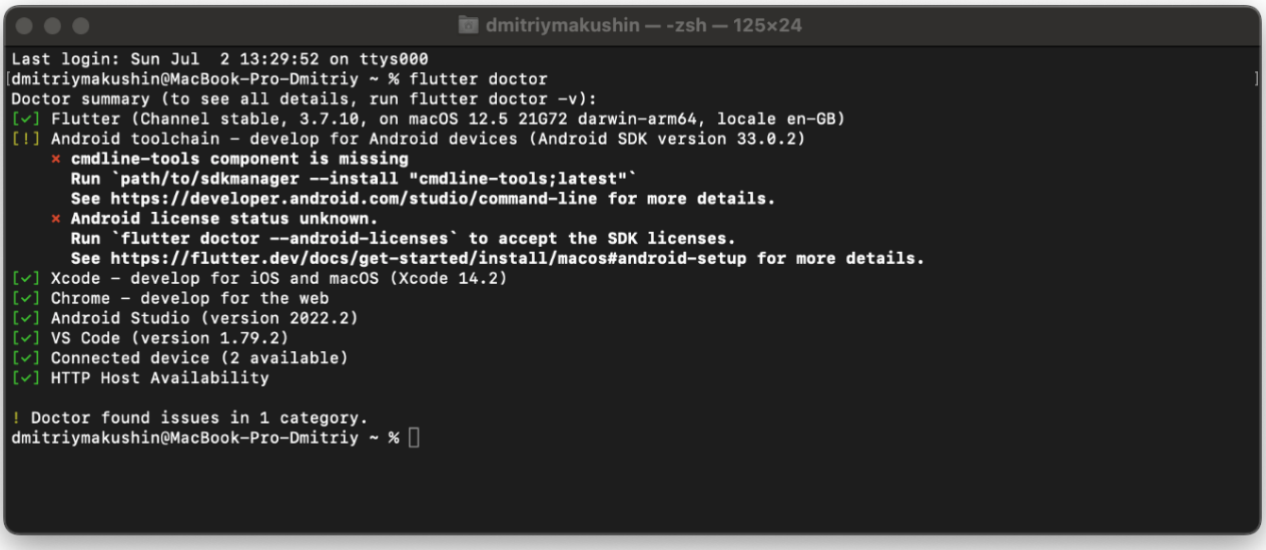

Рисунок 2.1 — Відображення встановленого та настроєного оточення для Flutter

На час написання цієї роботи актуальна версія була 3.10.5

Для написання коду я використовував редактор Android Studio в яку було додатково встановлено плагин Dart.

2.3 Встановлення емуляторів для розробки та первинний запуск застосунку

Для встановлення емуляторів для iOS достатньо встановити програмне забезпечення Xcode, яке містить емулятор iOS Simulator. Його можна завантажити з App Store або з офіційного веб-сайту розробника Apple (https://developer.apple.com/xcode/resources/).

Щоб встановити емулятори для Android, вам необхідно встановити Android Studio, яке містить емулятор Android Studio Emulator. Завантажити Android Studio можна з офіційного веб-сайту розробника Android (https://developer.android.com/studio).

2.3.1 Запуск застосунку на емуляторі iOS та Android розробленого на React Native

Для того щоб зробити стартовий проект треба ввести команду у терміналі:

$ npx react-native@latest init <Назва проекту>

Дуже важливо назва проекту не повинна складатися з двох окремих слів та нести у собі спеціальні символи.

Після створення первинного застосунку структура проекту буде виглядати як показано на рисунку 2.2.

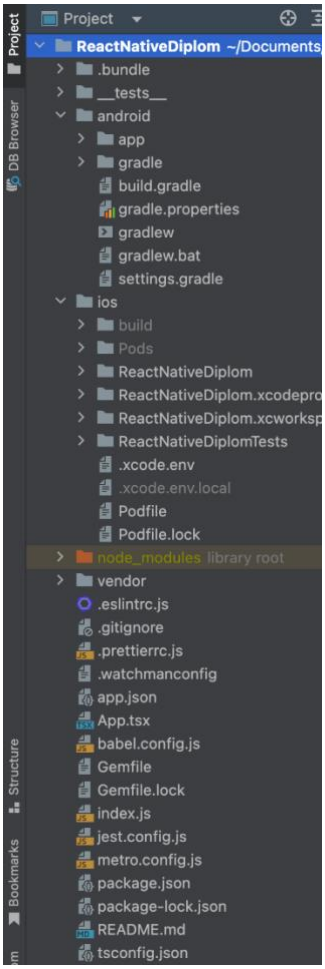

Рисунок 2.2 — Первинна структура проекту на React Native

Як ми бачимо з рисунку 2.2 у структурі проекту відразу створюються як нативні частини це папки “ios” та “android” так і сам початковий код JavaScript. Розберемо первинну структуру:

папки ios та android це папки з нативним кодом куди будуть додаватися частини коду після кожної збірки JavaScript.

.prettierrc.js — використовується для налаштування параметрів форматування коду за допомогою інструменту Prettier. У цьому файлі можна вказати різні правила форматування, такі як відступи, використання одинарних або подвійних лапок для рядків, розмір відступу, розміщення фігурних дужок тощо.

.watchmanconfig — файл використовується для налаштування Watchman, що є інструментом для стеження за змінами в файловій системі. Цей файл містить конфігураційні параметри для Watchman, такі як список каталогів, за якими

потрібно стежити, правила ігнорування файлів, налаштування відстеження змін та інші параметри, необхідні для коректної роботи інструменту.

app.json — використовується для зберігання конфігураційних параметрів застосунку. Він містить різноманітні налаштування, такі як ім'я застосунка, версія, інформація про автора, конфігурація сторінок, налаштування віддалених серверів та багато іншого.

App.tsx — файл є основним вихідним файлом застосунка. Він містить визначення кореневого компонента застосунка, який буде відображений при запуску застосунку. Цей файл відповідає за створення головного компонента застосунку та налаштування його роботи.

babel.config.js — файл використовується для налаштування Babel, який є інструментом для трансформації сучасного JavaScript коду в стандартний JavaScript код, який може бути розпізнаний та виконаний у всіх браузерах або середовищах виконання.

Gemfile — файл використовується для визначення та управління залежностями Ruby-програмного забезпечення, яке може бути потрібним для роботи з проектом React Native. Основна функція файлу Gemfile полягає в тому, щоб забезпечити управління різноманітними бібліотеками, фреймворками та іншими залежностями, які можуть використовуватися під час розробки, тестування та виробництва проекту.

Gemfile.lock — файл використовується для фіксації конкретних версій всіх залежностей, вказаних у файлі Gemfile. Основна функція файлу Gemfile.lock - це забезпечити стабільність та репродуктивність середовища розробки, зберігаючи точні версії кожного компонента, необхідного для проекту.

index.js — файл виступає як вхідна точка для початку виконання застосунку React Native і містить основну логіку, необхідну для правильного його функціонування.

jest.config.js — файл дозволяє налаштувати тестувальне середовище в проєкті React Native з використанням Jest, щоб забезпечити ефективне та зручне тестування застосунків.

metro.config.js — файл використовується для налаштування та конфігурації Metro Bundler в проекті React Native з метою оптимізації та покращення процесу пакування та збірки JavaScript-коду.

package.json — є центральним файлом у проекті React Native, який містить ключову інформацію про проект та допомагає керувати його залежностями, конфігурацією та автоматизацією процесів розробки.

package-lock.json — файл забезпечує фіксацію версій пакетів та їх залежностей, що дозволяє забезпечити стабільність та відтворюваність проекту під час розробки та встановлення залежностей.

tsconfig.json — цей файл у проекту React Native дозволяє налаштовувати та контролювати процес компіляції TypeScript коду, що допомагає забезпечити стабільність та ефективність розробки.

Для того щоб запустити застосунок є два способи.

Перший статус це запуск застосунку через термінал командую наприклад для запуску iOS застосунку:

$ npx react-native run-ios

для запуску застосунку на Android потрібна команда:

$ npx react-native run-android

Другий способу це запуск через середовище розробника. Для запуску застосунку, на емуляторі iOS потрібно відкрити проект у засобі розробки Xcode, після чого обрати емулятор iOS Simulator та запустити застосунок.

Для запуску на емуляторі Android потрібно відкрити проект у засобі розробки Android Studio, після чого обрати емулятор Android Studio Emulator та запустити застосунок.

Після запуску першого початкового застосунку можна побачити простий застосунок який буде виглядати як зображено на рисунку 2.2. для iOS та рисунку 2.3 для Android.

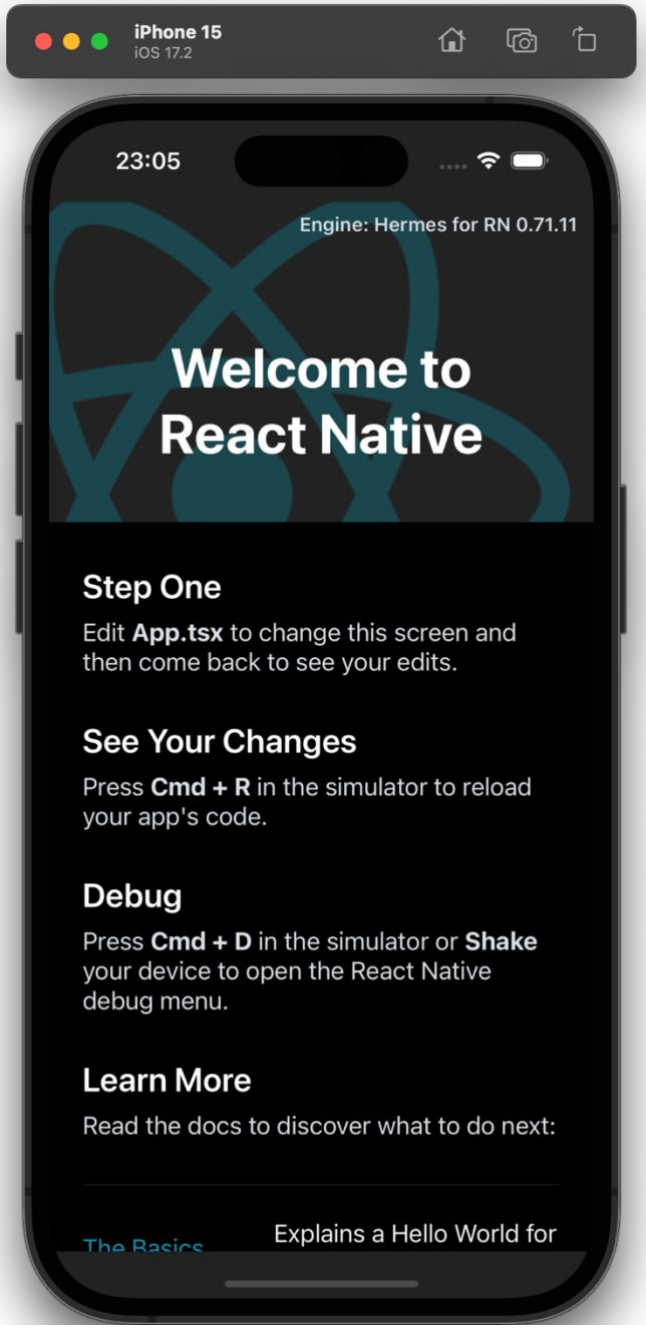

Рисунок 2.3 — Стартовий застосунок на операційній системі iOS створений на React Native

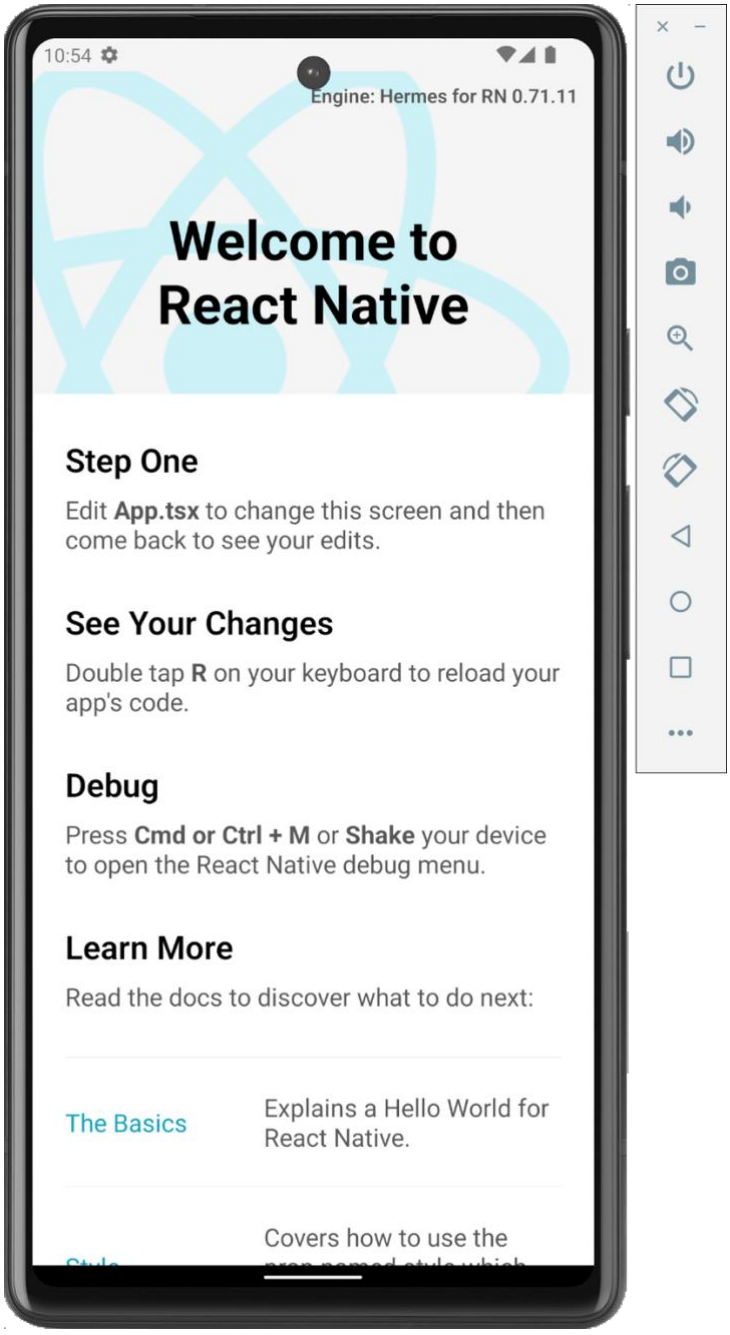

Рисунок 2.4 — Стартовий застосунок на операційній системі Android створений на React Native

2.3.2 Запуск застосунку на емуляторі iOS та Android розробленого на Flutter.

Для того щоб створити перший застосунок на Flutter достатньо вибрати при створенні проекту у Android Studio пункт створити проект на Flutter. Структура первинного проекту представлено на рисунку 2.5.

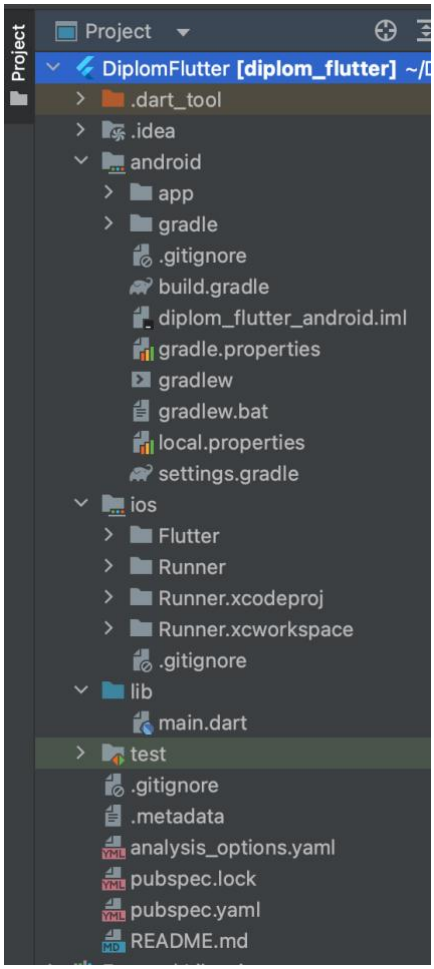

Рисунок 2.5 — Первинна структура проекту на Flutter

Описуємо первинну структуру:

папки “ios” та “android” — ці папки містять файли для конфігурації та побудови додатку для платформ iOS та Android відповідно. Вони містять конфігураційні файли, які можна налаштувати для встановлення підписів, версій SDK, налаштувань зображень тощо;

папка “lib” — це основна папка, де зазвичай зберігається весь вихідний код Flutter застосунку. Вона містить файли Dart, які визначають віджети, екрани, логіку застосунку, стилі, переклади тощо. С початку вона містить лише один файл main.dart який відповідає за встановлення і запуск основного віджета (widget) застосунку;

pubspec.yaml — цей файл визначає конфігурацію проекту Flutter, включаючи залежності пакетів, ресурси, версію проекту, інформацію про автора

тощо. В ньому вказуються всі залежності, які використовуються у проекті, і він дозволяє встановлювати нові пакети з Flutter із репозиторію pub.dev;

папка assets — ця папка містить ресурси, такі як зображення, шрифти, аудіо та відео файли, які використовуються у застосунку. Вони можуть бути завантажені та використані у застосунку за допомогою шляху до них у коді;

папка test — ця папка містить юніт-тести, які використовуються для тестування функцій та компонентів Flutter застосунку. Тести дозволяють перевіряти, чи працює код правильно під час розробки та рефакторингу.

Для того щоб запустити застосунок розроблений на Flutter у редактору Android Studio достатньо вибрати емулятор у редактору та натиснути запуск після чийого буде скомпільована версія застосунку для вибраного емулятору.

Після запуску першого початкового застосунку можна побачити простий додаток який буде виглядати як зображено на рисунку 2.4. для iOS та рисунку 2.5 для Android.

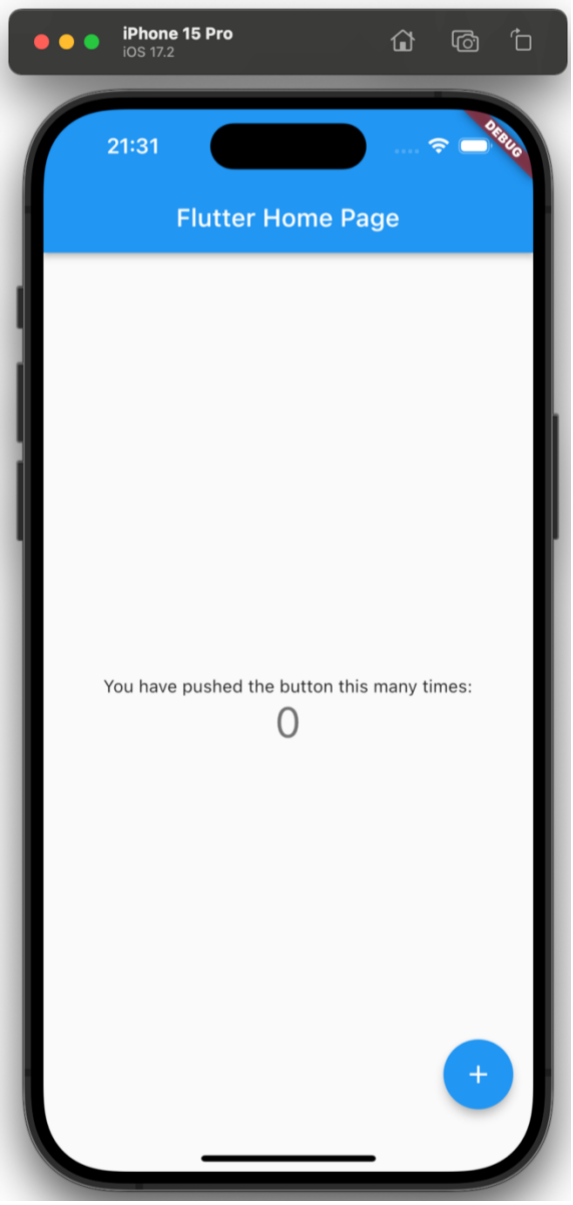

Рисунок 2.6 — Стартовий застосунок на операційній системі iOS на Flutter

Рисунок 2.7. Стартовий застосунок на операційній системі Android на Flutter

2.4 Розробка функціональних вимог та вимог до інтерфейсу користувача мобільного застосунку

Розроблення функціональних вимог та вимог до інтерфейсу користувача мобільного застосунку передбачає аналіз можливостей фреймворків та їхніх особливостей для глибшого дослідження. Визначено наступні функціональні вимоги та вимоги до інтерфейсу користувача:

– “Заставка запуску застосунку” — це перша сторінка, яка відображається при запуску додатка;

- “Головний екран з переліком екранів” — стартова сторінка, на якій є перелік доступних екранів з коротким описом і посиланнями на них;
- “Екран стандартних функцій” — Відображення переліку елементів, форм, вікон повідомлень та кнопок;
- “Екран використання датчиків” — сторінка, що демонструє використання датчиків пристрою;
- “Екран інтеграції сторонніх сервісів” — сторінка, що показує інтеграцію зі сторонніми сервісами;
- “Екран роботи з базою даних” — відображення можливостей роботи з нераліційною базою даних на основі Firebase Cloud Firestore. База даних матиме лише одну таблицю “Tasks” та має такі поля: id, taskDesc та taskName.

Деякі з цих пунктів мають додаткові підпункти, такі як "Прості елементи" та "Вікна повідомлень".

Для реалізації вище зазначених вимог, використовуються різні технології та сервіси, зокрема Firebase для збереження та обробки даних, Stripe для обробки платежів, а також Google Maps та Mapbox картографічних сервісів.

Висновки до розділу 2

Виконана підготовка робочого середовища для розробки мобільних застосунків за допомогою React Native та Flutter. Обрано MacOS як операційну систему для розробки, а також встановлено необхідні додаткові інструменти, такі як Xcode, Xcode Command Line Tools, Android Studio, Node.js, Watchman та інші.

У контексті React Native було вирішено використовувати React Native CLI замість Expo Go, враховуючи більш гнучкі можливості та менші обмеження React Native CLI.

Для розробки коду обрано інтегроване середовище розробки (IDE) WebStorm від JetBrains.

Для Flutter було виконано встановлення та налаштування робочого середовища згідно з інструкціями на офіційному сайті. Додатково настроєне інтегроване середовище розробки Android Studio.

Визначено функціональні вимоги та вимоги до інтерфейсу демонстраційних мобільних застосунків, які мають на меті охопити широкий спектр функціональних можливостей, включаючи інтеграцію зі сторонніми сервісами, роботу з базами даних, використання датчиків та інше.

РОЗДІЛ 3 РОЗРОБКА МОБІЛЬНИХ ЗАСТОСУНКІВ

3.1 Стек технологій

3.1.1 Stripe

Stripe — це платформа для обробки онлайн-платежів, яка надає інструменти та сервіси для прийому платежів через Інтернет. Stripe дозволяє підприємствам та розробникам отримувати платежі від різних джерел, таких як кредитні карти, мобільні платежі та інші електронні платіжні системи [17].

Основні можливості Stripe включають:

- Інтеграція платежів: Stripe надає простий спосіб інтеграції платіжної системи у ваш додаток або веб-сайт. Вона має документацію та набір інструментів для різних платформ, включаючи мови програмування, такі як JavaScript, Python, Ruby та інші.
- Обробка різних типів платежів: Stripe підтримує оплату кредитними та дебетовими картками (Visa, Mastercard, American Express та іншими), мобільними платежами (Apple Pay, Google Pay) та іншими електронними платіжними системами.
- Безпека та відповідність: Stripe забезпечує безпеку платежів через застосування шифрування та інших заходів безпеки для захисту конфіденційної інформації. Крім того, Stripe відповідає платіжним стандартам, таким як PCI DSS (Стандарт безпеки даних платіжної картки).

Stripe є популярним вибором для багатьох компаній та розробників через його простоту використання, гнучкість та надійність. Вона забезпечує зручний спосіб отримання платежів в Інтернеті та спрощує процес обробки транзакцій для бізнесів будь-якого розміру.

3.1.2 Firebase

Firebase — це платформа від Google, яка надає широкий спектр інструментів для розробки веб та мобільних застосунків [30]. Вона пропонує різноманітні

сервіси, починаючи від хостингу та зберігання баз даних до автентифікації користувачів, аналізу, зберігання файлів, сповіщень та багато іншого.

Основні можливості Firebase включають:

- Бази даних в реальному часі: Firebase Realtime Database - це облікова система бази даних в реальному часі, яка дозволяє зберігати та синхронізувати дані між вашими клієнтськими та серверними додатками в реальному часі.
- Автентифікація: Firebase надає сервіси автентифікації, які дозволяють впроваджувати різні методи входу, такі як електронна пошта, Google, Facebook, Twitter тощо.
- Хостинг та зберігання файлів: Firebase Hosting дозволяє розгортати та хостити веб-сайти з HTTPS. Firebase також забезпечує зберігання файлів і зображень.
- Аналітика: Firebase Analytics надає засоби для відстеження та аналізу дій користувачів у вашому додатку, що допомагає вдосконалити його функціональність та взаємодію.
- Хмарні повідомлення: Firebase Cloud Messaging (FCM) дозволяє надсилати повідомлення на мобільні пристрої звідки завгодно в хмарі.

Firebase - це інтегрована платформа, яка спрощує процес розробки та підтримки додатків шляхом надання різноманітних інструментів та сервісів для розробників та підприємств.

3.1.3 Google Maps

Google Maps — це веб-сервіс та мобільний додаток, розроблений компанією Google, який надає можливість переглядати мапи, отримувати інформацію про місця, шляхи проїзду та інші географічні дані [31]. Сервіс Google Maps включає в себе широкий спектр функцій:

Картографічні дані: Google Maps надає користувачам можливість переглядати мапи з різними видами картографічних даних, таких як карти з рельєфом, супутникові знімки, зображення вулиць тощо.

Навігація та маршрутизація: Користувачі можуть шукати маршрути, використовуючи різні види транспорту, отримувати вказівки проїзду, розраховувати час подорожі та вибирати оптимальний маршрут.

Інформація про місця: Google Maps надає детальну інформацію про різноманітні місця, такі як ресторани, готелі, магазини, атракції та інші об'єкти, включаючи адреси, відгуки користувачів, рейтинги та фотографії.

Інтеграція з мобільними пристроями: Пристрої з встановленим додатком Google Maps можуть використовувати його для навігації у режимі реального часу за допомогою GPS, а також для відстеження розташування користувача.

Google Maps є одним із найбільш популярних та універсальних сервісів картографії, що надає різноманітні можливості для використання географічних даних у повсякденному житті користувачів.

3.1.4 Mapbox

Mapbox — це платформа для картографії та геопросторових аналітичних сервісів, яка надає інструменти для створення та розгортання інтерактивних карт із високоякісними географічними даними [26].

Основні функції та можливості Mapbox включають:

Картографічні дані: Mapbox надає доступ до різних стилів та видів карт, включаючи супутникові знімки, топографічні карти, вуличні карти та інші варіації, які можна легко інтегрувати в веб-додатки та мобільні застосунки.

Кастомізація карт: Платформа дозволяє користувачам створювати та кастомізувати свої власні стилі карт, змінювати вигляд об'єктів на мапі, розміщати маркери, додавати анімації та інші елементи.

Геопросторові сервіси: Mapbox пропонує API для роботи з геоданими, включаючи маршрутизацію, геокодування, аналіз даних та інші сервіси, які дозволяють виконувати операції з геоданими у своїх застосунках.

Інтеграція з мобільними пристроями: Mapbox SDK надає інструменти для інтеграції в мобільні застосунки, що дозволяє використовувати картографічні

можливості, включаючи навігацію, створення маршрутів та інші сервіси на базі геоданих.

Mapbox використовується розробниками та компаніями для створення інтерактивних карт та інтеграції геоданих у своїх проектах. Ця платформа надає багато функціоналу для розробки різноманітних додатків, які потребують використання географічних даних.

3.2 Розробка мобільних застосунків

Після створення початкового налаштування оточення для розробки, було створено два проекта з початковими налаштуваннями які були розроблені у розділі 2 .

При створенні застосунку перше, що треба зробити так це налаштувати маршрутизацію або роутинг між екранами.

3.2.1 Налаштування маршрутизації у застосунках на React Native

Для розробки застосунку на React Native не має стандартного функціоналу, а требо використовувати сторонню бібліотеку. Існує декілька бібліотек для React Native, а саме: React Navigation, React Native Navigation, ExNavigation, React Native Router Flux та React Native Navigation Toolbox.

Для реалізації маршрутизації було використано бібліотеку React Navigation тому що вона являється самою популярною у розробці та підтримується більшістю сполученням розробників React Native.

Реалізація маршрутизації складається з декілька етапів:

- створення самого компоненту який буде екраном;
- створення типу у якому будуть зазначені усі назви сторінок у нашому випадку це “MainStackTypeParamList”;
- створення окремого компоненту у якому буде зазначене усі маршрути.

Приклад реалізації типу “MainStackTypeParamList” показано у лістингу 1, а приклад маршрутизації наводжу у лістингу 2,

Лістинг 1 типу “MainStackTypeParamList”.

```
export type MainStackTypeParamList = {
  Home: undefined;
  ScreenStandard: undefined;
  OtherServices: undefined;
  MapsList: undefined;
  Details: undefined;
  Auth: undefined;
  Database_Screen: undefined;
  Device_Screen: undefined;
  SimpleElements: undefined;
  StripeScreen: undefined;
  GoogleMapsScreen: undefined;
  MapboxScreen: undefined;
  ScreenWindowAlert: undefined;
};
```

Лістинг 2 Приклад маршрутизації.

```
function App() {
  const Stack = createStackNavigator<MainStackTypeParamList>();
  const { t } = useTranslation();
  return (
    <I18nextProvider i18n={i18n}>
      <NavigationContainer>
          <Stack.Navigator
            screenOptions={{
              gestureEnabled: true,
              headerStyle: {
                backgroundColor: 'rgb(66, 140, 255)',
              },
              headerTintColor: '#fff',
              headerTitleStyle: {
                fontWeight: 'bold',
              },
              ...TransitionPresets.SlideFromRightIOS,
```

```
        }}>
          <Stack.Screen
            name="Home"
            options={ () => (
              {
                title: t('main screen'),
                headerRight: () => (
                  <SelectLanguage />
                )
              }
            )}
            component={MainPage} />
          <Stack.Screen
            options={{
              headerBackTitle: 'Home',
              headerBackTitleVisible: false,
              title: 'Деталі',
              headerRight: () => (
                <SelectLanguage />
              )
            }}
            name="Details"
            component={DetailsScreen} />
```

У цьому лістингу 2 приведено приклад лише двох екранів та показано як інтегрувати переклади у маршрутизацію. Інші маршрути майже не будуть змінюватися ось ще приклад декілька екранів у лістингу 3.

Лістинг 3 Приклад маршрутизації декілька екранів.

```
<Stack.Screen
    options={({ navigation }) => ({
      headerLeft: () => (
        <ButtonBack onPress={() => navigation.goBack()}/>
      ),
```

```
        headerBackVisible: false,
        title: t('standard screen'),
        headerRight: () => (
          <SelectLanguage />
        )
      })}
      name="ScreenStandard"
      component={ScreenStandard} />
<Stack.Screen
  options={({ navigation }) => ({
    headerLeft: () => (
      <ButtonBack onPress={() => navigation.goBack()}/>
    ),
    headerBackVisible: false,
    title: t('other services'),
    headerRight: () => (
      <SelectLanguage />
    )
  })}
  name="OtherServices"
  component={OtherServices} />
<Stack.Screen
  options={({ navigation }) => ({
    headerLeft: () => (
      <ButtonBack onPress={() => navigation.goBack()}/>
    ),
    headerBackVisible: false,
    title: t('cartographic services screen'),
    headerRight: () => (
      <SelectLanguage />
    )
  })}
  name="MapsList"
  component={MapsList} />
```

У представленому лістингу 3 відображається класичний приклад навігації, але зазвичай маршрутизація у комерційних застосунках може виглядати так як вказано у лістингу 4.

Лістинг 4.

```
<Stack.Screen
  name="Database_Screen"
  component={DatabaseScreen} />
<Stack.Screen
  name="Device_Screen"
  component={DeviceScreen} />
<Stack.Screen
  name="SimpleElements"
  component={SimpleElements} />
<Stack.Screen
  name="StripeScreen"
  component={MainStripeScreen} />
<Stack.Screen
  name="ScreenWindowAlert"
  component={WindowAlertScreen} />
```

У цьому прикладу вказано лише назва типу екрану та компонент який відповідає за відображення цього екрану, що значно скорочує напис та головне усі налаштування знаходяться у самому компоненту, та надає можливість зробити кожен компонент унікальним у плані відображення на екрані пристрою.

3.2.2 Налаштування маршрутизації у застосунках на Flutter

Для розробки маршрутизації у застосунку на основі Flutter використовується стандартний функціонал, тому був створений файл app.dart у якому була створена маршрутизація між екранами. Структура маршрутизації показана на лістинг 5.

Лістинг 5. Приклад маршрутизації у додатку написаному на Flutter

```
Import 'package:example_app/generated/l10n.dart';
import
'package:example_app/pages/standard_functions/dialogs_page.dart';
import
'package:example_app/pages/standard_functions/standard_functions_pa
ge.dart';
import
'package:example_app/pages/third_party_services/auth/auth_email.dar
t';
import 'package:example_app/pages/main_page.dart';
import
'package:example_app/pages/sensors_example/sensors_example.dart';
import 'package:example_app/pages/splash_page.dart';
import
'package:example_app/pages/third_party_services/maps/maps_page.dart
';
import
'package:example_app/pages/third_party_services/stripe/stripe_examp
le_page.dart';
import
'package:example_app/pages/third_party_services/third_party_service
s_page.dart';
import
'package:example_app/pages/standard_functions/widgets_page.dart';
import
'package:example_app/pages/work_with_database/database_example_page
.dart';
import 'package:flutter/material.dart';
import 'package:flutter_localizations/flutter_localizations.dart';

class App extends StatelessWidget {
 const App({super.key});

 // This widget is the root of your application.
```

```
@override
Widget build(BuildContext context) {
    return MaterialApp(
    title: 'Flutter Demo',
     theme: ThemeData(
     primarySwatch: Colors.blue,
    ),
          debugShowCheckedModeBanner: false,
          initialRoute: SplashPage.path,
          routes: <String, WidgetBuilder>{
          SplashPage.path: (context) => const SplashPage(),
          MainPage.path: (context) => const MainPage(),
          AuthEmailPage.path: (context) => const AuthEmailPage(),
          WidgetsPage.path: (context) => WidgetsPage(),
          MapsPage.path: (context) => MapsPage(),
          StandardFunctionsPage.path: (context) =>
    StandardFunctionsPage(),
          StripeExamplePage.path: (context) =>
    StripeExamplePage(),
          ThirdPartyServicesPage.path: (context) =>
    ThirdPartyServicesPage(),
          DatabaseExamplePage.path: (context) =>
    DatabaseExamplePage(),
          SensorsExamplePage.path: (context) =>
    SensorsExamplePage(),
          DialogsPage.path: (context) => DialogsPage(),
    },
     localizationsDelegates: const [
          S.delegate,
          GlobalMaterialLocalizations.delegate,
          GlobalWidgetsLocalizations.delegate,
          GlobalCupertinoLocalizations.delegate,
    ],
     supportedLocales: S.delegate.supportedLocales,
    );
```

```
        }
}
```

3.2.3 Розробка головного екрану

На головного екрані відображається зміст усіх екранів які є розділами та короткий опис з кнопкою посилання на інший екран. Приклад відображення екрану на рисунку 3.1 для React Native та для Flutter на рисунку 3.2.

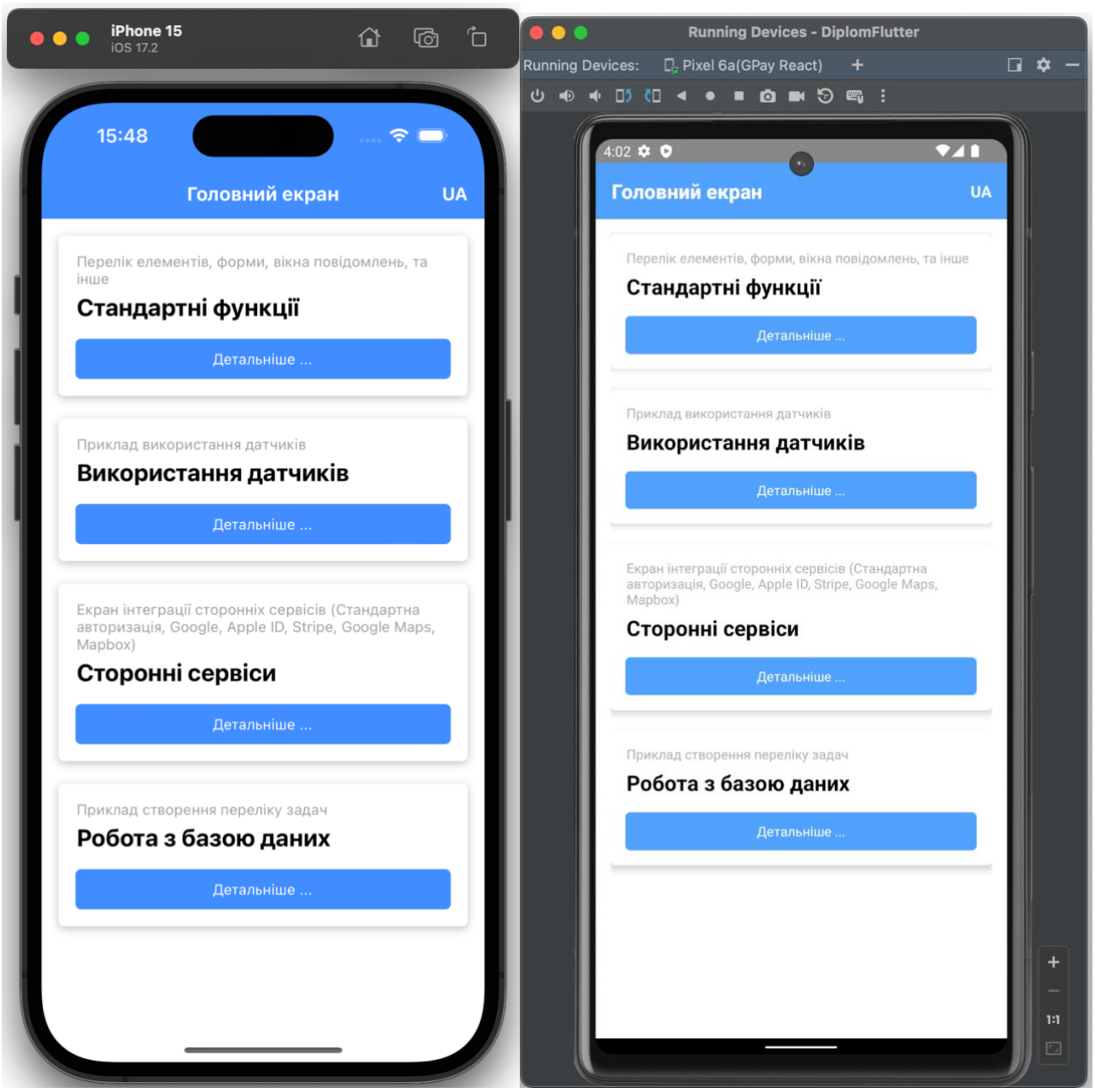

Рисунок 3.1 — Відображення головного екрану на платформі React Native для iOS та Android

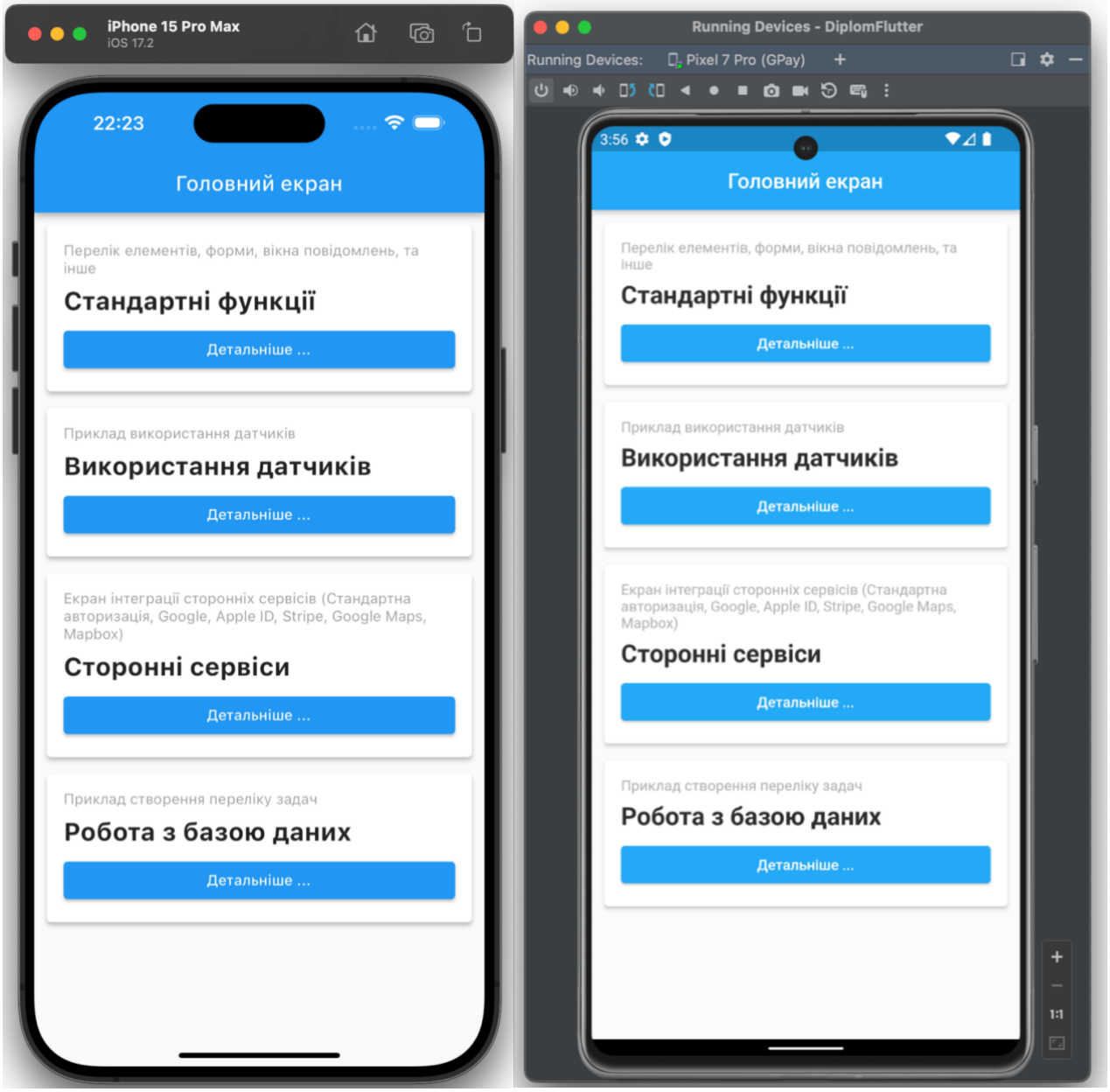

Рисунок 3.1 — Відображення головного екрану на платформі Flutter для iOS та Android

Для того щоб відобразити ці елементи було розроблено код для React Native приклад коду буде у лістингу 6, а для Flutter у лістингу 7.

Лістинг 6. Приклад реалізації головного екрану для React Native.

```
import React from 'react';
import { SafeAreaView, ScrollView, StatusBar, useColorScheme, View
} from 'react-native';
import styles from './Main.styles';
```

```
import { Colors } from 'react-native/Libraries/NewAppScreen';
import { Pages, PageItem } from '../../constants/pages'
import CardNavigation from '../../share/cart-navigate/cart-
navigate';
import { useTranslation } from "react-i18next";
import { StackScreenProps } from '@react-navigation/stack';
import { MainStackTypeParamList } from "../../../App";
type Props = StackScreenProps<MainStackTypeParamList, 'Home'>;

function MainPage ({ navigation }: Props) {
  const { t } = useTranslation();
  const isDarkMode: boolean = useColorScheme() === 'dark';
  const backgroundStyle = {
    backgroundColor: isDarkMode ? Colors.darker : '#fff',
  };

  const NavigateTo = (screen: any): void => {
    navigation.push(screen)
  };

  const dataSources: PageItem[] = Pages.main;

  return (
    <SafeAreaView style={[styles.safeAreaContainer,
backgroundStyle]}>
      <StatusBar barStyle={isDarkMode ? 'dark-content' : 'light-
content'} />
      <ScrollView style={styles.scrollStyle}>
        <View>
          {
            dataSources.map(
              (item, index) => (
                <CardNavigation
                  key={index}
                  onPress={() => NavigateTo(item.page)}
```

```
                    children={t(item.subtitle)}
                    title={ t(item.title)} />
                )
              )
            }
          </View>
        </ScrollView>
      </SafeAreaView>
    );
}

export default MainPage;
```

Лістинг 7. Приклад реалізації головного екрану для Flutter.

```
import 'package:example_app/generated/l10n.dart';
import
'package:example_app/pages/sensors_example/sensors_example.dart';
import
'package:example_app/pages/standard_functions/standard_functions_pa
ge.dart';
import
'package:example_app/pages/third_party_services/third_party_service
s_page.dart';
import
'package:example_app/pages/work_with_database/database_example_page
.dart';
import 'package:example_app/widgets/card_widget.dart';
import 'package:flutter/material.dart';

class MainPage extends StatelessWidget {
 const MainPage({super.key});

 static const path = '/main';

 @override
```

```
Widget build(BuildContext context) {
final navigator = Navigator.of(context);

return Scaffold(
appBar: AppBar(
title: Text(S.of(context).mainPage),
centerTitle: true,
),
body: ListView(
padding: EdgeInsets.all(4),
children: [
CardWidget(
title: S.of(context).standardFunctions,
description: S.of(context).listOfElements,
onPressed: () => navigator.pushNamed(StandardFunctionsPage.path),
),
CardWidget(
title: S.of(context).useOfSensors,
description: S.of(context).exampleOfUseOfSensors,
onPressed: () => navigator.pushNamed(SensorsExamplePage.path),
),
CardWidget(
title: S.of(context).thirdPartyServices,
description: S.of(context).thirdpartyServicesIntegrationScreen,
onPressed: () => navigator.pushNamed(ThirdPartyServicesPage.path),
),
CardWidget(
title: S.of(context).workingWithTheDatabase,
description: S.of(context).exampleOfCreatingAListOfTasks,
onPressed: () => navigator.pushNamed(DatabaseExamplePage.path),
),

],
),
);
```

```
  }
}
```

3.2.4 Розробка екрана стандартних елементів

У багатьох застосунках є багато елементів вводу та відображення вибору різних даних. Для розробки цього екрану було вирішено розбити його на два окремих екрана: прості елементи та вікна повідомлень.

На екрані “прості елементи” були використані елементи які є у самому фреймворку для вводу даних або їх обробки, або вибору, приклад відображення цих елементів на рисунку 3.3 для React Native та для Flutter на рисунку 3.4.

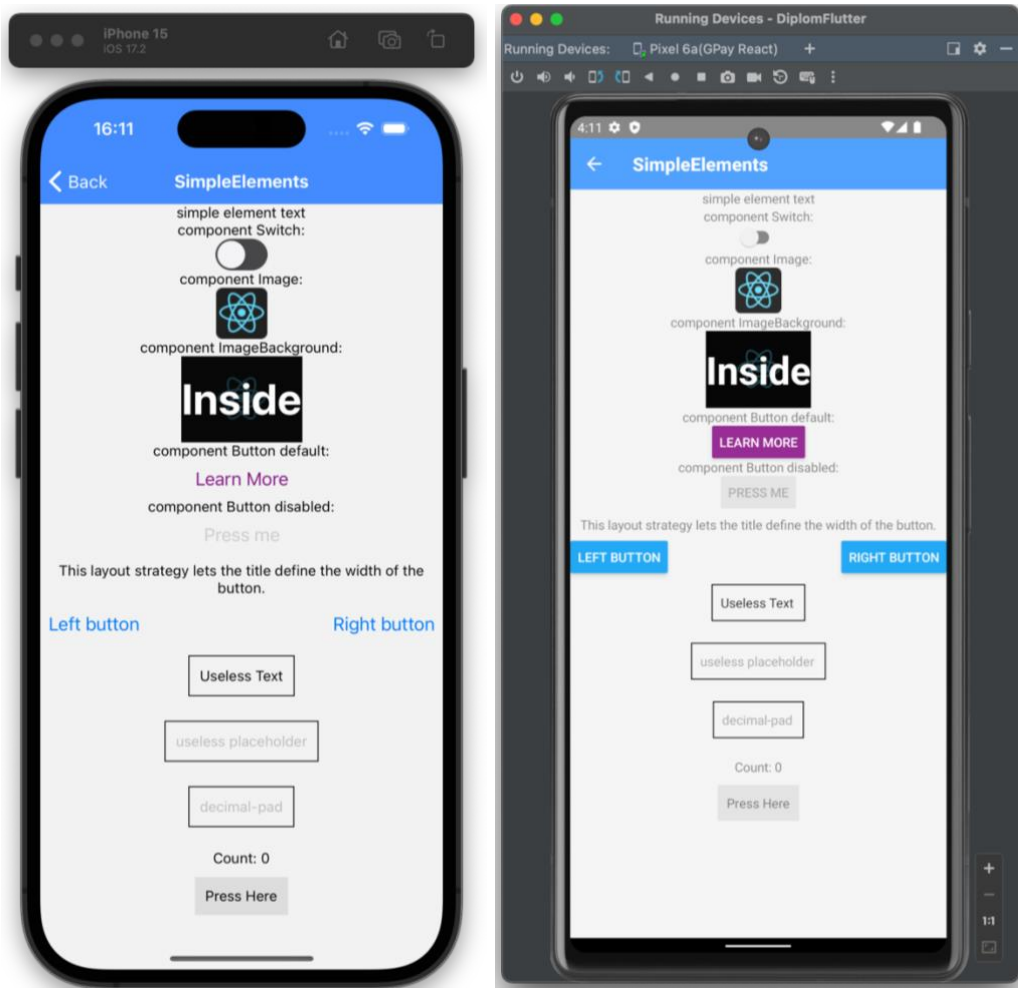

Рисунок 3.3 — Приклад відображення екрану “прості елементи” для React Native

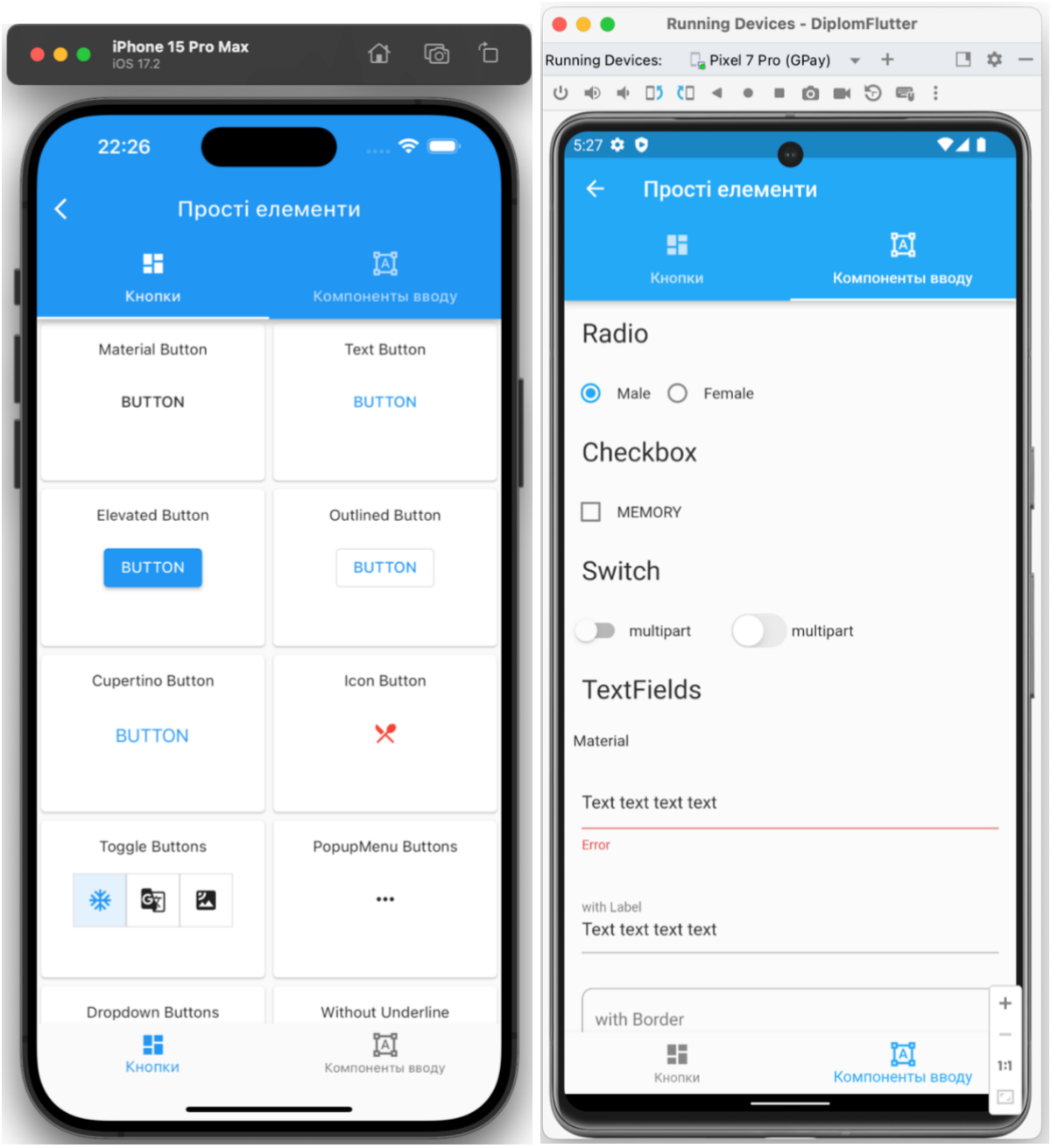

Рисунок 3.4 — Приклад відображення екрану “прості елементи” для Flutter

Приклад коду для цього відображено у лістингу 8 для React Native та лістингу 10 для Flutter.

Лістинг 8 Приклад коду “прості елементи” для React Native з імпортом.

```
import React, { useState } from 'react';
import { SafeAreaView, View, ScrollView, useColorScheme, StatusBar,
ActivityIndicator, Text, TouchableOpacity,
 Switch, Image, ImageBackground, Button, Alert, TextInput,
```

```
} from 'react-native';
import stylesSimpleElements from './SimpleElements'
import { useTranslation } from 'react-i18next';
import { Colors } from 'react-native/Libraries/NewAppScreen';
function SimpleElements (): React.JSX.Element {
  const { t } = useTranslation();
  const isDarkMode: boolean = useColorScheme() === 'dark';
  const backgroundStyle = {
    backgroundColor: isDarkMode ? Colors.darker : '#fff',
  };
  const [isEnabled, setIsEnabled] = useState(false);
  const toggleSwitch = () => setIsEnabled(previousState =>
!previousState);
  const image = {uri: 'https://legacy.reactjs.org/logo-og.png'};
  const [text, onChangeText] = React.useState('Useless Text');
  const [number, onChangeNumber] = React.useState('');
  const [count, setCount] = useState(0);
  const onPress = () => setCount(prevCount => prevCount + 1);
```

У цьому лістингу 8 приведено імпорті та константи.

Лістинг 9 Приклад коду “прості елементи” для React Native продовження .

```
return (
  <SafeAreaView>
    <StatusBar barStyle={isDarkMode ? 'dark-content' : 'light-
content'} />
    <ScrollView>
      <View style={stylesSimpleElements.container}>
        <Text>
          simple element text
        </Text>
      </View>
      <View style={stylesSimpleElements.container}>
        <Text>
          component Switch:
```

```
          </Text>
          <Switch
            trackColor={{false: '#767577', true: '#81b0ff'}}
            thumbColor={isEnabled ? '#f5dd4b' : '#f4f3f4'}
            ios_backgroundColor="#3e3e3e"
            onValueChange={toggleSwitch}
            value={isEnabled}
          />
        </View>
        <View style={stylesSimpleElements.container}>
          <Text>
            component Image:
          </Text>
          <Image
            style={stylesSimpleElements.tinyLogo}
            source={{
              uri: 'https://reactnative.dev/img/tiny_logo.png',
            }}
          />
        </View>
        <View style={stylesSimpleElements.container}>
          <Text>
            component ImageBackground:
          </Text>
          <ImageBackground source={image} resizeMode="cover"
style={stylesSimpleElements.image}>
            <Text style={stylesSimpleElements.text}>Inside</Text>
          </ImageBackground>
        </View>
.....
         <View style={stylesSimpleElements.container}>
            <View style={stylesSimpleElements.countContainer}>
              <Text>Count: {count}</Text>
            </View>
            <TouchableOpacity style={stylesSimpleElements.button}
```

```
onPress={onPress}>
            <Text>Press Here</Text>
          </TouchableOpacity>
        </View>
      </ScrollView>
    </SafeAreaView>
  )
}
```

У цьому коду немає усіх прикладів вводу даних у застосунку, але показано основний принцип.

Лістинг 10 Приклад коду “прості елементи” для Flutter.

```
import 'package:example_app/generated/l10n.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';

class WidgetsPage extends StatefulWidget {
 const WidgetsPage({Key? key}) : super(key: key);
 static const path = '/widgets';

 @override
 State<WidgetsPage> createState() => _WidgetsPageState();
}

class _WidgetsPageState extends State<WidgetsPage>
 with TickerProviderStateMixin {
 final _pages = [
 const ButtonsExampleWidget(),
 const InputsExampleWidget(),
 ];
 int _selectedIndex = 0;

 late final TabController _tabController;

 @override
```

```
void initState() {
super.initState();
_tabController = TabController(vsync: this, length: 3);
_tabController.addListener(() {
setState(() {
_selectedIndex = _tabController.index;
});
});
}
```

У цьому коду приведено імпорті та константи і стандартні елементи вводу та кнопки ще checkbox та інші.

3.2.5 Розробка екрану використання датчиків

Для реалізації цього екрану було обрано інтегрувати карту Google Maps з використанням GPS навігації.

У сучасних мобільних додатках локаційний сервіс з використанням карт Google Maps є важливою складовою для забезпечення користувачам можливості відстеження своєї геолокації. У даному дослідженні розглядається інтеграція карти Google Maps у фреймворках Flutter та React Native з використанням GPS-датчика для отримання актуальних географічних даних на операційних системах iOS та Android.

Для початку інтеграції карт Google Maps у фреймворк Flutter, необхідно встановити відповідний плагин, який надає доступ до функціоналу карт Google Maps. Плагін Flutter Google Maps дозволяє впроваджувати карти в мобільні додатки та працювати з ними на платформах iOS та Android.

Після встановлення плагина необхідно налаштувати його для використання GPS-датчика. Це може бути досягнуто шляхом використання вбудованих функцій Flutter для доступу до локаційних послуг. Додавання залежностей та налаштування дозволять отримувати дані про місцезнаходження користувача за допомогою GPS-датчика на пристрої.

Після успішної інтеграції карт Google Maps та GPS-датчика, додаток зможе відображати актуальну геолокацію користувача на карті Google Maps на основі отриманих даних з GPS-датчика. Це дозволить користувачам взаємодіяти з картами та використовувати їх для навігації та відстеження своєї реальної географічної позиції в мобільному додатку на платформах iOS та Android.

Цей підхід до інтеграції карт Google Maps та використання GPS-датчика у фреймворку Flutter створює можливості для створення функціональних та інтуїтивно зрозумілих мобільних додатків з високоякісними сервісами локаційного відстеження для користувачів.

Приклад встановлення залежностей наведено у лістингу 11.

Лістинг 11 Встановлення залежностей

```
import 'dart:async';
import 'package:app_settings/app_settings.dart';
import 'package:example_app/generated/l10n.dart';
import 'package:flutter/material.dart';
import 'package:geolocator/geolocator.dart';
import 'package:google_maps_flutter/google_maps_flutter.dart';
```

Для розробки у Flutter використовують підхід створення віджетів тому для реалізації наприклад відображення карти було створено віджет лістинг якого наведено у лістингу 12

Лістинг 12

```
@override
Widget build(BuildContext context) {
 return Scaffold(
 appBar: AppBar(
      title: Text(S.of(context).useOfSensors),
      centerTitle: true,
 ),
 body: GoogleMap(
```

```
        myLocationButtonEnabled: true,
        myLocationEnabled: true,
        mapType: MapType.normal,
        initialCameraPosition: _kyivCenter,
        minMaxZoomPreference: const MinMaxZoomPreference(5, 18),
        onMapCreated: (GoogleMapController controller) async {
        _controller = controller;
        checkLocationPermissions()
        .then((value) => _checkLocationPermissionsResultOnInit());
        },
 ),
 );
}
```

Для початку інтеграції карт Google Maps в додаток React Native необхідно встановити відповідний пакет, який забезпечить доступ до функціоналу карт Google Maps. Найчастіше для цього використовують плагін "react-native-maps", який дозволяє легко впроваджувати та взаємодіяти з картами Google Maps в додатках React Native.

Після встановлення плагіна необхідно налаштувати його для використання GPS-датчика. Це досягається за допомогою використання вбудованих API для доступу до локаційних послуг у React Native. Це дозволяє отримувати дані про геолокацію користувача на пристрої через GPS-датчик.

Після успішної інтеграції карт Google Maps та GPS-датчика, додаток може відображати актуальну геолокацію користувача на карті Google Maps на основі отриманих даних з GPS-датчика. Це дозволяє користувачам взаємодіяти з картами та використовувати їх для навігації та відстеження їх реальної географічної позиції у мобільному додатку на платформах iOS та Android.

Цей підхід до інтеграції карт Google Maps та використання GPS-датчика у фреймворку React Native створює можливості для створення функціональних та інтуїтивно зрозумілих мобільних додатків з високоякісними сервісами локаційного відстеження для користувачів.

Приклад реалізації цього екрану відображається на рисунку 3.5. для React Native та екран реалізований на Flutter на рисунку 3.6

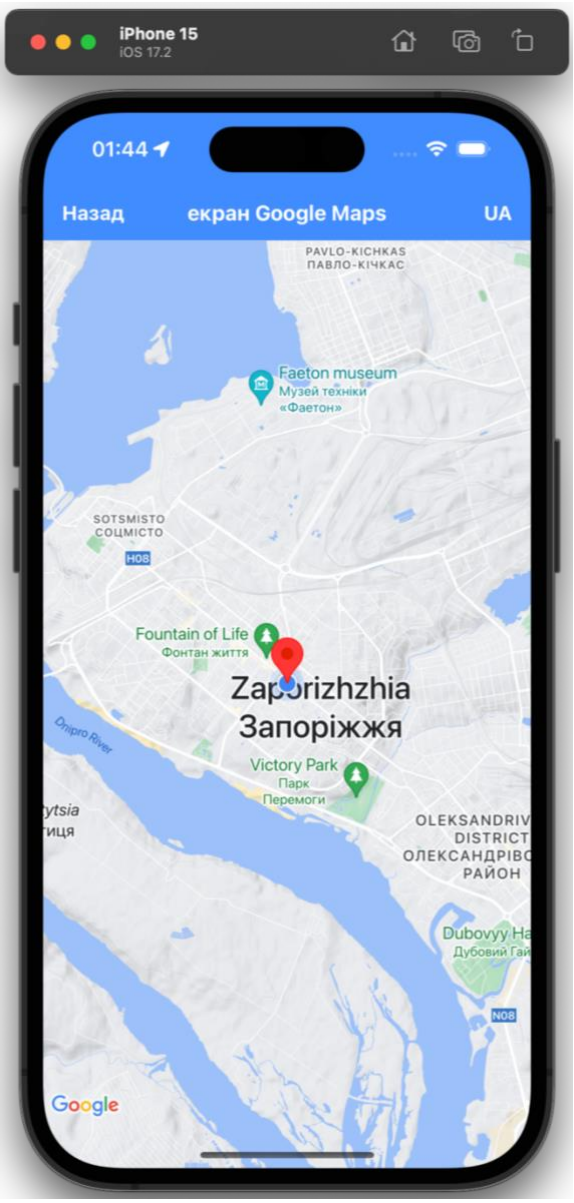

Рисунок 3.5 — Приклад відображення екрану “використання датчиків” для React Native

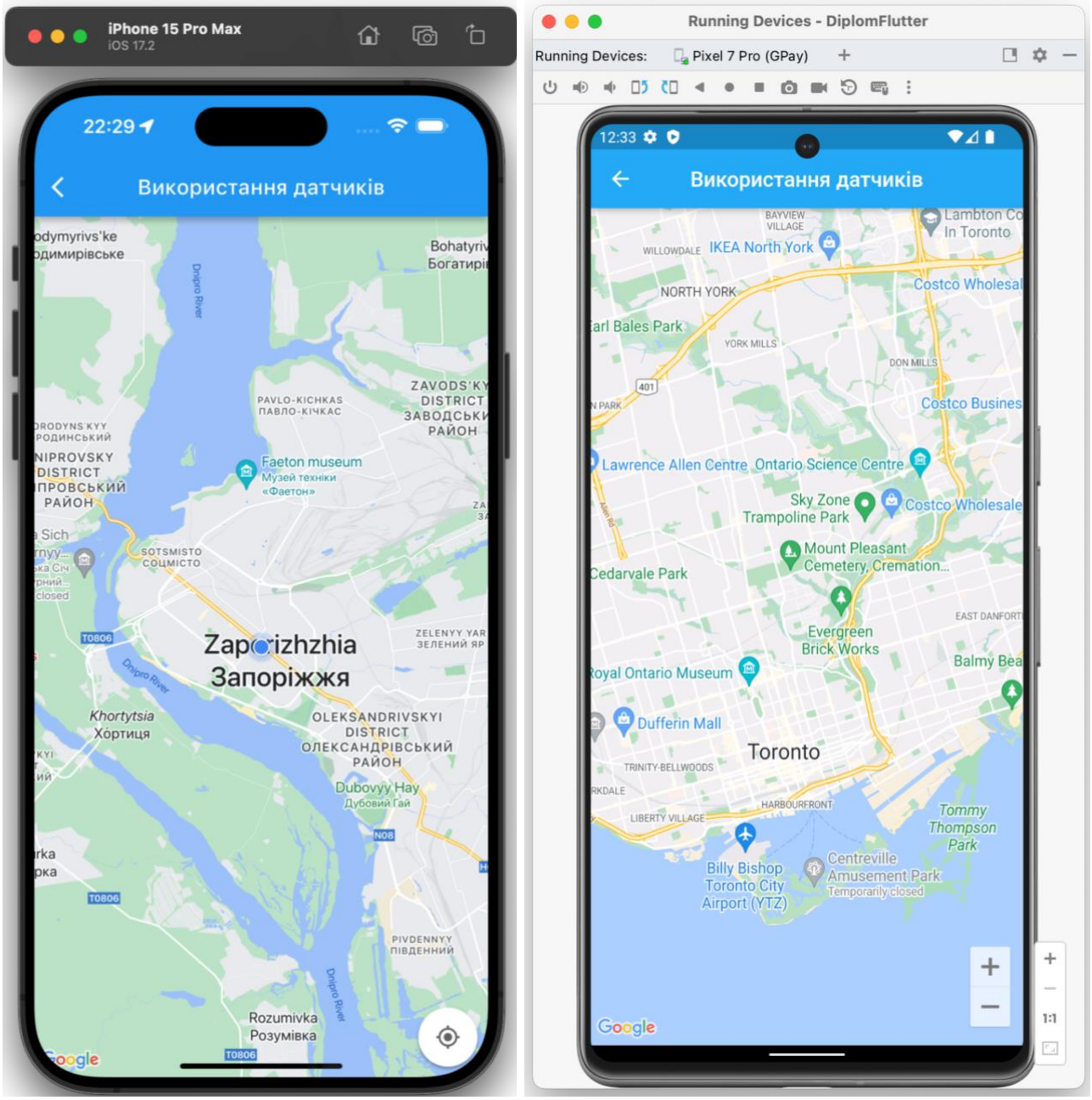

Рисунок 3.6 — Приклад відображення екрану “використання датчиків” для Flutter

3.2.6 Розробка екрану “Сторонні сервіси”.

Більшість додатків використовують бібліотеки та компоненти, які не є частиною фреймворку, на якому вони розроблені. Для втілення цього екрану було вирішено здійснити інтеграцію платіжного сервісу Stripe та картографічних сервісів Google Maps та Mapbox.

Документація Stripe зазначає, що всі платежі повинні бути підтверджені лише на бекенді [17]. Для реалізації цього було створено дві функції Firebase, де

відбувається підтвердження платежу як для карток, так і для сервісів Apple Pay та Google Pay. Ці функції становлять частину бекенду, розташованого на серверах Firebase. Приклади цих функцій можна побачити у лістингу 13 для оплати картками та у лістингу 14 для оплати через сервіси Apple Pay та Google Pay.

Лістинг 13 Приклад функції Firebase для платіжного сервісу Stripe для банківських кратоk.

```
app.post("/create-payment-intent", async (req: express.Request,
res) => {
  const stripe = new Stripe(process.env.STRIPE_SECRET_KEY as
string, {
    apiVersion: "2023-08-16",
    typescript: true,
    stripeAccount: "acct_XXXXXX",
  });

  const paymentIntent = await stripe.paymentIntents.create({
    amount: req.body.amount,
    currency: req.body.currency,
    payment_method_types: ["card"],
  });

  const clientSecret = paymentIntent.client_secret;

  if (clientSecret) {
    res.status(200).json({clientSecret});
  }
  const errorPay = {
    status: false,
    code: "404",
    message: "Error payment clientSecret",
  };
  res.status(404).json({errorPay});
});
```

Лістинг 14 Приклад функції Firebase для платіжного сервісу Stripe для сервісів Appele Pay та Google Pay.

```
app.post("/pay", async (req: express.Request, res) => {
  const stripe = new Stripe(process.env.STRIPE_SECRET_KEY as
string, {
    apiVersion: "2023-08-16",
    typescript: true,
    stripeAccount: "acct_XXXXX",
  });
  const customer = await stripe.customers.create();
  const ephemeralKey = await stripe.ephemeralKeys.create(
    {customer: customer.id},
    {apiVersion: "2023-08-16"},
  );
  const paymentIntent = await stripe.paymentIntents.create({
    amount: req?.body?.amount,
    currency: req?.body?.currency,
    customer: customer.id,
    automatic_payment_methods: {
      enabled: true,
    },
  });
  res.status(201).json({
    clientSecret: paymentIntent.client_secret,
    ephemeralKey: ephemeralKey.secret,
    customer: customer.id,
  });
});
```

Розробка самих застосунків для використання платіжного сервісу Stripe у кожному фреймворку відбувається по різному. У документації Stripe зазначено багато прикладів як можна інтегрувати цю платіжну систему для обох фреймворків. Приклади відображення у застосунках надаються на рисунку 3.7 для

React Native та на рисунку 3.8 для Flutter. Приклади інтеграції коду приведені у лістингах, для реалізації React Native у лістингу 15, а для реалізації Flutter у лістингу 16 та 17.

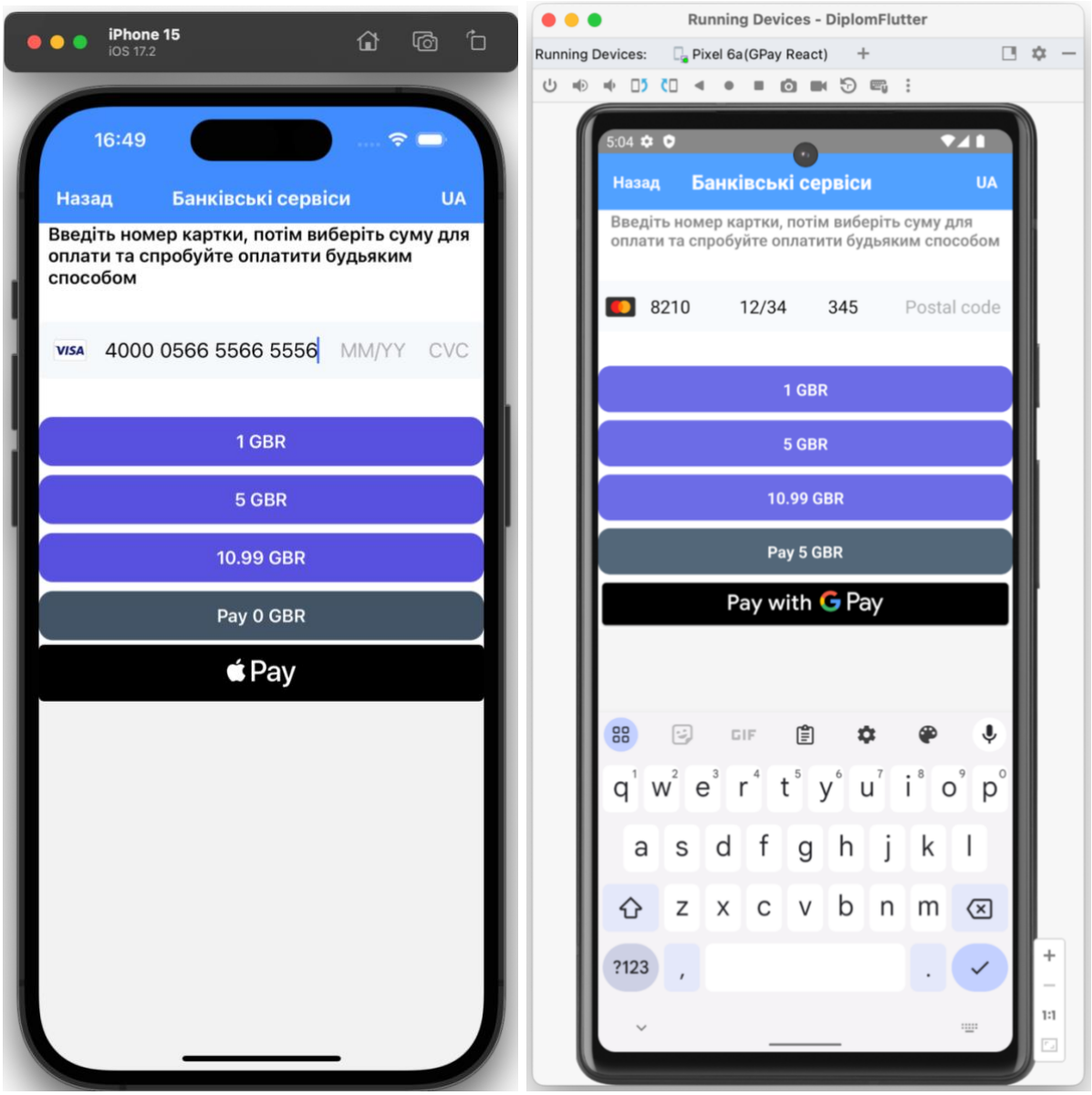

Рисунок 3.7 — Приклад реалізації інтеграції Stripe для React Native

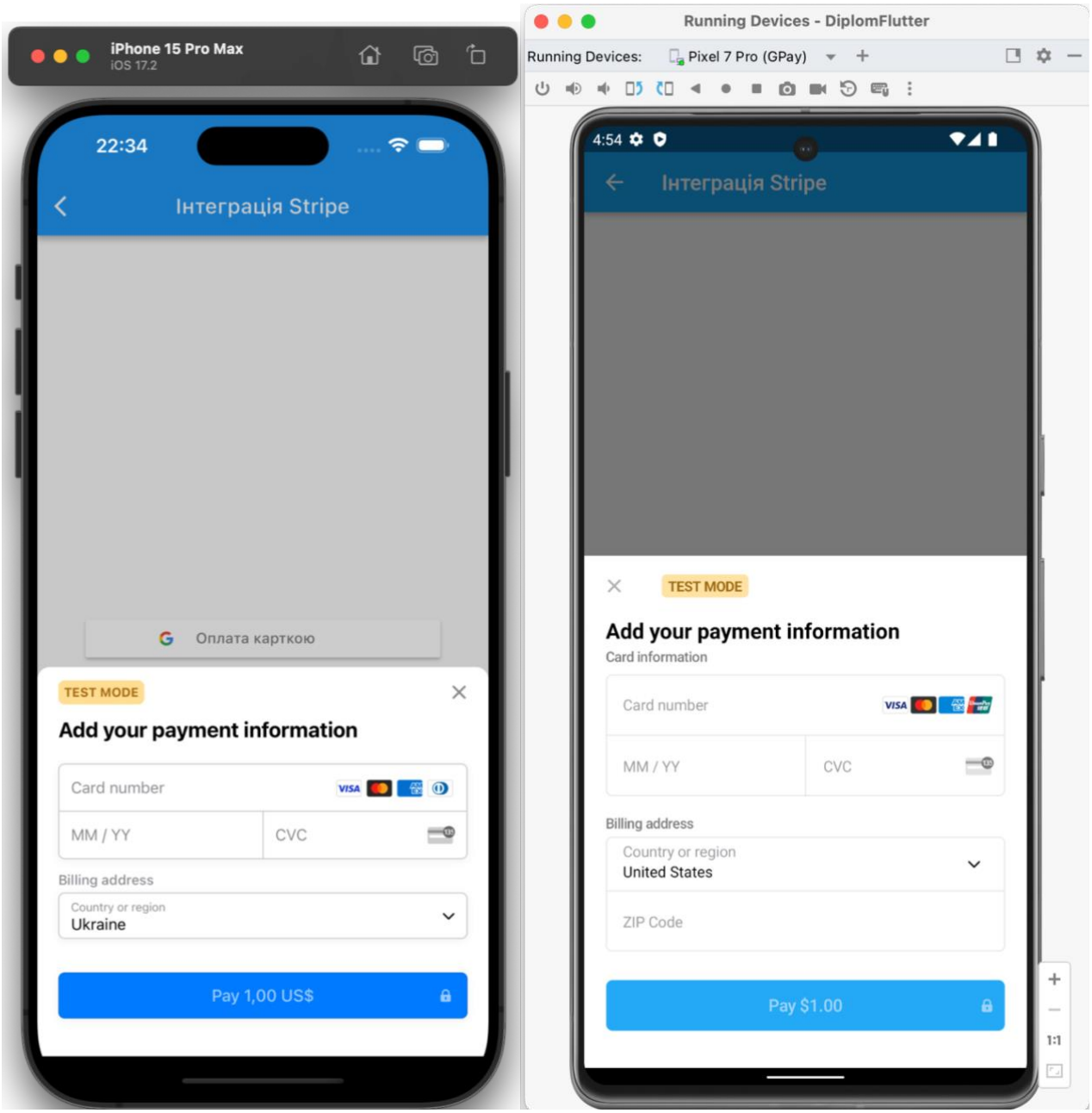

Рисунок 3.8 — Приклад реалізації інтеграції Stripe для Flutter

Лістинг 15. Приклад інтеграції Stripe для React Native.

```
function MainStripeScreen(): React.JSX.Element {
  const { t } = useTranslation();
  const isDarkMode: boolean = useColorScheme() === 'dark';
  const backgroundStyle = {
    backgroundColor: isDarkMode ? Colors.darker : '#fff',
  };
  const [publishableKey, setPublishableKey] = useState('');
  const [amount, setAmount] = useState(0);
```

```
  useEffect(() => {
    fetchPublishableKey();
  }, []);

  const fetchPublishableKey = async () => {
    const key: string = process.env.STRIPE_PUBLIC_KEY;
    setPublishableKey(key);
  };

  const setAmountSelect = (value: number) => {
    console.info('setAmountSelect: ', value);
    setAmount(value);
  }

  return (
    <SafeAreaView style={[backgroundStyle]}>
      <StatusBar barStyle={isDarkMode ? 'dark-content' : 'light-
content'} />
      <ScrollView>
        <View style={stylesSimpleElements.container}>
          <Text style={{fontWeight: '600', fontSize: 16}}>
            { t('Payment text') }
          </Text>
        </View>
        <StripeProvider
          publishableKey={publishableKey}
          merchantIdentifier="stripeDemo://openApp" // required for
Apple Pay
          urlScheme="merchant.com.demo" // required for 3D Secure
and bank redirects
        >
          <PaymentScreen sendAmount={setAmountSelect}/>
          <GooglePayPaymentScreen amount={amount}/>
        </StripeProvider>
```

```
      </ScrollView>
    </SafeAreaView>
  );
}
```

Лістинг 16. Приклад інтеграції Stripe для Flutter.

```
class _StripeExamplePageState extends State<StripeExamplePage> {
 final _formKey = GlobalKey<FormState>();

 final kStripePublishableKey = 'pk_test_5XXXXXXXX';
 final kStripeSecretKey = 'sk_test_XXXXXXXX';

 @override
 void initState() {
 // TODO: implement initState
 super.initState();

 WidgetsBinding.instance.addPostFrameCallback((timeStamp) async {
 Stripe.publishableKey = kStripePublishableKey;
 // Stripe.merchantIdentifier = 'merchant.flutter.stripe.test';
 // Stripe.urlScheme = 'flutterstripe';
 await Stripe.instance.applySettings();
 });
 }

 @override
 Widget build(BuildContext context) {
 return Scaffold(
 appBar: AppBar(title: const Text('Інтеграція Stripe') ), //
Text(S.of(context).authorization)
 body: Center(
 child: Form(
 key: _formKey,
 child: Padding(
 padding: const EdgeInsets.symmetric(horizontal: 46),
```

```
child: Column(
mainAxisAlignment: MainAxisAlignment.center,
children: <Widget>[
// const Expanded(child: SizedBox()),
SizedBox(
width: double.infinity,
child: SignInButton(
Buttons.Google,
text: "Оплата карткою",
onPressed: makeCardPayment,
),
),

if (Platform.isAndroid)
SizedBox(
width: double.infinity,
child: SignInButton(
Buttons.Google,
text: "+ GooglePay",
onPressed: makeGPayment,
),
),
// const SizedBox(height: 30),

if (Platform.isIOS)
SizedBox(
width: double.infinity,
child: SignInButton(
Buttons.Apple,
text: "+ ApplePay",
onPressed: makeAPayment,
),
),
],
),
```

```
 ),
 ),
 ),
 );
 }
```

Лістинг 17. Продовження прикладу інтеграції Stripe для Flutter.

```
Future<void> makeCardPayment() async {
 try {
 final paymentIntent = await createPaymentIntent('100', 'USD');

 await Stripe.instance
 .initPaymentSheet(
 paymentSheetParameters: SetupPaymentSheetParameters(
 appearance: const PaymentSheetAppearance(
 primaryButton: PaymentSheetPrimaryButtonAppearance(
 colors: PaymentSheetPrimaryButtonTheme(
 light: PaymentSheetPrimaryButtonThemeColors(
 background: Colors.blue,
 ),
 ),
 ),
 ),
 paymentIntentClientSecret: paymentIntent!['client_secret'],
 style: ThemeMode.light,
 merchantDisplayName: 'Merchant Display Name',
 ),
 )
 .then((value) {});

 displayPaymentSheet();
 } catch (err) {
 throw Exception(err);
 }
}
```

```
Future<void> makeGPayment() async {
 try {

 final paymentIntent = await createPaymentIntent('100', 'USD');

 await Stripe.instance
 .initPaymentSheet(
 paymentSheetParameters: SetupPaymentSheetParameters(
 appearance: const PaymentSheetAppearance(
 primaryButton: PaymentSheetPrimaryButtonAppearance(
 colors: PaymentSheetPrimaryButtonTheme(
 light: PaymentSheetPrimaryButtonThemeColors(
 background: Colors.blue,
 ),
 ),
 ),
 ),
 paymentIntentClientSecret: paymentIntent!['client_secret'],

 googlePay: PaymentSheetGooglePay(
 merchantCountryCode: 'US',
 testEnv: true,
 ),
 style: ThemeMode.light,
 merchantDisplayName: 'Merchant Display Name',
 ),
 )
 .then((value) {});
}
}
```

Далі були розроблені екрани для відображення картографічних сервісів Google Maps та Mapbox. Приклад розробки для Google Maps такий же як і для екрану “використання датчиків” різниця у тому що на цьому екрані немає опції

використання датчиків GPS. Тут зосередимося на розробці екрану з використанням сервісу Mapbox.

Для реалізації коду інтеграції картографічного сервісу Mapbox потрібно було зареєструватися на офіційному сайті Mapbox та створити токени доступу. Для реалізації на фреймворку React Native потрібно встановити бібліотеку @rnmapbox/maps та додати ключ до настроєк. Приклад реалізації коду приведен у лістингу 18.

Лістинг 18 Приклад коду реалізації інтеграції Mapbox для React Native

```
import React, { useEffect } from 'react';
import { StyleSheet, Text, View } from 'react-native';
import MapboxGL from '@rnmapbox/maps';

const tokenMapbox = ' <YOUR_ACCESSTOKEN>';

MapboxGL.setAccessToken(tokenMapbox).then();

const styles = StyleSheet.create({
  page: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    backgroundColor: "#F5FCFF"
  },
  container: {
    height: '100%',
    width: 375,
    backgroundColor: "tomato"
  },
  map: {
    flex: 1
  }
});
```

```
const Mapbox = () => {
  useEffect(() => {
    MapboxGL.setTelemetryEnabled(false);
  }, []);

  return (
    <View style={styles.page}>
      <View style={styles.container}>
        <MapboxGL.MapView style={styles.map} >
          <MapboxGL.Camera zoomLevel={12}
centerCoordinate={[35.1325, 47.8422]} />
        </MapboxGL.MapView>
      </View>
    </View>
  );
};
```

Крім цього коду у частині JavaScript ще треба додавати код у нативну частину наприклад для реалізації iOS частини не требо нічого додавати, а ось до частини Android токен у файл AndroidManifest.xml це буде виглядати як показано у лістингу 19.

Лістинг 19 Прикладу додавання токену до файлу AndroidManifest.xml.

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.example.example_app">
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
    <application
```

```
        <meta-data
                android:name="setAccessToken"
                android:value="sk.eXXXXXXXX" />
    </application>
</manifest>
```

Для реалізації на фреймворку Flutter потрібно додати залежність до файлу pubspec.yaml:

dependencies:

mapbox_gl: ^0.16.0

Також додати импорти та залежності.

Приклад самого класу який відображає картографічний сервіс наведено у лістингу 20.

Лістинг 20 Приклад коду для реалізації Mapbox для Flutter

```
class _MapboxPageState extends State<MapboxPage>
 with AutomaticKeepAliveClientMixin {
 @override
 bool get wantKeepAlive => true;
 @override
 Widget build(BuildContext context) {
 super.build(context);
 return Stack(
 children: [
 Positioned.fill(
 child: FlutterMap(
 options: MapOptions(
 minZoom: 5,
 maxZoom: 18,
 zoom: 12,
 center: LatLng(50.444798, 30.521361),
 ),
 children: [
 TileLayer(
```

```
 urlTemplate:

"https://api.mapbox.com/styles/v1/{userId}/{mapStyleId}/tiles/256/{
z}/{x}/{y}@2x?access_token={accessToken}",
 additionalOptions: {
 'userId':'<Id User>v',
 'mapStyleId': ' <YOUR_STYLE>’ ',
 'accessToken': ' <YOUR_ACCESSTOKEN>’ ',
 },
 ),
 ],
 ),
 ),
 ],
 );
 }
}
```

Кінцевий результат обох фреймворків можна побачити на рисунку 3.9 для React Native та на рисунку 3.10 для Flutter.

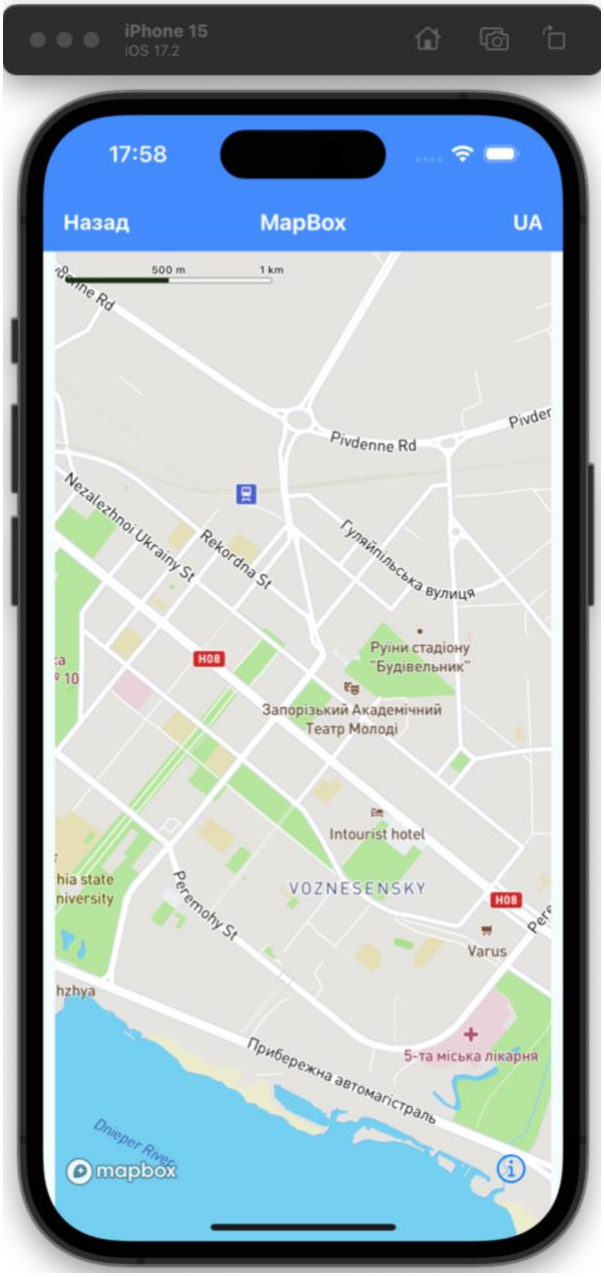

Рисунок 3.9 — Приклад інтеграції Mapbox для React Native

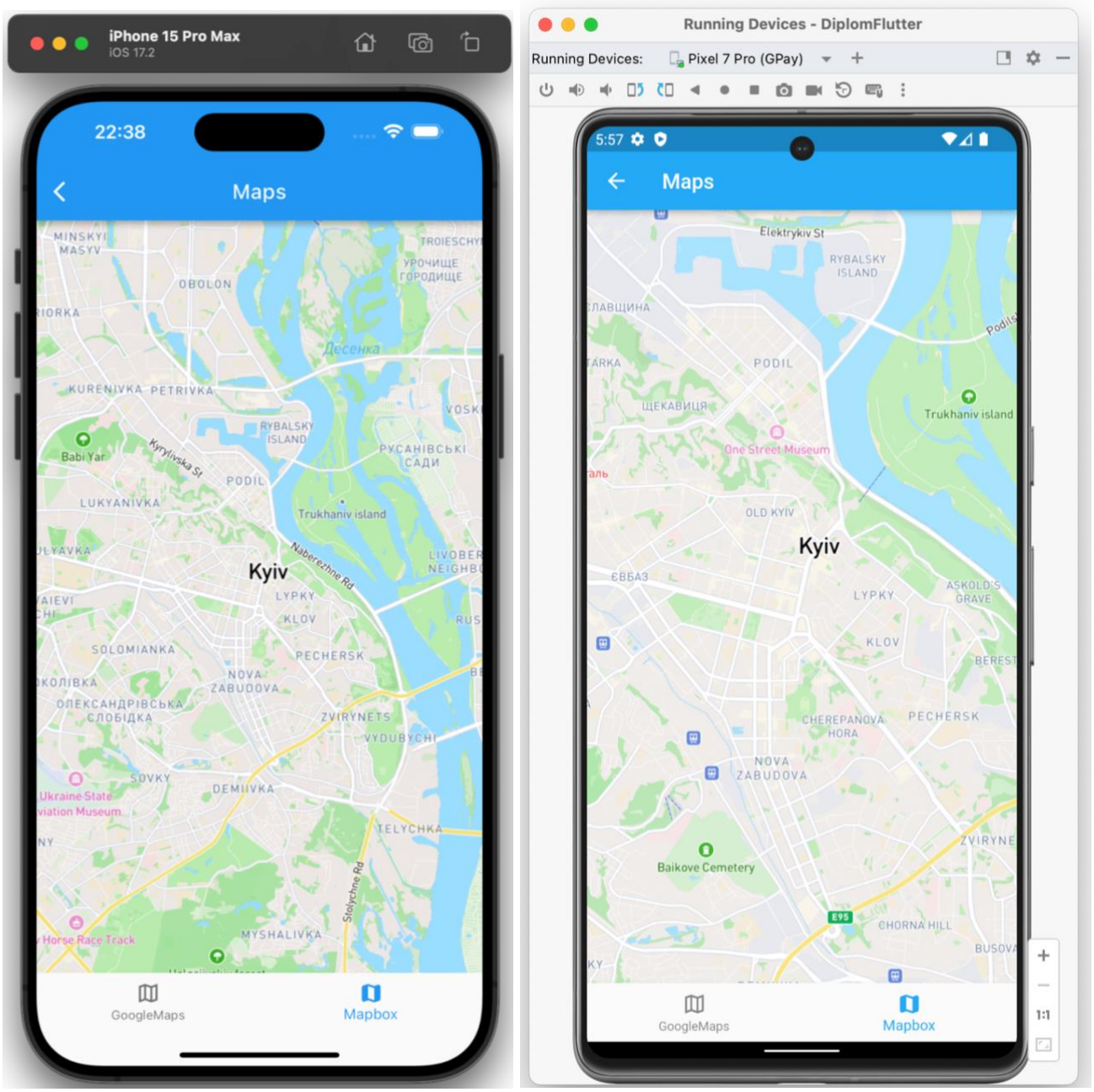

Рисунок 3.10 — Приклад інтеграції Mapbox для Flutter

3.2.7 Розробка екрану “Робота з базою даних”

Для реалізації цього екрану було створено базу даних у Firebase Cloud Firestore яка знаходиться у хмарі. Дані цієї базі доступні у обох додатках розроблених різними фреймворками. Сама база складається з одної таблиці та має вигляд як зображено на рисунку 3.11.

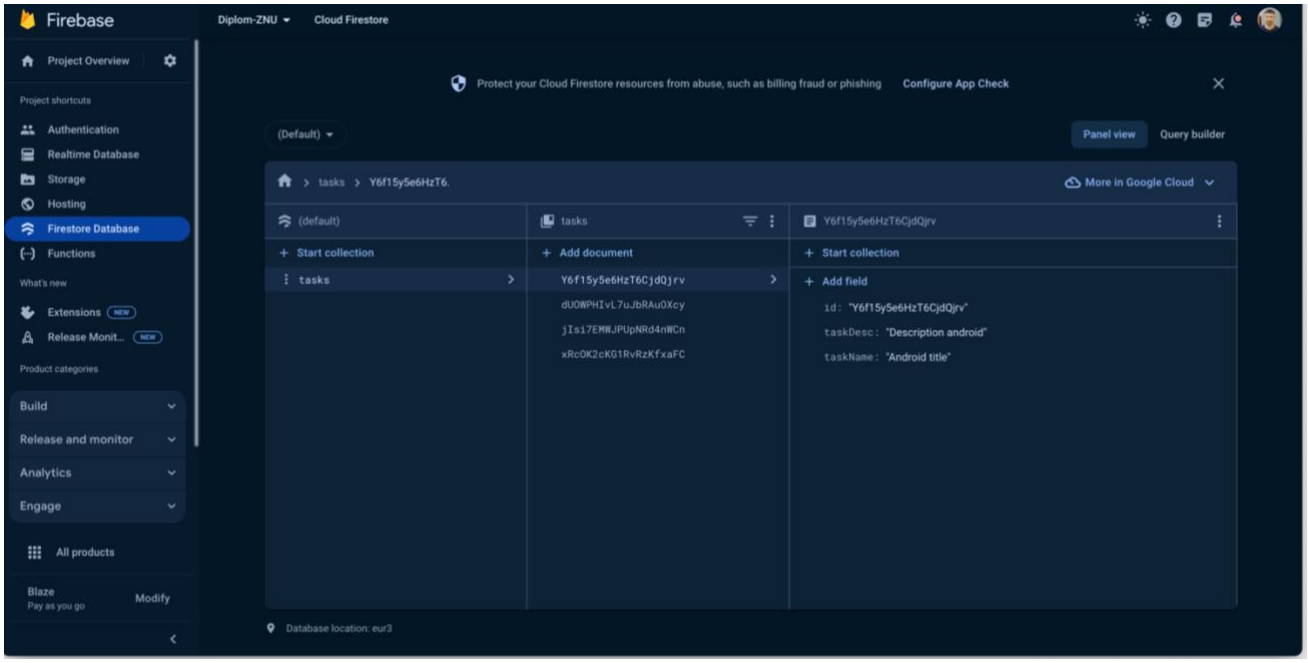

Рисунок 3.11 — Приклад бази даних у Firebase Cloud Firestore

Як видно з рисунку у цієї бази є три поля даних які мають тип “string”: id, taskDesc та taskName. Приклад використання бази даних реалізується на принципі додавання нотаток до бази з можливістю редагувати та виділяти їх з бази, крім того відображається увесь список цих нотаток.

Для реалізації коду для React Native було встановлено пакет @react-native-firebase/firestore та увесь код було розділено на декілька функцій:

- функція додавання та редагування запису;
- функція виділення запису;
- функція яка відображає усі записи та має посилання на інші функції.

Приклад коду представлено частково у лістингу з 20 по 22 де відображається основна частина де відображається список усіх нотаток.

Лістинг 20. Приклад коду відображення списку нотаток

```
function DatabaseScreen(): React.JSX.Element {
    const isDarkMode: boolean = useColorScheme() === 'dark';
    const backgroundStyle = {
        backgroundColor: isDarkMode ? Colors.darker : '#fff',
    };
```

```
    const [showLoading, setShowLoading] = useState(false);
    const [dataArray, setDataArray] = useState<TaskTodo[]>([]);
    const [dataItem, setDataItem] = useState<TaskTodo>({
      idN: 'generate',
      id: 'generate',
      taskDesc: 'task Desc:',
      taskName: 'task Name:',
    });
    const [isPopupVisible, setPopupVisible] = useState(false);
    const [isAddItemVisible, setAddItemVisible] = useState(false);
    const [isUpdateItemVisible, setUpdateItemVisible] =
useState(false);

    const togglePopup = (element: TaskTodo) => {
        setPopupVisible(!isPopupVisible);
        if (isPopupVisible) {
          return;
        }
        setDataItem({...element})
    };
```

Нажаль увесь код дуже великий тому продовження у наступному лістингу.

Лістинг 21 Продовження прикладу коду відображення списку нотаток

```
const toggleAddItem = () => {
    setAddItemVisible(!isAddItemVisible);
    setUpdateItemVisible(false)
    if (!isAddItemVisible) {
      setDataItem({
        idN: '',
        id: '',
        taskDesc: '',
        taskName: '',
      })
    }
```

```
};

const openUpdateItem = (data: TaskTodo) => {
  console.log('openUpdateItem data: ', data);
  setDataItem(data);
  setUpdateItemVisible(true);
  setAddItemVisible(true)
}

useEffect(() => {
    const unsubscribe = firestore()
        .collection('tasks')
        .onSnapshot(querySnapshot => {
            const newData: TaskTodo[] = [];
            setShowLoading(true);
            querySnapshot.forEach(documentSnapshot => {
                const element: TaskTodo = {
                    idN: documentSnapshot.id,
                    id: documentSnapshot.data().id,
                    taskDesc: documentSnapshot.data().taskDesc,
                    taskName: documentSnapshot.data().taskName,
                }
                newData.push({...element});
            });
            setDataArray(newData);
            setShowLoading(false);
        });
    return () => unsubscribe();
}, []);
```

Лістинг 22 Продовження прикладу коду відображення списку нотаток

```
return (
        <SafeAreaView style={[backgroundStyle]}>
            <StatusBar barStyle={isDarkMode ? 'dark-content' :
'light-content'} />
```

```
            <ScrollView style={styles.container}>
                <View>
                    {dataArray.map(item => (
                      <ElementItem key={item.idN}
                                   item={item}
                                   togglePopup={togglePopup} />
                    ))}
                </View>
                {showLoading &&
                    <View style={styles.activity}>
                        <ActivityIndicator size="large"
color="#0000ff" />
                    </View>
                }
            </ScrollView>
            <TouchableOpacity style={styles.buttonBlock}
onPress={toggleAddItem}>
                <Text style={styles.buttonText}>+</Text>
            </TouchableOpacity>
            <Popup
                 visible={isPopupVisible}
                 onClose={togglePopup}
                 item={dataItem}
                 update={openUpdateItem}     />
            <AddElementItem
                            visible={isAddItemVisible}
                            onClose={toggleAddItem}
                            data={dataItem}
                            edit={isUpdateItemVisible} />
        </SafeAreaView>
    )
}
```

```
export default DatabaseScreen;
```

Для реалізації коду для Flutter увесь код було розділено на декілька частин:

- додавання та редагування запису;
- віджато виділення запису;
- віджато яка відображає усі записи та має посилання на інші функції.

Приклад коду представлено частково у лістингу з 23 по 25 де відображається основна частина де відображається список усіх нотаток.

Лістинг 23. Приклад коду відображення списку нотаток відображення импортів.

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:example_app/generated/l10n.dart';
import
'package:example_app/pages/work_with_database/dialogs/add_task_dial
og.dart';
import
'package:example_app/pages/work_with_database/dialogs/delete_task_d
ialog.dart';
import
'package:example_app/pages/work_with_database/dialogs/update_task_d
ialog.dart';
import 'package:flutter/material.dart';

class DatabaseExamplePage extends StatefulWidget {
 const DatabaseExamplePage({super.key});

 static const path = '/databaseExample';

 @override
 _DatabaseExamplePageState createState() =>
```

```
_DatabaseExamplePageState();
}
```

Увесь код дуже великий тому я розділив його наступна частина лістинг 24.

Лістинг 24. Приклад коду відображення списку нотаток продовження.

```
class _DatabaseExamplePageState extends State<DatabaseExamplePage>
{
 final fireStore = FirebaseFirestore.instance;

 @override
 Widget build(BuildContext context) {
 return Scaffold(
 appBar: AppBar(
 title: Text(S.of(context).workingWithTheDatabase),
 centerTitle: true,
 ),
 body: StreamBuilder<QuerySnapshot>(
 stream: fireStore.collection('tasks').snapshots(),
 builder: (context, snapshot) {
 if (!snapshot.hasData) {
 return Center(child: Text(S.of(context).taksIsEmpty));
 } else {
 final items = snapshot.data!.docs;

 if (items.isEmpty) {
 return Center(child: Text(S.of(context).taksIsEmpty));
 }

 return ListView.builder(
 itemCount: items.length,
 padding: const EdgeInsets.all(8),
 itemBuilder: (BuildContext context, int index) {
 final task = TaskModel.fromJson(
 items[index].data()! as Map<String, dynamic>);
```

```
return Card(
child: Padding(
padding: const EdgeInsets.all(16),
child: Row(
crossAxisAlignment: CrossAxisAlignment.start,
children: [
Expanded(
child: Column(
children: [
Row(
children: [
Expanded(
child: Text(
task.taskName,
style: TextStyle(
fontSize: 18,
fontWeight: FontWeight.bold,),
)),
],
),
SizedBox(height: 12),
Row(
children: [
Expanded(
child: Text(
task.taskDesc,
maxLines: 2,
overflow: TextOverflow.ellipsis,
),
),
],
),
],
),
```

```
),
PopupMenuButton(
padding: EdgeInsets.zero,
itemBuilder: (context) {
return [
PopupMenuItem(
value: 'edit',
child: Text(
S.of(context).edit,
style: TextStyle(fontSize: 13.0),
),
onTap: () => _showUpdateTaskDialog(task),
),
PopupMenuItem(
value: 'delete',
child: Text(
S.of(context).delete,
style: TextStyle(fontSize: 13.0),
),
onTap: () => _showDeleteTaskDialog(task),
),
];
},
),
],
),
),
);
},
);
}
},
),
floatingActionButton: FloatingActionButton(
onPressed: _showTaskCreationDialog,
```

```
 child: Icon(Icons.add),
 ),
 );
 }
```

Лістинг 25. Приклад коду відображення списку нотаток продовження.

```
Future<void> _showTaskCreationDialog() async {
 await Future.delayed(
 const Duration(seconds: 0),
 () => showDialog(
 context: context,
 builder: (context) => const AddTaskAlertDialog(),), ); }

 Future<void> _showUpdateTaskDialog(TaskModel task) async {
 await Future.delayed(
 const Duration(seconds: 0),
 () => showDialog(
 context: context,
 builder: (context) => UpdateTaskAlertDialog(
 task: task,), ), ); }

 Future<void> _showDeleteTaskDialog(TaskModel task) async {
 await Future.delayed(
 const Duration(seconds: 0),
 () => showDialog(
 context: context,
 builder: (context) => DeleteTaskDialog(task: task), ), ); }
}

class TaskModel {
 TaskModel({
 required this.taskId,
 required this.taskName,
```

```
 required this.taskDesc,
 });

 factory TaskModel.fromJson(Map<String, dynamic> data) {
return TaskModel(
 taskId: (data['id']),

 taskName: (data['taskName']),
 taskDesc: (data['taskDesc']), ); }

 String taskId;
 String taskName; String taskDesc;}
```

Кінцевий результат обох фреймворків можна побачити на рисунку 3.12 для React Native та на рисунку 3.13 для Flutter.

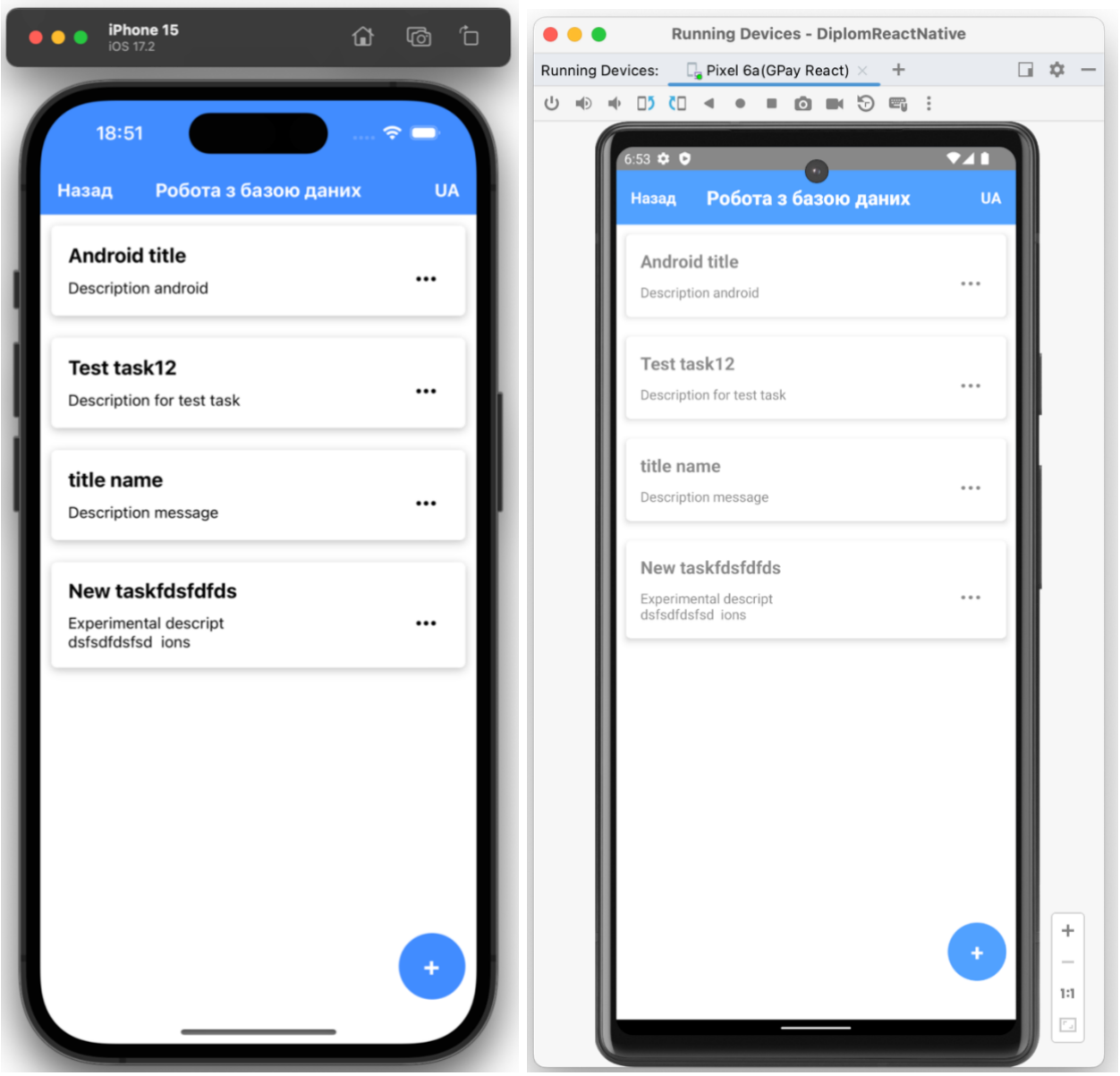

Рисунок 3.11 — Приклад реалізації “роботи з базою даних” для React Native

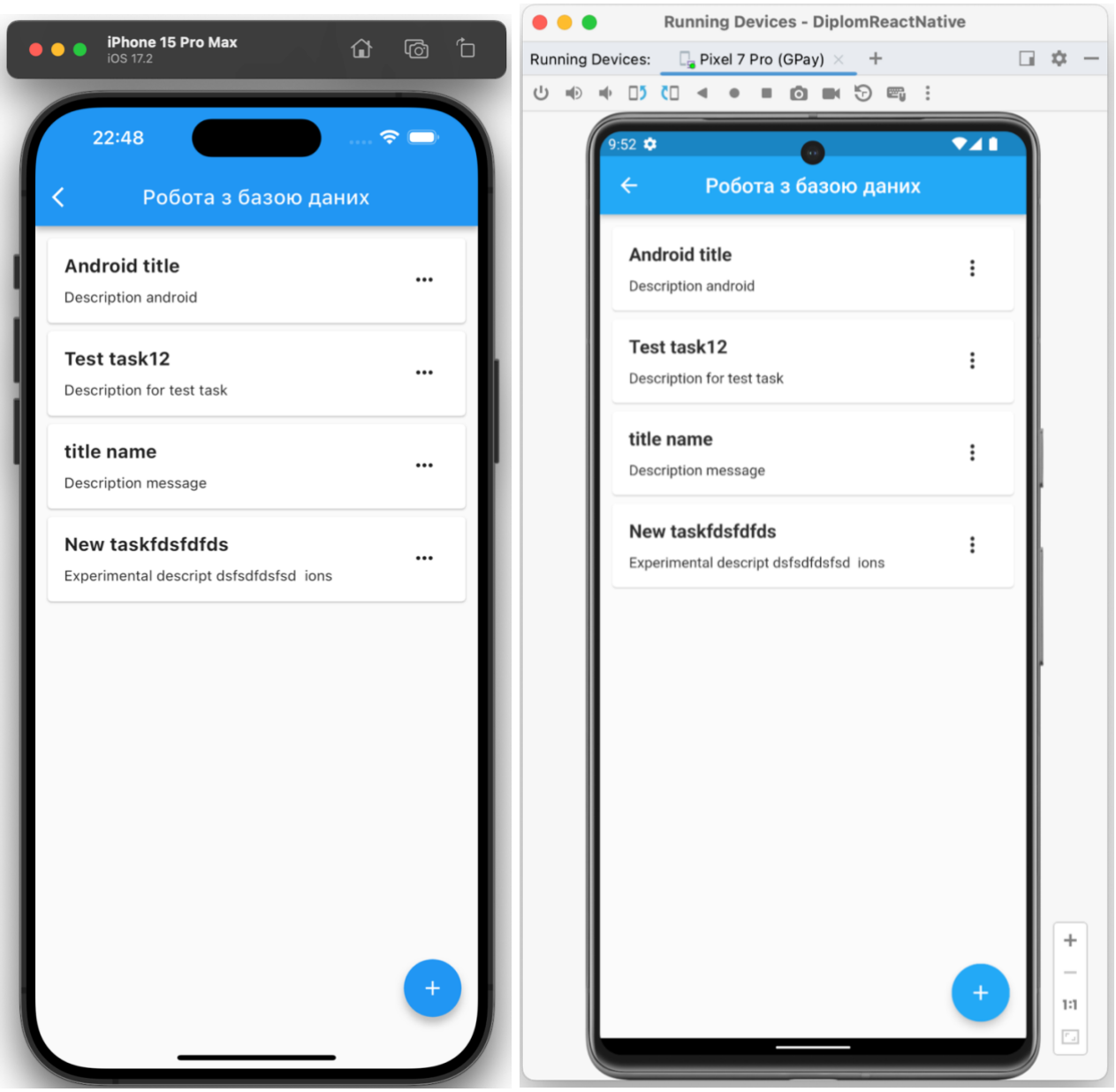

Рисунок 3.12 — Приклад реалізації “роботи з базою даних” для Flutter

Висновки до розділу 3

У цьому розділі було описано стек технологій, який використовується для створення мобільних застосунків. Було описано створення екранів з відображенням коду та знімків екрану, як це буде виглядати. Також були описані ключові моменти реалізації цих екранів, а також інтеграція бібліотек Stripe, Firebase, Google Maps і Mapbox, а також опис їхньої інтеграції.

РОЗДІЛ 4 АНАЛІЗ СТВОРЕННИХ ЗАСТОСУНКІВ

4.1 Аналіз інтегрування навігації у фреймворках.

У React Native для навігації між екранами зазвичай використовуються бібліотеки, такі як React Navigation, React Router Native, або навіть нативні навігаційні рішення. React Navigation є однією з найпопулярніших бібліотек навігації для React Native. Вона надає широкий спектр навігаційних рішень, включаючи стекову, табовану, вбудовану та інші навігаційні конфігурації. Реалізація навігації в React Native базується на стекових навігаторах, які дозволяють зручно керувати стеком екранів.

У Flutter навігація вбудована в SDK і базується на різних класах та об'єктах, таких як Navigator, Route та MaterialApp. Для навігації між екранами Flutter використовує стековий підхід, де кожен екран додається в стек навігації. Навігація в Flutter може бути реалізована за допомогою вбудованих класів, таких як MaterialPageRoute та CupertinoPageRoute, або використовуючи сторонні бібліотеки, такі як Flutter Navigator 2.0 або GetX.

Обидва фреймворки, React Native та Flutter, мають зручні засоби для реалізації навігації в мобільних застосунках. React Native надає більше гнучкості і можливостей завдяки широкому вибору бібліотек навігації, тоді як у Flutter навігація вбудована в SDK і має простий та зрозумілий інтерфейс. Обидва фреймворки підтримують стековий підхід до навігації, що спрощує управління стеком екранів у застосунку.

4.2 Аналіз порівняння стандартних елементів у фреймворках

У React Native стандартні нативні елементи, такі як кнопки, текстові поля, списки та інші, відтворюються за допомогою компонентів React Native, які мають вигляд та функціональність схожу на елементи, що використовуються в операційних системах iOS та Android. Однак, інтерфейсні компоненти React

Native не є абсолютно однаковими з нативними елементами і можуть мати невеликі відмінності в зовнішньому вигляді та поведінці. Крім того у останніх версіях фреймворку деякі елементи були видалені, а саме:

- Navigator — компонент навігації, який був замінений на React Navigation. Був видалений в реалізі 0.44 у 2017 році;
- PropTypes — типізація props для компонентів. Видалені у версії 0.63 в 2019 році та замінені на реалізацію з пакету prop-types;
- ListView — компонент списку, який був замінений на FlatList та SectionList в 2016 році через проблеми з продуктивністю;
- ToolbarAndroid — видалений у версії 0.62 в 2019 році. Замість нього радять використовувати компоненти для кожної платформи: ToolbarIOS та React Native Paper;
- ViewPagerAndroid — компонент горизонтального слайдера на Android. Був видалений у версії 0.62;
- WebView — компонент веб переглядача. Був видалений у версії 0.62 через складнощі підтримки;.
- RadioButton існував в ранніх версіях фреймворку. Він дозволяв створювати радіо-кнопки для вибору параметрів. Але починаючи з версії 0.44 в 2017 році його було видалено. Замість RadioButton рекомендується використовувати компоненти Switch або окремі кнопки і контролювати їх стан.
- Checkbox — існував в React Native до версії 0.29 але у 2018 році його видалили. Замість нього радять застосовувати компонент Switch, який виглядає і поводиться як чекбокс.

Також було видалено чимало застарілих стилів та властивостей за час існування React Native. Розробники фреймворку намагаються оптимізувати та спростити його з кожною новою версією.

У Flutter стандартні елементи інтерфейсу, відомі як "віджети", реалізовані як частина SDK і надають повну підтримку для побудови інтерфейсу, який виглядає та працює однаково на різних платформах. Віджети Flutter намагаються

максимально відтворити вигляд та поведінку нативних елементів інтерфейсу, що робить застосунки, розроблені з використанням Flutter, майже не відрізняються від нативних застосунків.

Якщо вам потрібно максимально точно відтворити зовнішній вигляд та поведінку нативних елементів інтерфейсу, то Flutter може бути кращим вибором, оскільки він надає повністю налаштовані віджети, що ідеально підходять для цієї мети. У React Native також можливо відтворити нативний вигляд елементів, але це може вимагати додаткових зусиль для досягнення такого результату.

Приводжу порівняльну таблицю усіх нативних елементів для React Native і Flutter у таблиці 1.

Таблиця 1 — Порівняння стандартних елементів для React Native та Flutter

Нативний елемент	iOS	Android	React Native	Flutter
Button	Так	Так	Так	Так
TextInput	Так	Так	Так	Так
RadioButton	Так	Так	Ні	Так
CheckBox	Так	Так	Ні	Так
Image	Так	Так	Так	Так
Dropdown	Так	Так	Ні	Так
View	Так	Так	Так	Так
Toggle Buttons	Так	Так	Ні	Так
ScrollView	Так	Так	Так	Так
ListView	Так	Ні	Так	Ні
TabBar	Так	Так	Так	Так
NavigationBar	Так	Так	Так	Так
SegmentedControl	Так	Так	Так	Так
Slider	Так	Так	Так	Так
Switch	Так	Так	Так	Так

Picker	Так	Так	Так	Так
DatePicker	Так	Так	Так	Так
TimePicker	Так	Так	Так	Так
Alert	Так	Так	Так	Так
ActionSheet	Так	Так	Так	Так
ProgressView	Так	Так	Так	Так
ActivityIndicator	Так	Так	Так	Так
ProgressHUD	Так	Ні	Ні	Ні
WebView	Так	Так	Так	Так
MapView	Так	Так	Так	Так
Geolocation	Так	Так	Так	Так
SpeechRecognizer	Так	Ні	Ні	Ні
BarcodeScanner	Так	Ні	Ні	Ні
Camera	Так	Ні	Ні	Ні
FilePicker	Так	Ні	Ні	Ні
Notifications	Так	Так	Так	Ні
Share	Так	Так	Так	Так
Clipboard	Так	Так	Так	Так
Accessibility	Так	Так	Так	Так

4.3 Аналіз мов програмування, які використовуються у Flutter (Dart) та React Native (JavaScript/TypeScript)

Порівняльна характеристика мов програмування Dart та JavaScript/TypeScript, що використовуються відповідно у Flutter та React Native:

Dart (для Flutter):

- створена спеціально для розробки мобільних та веб-застосунків;
- об'єктно-орієнтована, з підтримкою функціонального програмування;
- строга типізація (на відміну від динамічної типізації JS);
- швидкість роботи на рівні Java та JavaScript (JIT-компіляція);
- підтримка асинхронного програмування через async/await;
- вбудована підтримка імутабельних структур даних.

JavaScript/TypeScript (для React Native):

- мова загального призначення для веб та сервера
- динамічна типізація в JS, строга типізація в TS
- прототипне ООП
- асинхронне програмування через проміси та async/await
- величезна кількість бібліотек та інструментів
- в React Native використовується JS, обгорнутий в нативні компоненти

Dart оптимізований спеціально під мобільну та веб-розробку, в той час як JS/TS - універсальні мови програмування зі своїми перевагами та недоліками. Вибір залежить від конкретних потреб та переваг розробника.

4.4 Аналіз використання сторонніх бібліотек

Аналіз інтеграції сторонніх бібліотек робиться на основі інтеграції таких бібліотек: Stripe, Firebase та Firebase Cloud Firestore, Google Maps та Mapbox.

Stripe:

- React Native використовуються бібліотеки, такі як «tipsi-stripe» або «stripe-react-native», які дозволяють зручно взаємодіяти з платіжним сервісом у мобільних застосунках. Для дослідження було вибрано stripe-react-native;
- Flutter використовується пакет «flutter_stripe», який надає API для здійснення платежів та роботи з картками безпосередньо з Flutter застосунками.

Google Maps:

- React Native використовується бібліотека `react-native-maps`, яка дозволяє вбудовувати карти Google у мобільні застосунки та взаємодіяти з ними;
- Flutter використовується пакет `google_maps_flutter`, який надає зручні інструменти для роботи з картами Google безпосередньо в Flutter застосунках.

Mapbox:

- React Native зазвичай відбувається за допомогою бібліотеки `@react-native-mapbox-gl/maps`, яка надає доступ до функціональності Mapbox у мобільних застосунках але нова версія цієї бібліотеки рекомендує використовувати «@rnmapbox/maps»:
- Flutter використовується пакет `flutter_mapbox_gl`, який дозволяє вбудовувати та взаємодіяти з картою Mapbox у мобільних додатках на Flutter.

Firebase та Firebase Cloud Firestore:

- React Native має пакет `react-native-firebase`, який надає доступ до багатьох функцій Firebase, включаючи Cloud Firestore. Цей пакет дозволяє легко інтегрувати функціональність Firebase у React Native застосунки
- Flutter використовується пакет `firebase_core` для інтеграції Firebase та `cloud_firestore` для роботи з Firebase Cloud Firestore. Ці пакети дозволяють легко використовувати функціональність Firebase у Flutter застосунках.

Обидва фреймворки, React Native та Flutter, мають потужні інструменти для інтеграції сторонніх бібліотек, таких як Stripe, Firebase, Google Maps та Mapbox. Обираючи між ними, розробники повинні враховувати специфічні потреби свого проекту та особливості кожного фреймворку для оптимальної інтеграції та розробки мобільних застосунків.

.

Висновок до розділу 4

У даному розділі було проведено аналіз інтеграції навігації та порівняння стандартних елементів у фреймворках React Native та Flutter, а також порівняння мов програмування, які використовуються у цих фреймворках.

Щодо інтеграції навігації, обидва фреймворки мають зручні засоби для реалізації навігації в мобільних застосунках. React Native надає більшу гнучкість завдяки широкому вибору бібліотек навігації, тоді як у Flutter навігація вбудована в SDK і має простий та зрозумілий інтерфейс. Обидва фреймворки підтримують стековий підхід до навігації, що спрощує управління стеком екранів у застосунку.

У порівнянні стандартних елементів, було зазначено, що React Native та Flutter намагаються максимально відтворити вигляд та поведінку нативних елементів інтерфейсу. У React Native стандартні нативні елементи відтворюються за допомогою компонентів, але можуть мати деякі відмінності в зовнішньому вигляді та поведінці. У Flutter стандартні елементи інтерфейсу реалізовані як частина SDK і мають повну підтримку для побудови інтерфейсу, що є однаковим на різних платформах.

Щодо мов програмування, використованих у фреймворках, Dart (для Flutter) та JavaScript/TypeScript (для React Native), було виявлено, що Dart спеціально оптимізований для мобільної та веб-розробки, має строгу типізацію та інші особливості, які полегшують розробку. У той час, як JavaScript/TypeScript є універсальними мовами програмування зі своїми перевагами та недоліками.

Обидва фреймворки мають свої переваги та недоліки, і вибір між ними залежить від конкретних потреб та вимог проекту.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи можна зробити висновки, для яких проектів та з якими потребами краще використовувати фреймворки.

Використання React Native буде корисним:

- для розробки великих та складних мобільних застосунків завдяки великій екосистемі та спільноті;
- для проектів у яких є веб-застосунок на React, використання React Native може значно прискорити розробку мобільної версії;
- для проектів в яких потрібна інтеграція використання величезної кількісті JavaScript-бібліотек робить React Native привабливим для застосунків, що вимагають інтеграцію з різними зовнішніми сервісами та API.

Використання Flutter буде корисним:

- для створення складних та гнучких користувацьких інтерфейсів завдяки власним «віджетам» (SDK);
- для проектів де важлива швидкість розробки наприклад стартапів, так як має дуже зручне SDK та швидке розгортання середовища для розробки;
- інтеграція таких бібліотек як Stripe може бути складним так як в офіційній документації немає підтримки.

Рекомендації щодо використання:

- розгортання середовища для розробки на React Native може бути складнішим так як потрібно встановити та налаштувати Node.js, Watchman та можливо оновити версію Ruby та ще вибрати, яке саме середовище в конкретному вашому випадку треба використовувати;
- при виборі Flutter для розробки будьте готові до того що мова Dart не настільки поширена як JavaScript тому знайти розробника може бути складніше ніж React Native.

В цілому, вибір між Flutter та React Native залежить від конкретних вимог проекту, наявних ресурсів та досвіду команди розробників.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Miola A. Flutter. Complete Reference: Create beautiful, fast and native apps for any device. Independently published, 2020. 765 p.

2. Dabit N. React Native in Action. 1st Edition. Manning, 2019. 320 p.

3. Офіційний сайт для розробників React Native URL: https://reactnative.dev/ (дата звернення: 20.01.2024)

4. Офіційний сайт для розробників Flutter URL: https://flutter.dev/ (дата звернення: 17.01.2024)

5. Офіційний сайт для розробників Apache Cordova URL: https://cordova.apache.org/ (дата звернення: 10.01.2024)

6. Flutter vs React Native – What to choose in 2023? URL: https://www.thedroidsonroids.com/blog/flutter-vs-react-native-what-to-choose (дата звернення: 03.01.2024)

7. Flutter Pros & Cons for Mobile App Owners URL: https://www.thedroidsonroids.com/blog/flutter-pros-and-cons (дата звернення: 13.01.2024)

8. FAQ Flutter URL: https://docs.flutter.dev/resources/faq (дата звернення: 13.01.2024)

9. Код та приклади застосунків URL: https://github.com/flutter/flutter (дата звернення: 14.01.2021)

10. «IPST-2020» – дев'ята між народна науково-технічна конференція НТУ «ХПІ» м. Харків, “Порівняльній аналіз фреймворків для розробки крос-платформенних мобільних додатків, URL: https://repository.kpi.kharkov.ua/server/api/core/bitstreams/7da94830-09cd-40ac-b3bc-121ce9781581/content (дата звернення: 10.01.2024)

11. H. Hussain та K. Khan, "Comparative Study of Android Native and Flutter App Development", KSII The 13 th International Conference on Internet (ICONI) 2021, № 13, с. 99–100, 2021.

12. Demand statistics for the development of cross-platform applications URL: https://insights.stackoverflow.com/trends?tags=xamarin.forms%2Cnativescript%2Ccordova%2Cqt%2Creact-native%2Cflutter%2Cionic-framework (дата звернення: 20.01.2024)

13. Макушин Д. Особливості застосування фреймворків Flutter та React Native для створення мобільних застосунків. Молода наука-2023 : Збірник наукових праць студентів, аспірантів, докторантів і молодих вчених. Т.5, 17-22 квітня 2023 р. Запоріжжя : ЗНУ, 2023. С. 101-102. URL: https://www.znu.edu.ua/ii_znu/nauka/conf5/tom_5_2023_26_05_2023.pdf дата звернення: 17.01.2024).

14. Shaun Lewis and Mike Dunn “Native Mobile Development. A Cross-Reference for iOS and Android” O’Reilly Media, Inc. 2020 367p.

15. Marco L. Napoli. Beginning Flutter: A Hands On Guide to App Development. – Chapter 16 – 2019 – P. 432.

16. Nate Ebel. Mastering Kotlin. – Chapter 13 – 2019 – P. 267.

17. Stripe. URL: https://stripe.com/ (дата звернення: 20.01.2024).

18. Макушин Д.В. – магістрант, Коломоєць Г.П. доцент, канд. фіз.-мат. наук – науковий керівник. Матеріали III Всеукраїнської науково-практичної конференції за участю молодих науковців «Актуальні питання сталого науково-технічного та соціально-економічного розвитку регіонів України». Запорізький національний університет. Запоріжжя: ЗНУ, 2023. ??? с.

19. Офіційний сайт фреймворку Ionic URL: https://ionicframework.com/docs/ (дата звернення: 20.01.2024).

20 Офіційний сайт Capacitor URL: https://capacitorjs.com/ (дата звернення: 20.01.2024).

21. Офіційний сайт Dart URL: https://dart.dev/ (дата звернення: 20.01.2024).

22. Офіційний сайт Typescript. URL: https://www.typescriptlang.org/docs/ (дата звернення: 20.01.2024).

23. JavaScript Tutorial. URL: https://www.javascripttutorial.net/ (дата звернення: 20.01.2024).

24. Офіційна статистика по мовам програмування та фреймворкам URL: https://www.statista.com/ (дата звернення: 20.01.2024)

25. Офіційний сайт Mapbox URL: https://www.mapbox.com/ (дата звернення: 20.01.2024)

26. 4 Most Popular Cross-Platform App Development Frameworks for 2024 URL: https://www.thedroidsonroids.com/blog/top-cross-platform-app-development-frameworks (дата звернення: 4.02.2024)

27. React Native at Instagram URL: https://engineering.instagram.com/react-native-at-instagram-dd828a9a90c7 (дата звернення: 4.02.2024).

28. Офіційне підтвердження від Microsoft об використанню у розпробці Skype React Native URL: https://microsoft.github.io/reactxp/blog/2018/04/10/skype-react-native.html (дата звернення: 4.02.2024)

29. Офіційний список застосунків які розроблені на Flutter URL: https://flutter.dev/showcase (дата звернення: 4.02.2024)

30. Офіційний сайт Firebase URL: https://firebase.google.com/docs?hl=en (дата звернення: 10.02.2024).

31. Офіційний сайт Google Maps URL: https://developers.google.com/maps/documentation?hl=en (дата звернення: 10.02.2024).