Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів розробки графічного інтерфейсу користувача в кроссплатформенних додатках
(тема)

Виконав:
студент (ка) 2 курсу, групи ІПЗм-22-4

Моруга Д.І.
(прізвище, ініціали)

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

Керівник доц. Ревенчук І.А.
(посада, прізвище, ініціали)

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

2024 р.

Харківський національний університет радіоелектроніки

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

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Морузі Дмитру Ігоровичу
(прізвище, ім’я, по батькові)

1. Тема роботи «Дослідження методів розробки графічного інтерфейсу користувача в кроссплатформенних додатках»
Затверджена наказом по університету від 29.03.2024р. № 250 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 08.06.2024
3. Вихідні дані до роботи технології розробки кросплатформенного графічного інтерфейсу, опис досліджуваних фремворків, вимоги до кросплатформенного графічного інтерфейсу, мови програмування Dart та JavaScript, технології Flutter та React Native, середовище розробки Visual Studio Code
4. Перелік питань, що потрібно опрацювати в роботі
дослідження та порівняльний аналіз методів та засобів розробки графічного інтерфейсу користувача в кроссплатформенних додатках, огляд та аналіз літературних джерел, практичне дослідження методів розробки графічного інтерфейсу користувача в кроссплатформенних додатках

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі та постановка задачі	01.04 – 15. 04.24	*виконано*
2	Аналіз та вибір фреймворків для дослідження	16.04 – 23.04.24	*виконано*
3	Аналіз та дослідження літератури	24.04 – 30.04.24	*виконано*
4	Планування експериментів	01.05 – 05.05.24	*виконано*
5	Програмна реалізація кожного з обраних для дослідження фремворків	06.05 – 15.05.24	*виконано*
6	Експериментальні дослідження	16.05 – 20.05.24	*виконано*
7	Аналіз результатів експериментальних досліджень	21.05 – 24.05.24	*виконано*
8	Написання та оформлення статті та тез доповіді	25.05 – 30.05.24	*виконано*
9	Підготовка пояснювальної записки	31.05 – 04.06.24	*виконано*
10	Підготовка презентації та доповіді	05.06 – 07.06.24	*виконано*
11	Нормоконтроль	08.06 – 09.06.24	*виконано*
12	Рецензування	10.06 – 11.06.24	*виконано*
13	Занесення диплома в електронний архів	11.06.2024	*виконано*
14	Попередній захист	12.06.2024	*виконано*
15	Допуск до захисту у зав. кафедри	20.06.2024	*виконано*

Дата видачі завдання 01 квітня 2024р.

Студент (ка) _____ Моруга Д.І.
(підпис)

Керівник роботи _____ доц. Ревенчук І.А.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 62 с., 21 рис., 5 табл., 27 джерел.

ГРАФІЧНИЙ ІНТЕРФЕЙС, КРОСПЛАТФОРМЕННА РОЗРОБКА, REACT NATIVE, FLUTTER, XAMARIN, QT, ELECTRON.

Об’єктом дослідження є методи розробки графічного інтерфейсу користувача в кроссплатформенних додатках.

Метою роботи є проведення дослідження придатності методів розробки графічного інтерфейсу користувача в кроссплатформенних додатках для найбільшої кількості платформ.

Методами розробки та проектування є аналіз проблемної області дослідження, вибір фреймворків для проведення порівняльного дослідження шляхом вирішення багатокритеріальної задачі прийняття рішень.

У результаті кваліфікаційної роботи було досліджено методи та засоби розробки графічного інтерфейсу користувача в кроссплатформенних додатках, був виконаний їх порівняльний аналіз.

GUI, CROSS-PLATFORM DEVELOPMENT, REACT NATIVE, FLUTTER, XAMARIN, QT, ELECTRON.

The object of research is methods for development of graphical user interface in cross-platform applications.

The purpose of the work is to conduct a study of the suitability of methods for developing a graphical user interface in cross-platform applications for the largest number of platforms.

The solution methods are the analysis of the problem area of the study, the selection of the frameworks for conducting the comparative analysis by solving a multi-criteria decision-making problem.

As a result of the work, the methods and tools for developing a graphical user interface in cross-platform applications were investigated, their comparative analysis was performed.

Заява щодо самостійного виконання кваліфікаційної роботи та можливості її публікації в електронному архіві відкритого доступу ElArKhNURE.

Я, Моруга Дмитро Ігорович, студент(ка) гр. ІПЗм-22-4, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів розробки графічного інтерфейсу користувача в кроссплатформенних додатках», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

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

ЗМІСТ

ВСТУП

Графічний інтерфейс користувача є основним засобом взаємодії користувача з програмним забезпеченням. У сучасному повсюдному обчислювальному ландшафті додатки повинні безперебійно функціонувати на різноманітних платформах, від мобільних пристроїв і настільних комп'ютерів до вбудованих систем. Це вимагає використання кроссплатформенних графічних інтерфейсів, які спрощують процес розробки, забезпечуючи при цьому узгодженість взаємодії користувачів в різних операційних системах.

Фреймворки кроссплатформенних графічних інтерфейсів пропонують набір інструментів розробки та бібліотек, спеціально розроблених для створення інтерактивних графічних інтерфейсів. Вони надають абстракції від базових функцій, що залежать від платформи, дозволяючи розробникам зосередитися на основній логіці програми та дизайні взаємодії з користувачем. Вибір відповідної платформи відіграє вирішальну роль у успіху проекту, впливаючи на такі фактори, як ефективність розробки, продуктивність додатків та залучення користувачів.

Вибір найбільш підходящого кроссплатформенного графічного інтерфейсу вимагає всебічного процесу оцінки. Систематично оцінюючи критерії відповідно до потреб конкретного проекту та набору навичок команди, розробники можуть приймати обґрунтовані рішення. Найбільш підходяща структура, швидше за все, включатиме компроміси, що забезпечують баланс між ефективністю розробки, бажаним досвідом користувачів, цільовими платформами та довгостроковими цілями проекту. У цій роботі пропонується система порівняння, орієнтована на ключові критерії, які безпосередньо впливають на ефективність розробки та зручність роботи з користувачем на основі задачі критеріального вибору та теорії корисності. Далі проводиться порівняльний аналіз обраних кроссплатформенних графічних фреймворків: В аналізі досліджуються їх сильні та слабкі сторони щодо ключових аспектів. Розуміючи унікальні характеристики кожного фреймворку, розробники можуть приймати обґрунтовані рішення, які відповідають їхнім конкретним вимогам до проекту та набору навичок команди.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі дослідження

Сучасний ринок пропонує широкий спектр інструментів для кросплатформенної GUI-розробки. Вибір оптимального рішення залежить від специфіки проекту, платформ, що підтримуються, та навичок команди. Ретельний аналіз та порівняння доступних варіантів дозволить зробити оптимальний вибір та створити якісний інтерфейс користувача для додатку.

Настільні операційні системи, маючи значно більший термін існування, зумовили ранній розвиток інструментів для кросплатформенної розробки GUI. З часом мобільні платформи, завдяки стрімкому зростанню та поширенню, вийшли на перший план, ставши домінуючими у сфері персональних комп’ютерних систем.

Зростання популярності мобільних пристроїв стимулювало активний розвиток нових інструментів для кросплатформенної GUI-розробки під мобільні ОС. Завдяки більшій кількості користувачів, мобільні платформи наразі мають більш динамічний розвиток у цій сфері.

Деякі з інструментів кросплатформної GUI-розробки також підтримують розробку веб-додатків. Це розширює можливості та робить їх універсальними рішеннями для створення інтерфейсів користувача на різних платформах.

Qt – фреймворк мови програмування C++, призначений для розробки кроссплатформенних застосунків [1]. Qt дозволяє розробку додатків, які можуть працювати на різних операційних системах, таких як Windows, macOS, Linux, Android та iOS. Qt головним чином ґрунтується на мові програмування C++, надаючи продуктивність та гнучкість низькорівневого програмування. Qt пропонує два основних підходи для створення інтерфейсів користувача:

– Qt Widgets – традиційні компоненти інтерфейсу, які підходять для застосунків з складними інтерфейсами;

– Qt Quick – декларативна мова для створення плинних, анімованих інтерфейсів, часто використовується в мобільних та сенсорних застосунках.

Qt доступний як у відкритому вигляді (Qt Open Source), так і у комерційних версіях, які надають додаткові можливості та підтримку. Qt має офіційним інтегрованим середовищем розробки Qt Creator. Qt є модульним, існують різні модулі для різних аспектів розробки застосунків, такі як QtCore (основна функціональність), QtGui (функціоналість GUI), QtWidgets (компоненти для десктопу) та інші.

Electron – відкритий фреймворк, який дозволяє розробникам будувати кроссплатформенні настільні застосунки, використовуючи веб-технології, такі як HTML, CSS та JavaScript [2]. Electron дозволяє розробку настільних застосунків для різних операційних систем, включаючи Windows, macOS та Linux, з використанням однієї кодової бази. Веб-технології використовуються для створення однакових користувацьких інтерфейсів на різних платформах. Додатки на Electron суттєво є веб-додатками, які працюють в вікні браузера на базі Chromium. Розробники використовують HTML, CSS та JavaScript для побудови інтерфейсу користувача та функціоналу. Двигун Chromium забезпечує сумісність із сучасними веб-стандартами. Electron інтегрується з Node.js, що дозволяє розробникам використовувати JavaScript для отримання доступу до можливостей нативної системи.

Xamarin – кроссплатформенний фреймворк для розробки додатків для платформ iOS, Android та Windows, використовуючи одну кодову базу [3].

Xamarin дозволяє розробку кроссплатформенних додатків з використанням мови програмування C# та програмної птаформи .NET. Через компіляцію коду у нативні бінарні файли Xamarin дозволяє досягти майже нативної продуктивності. Також додатки Xamarin мають доступ до нативних API, що забезпечує нативний вигляд та відчуття на кожній платформі.

Xamarin використовує один код для бізнес-логіки, і код специфічний для платформи, може бути написаний з використанням Xamarin.Forms або

Xamarin.Native. Xamarin.Forms – набір інструментів для розробки UI, який дозволяє створювати єдиний інтерфейс користувача, який працює на різних платформах. Він спрощує процес розробки інтерфейсу користувача за допомогою єдиного коду XAML для визначення інтерфейсу. Xamarin.Native дозволяє розробникам створювати окремі інтерфейси для кожної платформи, забезпечуючи більше контролю над виглядом та відчуттям додатка на кожному пристрої.

Розробка Xamarin може виконуватися за допомогою Visual Studio, що надає широкий спектр інструментів для проектування, кодування, тестування та налагодження мобільних додатків.

React Native – відкритий фреймворк, розроблений компанією Facebook, який дозволяє розробникам будувати додатки з використанням JavaScript і React [4]. React Native дозволяє розробку кроссплатформенних додатків для iOS, Android, веб, macOS та Windows. React Native використовує JavaScript разом з бібліотекою React для створення користувацьких інтерфейсів. Це дозволяє залучити розробників, які володіють React для веб-розробки.

React Native дозволяє використовувати нативні компоненти, що забезпечує інтерфейс додатка виглядатиме як нативний застосунок. Основні компоненти написані на нативних мовах (Java для Android, Objective-C або Swift для iOS) і пакуються разом із кодом JavaScript. React Native ставить за мету наближення до нативної продуктивності за допомогою використання нативних компонентів та оптимізації виконання JavaScript. React Native слідує декларативному підходу до створення інтерфейсу, де розробники описують бажаний стан, а фреймворк відповідає за його відтворення. Компонентна архітектура сприяє модульності та повторному використанню, дозволяючи розробникам будувати складні інтерфейси зі складених компонентів.

Flutter – відкритий фреймворк для розробки користувацького інтерфейсу, створений компанією Google для побудови нативно скомпільованих застосунків з єдиної кодової бази [4]. Підтримуються наступні платформи iOS, Android, веб та настільні комп’ютери. Підхід написати один раз, запустити скрізь спрощує розробку та зменшує необхідність в окремих кодових базах для кожної

платформи. Flutter використовує мову програмування Dart, що призначена для створення сучасних, оптимізованих клієнтських застосунків з акцентом на продуктивність. Але через новизну кількість розробників на ринку, що володіють неб не велика. Flutter використовує архітектуру на основі віджетів, де весь інтерфейс користувача складається з невеликих, модульних компонентів, які називають віджетами. Дані елементи є повторно використовуваними, що полегшує створення складних та інтерактивних інтерфейсів користувача. Flutter надає широкий набір віджетів для створення різних елементів інтерфейсу користувача, від основних кнопок до складних анімацій.

1.2 Актуальність дослідження

Кроссплатформенна розробка надає значні переваги при розробці додатків для декількох платформ. Але впродовж більшості історії свого існування кросплатформенні фреймворки для розробки графічного інтерфейсу були призначені для платформ з подібними інтерфейсами взаємодії з користувачем. Однією з останніх інновації фреймворків кросплатформенного графічного інтерфейсу є підтримка платформ з різними інтерфейсами взаємодії з користувачем. Але більшість досліджень пов’язаних з цими фреймворками зосереджуються на одній платформі, що пов’язано з конкретними потребами бізнесу. Незважаючи на це існує потреба підтримки платформ з різними інтерфейсами взаємодії з користувачем, що потребує дослідження актуальних фреймворків.

1.3 Аналіз літератури

В області кроссплатформенної розробки графічного інтерфейсу спостерігається сплеск публікацій і цілеспрямованих дослідницьких зусиль. Цей зростаючий обсяг знань обумовлений зростаючим попитом на додатки, які без проблем функціонують на різних платформах, від мобільних пристроїв до настільних комп’ютерів. Дослідники заглиблюються в порівняльний аналіз відомих фреймворків, аналізуючи їх сильні та слабкі сторони в таких ключових

сферах, як придатність мови програмування, контрольні показники продуктивності, оптимізація швидкості розвитку та ступінь доступної підтримки спільноти.

Ці публікації служать безцінними ресурсами для розробників, надаючи їм необхідну інформацію для прийняття обґрунтованих рішень при виборі найбільш підходящої платформи для конкретних потреб проекту. Дослідження зосереджені не лише на аналізі існуючих фреймворків, але й на нових тенденціях, таких як рішення з низьким вмістом коду та гібридні підходи до розробки додатків, які формують майбутній ландшафт розробки графічного інтерфейсу.

Перспективним підходом до розробки мобільних додатків щодо продуктивності, у контексті нових технологій, таких як віртуальна реальність, доповнена реальність та Інтернет речей є гібридні та кросплатформенні додатки [5, 6]. В той же час ці підходи є недостатньо дослідженими в промислових контекстах [7].

Перспективним підходом до розробки додатків для настільних комп’ютерів, у контексті нових технологій є кросплатформенні фреймворки та платформи, але в порівнянні з мобільними фреймворками вони не мають поширеності в комерційній розробці графічного інтерфейсу [8-10].

Перспективною є розробка веб-додатків для настільних платформ, в той час як на мобільних платформах цей метод є більш обмеженим [11, 12].

В підсумку помітне домінування фреймворків, що виникли у галузі мобільної розробки, для розробки кросплатформенного графічного інтерфейсу користувача.

1.4 Вибір фреймворків для дослідження

Для прийняття обґрунтованого рішення пропонується використовувати векторну оптимізацію та апарат теорії корисності. Аргументація вибору цих методів, та детальні обчислення надані у [13]. Були обрані наступні критерії вибору:

– популярність методів, цей фактор відображає поширеність та визнання фреймворка у спільноті розробників [14];
– кількість підтримуваних платформ, важливий аспект, що визначає можливість кросплатформної розробки;
– популярність мови програмування, впливає на доступність розробників з відповідними навичками та наявність бібліотек [15];
– якість інструментів та екосистеми, оцінює зручність та ефективність інструментів, доступність документації та активність спільноти;
– тип ліцензії, визначає можливі сценарії використання та потенційні обмеження.

Задача вибору фреймворка може бути представлена як задача векторної оптимізації, де кожен критерій виступає як векторна компонента. Для прийняття остаточного рішення використовується теорія корисності.

Для оцінки кожного критерія буде використана номінальна шкала.

Популярність методів, оцінка дорівнює відсотку популярності помноженому на 100.

Кількість платформ що підтримуються:

– iOS – 3 бали;
– Android – 3 бали;
– macOS – 2 бали;
– Windows – 2 бали;
– Linux – 1 бал;
– IoT(watchOS, tvOS, Wear OS та Android TV) – 1 бал.

Оцінкою за шкалою є сума підтримуваних платформ.

Популярність мови програмування, оцінка дорівнює відсотку найпопулярнішої мови програмування, що використовує метод помножене на 100.

Якість інструментів та екосистеми:

– інструменти доступні для комерційної розробки без обмежень – 3;
– частина інструментів потребує окремої ліцензії – 2;
– немає загальнодоступних інструментів – 1.

Вид ліцензії:

– open source – 3;

– partial open source – 2;

– commercial – 1.

Приведемо критерії до одного принципу оптимізації, економії ресурсів, маємо наступний векторний опис альтернатив таблиця 1.1.

Таблиця 1.1 – Векторний опис альтернатив

	Економія на навчанні розробників методу	Перевага у платформах	Навчання розробників мові програмування	Економія на інструментах	Економія на ліцензії
QT	323	10	1640	0	0
Electron	365	0	5759	2	1
Flutter	580	10	0	2	1
React Native	511	8	5759	2	1
Xamarin	0	9	2160	1	1

Визначаємо множину Парето для альтернатив.

Xamarin – по кожному критерію має альтернативу, що краща або має таку саму перевагу.

Electron – навчання розробників мові програмування, таке саме, як у React Native але програє по економії на навчанні розробників методу.

В результаті маємо наступні альтернативи:

– QT – 3 бали;

– Flutter – 3 бали;

– React Native – 3 бали.

Після нормування критеріїв та розрахування корисності альтернативі за допомогою адитивної лінійної згортки з ваговими коефіцієнтами отримаємо фреймворк з найвищою корисністю, який буде вважатися оптимальним вибором таблиця 1.2. Для критеріїв були використані наступні вагові коефіцієнти:

– перевага у платформах є основним критерієм при розробці кроссплатформенних додатків – 0.4;
– економія на навчанні розробників методу – 0.2;
– навчання розробників мові програмування – 0.2;
– економія на інструментах – 0.1;
– економія на ліцензії – 0.1.

Таблиця 1.2 – Результат адитивної згортки

	Результат згортки
QT	$0.4 * \frac{0}{0+1+\frac{188}{257}} + 0.2 * \frac{1}{1+1+0} + 0.2 * \frac{\frac{1640}{5759}}{\frac{1640}{5759}+0+1} + 0.1 * \frac{0}{0+1+1} + 0.1 * \frac{0}{0+1+1} = 0.144$
Flutter	$0.4 * \frac{1}{0+1+\frac{188}{257}} + 0.2 * \frac{1}{1+1+0} + 0.2 * \frac{0}{\frac{1640}{5759}+0+1} + 0.1 * \frac{1}{0+1+1} + 0.1 * \frac{1}{0+1+1} = 0.431$
React Native	$0.4 * \frac{\frac{188}{257}}{0+1+\frac{188}{257}} + 0.2 * \frac{0}{1+1+0} + 0.2 * \frac{1}{\frac{1640}{5759}+0+1} + 0.1 * \frac{1}{0+1+1} + 0.1 * \frac{1}{0+1+1} = 0.425$

В результаті було отримано, що альтернативи, які найбільш задовольняють вимогам є Flutter та React Native.

1.5 Аналіз поточних досліджень обраних фреймворків

Поява мобільних пристроїв як основної обчислювальної платформи спричинила сплеск досліджень та публікацій, порівняння React Native та Flutter, два домінуючі фреймворки для розробки кроссплатформенного графічного інтерфейсу. Цей сплеск інтересу безпосередньо пов’язаний з популярністю обох фреймворків, оскільки вони пропонують розробникам можливість створювати багатофункціональні мобільні додатки з єдиною кодовою базою, оптимізуючи процеси розробки і розширюючи їх цільову аудиторію в різних операційних системах. Дослідники заглиблюються в порівняльний аналіз React Native і Flutter, щоб визначити оптимальний фреймворк для конкретних потреб розробки мобільних додатків, враховуючи такі фактори, як швидкість розробки, продуктивність і доступ до вбудованих функцій пристроїв.

Дослідження продуктивності показують, що у деяких випадках Flutter має перевагу над React Native, але різниця в продуктивності може залежати від багатьох факторів, таких як сценарій використання, цільова платформа, та досвід розробників з фреймворком [16-18].

Загальні дослідження використовують більше критеріїв, такі як зацікавленість розробників, використовувані мови програмування, середовище розробки, підтримувані операційні системи, підтримка програмного забезпечення сторонніх бібліотек, розмір додатків, перспективи розвитку та перспективи застосування [19-21]. Ці дослідження надають перевагу React Native через використання JavaScript, знайомої та широко використовуваної мови, більш розвинену екосистему та спільноту.

На відміну від великої кількості досліджень, присвячених кроссплатформенним графічним інтерфейсам, орієнтованим на мобільні пристрої, область кроссплатформенних графічних інтерфейсів для настільних комп'ютерів відчуває відносну нестачу публікацій і спеціалізованих досліджень. Цю невідповідність можна пояснити сучасним домінуванням мобільних пристроїв як основної обчислювальної платформи. Оскільки користувачі все частіше переходять на смартфони та планшети для вирішення різних завдань, дослідницькі зусилля та публікації, природно, тяжіють до фреймворків, спеціально розроблених для розробки мобільних додатків. Однак ця тенденція не скасовує потенційної цінності кроссплатформенних графічних інтерфейсів для настільних комп'ютерів, і необхідні подальші дослідження для вивчення їх ефективності та визначення найбільш підходящих варіантів для розробників, які створюють настільні програми в різних операційних системах.

Дослідження застосування React Native на настільних платформах відсутні, в той час як Flutter порівнюється з WPF на настільній платформі і з Xamarin одночасно на настільній та мобільній платформі [22, 23]. Дані дослідження надають перевагу Flutter в усіх випадках, через більшу продуктивність та підтримку сучасних функцій.

В підсумку можна зробити висновок, що більшість досліджень зосереджена на порівнянні фреймворків лише на одній платформі.

1.6 Постановка задачі

Розвиток програмного забезпечення набуває різноманітних форм, охоплюючи, але не обмежуючись, такими платформами як настільні комп’ютери, Інтернет та мобільні пристрої, а також програмування для вбудованих пристроїв. React Native і Flutter зарекомендували себе як потужні конкуренти, пропонуючи можливості кроссплатформенної розробки для мобільних, настільних і, все частіше, веб-додатків. Порівняння їх функціональних можливостей на цих різних платформах дає цінну інформацію розробникам, які шукають найбільш відповідне рішення для своїх конкретних потреб.

Отже, для проведення дослідження необхідно вирішити наступні задачі:

– провести аналіз та дослідити сфери використання фреймворків;
– розробити або обрати додатки для експериментального дослідження фреймворків;
– провести експериментальне дослідження фреймворків, проаналізувати продуктивність додатків та досвід розробки з використанням фреймворку для платформи.

Опрацювавши результати дослідження буде можливо зробити висновок, щодо оптимальних методів розробки кросплатформенного графічного інтерфейсу.

2 ОПИС ПРИЙНЯТИХ ПРОЕКТНИХ РІШЕНЬ, ПРОГРАМНА РЕАЛІЗАЦІЯ

2.1 Вибір технологій

Дослідження фреймворків кросплатформенної розробки графічного інтерфейсу користувача Flutter та React Native потребує розробки додатку доказу концепції з підтримкою максимальної кількості платформ. Цільовою мобільною платформою обраний Android, цільовою настільною платформою обраний Windows, через поширеність цих платформ, також фреймворки підтримують веб-додатки.

Метою додатку доказу концепції буде створення, отримання, оновлення та видалення нотаток. Додаток підтримуватиме CRUD операції, що поширеними для більшості комерційних додатків. Для перевірки продуктивності буде використаний окремий додаток бенчмарк з навантаженням на можливості рендерингу фреймворків. Через те, що метою роботи є кросплатформенні графічні інтерфейси користувача для серверної частини та зберігання даних буде використана хмарна платформа Firebase. Використання React Native та Flutter зумовлене метою дослідження.

Firebase – це потужне рішення, яке прискорює виведення мобільних і веб-додатків на ринок і спрощує процес розробки. Воно володіє такими можливостями, як синхронізація даних, хмарне сховище, аутентифікація користувачів в режимі реального часу і аналітика. Таким чином, використовуючи це програмне забезпечення, розробники можуть створювати додатки вищої якості і використовувати переваги потужної хмарної інфраструктури Google.

Для зберігання даних буде використано Firebase Firestore – це хмарна база даних NoSQL, яка полегшує масштабовану та гнучку розробку мобільних додатків, веб-додатків та серверів. Будучи частиною платформи Firebase, Firestore призначений для плавної інтеграції з іншими хмарними сервісами Firebase і Google. Можливості Firestore в режимі реального часу, підтримка в автономному режимі та розширені запити роблять його універсальним вибором для широкого кола додатків – від соціальних мереж та інструментів співпраці до аналітичних

панелей у режимі реального часу. Його структурована модель даних спрощує управління даними, а масштабованість гарантує, що база даних зростатиме разом із додатком. Функції безпеки забезпечують спокій, цілісність даних та конфіденційність користувачів.

React Native – популярний фреймворк для створення кроссплатформенних додатків за допомогою JavaScript та React. Архітектуру React Native можемо розділити на наступні розділи:

– Rendering;
– Build Tools.

Процес візуалізації в React Native має вирішальне значення для створення та оновлення інтерфейсу користувача. Він передбачає перетворення компонентів React у подання, що залежать від платформи. Ключові аспекти візуалізації в React Native наступні:

– Fabric;
– Render, commit, and mount;
– Cross Platform implementation;
– View Flattering Tools;
– Threading Model.

Fabric – нова система візуалізації React Native. Її основна мета-уніфікувати логіку рендеринга на C++ для підвищення продуктивності і результативності. Fabric дозволяє React Native виконувати той самий код React framework, що і React для Web, але відображає його в загальних поданнях платформи (host views) замість вузлів DOM. Візуалізатор тканини існує в JavaScript і націлений на інтерфейси, доступні за допомогою коду на C++.

Render, commit, and mount, засіб рендеринга React Native виконує послідовність робіт для рендеринга логіки React на хост-платформу. Ця послідовність робіт називається конвеєром рендеринга і виконується для початкового рендеринга і оновлення стану користувальницького інтерфейсу.

Процес рендеринга в React Native можна розбити на три етапи: рендеринг, фіксація і монтування.

Візуалізація: на етапі візуалізації React виконує логіку продукту та створює дерева елементів React у JavaScript. Потім рендеринг React Shadow Tree дерева на C++ для кожного компонента хоста.

Фіксація: після того, як React Shadow Tree повністю створено, запускається фіксація. Це просуває React Element Tree та React Shadow Tree як наступне дерево для встановлення. Розрахунок макета також запланований на цьому етапі.

Монтування, фаза монтування перетворює React Shadow Tree разом з інформацією про його компонуванні в Host View Tree, яке представляє пікселі, що відображаються на екрані. На цьому етапі налаштовується кожен вид вузла з використанням реквізитів з відповідного React Shadow Node.

React Native надає різні інструменти збірки для покращення процесу розробки. Одним з таких інструментів є Hermes, движок JavaScript з відкритим кодом, оптимізований для React Native.

Flutter – кроссплатформенний набір інструментів інтерфейсу користувача, призначений для повторного використання коду в операційних системах, таких як iOS та Android, а також дозволяє програмам безпосередньо взаємодіяти з базовими службами платформи. Мета Flutter полягає в тому, щоб дати розробникам можливість створювати високопродуктивні програми, які будуть відчувати себе природно на різних платформах, враховуючи відмінності там, де вони існують, і при цьому використовуючи якомога більше загального коду.

Під час розробки програми Flutter запускаються у віртуальній машині, яка забезпечує гаряче перезавантаження змін із збереженням стану без необхідності повної перекомпіляції. Для випуску програми Flutter компілюються безпосередньо в машинний код, будь то інструкції Intel x64 або ARM, або в JavaScript, якщо вони орієнтовані на веб.

На наступній діаграмі рисунок 2.1 представлений огляд компонентів, що складають звичайний додаток Flutter. На ній показано, де в цьому стеку знаходиться двигун Flutter, виділені межі API і вказані сховища, в яких знаходяться окремі компоненти.

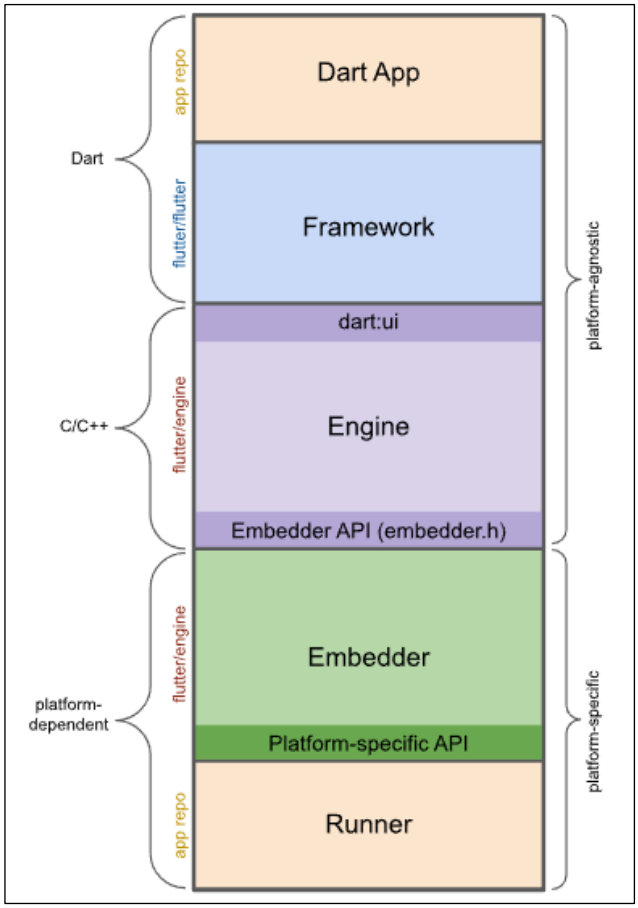

Рисунок 2.1 – React Native TODO додаток

Dart App компонує віджети в бажаний користувальницький інтерфейс, реалізує бізнес-логіку та належить розробнику додатка.

Framework надає API вищого рівня для створення високоякісних додатків (наприклад, віджети, тестування звернень, розпізнавання жестів, доступність, введення тексту), компонує дерево віджетів додатку в сцену.

Engine відповідає за растеризацію скомпонованих сцен, забезпечує низькорівневу реалізацію основних API Flutter (наприклад, графіка, верстка тексту, середовище виконання Dart). Надає свою функціональність фреймворку за допомогою Dart: UI API. Інтегрується з потрібною платформою, використовуючи API Embedder движка.

Embedder взаємодіє з базовою операційною системою для доступу до таких сервісів, як поверхні рендеринга, спеціальні можливості і введення та керує циклом подій. Надає API для конкретної платформи для інтеграції Embedder в додатки.

Runner об’єднує фрагменти, надані платформо-залежним API Embedder, в пакет додатку, доступний для запуску на цільовій платформі

2.2 Інструменти розробки

React Native та Flutter не мають власних спеціалізованих IDE, тому для розробки додатків була використана Visual Studio Code з розширеннями для React Native та Flutter.

Visual Studio Code – це універсальний і широко використовуваний редактор коду, розроблений Microsoft. Він став популярним інструментом серед розробників завдяки багатому набору функцій, розширюваності та зручному дизайну.

Для розробки Flutter додатків Visual Studio Code потребує розширень Dart та Flutter. Дані розширення додають підтримку синтаксису Dart, вибору цільових платформ для налагодження додатку та інші функції для підвищення якості процесу розробки.

Для розробки React Native додатків Visual Studio Code потребує розширень React Native Tools та Babel Javascript, залежно від того які функції Javascript знайомі розробнику. React Native Tools надає функції для налагодження додатку та інші функції для підвищення якості процесу розробки.

Для швидкої розробки додатків екосистема React Native має платформу Expo. Expo – це потужна та універсальна платформа для розробки React Native, яка спрощує процес створення, розгортання та керування мобільними додатками. Він особливо привабливий для розробників завдяки своїй простоті використання, надійному набору функцій і широкому інструментарію. Серед недоліків Expo відсутність підтримки Windows та нативних модулів.

2.3 Використані бібліотеки

Для розробки Flutter додатку були використані наступні бібліотеки:

– firebase_core, Flutter плагін для використання Firebase Core API, які здатні підключатися до багатьох Firebase додатків;

– cloud_firestore, плагін Flutter для використання API Cloud Firestore;
– fluttertoast, бібліотека нотифікацій.

Бібліотеки Firebase мають підтримку усіх цільових платформ, але підтримка Windows та веб знаходиться в бета стадії, приклад використання наведено на рисунку 2.2. В той час fluttertoast має підтримку лише мобільної та веб платформи, що потребуватиме відключення її функцій для настільних платформ, приклад використання наведено на рисунку 2.3.

```
Run | Debug | Profile
Future main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(options: DefaultFirebaseOptions.currentPlatform);
  runApp(const MyApp());
}
```

Рисунок 2.2 – Ініціалізація Firebase

```
void toast({required String message}) {
  Fluttertoast.showToast(
      msg: message,
      toastLength: Toast.LENGTH_SHORT,
      gravity: ToastGravity.BOTTOM,
      timeInSecForIosWeb: 1,
      backgroundColor: Colors.cyan,
      textColor: Colors.white,
      fontSize: 16.0);
}
```

Рисунок 2.3 – Нотифікації для користувача

Для розробки React Native додатку були використані наступні бібліотеки:

– firebase, надає інструменти та інфраструктуру. Цей пакет підтримує веб-клієнти, мобільний клієнти і сервер;
– react-navigation/native, навігація у React Native додатку;
– react-navigation/stack, навігація на основі структури стек для React Navigation;
– shopify/flash-list, компонент для відображення списків.

Недоліком React Native виявилась відсутність підтримки Windows додатків у бібліотеці firebase. Інші бібліотеки не мають залежностей від платформи, приклади використання рисунок 2.4 та 2.5.

```
<FlashList
    data={notes}
    numColumns={1}
    estimatedItemSize={100}
    renderItem={(item) => (
        <View style={styles.noteView}>
            <Pressable
                onPress={() => navigation.navigate('Detail', { item })}
            >
                <Text style={styles.noteTitle}>
                    {item.item.title}
                </Text>
                <Text style={styles.noteDescription}>
                    {item.item.note}
                </Text>
            </Pressable>
        </View>
    )}
/>
```

Рисунок 2.4 – Список нотаток у React Native додатку

```
export default function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen
          component={Home} ···
          }}
        />
        <Stack.Screen
          component={NoteAdd} ···
          }}
        />
        <Stack.Screen
          component={Detail} ···
          }}
        />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

Рисунок 2.5 – Навігація у React Native додатку

У підсумку екосистема React Native має більшу кількість бібліотек, але через відсутність підтримки настільних платформ від розробників підтримка цих платформ є ресурсномісткою. Flutter надає кращу підтримку для різноманітних платформ проте деякі з них знаходяться в стадії бета.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

У цьому розділі буде описана реалізація додатків, що були розробленні для отримання досвіду розробки кросплатформенного графічного інтерфейсу. Були розроблені додатки для створення нотаток за допомогою фреймворків React Native та Flutter. Для тесту продуктивності фреймворків були розроблені бенчмарки за зразком [24]. Тести, орієнтовані на складні обчислення, не обов'язково відображають типове використання мобільних додатків, оцінка ефективності фреймворку в сценарії, який безпосередньо впливає на досвід користувача є важливою. Отже було вирішено оцінити можливості рендеринга, важливий аспект взаємодії з користувачем. В рамках кожного фреймворку були реалізовані ідентичні рудиментарних додатки. Ці програми відображали сітку з 200 зображень і безперервно обертали їх у циклі. Цей тест, хоча і не є вичерпною оцінкою, служить орієнтиром для реальних сценаріїв, що включають численні анімації та інтенсивні оновлення інтерфейсу користувача.

3.1 Архітектура додатку

Клієнт серверна архітектура з тонким клієнтом централізує основну логіку та обробку даних на віддаленому сервері, забезпечуючи легкий та послідовний інтерфейс користувача на різних платформах. Архітектура тонкого клієнта, при якій на стороні клієнта виконується мінімальна обробка, а сервер виконує більшу частину обчислень і зберігання, забезпечує ефективне рішення при потребі в безперебійній і узгодженій взаємодії користувачів на декількох пристроях.

В якості серверної частини було використано платформу Firebase. Серед функцій платформи Firebase додатки використовують Cloud Firestore, базу даних для зберігання нотаток. Інтерфейс на стороні клієнта було розроблено з використанням Flutter для усіх платформ і React Native для мобільної платформи та веб платформи, відсутність повноцінної підтримки настільна платформа React Native не має підтримки Firebase. Схема архітектури наведена на рисунку 3.1.

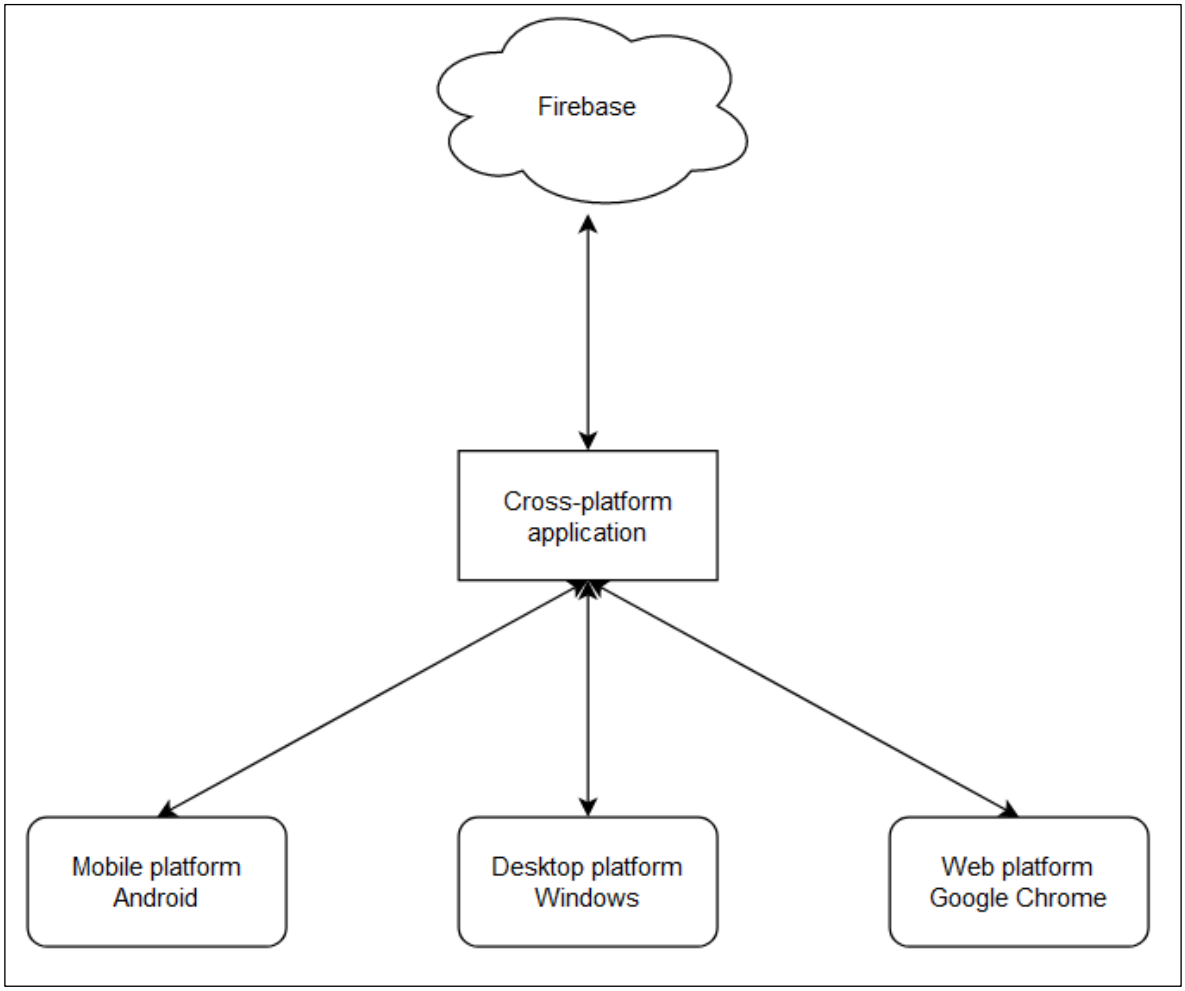

Рисунок 3.1 – Схема архітектури додатку

Описана Архітектура тонкого клієнта забезпечує високу адаптивність, зручність обслуговування і масштабованість рішення для кроссплатформенного додатку для створення нотаток. Завдяки централізації обробки даних на серверній частині і легкій вазі клієнтських додатків, архітектура забезпечує ефективну і узгоджену роботу користувачів на платформах Android, Windows і Web. Ця модель ефективно задовольняє зростаючу потребу в безперебійній роботі з декількома пристроями в сучасних додатках.

3.2 Програмна реалізація Flutter додатку

Для роботи з нотатками в додатку створена модель рисунок 3.2. Модель представлена класом NoteModel, який представляє нотатку. Кожна нотатку має ідентифікатор, заголовок, текст та колір. Усі ці поля можуть мати значення null.

Конструктор NoteModel приймає ці чотири поля як іменовані параметри, всі параметри є необов’язковими.

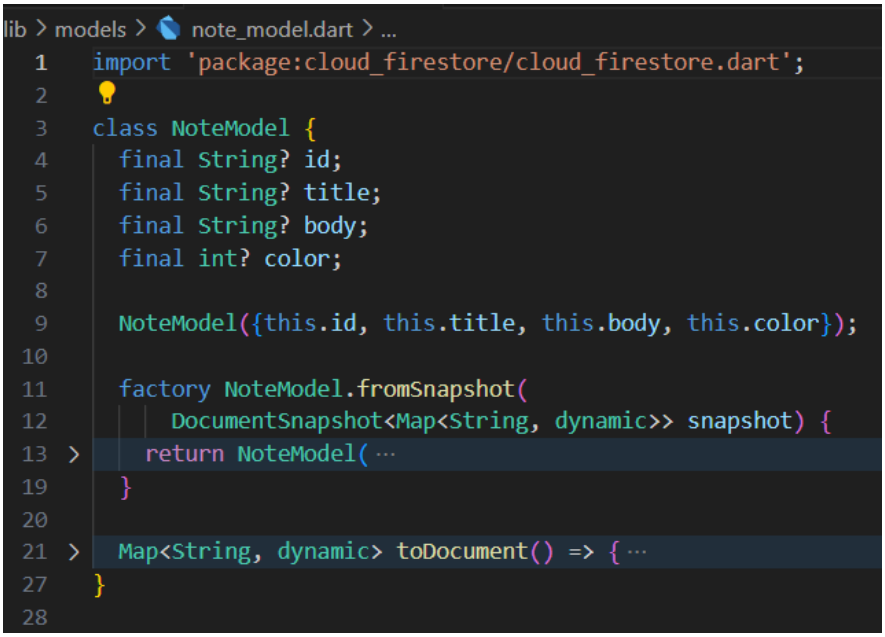

Рисунок 3.2 – Модель нотатки NoteModel

Фабричний конструктор NoteModel.fromSnapshot використовується для створення нового екземпляра NoteModel з об’єкта DocumentSnapshot. DocumentSnapshot – це тип даних із пакета cloud_firestore, який використовується для взаємодії з базою даних Firestore у Firebase. Цей конструктор витягує поля title, body, color та id із знімка та використовує їх для створення нової моделі NoteModel.

Метод toDocument перетворює екземпляр NoteModel у Map<String, dynamic>. Що використовується для зберігання екземпляру NoteModel у базі даних Firestore.

Взаємодія з базою даних Firestore виконується у класі FirestoreService рисунок 3.3. FirestoreService використовує модель NoteModel.

Метод createNote – це асинхронний метод, який приймає об’єкт NoteModel як аргумент і створює новий документ у колекції Firestore "notes". Він генерує новий ідентифікатор документа, створює новий об’єкт NoteModel з цим ідентифікатором, а потім перетворює цей об'єкт NoteModel у документ.

Метод getNotes – цей метод витягує всі документи з колекції "notes" у Firestore. Він повертає потік зі списку об’єктів NoteModel. Метод snapshots() повертає потік QuerySnapshot, який ми можемо зіставити з нашою моделлю нотаток. Цей потік буде генерувати новий список нотаток щоразу, коли колекція оновлюється у Firestore.

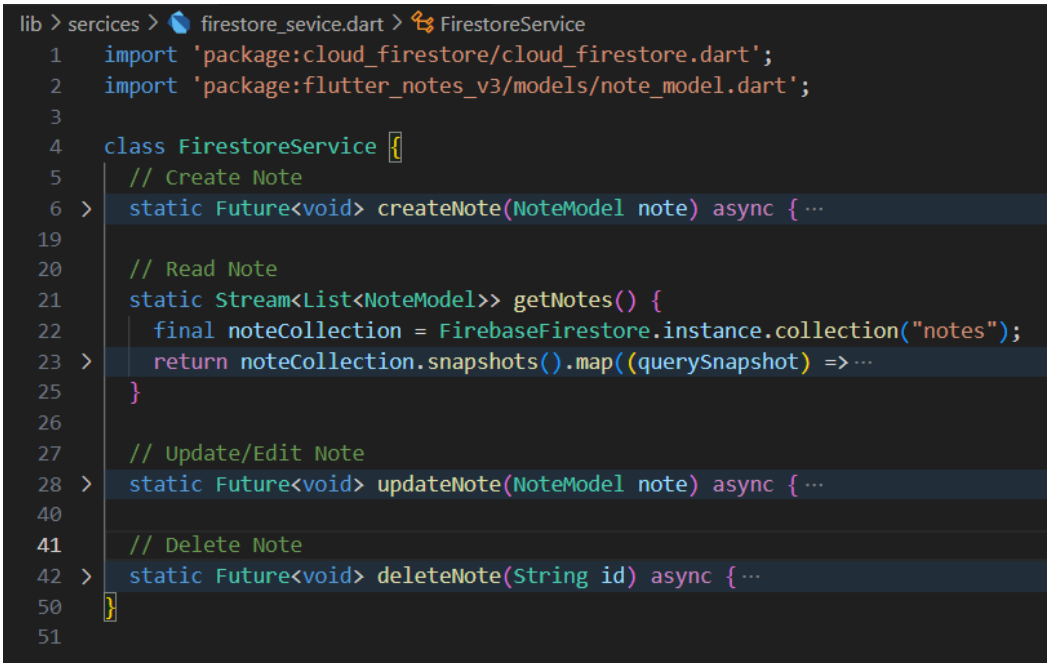

Рисунок 3.3 – Сервіс взаємодії з базою даних Firestore

Метод updateNote – цей метод приймає об’єкт NoteModel як аргумент. Спочатку він отримує посилання на колекцію Firestone "notes". Потім він створює новий об’єкт NoteModel з тим самим ідентифікатором, назвою, основною частиною та кольором, що і вхідна нотатка. Потім він оновлює документ у колекції Firestore, додаючи ідентифікатор нотатки та нові дані для нотатки. Якщо документ не існує, він буде створений.

Метод deleteNote – цей метод приймає ідентифікатор нотатки як аргумент. Він отримує посилання на колекцію Firestore "notes", а потім видаляє документ із зазначеним ідентифікатором із колекції. Якщо документ не існує, ця операція нічого не робить.

Для роботи з нотатками додаток має три віджети HomePage, CreateNotePage та EditNotePage. HomePage відображає нотатки надає і можливість взаємодії з

ними, одне натискання для редагування нотатки, два натискання для видалення нотатки, для створення нотаток використовується окрема кнопка рисунок 3.4.

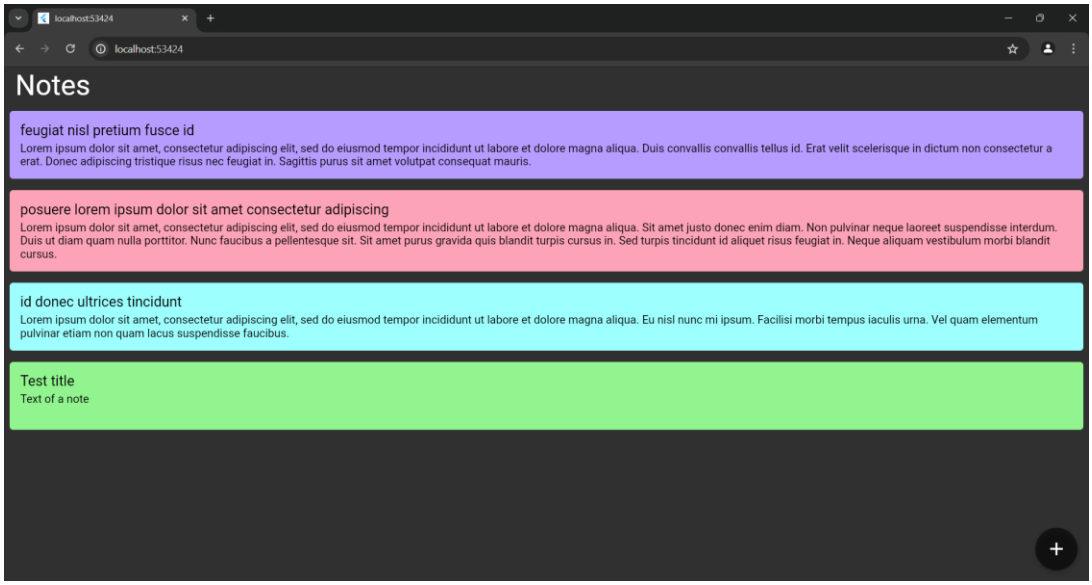

Рисунок 3.4 – Віджет HomePage

Віджети CreateNotePage та EditNotePage використовуються для створення та редагування нотаток відповідно.

Flutter додаток тестування рендерингу відображає постійно анімовану сітку елементів. Додаток використовує створює анімовану сітку в Flutter за допомогою віджета GridView для компонування і комбінації віджетів AnimationController, Tween і Transform для досягнення ефекту безперервного обертання рисунок 3.5.

```
class MyHomePage extends StatefulWidget {
  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage>
    with SingleTickerProviderStateMixin {
  late AnimationController controller;
  late Animation rotateAnimation;

  @override
  void initState() {···

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
          child: GridView.count(
        crossAxisCount: 10,
        children: [···
      )), // GridView.count // Center
    ); // Scaffold
  }
}
```

Рисунок 3.5 – Flutter додаток тесту рендерингу

Процес розробки Flutter додатку під велику кількість платформ має великий ступінь уніфікації. Сама розробка не мала проблем на усіх платформах, лише використання бібліотеки fluttertoast погребувало опрацювання того на якій платформі виконується додаток.

3.3 Програмна реалізація React Native додатку

React Native додаток складається з чотирьох компонентів Home, Header, NoteAdd та Details.

Компонент Home використовується для відображення списку нотаток, витягнутих з бази даних Firestore, і забезпечує навігацію по інших екранах рисунок 3.6, код наведено на рисунку 3.7.

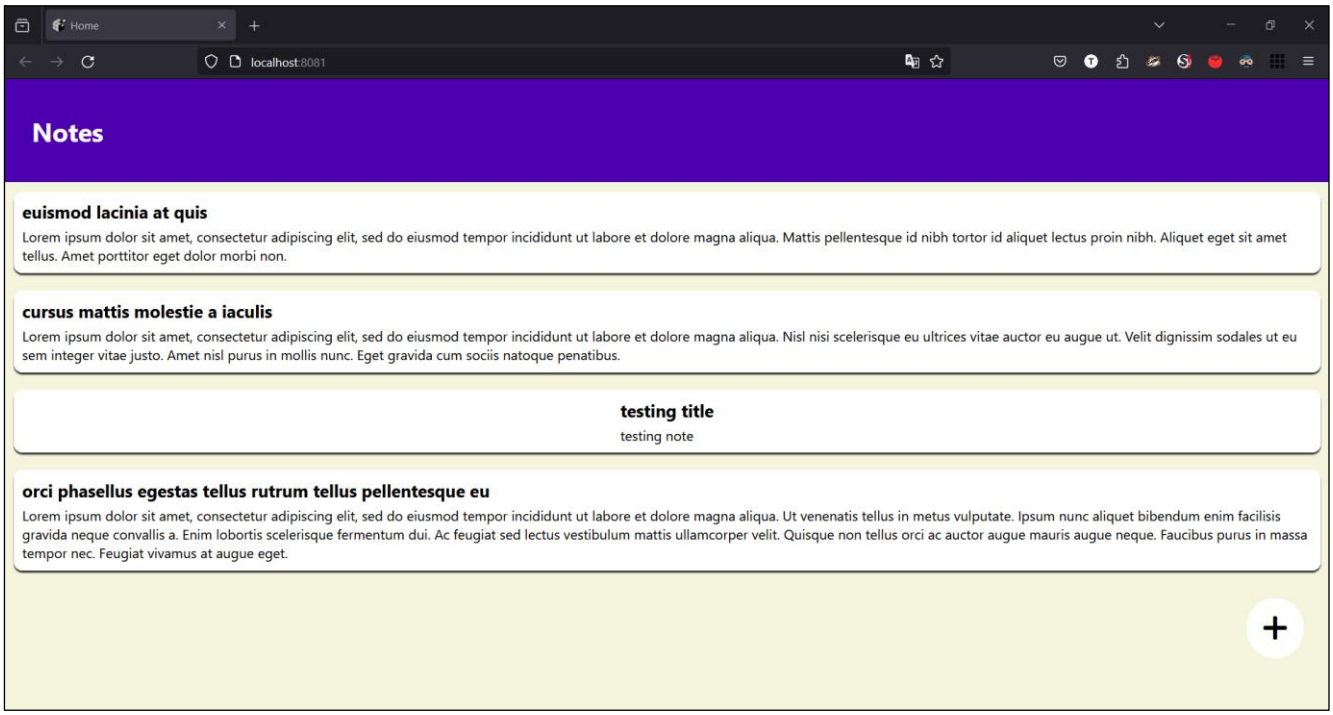

Рисунок 3.6 – Компонент Home

Всередині компонента Home оголошені дві змінні стану:

– notes, для зберігання списку нотаток;
– navigation, для управління навігацією між екранами.

Функція useEffect використовується для вилучення нотаток із бази даних Firestore під час підключення компонента.

Компонент повертає подання, що містить список FlashList і TouchableOpacity. FlashList використовується для відображення списку нотаток. Кожна нотатка – це компонент, який при натисканні переходить на екран Details. TouchableOpacity – це кнопка, яка при натисканні переходить на екран NoteAdd.

```
const Home = () => {

    const [notes, setNotes] = useState([]);
    const navigation = useNavigation();

    // fwetch the data from firestore

    useEffect(() => {
        firebase.firestore()
            .collection('notes')
            .onSnapshot((querySnapshot) => {
                const newNotes = [];

                querySnapshot.forEach((doc) => {···
                });
                setNotes(newNotes);
            });
    }, []);

    return (
        <View style={styles.container}>
            <FlashList
                data={notes}···
                )}
            />
            <TouchableOpacity
                style={styles.button}
                onPress={() => navigation.navigate('NoteAdd')}
            >
                <Entypo name="plus" size={45} color="black" />
            </TouchableOpacity>
        </View>
    )
}
```

Рисунок 3.7 – Код компоненту Home

Компонент Detail використовується для відображення детальної інформації про вибрану нотатку та надання опцій для оновлення або видалення нотатки rjl на рисунку 3.8.

Всередині компонента Detail оголошені дві змінні стану: noteText і noteTitle. Вони ініціалізуються за допомогою відомостей про примітки, переданих через route prop.

Функція handleUpdate оновлює нотатку в базі даних Firestore відповідно до поточного стану noteTitle та noteText. Після оновлення програма повертається на

головний екран. Якщо під час оновлення виникає помилка, вона видає повідомлення про помилку.

```
const Detail = ({ route }) => {

    const navigation = useNavigation();
    const [noteText, setNoteText] = useState(route.params.item.item.note);
    const [noteTitle, setNoteTitle] = useState(route.params.item.item.title);

    const handleUpdate = () => {···
    }

    const handleDelete = () => {···
    }

    return (
        <View style={styles.container}>
            <TextInput
                placeholder='Title'···
                style={styles.inputTitle}
            />
            <TextInput
                placeholder='Note'···
                multiline={true}
            />
            <View style={styles.buttonView}>
                <TouchableOpacity···
                </TouchableOpacity>
                <TouchableOpacity···
                </TouchableOpacity>
            </View>

        </View>
    )
}
```

Рисунок 3.8 – Код компоненту Detail

Функція handleDelete видаляє нотатку з бази даних Firestore, а потім повертає на головний екран. Якщо під час видалення виникає помилка, виводиться повідомлення про помилку.

Компонент повертає подання, що містить два компоненти TextInput для заголовка та тексту нотатки та два компоненти Touchable Capacity для дій оновлення та видалення.

Компонент NoteAdd використовується для додавання нової нотатки до бази даних Firestore код на рисунку 3.9.

```
const NoteAdd = () => {
    const [title, setTitle] = useState('');
    const [note, setNote] = useState('');
    const navigation = useNavigation();

    const handleAdd = () => {···
    }

    return (
        <View style={styles.container}>
            <TextInput
                placeholder='Title'···
                style={styles.inputTitle}
            />
            <TextInput
                placeholder='Note'···
                multiline={true}
            />
            <TouchableOpacity
                style={styles.button}
                onPress={handleAdd}>
                <Text style={styles.buttonText}>
                    Add
                </Text>
            </TouchableOpacity>
        </View>
    )
}
```

Рисунок 3.9 – Код компоненту NoteAdd

Всередині компонента NoteAdd оголошені дві змінні стану: title і note. Вони використовуються для зберігання введених користувачем даних про назву і зміст замітки.

Функція handleAdd додає нову нотатку до бази даних Firestore із поточним станом заголовка та примітки. Після додавання нотатки вона скидає змінні стану заголовка та примітки, вимикає клавіатуру та переходить назад на головний екран. Якщо під час додавання виникає помилка, виводиться повідомлення про помилку.

Компонент повертає подання, що містить два компоненти TextInput для заголовка та вмісту нотатки, а також компонент TouchableOpacity для дії додавання.

React Native додаток тестування рендерингу відображає постійно анімовану сітку елементів. Додаток використовує компонент FlatList для ефективного відображення сітки та бібліотеку Animatable для застосування безперервної анімації до окремих елементів рисунок 3.10.

```
export default class App extends React.Component {
  renderItem = ({ item, index }) => {
    if (item.empty === true) {
      return <View style={[styles.item, styles.itemInvisible]} />;
    }
    return (
      <View style={styles.item}>

        <Animatable.View style={{ width: '100%', height: '100%' }} animation="rotate"
          iterationCount="infinite" duration={1000} easing='linear'>
          <Image
            resizeMode={'cover'}
            style={{ width: '100%', height: '100%' }}
            source={require('./assets/logo.png')}
          />
        </Animatable.View>
      </View>
    );
  };

  render() {
    return (
      <FlatList
        data={elements}
        style={styles.container}
        renderItem={this.renderItem}
        numColumns={numColumns}
      />
    );
  }
}
```

Рисунок 3.10 – React Native додаток тесту рендерингу

Процес розробки React Native додатку під велику кількість платформ має уніфікацію для мобільних та веб платформи, але настільна платформа підтримується окремою організацією. Підтримка різними організаціями робить неможливим використання, деяких інструментів та бібліотек, більшість з яких не підтримують настільну платформу. Сама розробка не мала проблем на мобільній та веб платформі платформах, для настільної платформи був розроблений лише бенчмарк через відсутність підтримки Firebase та Expo.

4 ОПИС ЕКСПЕРЕМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

У цьому розділі будуть розглянуті результати досвіду розробки та експериментального порівняння додатків, що були розроблені з використанням React Native та Flutter, порівняння продуктивності фреймворків розглядається у [25-26]. Додатки було запущено на наступних пристроях. Смартфон Samsung Galaxy M52 Android 13. ПК з ОС Windows 11, AMD Ryzen 7 6800H, 32GB ОЗУ, Nvidia RTX 3050Ti. ПК вказаний раніше був використаний для тестування веб-застосунків у браузері Google Chrome.

4.1 Аналіз альтернатив для мобільної розробки

React Native і Flutter мають незаперечні переваги для кросплатформенної мобільної розробки. React Native ідеально підходить для розробників з досвідом роботи на JavaScript, яким потрібна розвинена екосистема. Flutter відрізняється своєю продуктивністю та цілісним досвідом розробки, хоча вимагає вивчення Dart. Вибір між ними повинен базуватися на вимогах проекту, наявному досвіді та конкретних вимогах до продуктивності та узгодженості інтерфейсу користувача.

React Native використовує JavaScript, використовуючи бібліотеку React для створення інтерфейсів користувача. Повсюдність JavaScript та популярність React роблять його доступним для широкого кола розробників, особливо тих, хто має досвід веб-розробки.

Flutter використовує мову Dart, розроблену Google. Dart призначений для розробки на стороні клієнта та пропонує функції, які полегшують створення мобільних додатків. Хоча синтаксис Dart менш поширений, ніж JavaScript, він знайомий розробникам, які мають досвід роботи з Java або C#.

Для перевірки додатки були перевірені на власному пристрої:

- смартфон: Samsung M52;
- версія Android: 13;
- програмне забезпечення для отримання інформації: GPUWatch.

Реалізовані додатки наведені на рисунках 4.1 та 4.2.

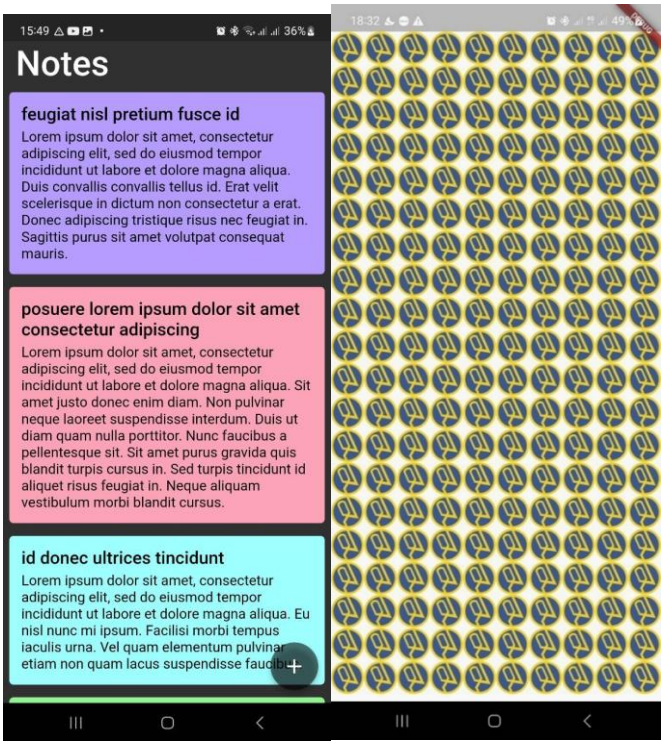

Рисунок 4.1 – Flutter мобільні додатки

Незважаючи на те, що реалізовані програми зазнали складного завдання візуалізації, що включає безперервне обертання 200 зображень у сітці, візуально помітних проблем із продуктивністю не спостерігалося в жодному фреймворку.

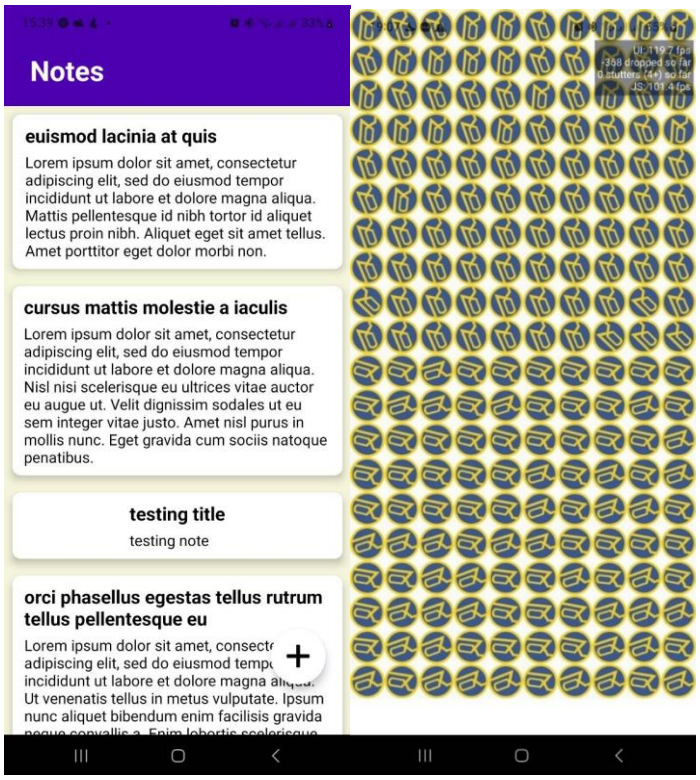

Рисунок 4.2 – React Native мобільні додатки

Це говорить про високий рівень оптимізації в межах кожного фреймворку для обробки складних оновлень інтерфейсу користувача.

Результати вимірювання на власному пристрої наведені у таблиці 4.1. З отриманих результатів можна отримати висновки, шо Flutter має просадки у FPS, в порівнянні з React Native, але Flutter використовує значно менше CPU, трохи менше пам’яті та значно менше GPU, ніж React Native.

Таблиця 4.1 – Результати вимірів GPUWatch

	Середній FPS	Використання CPU	Використання GPU
Flutter	80	30%	53 MB
React Native	103	44%	97 MB

З цього можна зробити наступний висновок, що для максимально плавної анімації, React Native може бути оптимальним вибором, але FPS, що має Flutter додаток є також прийнятним і переваги у менших вимогах до ресурсів пристрою, роблять Flutter кращим вибором.

4.2 Аналіз альтернатив для настільної розробки

Flutter і React Native, спочатку розроблені для мобільної розробки, також були розширені для підтримки настільних платформ в різному ступені. Flutter підтримує створення додатків для настільних платформ, таких як Windows, macOS та Linux. Google активно працює над розширенням можливостей Flutter для настільних середовищ поряд з мобільними. Незважаючи на те, що Flutter для настільних комп’ютерів досягає прогресу, вважається, що він все ще знаходиться на експериментальній стадії, і деякі функції можуть бути неповними або нестабільними React Native також має експериментальну підтримку для створення додатків для настільних платформ, таких як Windows, macOS та Linux. Але на відміну від Flutter в React Native ця підтримка забезпечується спільнотою. React Native для настільних комп’ютерів все ще знаходиться на експериментальній стадії, і підтримка може змінюватися залежно від платформи. Деякі функції

можуть бути неповними або нестабільними. В нашому випадку була відсутня підтримка хмарної платформи Firebase, що унеможливило розробку настільного додатку для зберігання нотаток. Оскільки React Native в першу чергу розроблявся для мобільних пристроїв, для деяких функцій, специфічних для настільних комп'ютерів, можуть знадобитися додаткові або власні модулі. Адаптовані до Windows додатки наведені на рисунках 4.3 та 4.4.

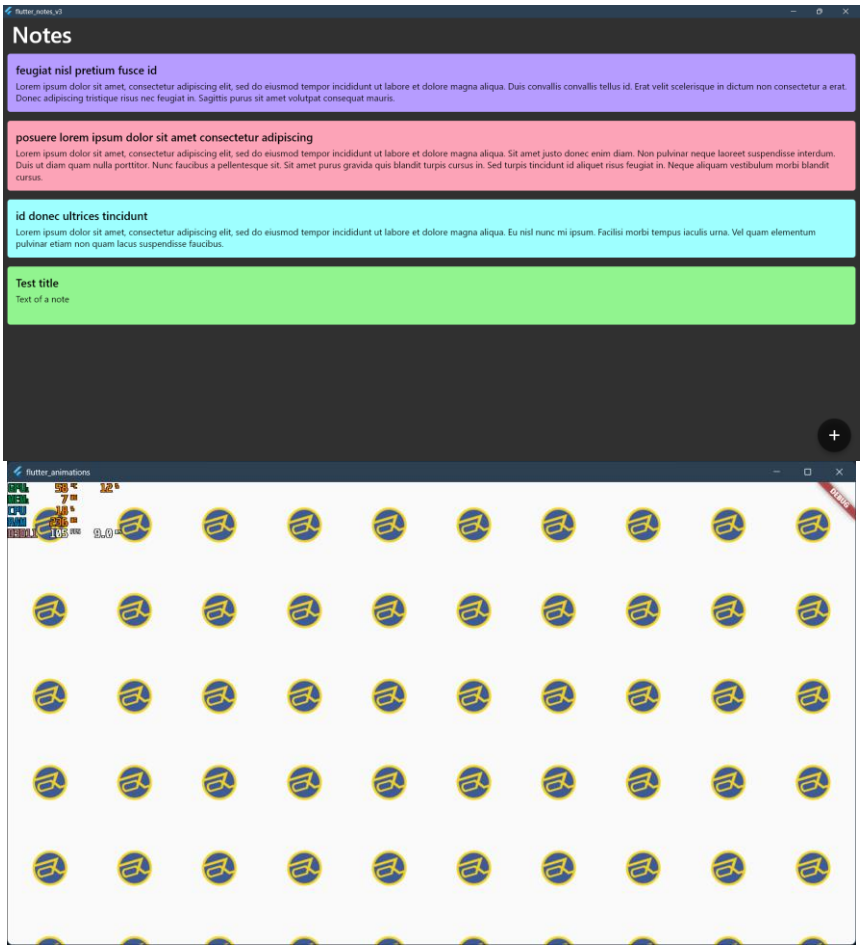

Рисунок 4.3 – Flutter настільні додатки

Flutter надає всебічний досвід розробки для створення власних настільних додатків у Windows. Досвід розробки був на рівні з мобільним додатком, що підтверджується досвідом інших розробників де в середньому 78% розробників настільних додатків Flutter повідомили, що вони дуже або певною мірою задоволені своїм досвідом розробки [27]. Результати показали, що задоволеність мала тенденцію до зниження в процесі розробки настільного додатка. Це викликано найменшим задоволенням користувачів при налагодженні

продуктивності та розгортанні додатку. Також зазначається, що підтримка настільних пакетів все ще наздоганяє підтримку мобільних пристроїв та веб-застосунків.

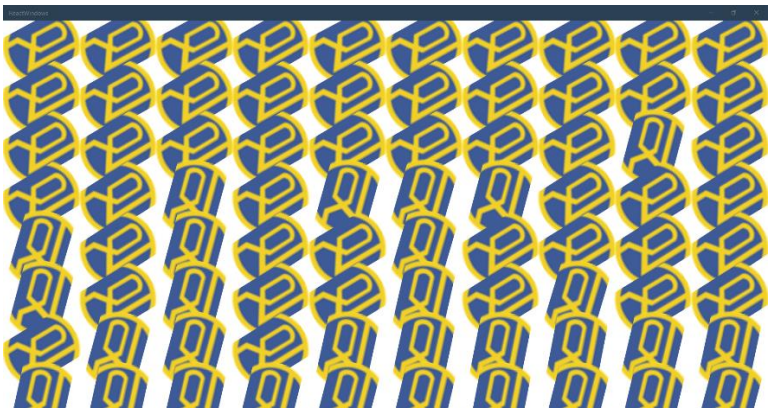

Рисунок 4.4 – React Native настільний додаток тестування рендерингу

У той час як React Native домінує в розробці мобільних додатків, перехід до розробки Windows з його допомогою являє собою унікальний досвід, пов’язаний як з проблемами, так і з потенційними перевагами. React Native сам по собі не пропонує великої вбудованої підтримки для розробки Windows. Для досягнення цієї мети ви в основному будете покладатися на сторонні бібліотеки та проекти, керовані спільнотою. Це призводить до потенційних проблем сумісності, і пошук підтримки може виявитися складнішим у порівнянні з встановленими платформами Windows. В нашому розробка настільного додатку для зберігання нотаток була немотовою через відсутність підтримки сторонніх бібліотек, також залежності для розробки Windows додатку за допомогою React Native склали 4 GB.

Результати порівняння представлені в таблиці 4.2.

Таблиця 4.2 – Результати порівняння

	FPS	Середнє використання CPU	Середнє використання пам’яті
Flutter	105	18%	200 MB
React Native	45	20%	160 MB

З отриманих результатів можна отримати висновки, шо Flutter має майже втричі більший FPS, ніж React Native, використовує трохи менше CPU але більше пам’яті, ніж React Native.

З цього можна зробити наступний висновок, що Flutter має кращу підтримку настільних платформ та може бути кращим вибором для складних інтерфейсів, що потребують високої частоти кадрів

4.3 Аналіз альтернатив для веб-розробки

Flutter, і React Native можна використовувати для веб-розробки, але вони мають різні сильні та слабкі сторони. Flutter for Web пропонує повторне використання коду та гарячу перезавантаження, але все ще знаходиться в бета-версії, тоді як React Native for Web надає знайомий досвід розробки React, але може вимагати додаткових налаштувань для веб-функцій. Адаптовані до Web додатки наведені на рисунках 4.5 та 4.6. Порівняння продуктивності Flutter, та React Native проводили за допомогою веб-браузеру Google Chrome.

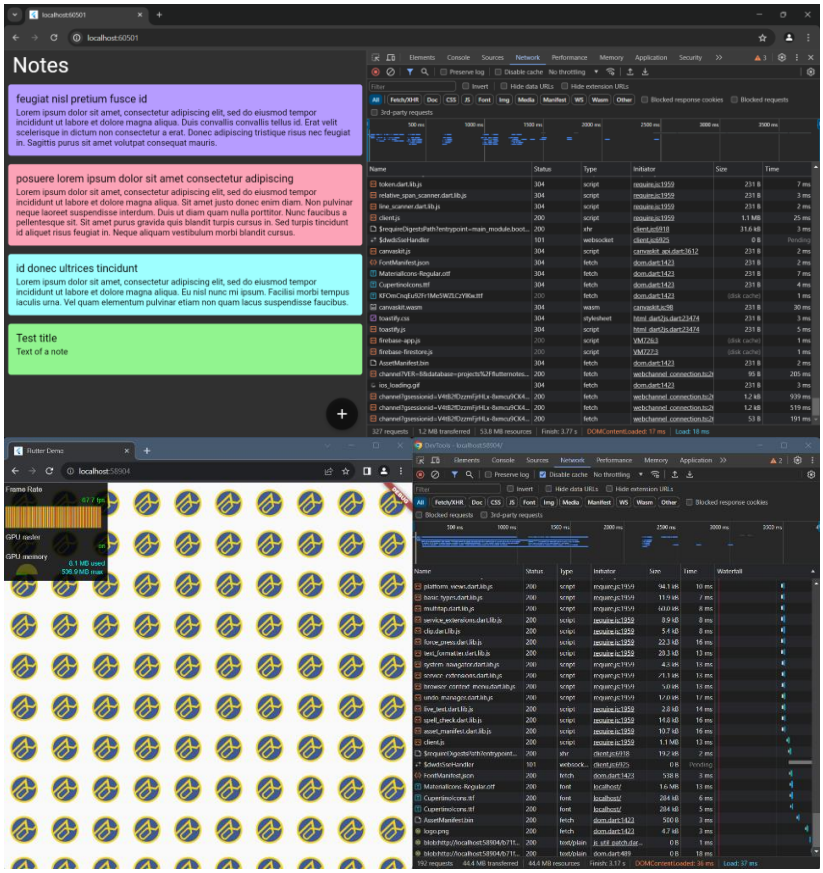

Рисунок 4.5 – Flutter веб-додатки

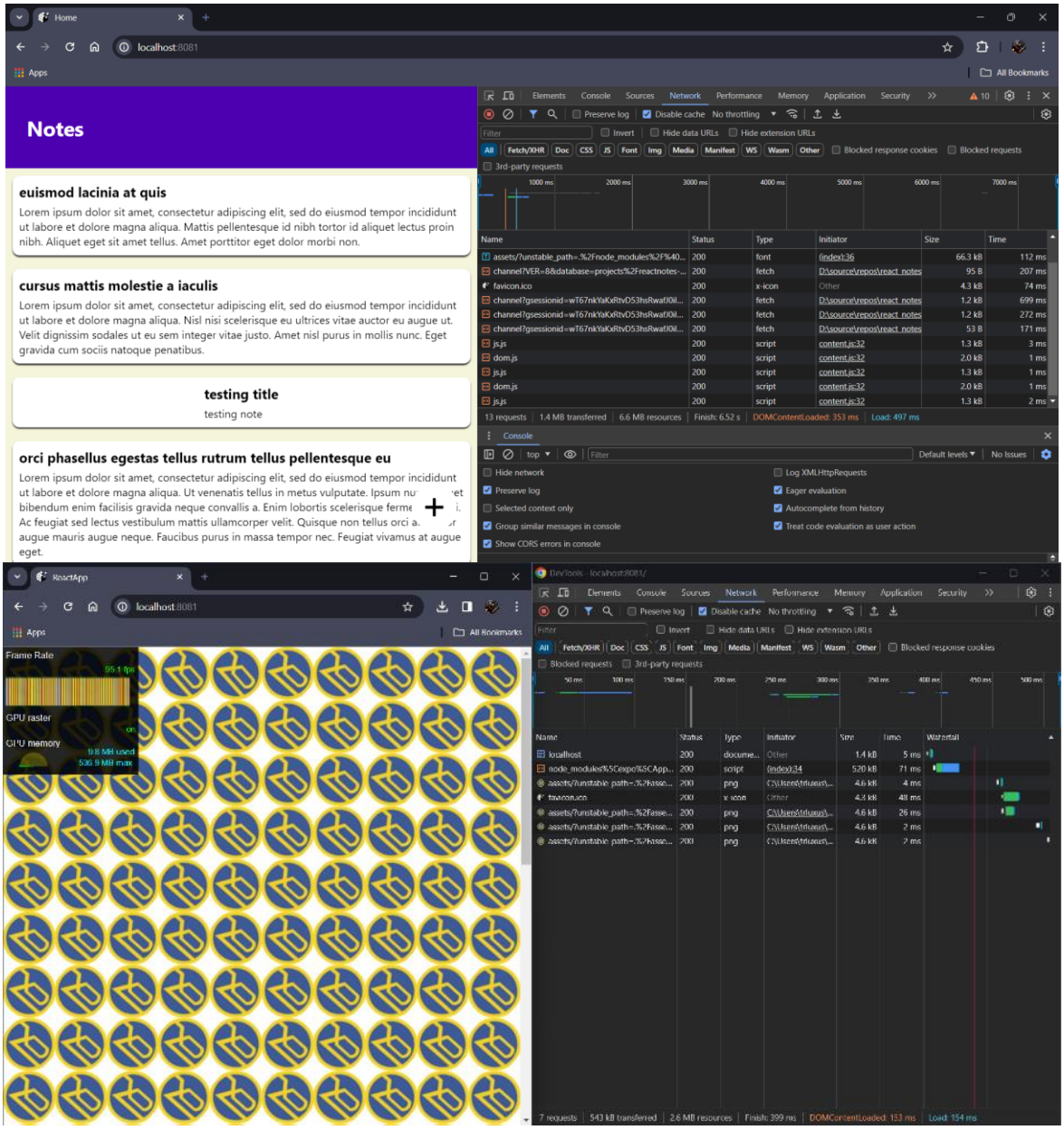

Рисунок 4.6 – React Native веб-додатки

Досвід розробки веб-додатку за допомогою Flutter не має особливих відмінностей від інших платформ. Проте саме з розробкою веб-додатків Flutter має певні недоліки. Сучасні користувачі очікують, що сайти будуть завантажуватися негайно, особливо при хорошому інтернет-з’єднанні. Користувач бачить посилання, натискає на нього і хоче щось отримати менш ніж за секунду. Якщо він цього не отримуєте, він переходить на наступний посилання. Все відбувається дуже швидко. Але веб-додатки, створені за допомогою Flutter, завантажуються приблизно за 3-5 секунд після оптимізації, поки ви не побачите

всю сторінку. Це може бути поліпшено використанням стороннього завантажувача, але не вирішено повністю, якщо користувачі часто використовують веб-додаток, повільний час завантаження буде для них поганим досвідом.

Стандартно клавіатура не працює з Flutter веб-додатком. Натискання вгору та вниз не впливає на смугу прокрутки. Що потребує розробки власного рішення. Також у веб-додатку Flutter стандартно текст, що відображається на екрані, насправді не текст, а зображення. Таким чином, користувач не можете шукати його, не може виділити його, і не може скопіювати його.

Загалом, React Native для веб-додатків забезпечує універсальний та ефективний спосіб створення веб-додатків. До недоліків React Native для розробки веб-додатків можна віднести те, що не всі компоненти та API React Native доступні в Інтернеті, тому може знадобитися знайти альтернативи або обхідні шляхи для деяких функцій.

Таблиця 4.3 – Результати порівняння

	FPS	Розмір додатку	Час завантаження	Використання GPU
Flutter	68	45 MB	3017 MS	8 MB
React Native	95	3 MB	400 MS	10 MB

React Native, і Flutter не є оптимальними засобами при розробці веб-додатків, але все ще надають перевагу при розробці, як кросплатформенні фреймворки. З отриманих результатів таблиця 4.3 можна отримати висновки що веб-додатки є платформою на якій React Native має значну перевагу над Flutter

ВИСНОВКИ

В роботі проаналізовані актуальні методи для розробки кроссплатформенних графічних інтерфейсів. Були обрані методи за допомогою векторної оптимізації та апарату теорії корисності. Були досліджені додатки розроблені за допомогою фреймворків React Native та Flutter для мобільних пристроїв, настільних комп'ютерів та Інтернет.

Було розроблено додатки для збереження нотаток, була використана клієнт серверна архітектура з тонким клієнтом для підтримки великої кількості платформ. Додатки були розроблені з використанням фреймворків React Native та Flutter, дані фреймворки виявилися найбільш оптимальними серед фреймворків представлених у галузі після застосування векторної оптимізації та апарату теорії корисності. Для порівняння продуктивності для кожного з фреймворків був розроблений окремий додаток бенчмарк.

Розробка додатків для збереження нотаток показала високий ступінь уніфікації при розробці з використанням фреймворку Flutter для настільної, мобільної та веб платформи. Незважаючи на те, що підтримка настільної та веб платформи знаходиться в ранній стадії додаток було без проблем реалізована на усіх трьох платформах. React Native має високий рівень підтримки мобільної та веб платформи, але підтримка настільної платформи спільнотою виявилася недостатньою для реалізації додатку на цій платформі.

Порівняння продуктивності показали перевагу Flutter на мобільній та настільній платформі, фреймворки показували приблизно однакову продуктивність, але Flutter витрачав менше ресурсів системи, на якій проводилось тестування. На веб платформі React Native мав невелику перевагу у продуктивності при значно менших вимогах до ресурсів.

В результаті було зроблено висновок, що через кращу підтримку різних платформ і продуктивність додатків Flutter, є оптимальним методом розробки кроссплатформенних графічних інтерфейсів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Krajewski, Marek. “Cross-platform development of the Smart Client application with Qt framework and QtQuick.” Vaasan ammattikorkeakoulu, 2016. https://www.theseus.fi/handle/10024/113519.

2. Alymkulov, Daniyar. “Desktop Application Development Using Electron Framework: Native Vs. Cross-Platform,” 2019. https://www.theseus.fi/handle/10024/167669.

3. Prajapati, Mukesh, Dhananjay Phadake, and Archit Poddar. "Study on xamarin cross-platform framework." Int J Tech Res Appl 4, no. 4 (2016): 13-8.

4. Wu, Wei-Tao. “React Native Vs Flutter, Cross-platforms Mobile Application Frameworks,” 2018. https://www.theseus.fi/handle/10024/146232.

5. Biørn-Hansen, Andreas, Christoph Rieger, Tor-Morten Grønli, Tim A. Majchrzak, and Gheorghiţă Ghinea. “An Empirical Investigation of Performance Overhead in Cross-platform Mobile Development Frameworks.” Empirical Software Engineering 25, no. 4 (June 9, 2020): 2997–3040. https://doi.org/10.1007/s10664-020-09827-6.

6. Shukla, Abhishek. “Comparative Analysis of Native and Hybrid Applications and Their Future.” Journal of Artificial Intelligence & Cloud Computing, December 31, 2022, 1–6. https://doi.org/10.47363/jaicc/2022(1)136.

7. Zohud, Tasnim, and Samer Zein. “Cross-Platform Mobile App Development in Industry: A Multiple Case-Study.” International Journal of Computing, March 30, 2021, 46–54. https://doi.org/10.47839/ijc.20.1.2091.

8. Cirani, Simone, Marco Picone, Luca Veltri, Luca Zaccomer, and Francesco Zanichelli. “ZWT: A New Cross-platform Graphical Interface Framework for Java Applications.” SoftwareX 12 (July 1, 2020): 100599. https://doi.org/10.1016/j.softx.2020.100599.

9. Thokal, Vrushali, Sunny Kirtkar, Abdul Qadir Tinwala, and Mohd Aatif Mohd Shakil. "A Modern GUI Desktop Application with Advance Features for GST Billing and Management."

10. Volianskyi, Vitalii. "Development of cross-platform file system manager."

(2021).

11. Boduch, Adam, and Roy Derks. React and react native: A complete hands-on guide to modern web and mobile development with react.js. Birmingham, U.K.: Packt Publishing Ltd., 2020.

12. Scoccia, Gian Luca, and Marco Autili. (2020). "Web frameworks for desktop apps: an exploratory study." Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM).

13. Моруга Д.І., Ревенчук І.А. Research of Methods for Development of Graphical User Interface in Cross-platform applications // Матер VII Міжнар наук.-практ конф. "Scientific Research: Theoretical Foundations and Practical Applications".- Відень 24-26.01.2024.- P.165-170.

14. Most used libraries and frameworks among developers, worldwide, as of 2023 [Електронний ресурс] – Режим доступу : www/ URL: https://www.statista.com/statistics/793840/worldwide-developer-survey-most-used-frameworks/ – 03.11. 2023 р. – Загол. з екрану.

15. Most used programming languages among developers worldwide as of 2023 [Електронний ресурс] – Режим доступу : www/URL: https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/ – 03.11.2023 р. – Загол. з екрану.

16. Tollin, Gustav, and Lidekrans Marcus. "React Native vs. Flutter: A performance comparison between cross-platform mobile application development frameworks." (2023).

17. Fentaw, Awel Eshetu. “Cross Platform Mobile Application Development : A Comparison Study of React Native Vs Flutter,” 2020. https://jyx.jyu.fi/handle/123456789/70969.

18. Stender, Simon, and Hampus Åkesson. “Cross-platform Framework Comparison : Flutter a React Native.” Blekinge Institute of Technology, 2020. http://www.diva-portal.org/smash/record.jsf?pid=diva2:1440825.

19. Hjort, Elin. “Evaluation of React Native and Flutter for cross-platform mobile application development.” Åbo Akademi, 2020.

https://www.doria.fi/handle/10024/180002.

20. Gülcüoğlu, Ekrem, Ahmet Üstün, and Neşet Seyhan. “Comparison of Flutter and React Native Platforms.” Internet Uygulamalari Ve Yönetimi, December 29, 2021. https://doi.org/10.34231/iuyd.888243.

21. Khan, Sharjeel Moqrab, Aftab Ul Nabi, and Tahir Hussain Bhanbhro. “Comparative Analysis Between Flutter and React Native.” International Journal of Artificial Intelligence & Mathematical Sciences 1, no. 1 (September 9, 2022): 15–28. https://doi.org/10.58921/ijaims.v1i1.19.

22. Zindl, Stefan. "Flutter on Windows Desktop: a use case based study." Bachelor's thesis, 2021.

23. Uciński, Mateusz, and Mariusz Dzieńkowski. “A Comparative Analysis of Performance of Flutter and Xamarin Development Frameworks.” Journal of Computer Sciences Institute 25 (December 30, 2022): 366–70. https://doi.org/10.35784/jcsi.3025.

24. Flutter vs React Native vs Qt in 2022 [Electronic Resource] (2022). Retrieved March 5, 2024, from https://scythe-studio.com/en/blog/flutter-vs-react-native-vs-qt-in-2022.

25. Моруга Д.І., Ревенчук І.А. Кордонні обчислення в якості основи розумного міста: матеріали 28-го Міжнарод. молодіжного форуму «Радіоелектроніка та молодь у XXI столітті». Зб. матеріалів форуму. Т. 6., Харків: ХНУРЕ. 2024. С.309-311.

26. Моруга Д.І., Ревенчук І.А. Unified Approach For Development Of Graphical User Interface In Cross-Platform Applications // Матер XXII Міжнар наук.-практ конф. "Modern Scientific Research: Theoretical and Practical Aspects".- Осло 08-10.05.2024.- Р.73-76.

27. How can we improve the Flutter experience for desktop? [Electronic Resource] (2022). Retrieved March 5, 2024, from https://medium.com/flutter/how-can-we-improve-the-flutter-experience-for-desktop-70b34bff9392.