Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів анімації з використанням фреймворку Flutter
(тема)

Виконав:
здобувачка 2 року навчання
групи ІПЗзм-23-1

Софія БИКОВСЬКА
(Власне ім'я, ПРІЗВИЩЕ)

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

Керівник доц. Олексій НАЗАРОВ
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми
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)

ЗАТВЕРЖДУЮ:
Зав. кафедри _____
(підпис)
«___»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
здобувачці *Биковської Софії Андріївни*
(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження методів анімацій з використанням фреймворку Flutter»

Затверджена наказом університету від "10" 04 20 25 р. № 55стз

2. Термін подання студентом роботи до екзаменаційної комісії "16" 06 2025 р.

3. Вихідні дані до роботи опис механізмів анімацій у Flutter (імпліцитні та експліцитні анімації), вимоги до реалізації доступного та продуктивного анімованого інтерфейсу, електронні ресурси за обраною темою (офіційна документація Flutter, Material Design, ARIA APG), мова програмування Dart, технологія Flutter SDK 3.x, середовище розробки Visual Studio Code, інструменти оцінки продуктивності та доступності: Flutter DevTools, Performance Overlay, Google Lighthouse, системні screen reader-и (TalkBack, VoiceOver).

4. Перелік питань, що потрібно опрацювати в роботі аналіз та порівняння типів анімацій у Flutter, вибір критеріїв для оцінки продуктивності та доступності, проектування та реалізація прикладових інтерфейсів з анімацією, тестування впливу анімацій на UX,проведення експериментів із застосуванням Flutter DevTools.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Позначка про виконання
1	Аналіз предметної галузі	08.04.2025 – 10.04.2025	*виконано*
2	Розробка постановки задачі	11.04.2025 – 16.04.2025	*виконано*
3	Аналіз середовища для проведення дослідження	17.04.2025 – 23.04.2025	*виконано*
4	Планування експериментального дослідження	24.04.2023 – 01.05.2025	*виконано*
5	Проектування середовища для дослідження	02.05.2025 – 06.05.2025	*виконано*
6	Програмна реалізація дослідження	06.05.2025 – 16.05.2025	*виконано*
7	Експериментальні дослідження та їх аналіз	17.05.2025 – 18.05.2025	*виконано*
8	Оформлення статті або тез доповіді	19.05.2025 – 24.05.2025	*виконано*
9	Підготовка пояснювальної записки	25.05.2025 – 06.06.2025	*виконано*
10	Підготовка презентації та доповіді	07.06.2025 – 10.06.2025	*виконано*
11	Нормоконтроль	10.06.2025	*виконано*
12	Рецензування	12.06.2025	*виконано*
13	Занесення диплома в електронний архів	13.06.2025	*виконано*
14	Попередній захист	14.06.2025	*виконано*
15	Допуск до захисту у зав. кафедри	15.06.2025	*виконано*

Дата видачі завдання 07 квітня 2025р.

Здобувачка _____ Софія БИКОВСЬКА

(підпис) (прізвище, ініціали)

Керівник роботи _____ доц.кафедри ПІ Олексій НАЗАРОВ

(підпис) (прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка кваліфікаційної роботи містить: 64 сторінок, 13 рисунків, 16 джерел.

АНІМАЦІЇ, КРОСПЛАТФОРМНА РОЗРОБКА, МОБІЛЬНА РОЗРОБКА, ФРЕЙМВОРК, ОПТИМІЗАЦІЯ, ПОРІВНЯННЯ, FLUTTER, DART

Об'єктом дослідження є анімації в мобільній розробці з використанням фреймворку Flutter, зокрема різні підходи до створення анімацій – implicit та explicit анімації, а також їх застосування для покращення інтерактивності та досвіду користувача.

Метою роботи є порівняння та дослідження переваг та недоліків використання implicit та explicit анімацій у Flutter, а також визначення оптимальних варіантів для різних типів інтерфейсів і сценаріїв.

Методами дослідження є аналіз існуючих підходів до анімацій в Flutter, вибір найбільш ефективних методів для впровадження анімаційних ефектів, а також оцінка їх впливу на продуктивність мобільних додатків.

У результаті дослідження було розглянуто два основні підходи до анімацій: використання implicit анімацій через анімаційні віджети, а також explicit анімацій, де контролюється кожен аспект анімації.

ANIMATIONS, CROSS-PLATFORM DEVELOPMENT, MOBILE DEVELOPMENT, FRAMEWORK, OPTIMIZATION, COMPARISON, FLUTTER, DART

The object of the study is animations in mobile development using the Flutter framework, in particular, different approaches to creating animations - implicit and explicit animations, as well as their application to improve interactivity and user experience.

The purpose of the work is to compare and study the advantages and disadvantages of using implicit and explicit animations in Flutter, as well as to determine the optimal options for different types of interfaces and scenarios.

The research methods are the analysis of existing approaches to animations in Flutter, the selection of the most effective methods for implementing animation effects, and the assessment of their impact on the performance of mobile applications.

As a result of the study, two main approaches to animations were considered: the use of implicit animations through animation widgets, as well as explicit animations, where every aspect of the animation is controlled.

Завідувачу кафедри
ПІ_____
(скорочена назва кафедри)
проф. Кирилу СМЕЛЯКОВУ_
(вчене звання, сласне ім'я, прізвище)

ЗАЯВА
щодо самостійності виконання кваліфікаційної роботи та можливості її публікації (та/або публікації анотації кваліфікаційної роботи) в електронному архіві відкритого доступу ElAr KhNURE

Я, Биковська Софія Андріївна, студентка гр. ІПЗзм-23-1, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: мій курсовий проєкт на тему «Дослідження методів анімацій з використанням фреймворку Flutter», що буде представлений для публічного захисту, виконаний самостійно, не містить елементи плагіату і може бути опублікований в електронному архіві з відкритим доступом ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

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

05.06.2025

ЗМІСТ

ВСТУП

Анімація відіграє життєво важливу роль у взаємодії з користувачем для сучасних мобільних додатків, оскільки вона не тільки підвищує привабливість інтерфейсу, але також допомагає або забезпечує більш природну та ефективну роботу користувача з додатком. Анімація відіграє кілька ролей у мобільних програмах: вона може бути простим візуальним елементом, а також служити механізмом для плавної навігації програмою. Flutter надає безліч ракурсів для виконання простих і складних анімацій завдяки гнучкості та вбудованим інструментам.

Ми можемо розділити анімацію у Flutter на два основні типи – неявну та явну. Найшвидший спосіб створення анімаційних ефектів – це неявна анімація. Хоча ці підходи мають переваги та недоліки, рівень контролю розробника та необхідна складність анімаційного ефекту визначатимуть, чи використовується прихована чи явна анімація.

Щоб досягти цього, прототипи дисплеїв будуть створені з найпопулярнішими анімаційними ефектами, такими як масштабування, рух елементів, зміни кольору тощо. Можна легко виміряти вплив кожного типу анімації на час відгуку програми, використання ресурсів та інтеграцію, оскільки вбудованих моделей.

Результати дослідження допоможуть визначити найкращі практики впровадження анімацій у мобільних додатках на основі Flutter, а також вибрати прозорі та чіткі анімації на основі конкретних потреб проекту. Ці знання можуть бути корисними для розробників, які хочуть покращити інтерактивність їхні програми й для тих, хто хоче покращити взаємодію з користувачем, і проблеми з оптимізацією функціональності Face під час роботи з анімацією.

Тому аналіз забезпечує глибше розуміння методів анімації у Flutter та їх впливу на кінцевий продукт, а також дає рекомендації щодо вибору методів анімації на основі цілей проекту та цілеспрямовано.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ І ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної галузі

Анімація є важливою частиною сучасних мобільних програм, оскільки вони можуть не тільки покращити естетичне сприйняття інтерфейсу, але й значно збільшити взаємодію користувача з програмою.

Для мобільних програм анімація діє як перехід інтерфейсу між станами програми. і забезпечує більш приємний досвід користувача.

У фреймворку Flutter анімація може бути реалізована двома основними способами: явна (explicit) та неявна (implicit) анімація.

Кожен із цих методів має свої унікальні особливості та використовується в різних середовищах розробки мобільних додатків.

У цьому дослідженні порівняються ці дві моделі, їхні переваги та недоліки, а також те, наскільки добре вони використовуються в реальних проектах.

У розробці програмного забезпечення анімація – це плавний перехід між різними станами елементів інтерфейсу, який створюється зміною властивостей віджетів, таких як розмір, колір, прозорість, положення елементів на екрані тощо.

Flutter пропонує широкий набір інструментів для створення анімацій та спеціальні віджети для автоматизації й поліпшення цього процесу, наприклад AnimatedContainer, AnimatedOpacity, AnimatedPositioned [1].

Анімації у Flutter можна розділити на дві категорії: явні та неявні. Вибраний тип анімації залежить від складності інтерфейсу, який буде використовуватися, і рівня контролю, який розробник хоче використовувати для відтворення.

Implicit тип – це анімації, які автоматично застосовуються до змін властивостей віджетів без необхідності вручну вказувати точні параметри анімації.

Ця анімація є найбільш корисною для простих ситуацій, коли під час розробки необхідно лише налаштувати зовнішній вигляд елемента, коли змінюється його стан (наприклад, змінюється розмір або колір).

Вбудована анімація використовує роль вбудованих механізмів керування анімацією, потреби розробника лише для того, щоб вказати змінні.

Explicit анімації дає розробнику повний контроль над процесом анімації.

Потрібна детальна настройка, оскільки будь-які зміни властивостей мають бути явно визначені AnimationController і об’єктом Tween.

Tween – це значення, яке вказує, як значення змінюється між двома точками (наприклад, від 0 до 1).

AnimationController є основним контролером для анімації: він визначає тривалість, випередження часу та інтервал анімації.

За допомогою CurvedAnimation, ми можемо додати додаткові ефекти, такі як прискорення або уповільнення анімації, що забезпечує плавні та динамічні переходи.

Але під час обрання того чи іншого анімації можуть виникнути дилеми й питання доцільності використання в різних ситуаціях.

Одне з найголовніших питань, котре необхідно розглянути, – це продуктивність, а саме те, наскільки добре виконується обробка анімації на різних пристроях.

Погано розроблені анімації можуть затримувати або сповільнювати програму, що особливо помітно на старих машинах або під великим навантаженням.

Пасивна анімація може бути дуже продуктивною для змін, які не викликають труднощів, оскільки Flutter сам обробляє ці анімації, зменшуючи навантаження на розробник [2].

Не менш важливою проблемою є складність підтримки анімації протягом тривалого часу, особливо у великих програмах.

Зменшення часу розробки, підвищена гнучкість анімації та ефективність використання системних ресурсів є важливими міркуваннями для розробників при виборі типу анімації, яку вони підтримують, щоб вирішити проблеми оптимізації взаємодії користувача та програми та підвищити загальну продуктивність.

Таким чином, вибір явної та неявної анімації залежить від конкретних вимог проекту та бажаного контролю над анімацією, а також від необхідності досягнення оптимальної продуктивності на різних пристроях.

1.2 Постановка задачі

В рамках цього курсового проєктування важливо організувати проєкти, спрямовані на детальне вивчення неявної та явної анімації в фреймворці Flutter.

Це дослідження спрямоване на вивчення об’єктів та об’єктів із реалізованою анімацією у Flutter, щоб вибрати між цими двома методами, найкращими для конкретного типи інтерфейсу та взаємодії з користувачем.

Під час цієї роботи наведені основні завдання, котрі необхідно вирішити:

- оцінка ефективності явної та неявної анімації в контексті Flutter;
- порівняння швидкості анімації на основі складності;
- огляди продуктивності анімації на різних пристроях;
- тестування анімації на пристроях різної продуктивності (від простих смартфонів до потужної графіки);
- визначення впливу різних анімацій на час відгуку користувача та частоту кадрів (FPS);
- аналіз використання системних ресурсів (CPU, GPU, пам'ять) під час анімації;
- критерії проведення (наприклад, складність анімації, кількість елементів на екрані, час анімації) для вибору відповідної технології.

Загалом робота має на меті визначити ключові принципи та прийоми ефективного використання анімації у фреймворці Flutter, які допоможуть досягти вищої продуктивності та зручності у розробці мобільних додатків.

2 ОПИС ПРИЙНЯТИХ ПРОЄКТНИХ РІШЕНЬ

2.1 Аналіз та вибір середовища для проведення дослідження

Для виконання експериментальної частини роботи, пов'язаної з дослідженням впливу анімації на доступність та продуктивність мобільних додатків, було необхідне середовище, яке підтримує гнучку та масштабовану реалізацію динамічних інтерфейсів та надає можливість відстежувати технічні показники на рівні рендерингу компонентів інтерфейсу.

Здатність фреймворку виконувати складну анімацію була лише одним з основних критеріїв вибору; інші фактори включали підтримку інструментів аналізу продуктивності, дотримання принципів інклюзивного дизайну та надійну екосистему з вільно доступною документацією та реальними прикладами.

На сьогоднішній день Flutter від Google та React Native від Meta є найпопулярнішими інструментами крос-платформної розробки.

Обидві платформи пропонують можливість створювати нативні мобільні додатки зі спільним кодом для Android та iOS, але мають різні підходи до архітектури рендерингу інтерфейсу, управління анімацією та доступу до нативних ресурсів платформи.

Згідно з результатами незалежного бенчмаркінгу, про який йдеться у статті «Flutter vs React Native Performance Benchmarks You Can't Miss» [3], Flutter має кращу продуктивність під час стресового рендерингу UI з анімацією.

Точніше, при відображенні великих анімованих списків Flutter підтримує стабільну частоту кадрів 60 FPS, в той час як React Native помітно падає до 25-30 FPS, за рахунок плавності інтерфейсу (див.рис.2.1, 2.2).

Даний бенчмаркінг підкреслює різні властивості кожної з платформ, враховуючи найважливіші критерії при порівнянні продуктивності.

У цьому тесті оцінювалася плавність прокрутки списку з великою кількістю елементів. Flutter показав стабільну частоту 60 кадрів на секунду (FPS) без втрат продуктивності навіть при інтенсивній взаємодії, тоді як React Native

продемонстрував зниження продуктивності до 25–30 FPS, що викликало помітні лаги. Це свідчить про ефективніший рендеринг та використання GPU у Flutter.

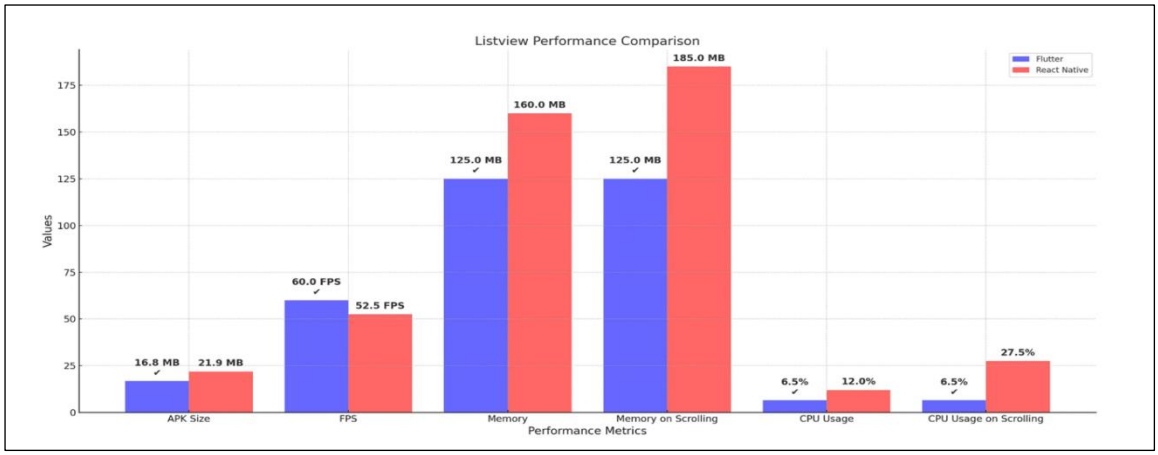

Рисунок 2.1 – Графік порівняння продуктивності (рисунок виконано самостійно)

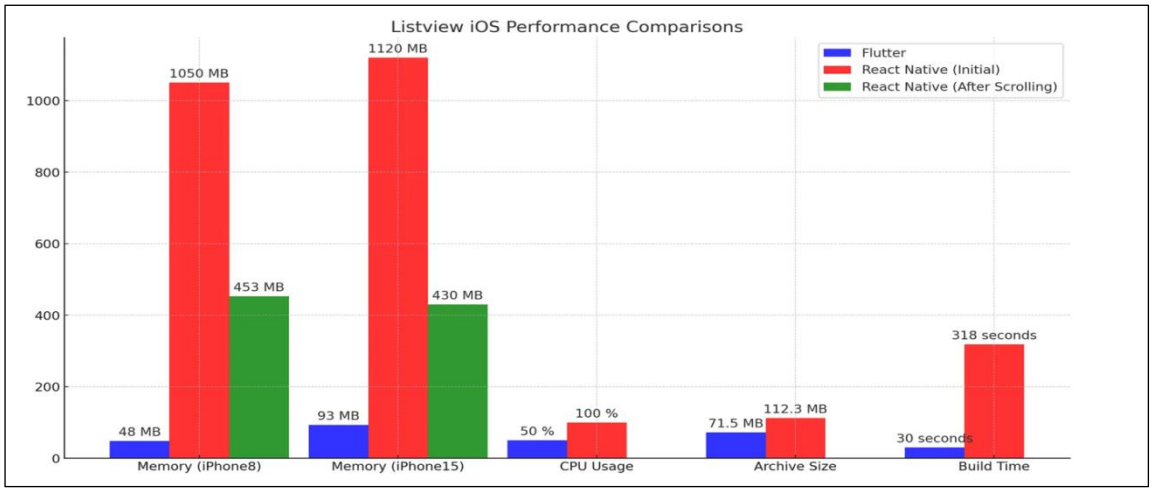

Рисунок 2.2 – Графік порівняння продуктивності (рисунок виконано самостійно)

У тесті з масовою анімацією зображень Flutter знову переважає: він забезпечив плавне відображення анімованих компонентів навіть за високого навантаження.

React Native продемонстрував затримки в рендерингу та збільшене навантаження на CPU, що негативно позначилось на інтерактивності застосунку. Flutter ефективніше працює з анімаціями завдяки використанню графічного рушія Skia (див.рис.2.3).

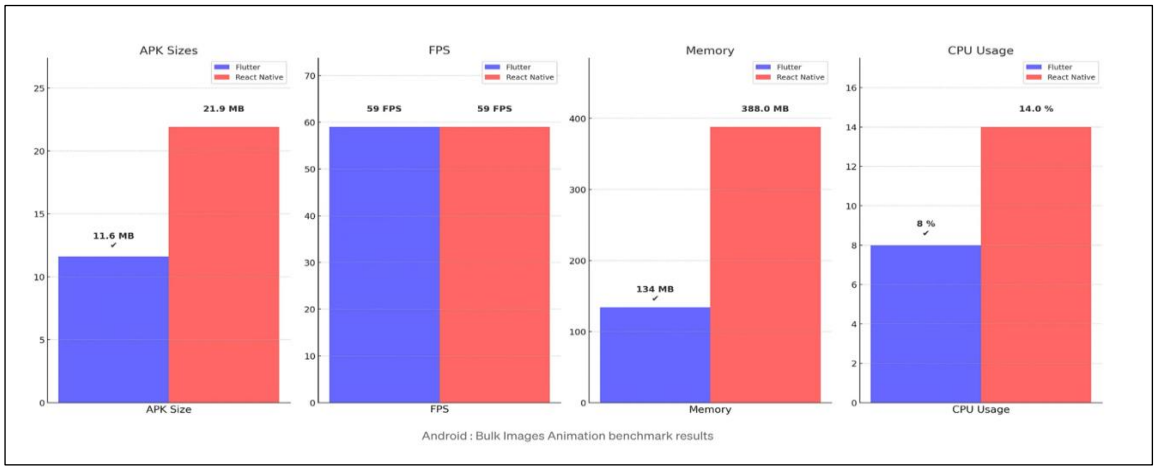

Рисунок 2.3 – Графік порівняння продуктивності (рисунок виконано самостійно)

Графік, наведений вище, ілюструє чотири основні метрики: розмір APK-файлу, кількість кадрів за секунду (FPS), використання оперативної пам’яті та завантаження процесора.

Результати демонструють переконливу перевагу Flutter у трьох із чотирьох параметрів. По-перше, розмір згенерованого APK-файлу для Flutter становить лише 11.6 МБ, тоді як для React Native – майже удвічі більше, а саме 21.9 МБ. Це є важливим чинником при розповсюдженні мобільних додатків, особливо в умовах обмеженого інтернет-з’єднання або браку вільного місця на пристрої.

По-друге, обидва фреймворки забезпечують однакову частоту оновлення зображення – 59 FPS, що свідчить про плавність анімацій незалежно від вибору технології. Проте за показником споживання оперативної пам’яті Flutter знову виявляється ефективнішим, 134 МБ проти 388 МБ у React Native. Це означає, що застосунок на Flutter краще масштабується та стабільніше працює на пристроях з обмеженими ресурсами.

Останній критерій – завантаження процесора – також демонструє перевагу Flutter: лише 8% у порівнянні з 14% у React Native. Нижче навантаження на CPU безпосередньо впливає на тривалість роботи батареї та загальну плавність системи.

Далі розглянемо сцеарій масової анімації зображень на пристроях Iphone 8 та Iphone 15 (див.рис.2.4)

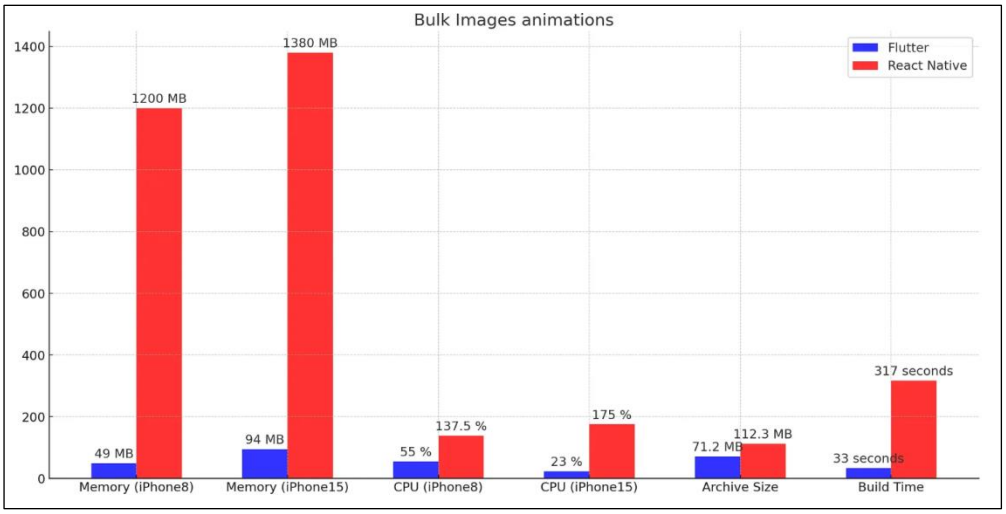

Рисунок 2.4 – Графік порівняння побудови анімацій (рисунок виконано самостійно)

Показники споживання пам’яті засвідчують значну перевагу Flutter. На iPhone 8 він споживає лише 49 МБ, тоді як React Native – 1200 МБ. Аналогічна картина спостерігається і на iPhone 15: Flutter – 94 МБ, React Native – 1380 МБ. Це свідчить про те, що Flutter зберігає стабільність і ефективність роботи незалежно від покоління пристрою, тоді як React Native вимагає значно більше ресурсів пам’яті.

Щодо завантаження процесора, то на iPhone 8 Flutter використовує 55%, тоді як React Native досягає 137.5%**,** що перевищує нормальний рівень і може призводити до перегріву пристрою чи просідання продуктивності. На iPhone 15 ця різниця ще виразніша: 23% у Flutter проти 175% у React Native. Такі цифри вказують на високий рівень оптимізації Flutter щодо використання обчислювальних ресурсів.

Ще одним важливим критерієм є розмір архіву, а саме 71.2 МБ у Flutter проти 112.3 МБ у React Native. Це має значення при розповсюдженні застосунку через App Store та впливає на швидкість продуктивності (див.рис.2.5).

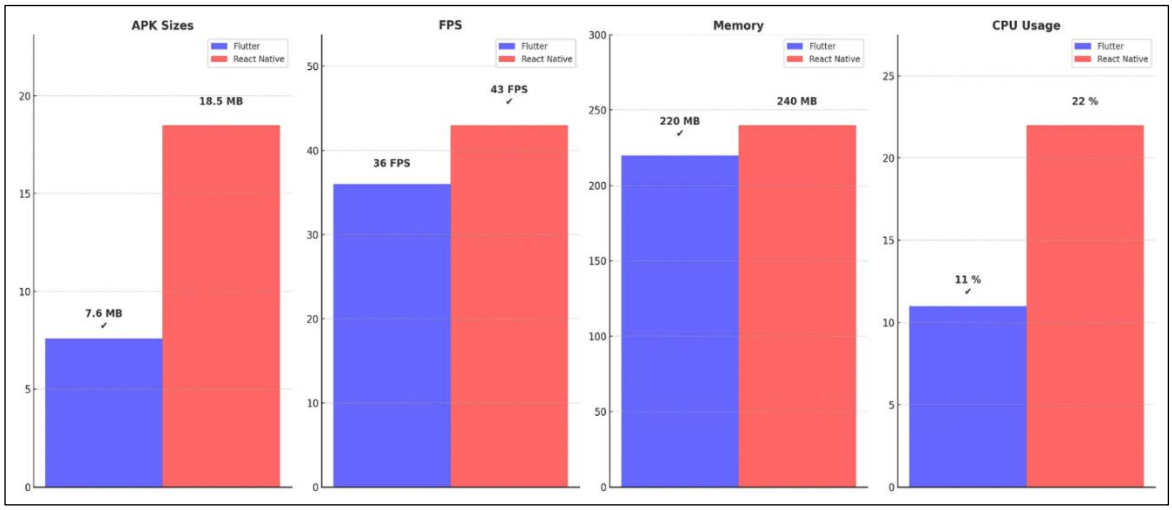

Рисунок 2.5 – Графік порівняння продуктивності (рисунок виконано самостійно)

У межах дослідження методів побудови анімацій у мобільних додатках важливо оцінити не лише візуальні можливості фреймворків, але й їхня ефективність з погляду використання ресурсів пристрою.

Результати порівняння зображеному на графіку демонструють збалансовану, але водночас контрастну картину. Зокрема, Flutter показує значно менший розмір згенерованого APK-файлу – лише 7.6 МБ, тоді як додаток на React Native займає 18.5 МБ. Така різниця є суттєвою при поширенні застосунків, особливо для користувачів з обмеженим доступом до високошвидкісного інтернету або на пристроях із браком вільної пам’яті. У контексті частоти оновлення зображення (FPS), перевагу отримує React Native, досягаючи показника 43 кадри на секунду, що перевищує результат Flutter, який становить 36 FPS.

Така відмінність може вказувати на дещо плавніший візуальний досвід у React Native при рендерінгу анімацій. Проте цю перевагу слід розглядати з огляду на інші показники системного навантаження. Наприклад, React Native потребує більше оперативної пам’яті 240 МБ, тоді як Flutter задовольняється 220 МБ. Попри відносно незначну різницю, вона підтверджує загальну тенденцію до економнішого споживання ресурсів у Flutter.

Ще більш яскраво це видно у завантаженні процесора: Flutter використовує лише 11% CPU, тоді як React Native – удвічі більше, 22%. Це не лише впливає на енергоспоживання, а й може визначати стабільність роботи застосунку на менш потужних пристроях.

Таким чином, хоча React Native демонструє вищу частоту кадрів у тестовому середовищі, Flutter загалом виглядає більш оптимізованим з точки зору ефективності: він споживає менше пам’яті, навантажує процесор меншою мірою й генерує набагато компактніший застосунок. Це робить Flutter технологією, яка краще відповідає вимогам сучасної мобільної розробки, де продуктивність і ощадливість мають критичне значення.

Розглянемо порівняння анімації Lottie (див.рис.2.6).

Анімації формату Lottie, завдяки своїй компактності та підтримці векторної графіки, широко застосовуються у мобільних застосунках для створення привабливого та легкого інтерфейсу. Проте ефективність їх реалізації суттєво залежить від фреймворку.

З метою оцінки продуктивності відтворення Lottie-анімацій було проведено експериментальне порівняння двох найпопулярніших кросплатформених інструментів – Flutter і React Native. Результати тестування подано на рисунку 2.6.

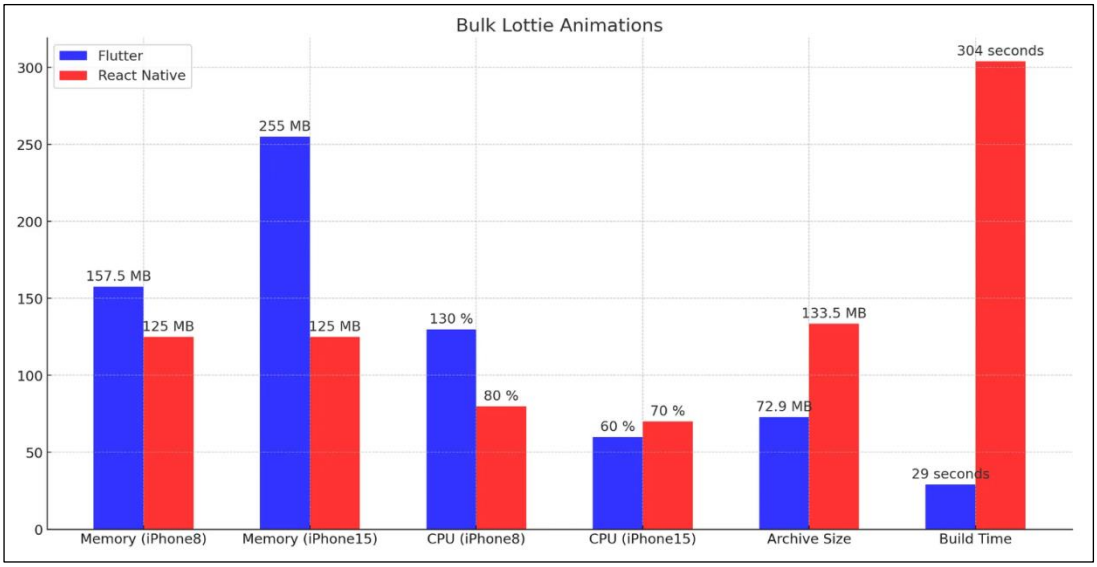

Рисунок 2.6 – Графік порівняння анімації Lottie (рисунок виконано самостійно)

Аналіз даних показує неоднозначну картину. Flutter виявився менш ощадливим у використанні оперативної пам’яті: на iPhone 8 він споживає 157.5 МБ проти 125 МБ у React Native, а на iPhone 15 ця різниця ще зростає – 255 МБ у Flutter проти стабільних 125 МБ у React Native.

Такий показник може свідчити про те, що Lottie-анімації у Flutter потребують додаткових обчислювальних структур або менш оптимізовані на даному етапі реалізації. Що стосується навантаження на процесор, то тут ситуація змінюється.

На iPhone 8 CPU-навантаження у Flutter становить 130%, тоді як у React Native – 80%. Проте на iPhone 15 ситуація вирівнюється: Flutter використовує 60%, а React Native – 70%. Це свідчить про те, що Flutter краще адаптується до нових пристроїв, тоді як React Native навпаки – демонструє вищу стабільність на старших моделях. Особливо варто звернути увагу на показники, які безпосередньо впливають на процес розробки та поширення застосунків.

Розмір архіву збірки у Flutter є 72.9 МБ, що майже вдвічі менше порівняно з 133.5 МБ у React Native. Але найсуттєвіша перевага Flutter полягає у часі збирання застосунку: лише 29 секунд проти надзвичайно тривалих 304 секунд у React Native.

Це не просто технічний нюанс – а важлива характеристика, яка безпосередньо впливає на ефективність команди розробки, швидкість ітерацій і цикл випуску оновлень.

Таким чином, хоча Flutter у даному тесті має вищі показники споживання пам’яті, він компенсує це кращою продуктивністю на нових пристроях, компактнішою збіркою та суттєво швидшим часом побудови. React Native, навпаки, демонструє стабільність на рівні ресурсозбереження, але потребує значно більше часу для компіляції та створює більший застосунок.

Цей аналіз дозволяє зробити висновок, що вибір фреймворку для Lottie-анімацій залежить від пріоритетів проєкту: оптимізація під нові пристрої та швидкість розробки – за Flutter, стабільне ресурсозбереження – за React Native.

Далі розглянемо вимірювання швидкості компіляції застосунку (див.рис.2.7).

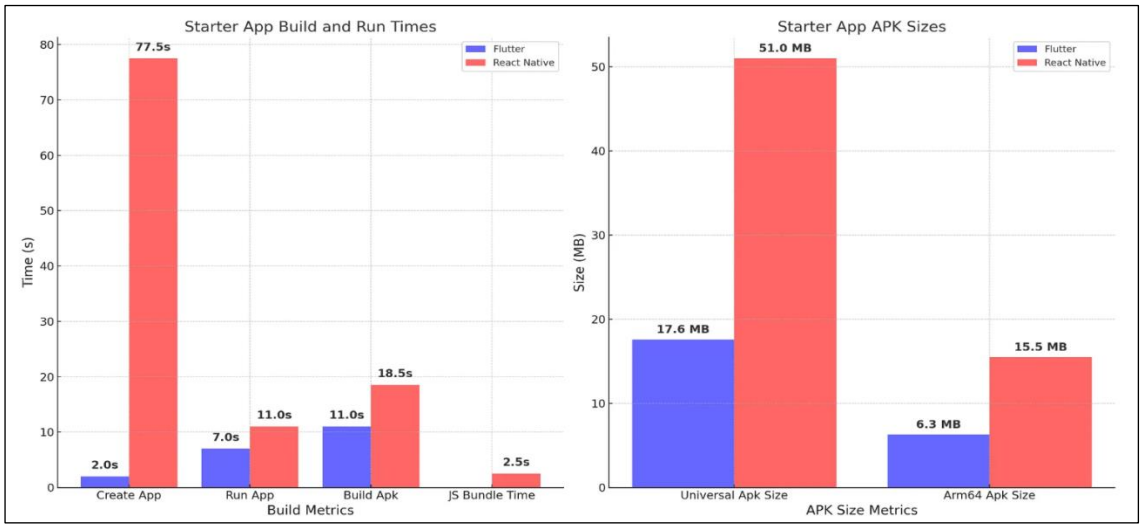

Рисунок 2.7 – Графік вимірювання швидкості компіляції застосунку (рисунок виконано самостійно)

Результати за даним графіком є однозначно на користь Flutter. Час створення нового застосунку у Flutter становив лише 2 секунди, тоді як у React Native – 77.5 секунд, що є вражаючим контрастом і демонструє помітну перевагу Flutter у швидкості ініціалізації проєкту.

Запуск застосунку також проходить швидше у Flutter – 7 секунд проти 11 секунд у React Native. Хоча різниця менш драматична, вона залишається важливою в умовах частого повторного запуску під час налагодження.

Цікаво, що час складання APK-файлу практично однаковий у обох фреймворках – 11 секунд у Flutter та 18.5 секунд у React Native. Проте в React Native додатково потребується ще 2.5 секунди для створення JS-бандла, що збільшує загальний час збірки і відображає складнішу архітектуру системи.

Праве поле графіка показує розміри зібраних APK-файлів. Flutter знову демонструє вищу ефективність: універсальний APK має розмір 17.6 МБ, тоді як у React Native – 51.0 МБ, що майже втричі більше. Навіть у оптимізованій версії під архітектуру ARM64 Flutter створює APK розміром 6.3 МБ, у той час як React Native – 15.5 МБ. Ці дані свідчать про те, що Flutter не лише швидше створює та запускає

початковий застосунок, але й формує значно компактніший пакет для розповсюдження.

Отже, як показують графіки в публікації, продуктивність Flutter також лідирує у використанні процесора, ефективності пам'яті та затримках при обробці взаємодій. Це свідчить про кращу оптимізацію графічного движка Skia, який використовується у Flutter для малювання об'єктів за межами рідних компонентів ОС.

На основі цих показників, а також офіційної підтримки розширеного набору інструментів профілювання, таких як Flutter DevTools, Performance Overlay, Frame Rendering Inspector та Timeline, ми вирішили обрати Flutter SDK як основне середовище розробки для реалізації дослідницького інтерфейсу.

Можливість швидкого прототипування та повторного використання шаблонів анімації була продумана до дрібниць.

У цьому випадку Flutter дозволяє використовувати як неявну анімацію (наприклад, AnimatedContainer і AnimatedOpacity), так і явні контролери анімації (наприклад, AnimationController, Tween і CurvedAnimation) [4].

Це дозволяє створювати адаптивні сценарії того, як інтерфейс реагує на дії користувача, враховуючи продуктивність і навантаження на потік інтерфейсу.

Отже, середовище Flutter мало потрібний рівень функціональності для виконання дослідницьких завдань.

Воно дозволило змішувати і поєднувати техніки анімації, бачити, як вони впливають на продуктивність, і перевіряти, чи відповідають вони стандартам доступності.

Це середовище також дозволяє проводити більше досліджень щодо створення інтерфейсів, які легко використовувати людям з особливими потребами.

2.2 Вибір критеріїв з оцінювання ефективності методів анімації

Щоб дійсно зрозуміти, які анімації у Flutter працюють найкраще, потрібно чітко визначити, за якими критеріями їх оцінювати.

Не достатньо просто подивитись, чи “гарно” виглядає – важливо враховувати технічну сторону, зручність у реалізації, вплив на продуктивність і загальний користувацький досвід [5].

Перший показник, який беремо до уваги – коефіцієнт плавності анімації. По суті, він показує, скільки кадрів встигли відрендеритись вчасно.

Якщо цей коефіцієнт високий – анімація виглядає гладкою, без смикань чи підвисань. Формула для розрахунку виглядає наступним чином:

$$F = \left(\frac{f_s}{f_t}\right) * 100\% \tag{2.1}$$

де: f_s – кількість кадрів, які з’явились вчасно,

f_t – загальна кількість кадрів, які мала показати анімація.

Наступний важливий параметр – коефіцієнт затримки анімації.

Цей показник відображає, скільки кадрів “запізнилися” – тобто були відображені з затримкою, що може вплинути на плавність взаємодії. Розраховується за формулою:

$$L = \left(\frac{f_d}{f_t}\right) * 100\% \tag{2.2}$$

де: f_d – кількість кадрів із затримкою ,

f_t – загальна кількість кадрів.

Також не варто забувати про час відгуку анімації – це час, який проходить від моменту дії користувача (наприклад, натискання кнопки) до фактичного старту анімації на екрані. Занадто довгий відгук створює відчуття “гальмування” застосунку [6].

Окремо варто звернути увагу на використання ресурсів пристрою.

Деякі анімації можуть бути візуально красивими, але надто “важкими” – споживають багато CPU або оперативної пам’яті, що в підсумку погано впливає на загальну швидкість застосунку. Особливо це критично для слабших смартфонів [7].

Ще один критерій, який часто недооцінюють – гнучкість та повторне використання анімації.

Якщо певну анімацію легко застосувати в різних частинах застосунку без переписування коду – це величезний плюс при розробці й підтримці проєкту.

Усі ці критерії дозволяють побачити повну картину – не лише “на око”, а й через об’єктивні вимірювання.

Щоб дослідження було ще більш ґрунтовним, варто проводити й стрес-тести інтерфейсу: перевірити, як анімації поводяться при великій кількості одночасних натискань, переходів чи зміни стану екрана.

2.3 Аналіз та моделювання предметної області для дослідження

У рамках даного дослідження було обрано не традиційну прикладну сферу на кшталт фінансів чи електронної комерції, а більш креативну та візуально виразну – мобільний застосунок, який представляє собою галерею фантазійних анімацій.

Такий вибір зумовлений кількома факторами.

По-перше, це дає змогу протестувати широкий спектр анімаційних технік Flutter у творчому, необмеженому середовищі.

По-друге, подібний застосунок орієнтований передусім на візуальне враження та користувацький досвід, отже роль анімацій тут є ключовою, а не допоміжною.

Обрана предметна галузь умовно імітує інтерактивну казкову мапу або чарівну експозицію, в якій користувач подорожує між локаціями, взаємодіє з магічними об’єктами та персонажами, відкриває нові сцени, а кожна така дія супроводжується анімаційною відповіддю.

В результаті, кожен екран або компонент застосунку стає полігоном для експериментів з анімаціями – від найпростіших (Fade, Scale, Slide) до складних кастомних послідовностей з AnimationController, TweenSequence, а також з використанням сторонніх бібліотек, таких як rive, Lottie чи flutter_animate [8].

На відміну від суворо функціональних застосунків, де анімація зазвичай виконує допоміжну роль (наприклад, вказує на зміну стану або покращує читабельність), у цьому випадку вона є центральним елементом досвіду.

Це дозволяє дослідити анімації не просто як "ефекти", а як інструмент взаємодії, навігації та оповіді.

До переваг такої предметної галузі варто віднести:

- гнучкість у сценаріях – немає обмежень з боку бізнес-логіки; можна створювати фантазійні переходи, незвичні візуальні ефекти, комбінувати трансформації, змінювати стан об'єкта без прив'язки до типових UX-патернів;
- інтенсивність взаємодії – кожен клік, свайп або дотик запускає нову анімацію, тому система постійно перебуває в русі;
- тестування навантаження – наявність постійних фонових анімацій (наприклад, "магічний пил", зоряне небо, коливання води) дає змогу вимірювати продуктивність, FPS, ресурсоспоживання при активному UI.

Основні екрани та взаємодії, що досліджуються:

- головна мапа – анімована сцена з переходами до інших розділів. Застосовується масштабування, рух камери, поява світлових ефектів при натисканні на точки інтересу;
- галерея істот – екран, на якому можна переглядати інтерактивних персонажів. Наприклад, при натисканні на лісовика він махає рукою, блимає очима або зникає у листі. Тут ідеально підходять керовані анімації з AnimationController, а також цикловані анімації;
- магічні об'єкти – інтерактивні елементи, які реагують на дотик: кристали, котли, магічні книги. Вони можуть змінювати колір, трансформуватись або випускати "енергію". Ці сцени дозволяють протестувати складні анімаційні послідовності та затримки (DelayedAnimations, Chained Tweens) [9];

- плавна навігація між світами – кожен перехід між екранами оформлений не як звичайна зміна маршруту, а як мандрівка: камера "злітає" над мапою, іскри змивають попередній екран і розкривають новий. Тут доречно досліджувати кастомні PageRoute та Hero-анімації;
- меню та діалоги – навіть стандартні елементи інтерфейсу (меню налаштувань, інформаційні підказки) відкриваються з використанням легких анімацій, що створюють ефект чарівності. Наприклад, діалог «випливає» з центру екрану або складається з частинок світла.

Незважаючи на казкову тематику, застосунок має важливу прикладну цінність у контексті дослідження. Він дозволяє не лише протестувати технічну реалізацію анімацій у Flutter, а і:

- порівняти підходи: декларативні vs. імперативні;
- оцінити продуктивність при високому навантаженні (фон + взаємодія + переходи);
- оцінити гнучкість повторного використання компонентів – наскільки легко винести анімовані віджети у спільні елементи;
- продемонструвати креативні можливості Flutter – як інструменту не лише для бізнес-додатків, а й для ігрових або експериментальних інтерфейсів.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Вибір технологій та середовища розробки

При створенні програмної частини дослідження було зроблено вибір на користь роботи з платформою Flutter від Google.

Найважливішою причиною цього є те, що Flutter може надавати нативний інтерфейс для різних платформ (Android, iOS, Web), використовуючи єдиний код, що насправді значно економить час розробки та робить проекти більш зручними в обслуговуванні. Flutter використовує мову Dart, яка була створена компанією Google.

Dart має підтримку пристойної продуктивності, підтримку асинхронного програмування і добре підходить для використання як імперативних, так і декларативних конструкцій, що відіграє важливу роль у випадку анімації.

В якості інтегрованого середовища розробки (IDE) було використано Android Studio, яке повністю підтримує Flutter і Dart, включаючи автоматичне завершення, налагодження, профілювання продуктивності та інтеграцію з Flutter DevTools.

Це дозволило легко керувати структурою проекту, покроково налагоджувати логіку анімації та тестувати їх вплив на продуктивність у режимі профілювання. DevTools від Flutter використовувався для аналізу ефективності анімації, а також для зручного моніторингу використання ресурсів для оцінки частоти кадрів (FPS), часу побудови кадрів, затримок рендерингу та обсягу використаної пам'яті [10].

3.2 Архітектура застосунку

Для забезпечення масштабованості, розширюваності та супроводжуваної було використано функціонально-орієнтований архітектурний стиль з підтримкою патернів повторного використання.

Цей підхід передбачає сегментацію програмного коду на цільові функціональні секції (наприклад, головна карта, галерея істот, магічні об'єкти тощо), що дозволяє легко локалізувати логіку, стилі та анімацію в межах однієї конструкції.

Додаток має модульну структуру (див. рис. 3.1) з наступними основними розділами:

– screens/ - екрани додатку, що відповідають за відображення відповідних сцен (мапа, галерея, об'єкти тощо);
– components/ - багаторазові компоненти інтерфейсу (наприклад, анімовані кнопки, кастомні діалоги, персонажі);
– animations/ - незалежний модуль для зберігання неявних та явних анімацій, що використовуються в додатку;
– services/ - доступ до зовнішніх сервісів (наприклад, Firebase, поштовий клієнт, генерація PDF);
– models/ - опис структури даних (наприклад, персонажів, об'єктів, точок мапи);
– providers/ - класи керування станом за допомогою Provider.

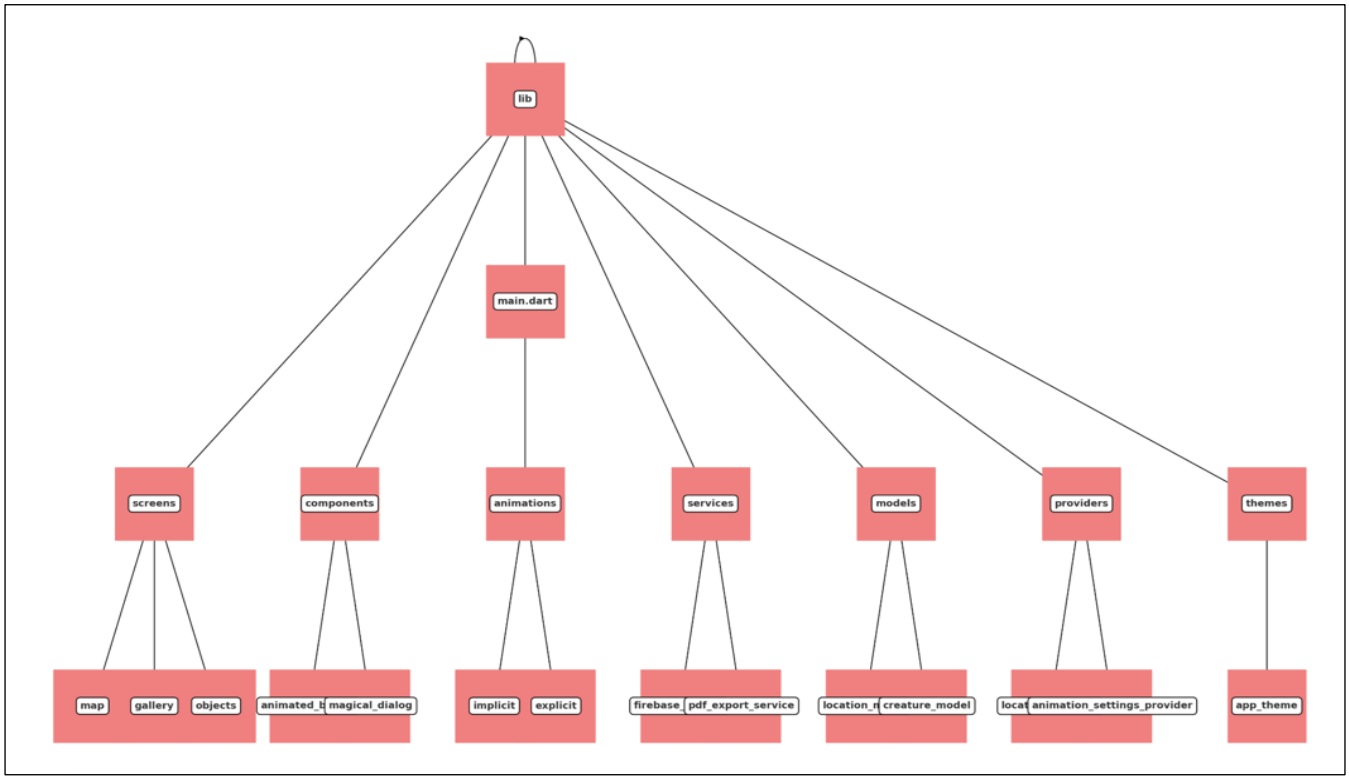

Рисунок 3.1 – Модульна структура (рисунок виконано самостійно)

Для управління станом додатку було обрано найпростіший спосіб, вбудований в сам Flutter - чисте управління станом за допомогою функції setState [11].

Цей підхід дозволяє змінювати стан віджетів безпосередньо під час їхнього життєвого циклу та ідеально підходить для додатків з переважно локальною логікою.

Метод setState використовується в StatefulWidget, де інтерфейс перебудовується через зміну даних, що стосуються екрану або іншого окремого компонента [12].

Метод має простоту реалізації та зрозумілість поведінки при розробці невеликих або замкнутих фрагментів інтерфейсу, таких як:

- реакція на жести (свайпи, тапи);
- зміна режиму анімації;
- управління анімацією в межах одного екрану.
- Цей метод активно використовувався в цьому проекті:
- для управління анімацією з AnimationController, де потрібно викликати setState при зміні ходу анімації;
- для обробки локальних подій взаємодії з віджетами (наприклад, при кліці на об'єкті або персонажі);
- у демонстраційних сценаріях, щоб побачити продуктивність декларативної (TweenAnimationBuilder) та імперативної (setState+AnimationController) анімації.

Хоча setState не найкраще підходить для складних або глобальних сценаріїв управління станами, коли мова йде про дослідження анімації, де основна увага приділяється автономній поведінці окремих компонентів, цей підхід дозволяє максимально точно керувати оновленнями інтерфейсу і динамічно контролювати хід анімації.

3.3 Реалізація основних екранів

3.3.1 Головна мапа

Головна карта є навігаційним центром. Візуально це магічна карта з точками інтересу, на які можна натиснути. Компоненти послідовно з'являються під час завантаження екрана за допомогою прозорості та анімації слайдів (Fade і Slide) через TweenAnimationBuilder.

Натискання на одну з точок переводить до відповідного пункту призначення через анімований маршрут PageRouteBuilder у поєднанні з Hero для безперервного потоку елементів по екранах (див.рис.3.2).

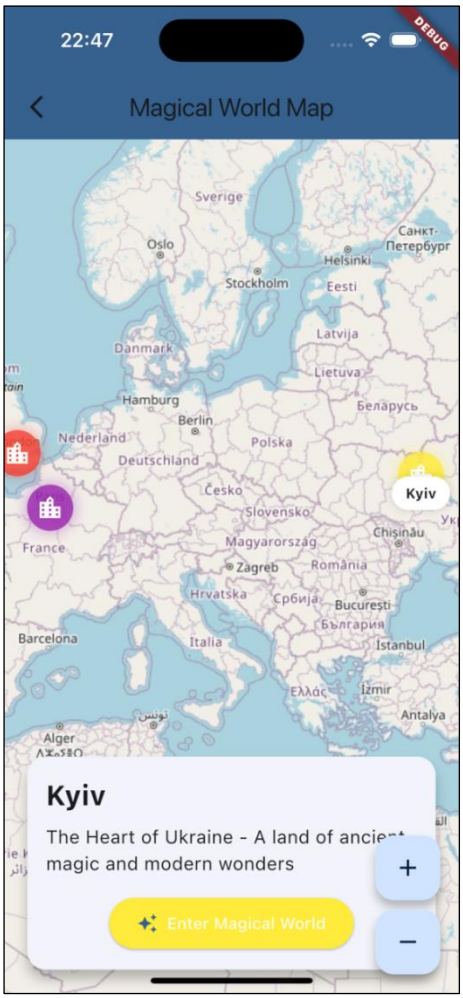

Рисунок 3.2 – Інтерфейс компоненту InteractiveViewer (рисунок виконано самостійно)

Карта підтримує взаємодію користувача за допомогою панорамування та масштабування, що стає можливим завдяки компоненту InteractiveViewer.

3.3.2 Галерея істот

Екран галереї істот служить тестовим майданчиком для контрольованих анімацій. Усі персонажі мають власний профіль анімації: деякі анімації є циклічними і запускаються автоматично, а інші запускаються після взаємодії користувача. У цьому випадку використовується AnimationController у поєднанні з Tween [13]. Натискання на істоту викликає перехід її стану – рух, миготіння, зникнення або появу інших візуальних ефектів (див.рис.3.3).

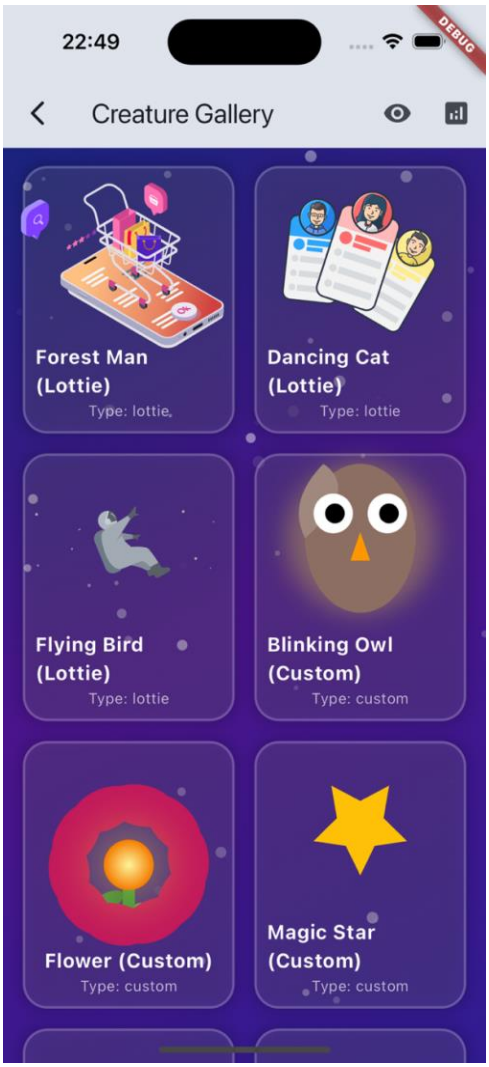

Рисунок 3.3 – Інтерфейс екрану галереї істот (рисунок виконано самостійно)

Анімації реалізовані за допомогою AnimatedBuilder або за допомогою пакета flutter_animate, який забезпечує лаконічну синтаксис і можливість комбінувати ефекти.

3.3.3 Складні анімації

Ще одним самостійним розділом реалізації є складні анімаційні переходи, які не є специфічними для екрану, а демонструють можливості налаштування анімацій у Flutter на вищому рівні. Зокрема, створено кілька типів переходів, які візуально виглядають як мерехтіння, іскристість або пікселізація. Було реалізовано ці ефекти за допомогою власних анімованих маршрутів через CustomPageRoute, ShaderMask та інші графічні утиліти. У деяких випадках також було використано функції Hero вищого рівня, такі як flightShuttleBuilder, де можна налаштувати поведінку в середині переходу (див.рис.3.4).

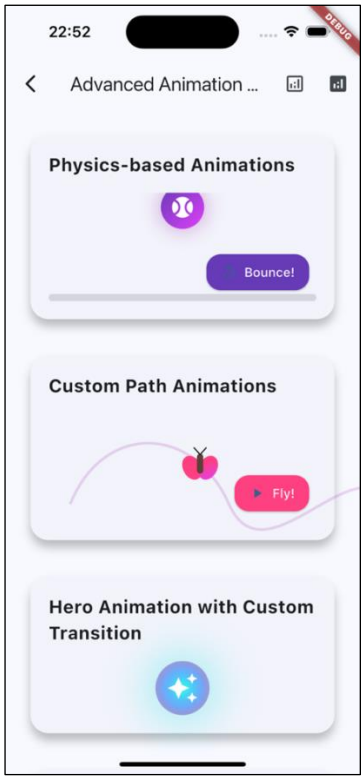

Рисунок 3.4 – Інтерфейс складних анімаційних переходів (рисунок виконано самостійно)

Особливу увагу було приділено забезпеченню продуктивності: анімації були пов'язані з TickerProvider, а для моніторингу продуктивності використовувався Flutter DevTools.

3.3.4 Дешборд анімацій

Було реалізовано спеціальний екран для технічного порівняння різних видів анімацій та моніторингу їхньої продуктивності. У межах цієї панелі анімацій передбачено можливість перемикання між сценаріями за допомогою різних підходів, зокрема setState, TweenAnimationBuilder та AnimationController. Це дозволяє дослідити як імпліцитні, так і експліцитні анімації в контексті практичного використання, а також спостерігати за їхнім впливом на продуктивність інтерфейсу користувача [14].

Для кожного типу анімації реалізовано візуальне представлення, що дозволяє не лише побачити зміну станів у динаміці, але й оцінити супутні метрики продуктивності. Основними параметрами для вимірювання виступили кількість кадрів за секунду (FPS), середній час побудови одного кадру (frame build time), а також завантаження ресурсів системи, включаючи пам’ять та CPU.

У процесі тестування було увімкнено профілювання, що дозволило отримати точні дані щодо рендерингу, роботи із GPU, затримок при побудові кадрів, часу виконання фреймів та викликів garbage collector. Ці метрики були записані в реальному часі під час запуску кожного анімаційного сценарію.

Зібрані значення були оброблені та представлені у графічному вигляді за допомогою бібліотеки fl_chart, яка була інтегрована у застосунок. Це забезпечило можливість наочної побудови діаграм і графіків, що відображають поведінку системи під час різних типів анімації. Зокрема, побудовано графіки зміни FPS, використання оперативної пам’яті та завантаження процесора протягом часу, що дало змогу точно порівняти імпліцитні та експліцитні підходи за ключовими критеріями продуктивності (див.рис.3.5).

Рисунок 3.5 – Інтерфейс дешборду анімацій (рисунок виконано самостійно)

Для вимірювання використовувалися Flutter DevTools та специфічні для платформи інструменти в SchedulerBinding. Такий підхід дозволяє не тільки порівняти явні та неявні анімації на практиці, але й отримати об'єктивні кількісні дані, які були використані в аналітичній частині дослідження.

4 ОПИС ЕКСПЕРЕМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

4.1 Проведення експериментальних досліджень

Мета цього дослідження є краще розуміння, як працюють різні методи анімацій у мобільних застосунках, створених за допомогою фреймворку Flutter.

Анімації – це не просто візуальні ефекти, а важлива частина взаємодії користувача із застосунком, тому важливо знати, які підходи до їх реалізації є найбільш ефективними, зручними й простими в обслуговуванні.

Для початку було вирішено створити окремі тестові Flutter-проєкти, в яких реалізуються різні типи анімацій.

Наприклад, імперативний підхід з використанням AnimationController, декларативний – через AnimatedWidget, AnimatedBuilder та інші вбудовані віджети, а також анімації за допомогою сторонніх бібліотек на кшталт flutter_animate чи rive.

Це дозволяє побачити сильні та слабкі сторони кожного з підходів на практиці.

Щоб оцінка була об'єктивною, усі анімації в цих проєктах реалізовані у схожому контексті – з однаковими сценаріями використання, стилістикою та навігаційною логікою.

Це дало змогу порівняти їх з погляду продуктивності (наприклад, FPS), навантаження на систему та складності реалізації.

Окремим етапом було визначення предметної області, в межах якої анімації справді мають значення.

Завдяки цьому вдалося оцінити, як різні способи анімацій поводяться в умовах, максимально наближених до реального використання.

4.2 Порівняння імпліцитних та експліцитних анімацій

Одним із напрямків дипломної роботи було дослідження технік анімації в Flutter, новому фреймворку для створення міжплатформних мобільних інтерфейсів користувача.

У кваліфікаційній роботи особливий акцент було зроблено на порівнянні двох основних механізмів, які Flutter підтримує для створення анімованих компонентів: неявного та явного.

Неявні анімації – це декларативний спосіб накладання анімованих змін.

Розробник визначає, яким має бути результат, а Flutter самостійно визначає проміжні стани та контролює параметри часу анімації. AnimatedContainer, AnimatedOpacity, AnimatedAlign, TweenAnimationBuilder тощо – це все неявні анімації [15].

Їх головна перевага – простота використання: достатньо змінити значення деякої властивості, і фреймворк автоматично плавно перейде до нового значення.

Явні анімації, навпаки, вимагають явного контролю.

Вони базуються на AnimationController, який вимагає налаштування тривалості, ініціалізації через TickerProvider, запуску, повторення, зупинки тощо.

Водночас розробник має повний контроль над анімацією, наприклад, може змінювати криві інтерполяції (Curves), слухати події, комбінувати кілька Tweens у TweenSequence, синхронізуватися з іншими процесами – і це відкриває цілий світ можливостей для побудови складної поведінки.

У практичній частині роботи було створено екран «Інтерактивний світ» як демонстраційне середовище для порівняння обох підходів в одному візуальному контексті.

Основними елементами екрану є: фон (небо), анімовані хвилі води, об'єкти сцени (птахи, дерева) та елементи керування (кнопка перемикання режиму). Всі ці елементи малюються вручну за допомогою CustomPainter, що дозволяє точно контролювати візуальні параметри.

Фон сцени, який чергується між світлим (день) і темним (ніч), досягається за допомогою TweenAnimationBuilder, тобто неявної анімації.

Розробник вказує початкові та кінцеві значення кольорів (ColorTween) і визначає тривалість анімації.

Все інше (інтерполяція, оновлення стану, плавність) відбувається самостійно. Результатом є плавні та приємні переходи освітлення з мінімальним кодом.

Хвилі на воді були реалізовані явно – за допомогою AnimationController, який повторюється нескінченно (repeat()).

Значення controller.value передається в режимі реального часу до CustomPainter, де використовується для обчислення зсуву фази в синусоїдальній функції.

Це дозволяє малювати хвилі, які постійно змінюють форму, імітуючи реальний рух води.

Явний підхід дозволяє точно контролювати частоту коливань, амплітуду та інші характеристики.

Код, що ілюструє поєднання обох підходів:

```
class AnimatedWorld extends StatefulWidget {
  @override
  State<AnimatedWorld> createState() => _AnimatedWorldState();
}

class _AnimatedWorldState extends State<AnimatedWorld> with
SingleTickerProviderStateMixin {
  late final AnimationController _controller;
  bool isNight = false;

  @override
  void initState() {
    super.initState();
    _controller = AnimationController(duration: const Duration(seconds: 3),
vsync: this)..repeat();
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Stack(
        children: [
          TweenAnimationBuilder<Color?>(
            duration: const Duration(milliseconds: 800),
            tween: ColorTween(
              begin: isNight ? Colors.indigo[900] : Colors.lightBlue,
              end: isNight ? Colors.lightBlue : Colors.indigo[900],
            ),
            builder: (_, color, __) => CustomPaint(
```

```
              painter:      WorldPainter(color:      color!,      waveOffset:
_controller.value),
              child: const SizedBox.expand(),
            ),
          ),
          Positioned(
            bottom: 50,
            left: 30,
            child: ElevatedButton(
              onPressed: () => setState(() => isNight = !isNight),
              child: Text(isNight ? 'Day' : 'Night'),
            ),
          )
        ],
      ),
    );
  }
}
```

Параметр isNight визначає, який колір неба буде застосовано. TweenAnimationBuilder забезпечує плавну анімацію фону.

AnimationController впливає на рух хвиль.

CustomPainter (WorldPainter) малює як фон, так і хвилі відповідно до встановлених параметрів.

Для дослідження впливу різних типів анімацій на продуктивність додатка було використано Flutter DevTools у профільному режимі (flutter run --profile).

Під час тестування відстежувалися найважливіші показники: частота кадрів (FPS), час побудови кадру та використання процесора.

Порівняння показало, що при використанні лише неявних анімацій система постійно працює з частотою 60 FPS, а середнє використання процесора становить близько 22–24%.

Однак, при запуску явних анімацій (хвилі + птахи) частота кадрів іноді падає до 56–59 FPS, а використання процесора зростає до 30–35%.

Інформація була представлена у вигляді графіку FPS – для відображення загальної стабільності рендерингу (див.рис.4.1).

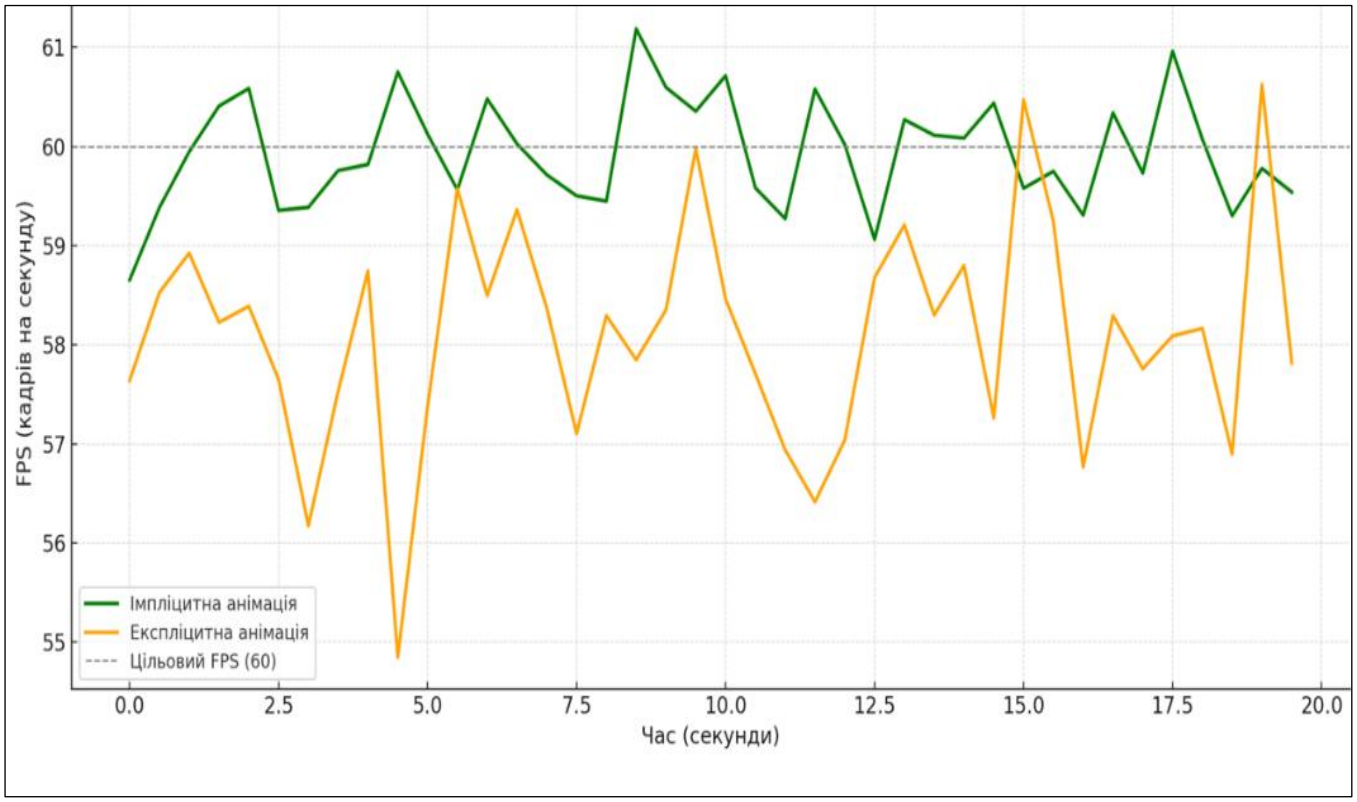

Рисунок 4.1 – Графік FPS (рисунок виконано самостійно)

Аналіз показує, що неявні анімації є зручним механізмом для додавання плавних, простих перетворень, ідеальних для фонових або стилістичних переходів.

Явні анімації, з іншого боку, дають розробнику набагато більше свободи при створенні складних, динамічних ефектів, дозволяючи досягти вражаючої візуальної глибини та налаштувати логіку.

Гібридне рішення, що інтегрує обидва методи на одному екрані, виявилося ефективним як з технічної точки зору, так і з точки зору естетики взаємодії.

У контексті теми магістерської роботи цей досвід вказує на потенціал Flutter не тільки як засобу розробки бізнес-додатків, але й як платформи для створення експериментальних, креативних інтерфейсів.

Таким чином, така комбінація дозволяє досягти балансу стабільності та гнучкості, заощадити ресурси на другорядних елементах і сконцентрувати обчислення на критично важливих елементах сцени.

Це особливо корисно при розробці додатків з високими візуальними вимогами, зокрема для ігор або симуляторів.

Крім того, експериментальний аналіз показав, що плавність і частота кадрів (FPS) строго залежать від обраної стратегії.

Неявна анімація, завдяки внутрішній оптимізації Flutter, забезпечує практично лінійну продуктивність незалежно від складності сцени, в той час як явна анімація вимагає правильного планування, особливо якщо є багато одночасно активних контролерів або складних малюнків.

На завершення слід підкреслити, що застосування анімації у Flutter виходить далеко за рамки візуалізації стану.

Вони стають життєздатним інструментом для планування сценаріїв, навігації, реактивності та занурення, і з цієї причини їх вивчення в контексті магістерської роботи не тільки застосовується, але і є стратегічно вигідним, коли мова йде про майбутній розвиток в практиці UI/UX.

ВИСНОВКИ

У результаті аналізу «Дослідження implicit та explicit анімації за допомогою фреймворку Flutter» можна зробити кілька основних висновків. Перш за все можна чітко зазначити, що анімація є важливим інструментом для покращення взаємодії з користувачем у мобільних додатках. Використання анімації дозволяє створювати плавний перехід між елементами інтерфейсу, покращувати концепцію програми та допомагати користувачам орієнтуватися в її функціональності. Зокрема, фреймворк Flutter надає можливість багатьом використовувати анімаційні ефекти, значно спрощуючи дизайн інтерактивних інтерфейсів.

Більш того, дослідження показали, що існує два основних способи використання анімації у Flutter: неявний і явний. Вбудована анімація є автоматичною та зручною для простого тексту, коли потрібно анімувати стандартні властивості віджетів, наприклад змінити розмір, колір або прозорість. Вони пропонують швидкий і простий спосіб створення графіки без особливого контролю над процесом, що робить їх ідеальними для простих ефектів інтерфейсу користувача, таких як кнопки або переходи між екранами.

Навпаки, явна анімація забезпечує більший контроль над кожним кроком графіки, що робить її кращою для складніших налаштованих середовищ. Вони дозволяють створювати точні анімаційні ефекти, які вимагають не лише зміни властивостей, але й складної логіки, як-от керування декількома анімаціями одночасно, часом виконання, початковим і кінцевим станами, взаємодіями. Це може включати анімацію для складних компонентів інтерфейсу користувача або інтенсивно реалізованих динамічних інтерфейсів, де кожен крок анімації має бути чітко визначений.

Однак вибір між implicit та explicit анімацією залежить від специфіки проекту та функціональних вимог програми. Вбудовані анімації зазвичай мають мінімальний вплив на продуктивність і ідеально підходять для простих ситуацій, коли анімації

переходу властивостей потрібно реалізувати без потреби в значних обчислювальних ресурсах. У складних анімаціях, наприклад для змін взаємодії або складних графічних елементів, явні анімації дозволяють краще контролювати та результати. Саме вони мають більше комп’ютерних ресурсів, і може знадобитися більш складний код.

Важливою частиною й те, що було зазначено це те, що Flutter надає потужні інструменти, щоб отримати максимальну віддачу від обох типів анімації за допомогою спеціалізованих бібліотек і API. Ці інструменти забезпечують гнучкість і налаштовують анімацію, дозволяючи вибрати найбільш підходящий варіант на основі завдання та вимог проєкту.

Таким чином, ми можемо зробити висновок, що найкращий вибір типу анімації значною мірою залежить від взаємодії, складності інтерфейсу користувача та вимог до продуктивності програми. Обидва види анімацій в фрейморці Flutter надають достатньо інструментів для оптимізації мобільного застосунку, а для розробників здатні розробляти високоякісні та продуктивні мобільні програми.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Cheng F. Flutter Recipes: Mobile Development Solutions for iOS and Android - Apress, 2019 – 555 P.

2. Tyagi P. Pragmatic Flutter: Building Cross-Platform Mobile Apps for Android, iOS, Web & Desktop – CRC Press, 2021 – 354 P.

3. Flutter vs React Native: Performance Benchmarks [Електронний ресурс] – Режим доступу до ресурсу: https://nateshmbhat.medium.com/flutter-vs-react-native-performance-benchmarks-you-cant-miss-%EF%B8%8F-2e31905df9b4 (дата звернення 12.02.2025).

4. Exploring Explicit Animations in Flutter [Електронний ресурс] – Режим доступу до ресурсу: https://www.blup.in/blog/flutter-animations-exploring-explicit-animations-in-flutter (дата звернення 02.03.2025).

5. Коба Ю.Ю., Афанасьєва І.В. , Онищенко К.Г. Аналіз та використання способів керування станом застосунку Flutter. Збірник наукових праць Харківського національного університету Повітряних сил. 2023. №2 (76). С. 75-81.

6. Flutter Documentation: Introduction to Animations [Електронний ресурс] – Режим доступу до ресурсу: https://docs.flutter.dev/ui/animations (дата звернення 16.03.2025).

7. Flutter Animation Fundamentals: Enhancing User Experience [Електронний ресурс] – Режим доступу до ресурсу: https://www.blup.in/blog/flutter-animation-fundamentals-enhancing-user-experience (дата звернення 24.03.2025).

8. Flutter Animation: What is the Difference Between Explicit and Implicit Animation? [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@saminchandeepa/flutter-animation-what-are-explicit-and-implicit-animation-31868f602fc6 (дата звернення 30.03.2025).

9. Kosynskii O. Android application for music recommendation based on group interests. Сателітна конференція «Інформаційні технології та застосування (IT&Is). 2023. С. 23-24.

10. How and When to Create Custom Implicit Animations [Електронний ресурс] – Режим доступу до ресурсу: https://invertase.io/blog/how-and-when-to-create-custom-implicit-animations (дата звернення 10.04.2025).

11. Flutter Codelabs: Advanced Animations [Електронний ресурс] – Режим доступу до ресурсу: https://codelabs.developers.google.com/advanced-flutter-animations https://www.instagram.com/p/DKIxIM6zMLr/?igsh=MWdmanVxM2J1ZXJoZQ== (дата звернення 22.04.2025).

12. Корецький О.А. Використання реактивного програмування у розробці мобільних застосунків. SCIENCE, RESEARCH, DEVELOPMENT. Technics and technology. #4 Barcelona. С. 79-83.

13. Alessandria S., Kayfitz B. Flutter Cookbook: Over 100 Proven Techniques and Solutions for App Development with Flutter 2.2 and Dart – Packt, 2021 – 646 P.

14. Animations in Flutter: Implicit and Explicit Animations [Електронний ресурс] – Режим доступу до ресурсу: https://www.newline.co/@isaaclyman/animations-in-flutter-implicit-and-explicit-animations--68e70be5 (дата звернення 03.05.2025).

15. The Ultimate Guide to Mobile App Animations [Електронний ресурс] – Режим доступу до ресурсу: https://www.justinmind.com/ui-design/mobile-app-animations (дата звернення 18.05.2025).

16. https://github.com/sofiabukovskaya/Diploma-master-IPZzm-23-1-Bykovska-Sofiia

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

5. Коба Ю.Ю. , Афанасьєва І.В. , Онищенко К.Г. Аналіз та використання способів керування станом застосунку Flutter. Збірник наукових праць Харківського національного університету Повітряних сил. 2023. №2 (76). С. 75-81.

9. Kosynskii O. Android application for music recommendation based on group interests. Сателітна конференція «Інформаційні технології та застосування (IT&Is). 2023. С. 23–24.

12. Корецький О.А. Використання реактивного програмування у розробці мобільних застосунків. SCIENCE, RESEARCH, DEVELOPMENT. Technics and technology. #4 Barcelona. С. 79-83.