Міністерство освіти і науки України

Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»

Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка

до кваліфікаційної
ОС Магістр

на тему: «Дослідження впливу способу управління станом Flutter додатку на його швидкодію та ресурсоємність»
за освітньою програмою: «121 Інженерія програмного забезпечення»
зі спеціальності: «121 Інженерія програмного забезпечення»

Виконав: студент групи ПЗ2421: / Юрій КРИВОНОСОВ /

Керівник: / Віктор ШАРАВАРА /

Нормоконтролер: / Світлана ВОЛКОВА /

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент

Дніпро – 2026 рік

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»

Department «Computer information technology»

Explanatory Note
to Masters's Thesis

on the topic: «Research into the impact of the Flutter application state management method on its performance and resource consumption»
according to educational curriculum «121 Software engineering»
in the Speciality: «121 Software engineering»

Done by the student of the group PZ2421: / Yurii KRYVONOSOV /

Scientific Supervisor: / Viktor SHARAVARA /

Normative controller: / Svitlana VOLKOVA /

Dnipro – 2026

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: «Комп'ютерні технології і системи»
Кафедра: «Комп'ютерні інформаційні технології»
Рівень вищої освіти: магістр
Освітня програма: «121 Інженерія програмного забезпечення»
Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри КІТ
_____ /Вадим ГОРЯЧКІН/
(підпис)

Дата _____

З А В Д А Н Н Я

на кваліфікаційну роботу Магістра

студенту Кривоносову Юрію Антоновичу

1. Тема роботи: «Дослідження впливу способу управління станом Flutter додатку на його швидкодію та ресурсоємність»
Керівник роботи: Шаравара Віктор Володимирович, старший викладач
затверджені наказом № 1401 ст від 02.10.2025.
2. Строк подання студентом роботи: 05.01.2026
3. Вихідні дані до роботи:
4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати):
вступ, теоретичні відомості, проєктування та розробка ПЗ, проведення експериментів, обробка отриманих даних, висновки, література.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): презентація.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вступ	16.10.2025 – 17.10.2025	
2	Вибір тестових випадків та способів управління станом	19.10.2025 – 24.10.2025	
3	Розробка ПЗ для проведення експериментів та роботи з даними	25.10.2025 – 16.11.2025	30%
5	Проведення експериментів	17.11.2025 – 28.11.2025	
6	Компонування даних	29.11.2025 – 08.12.2025	
7	Опис програмного забезпечення та експериментів	09.12.2025 – 16.12.2025	60%
8	Оцінка отриманих результатів	19.12.2025 – 26.12.2025	
9	Опис додатків	24.12.2025 – 27.12.2025	
10	Розробка демонстраційних матеріалів	28.12.2025 – 30.12.2025	100%
	Подання кваліфікаційної роботи до кафедри	05.01.2026 – 12.01.2026	
11	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	___.01.2026	

Студент _____ Юрій КРИВОНОСОВ
(підпис) (Ім’я ПРІЗВИЩЕ)

Керівник роботи _____ Віктор ШАРАВАРА
(підпис) (Ім’я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра 116 с., 46 рис., 36 табл., 12 джерел.

Об’єктом дослідження є різні способи управління станом у Flutter-додатках та їхній вплив на продуктивність. Метою кваліфікаційної роботи є порівняння ефективності та ресурсоємності обраних стейт-менеджерів. Методами дослідження є експериментальне тестування на спеціально розробленому програмному забезпеченні, що імітує різноманітні сценарії взаємодії з інтерфейсом. Випробування проводяться на емуляторі та реальному мобільному пристрої для оцінки впливу апаратного середовища. Результатами та їхньою новизною є виявлення статистично значущих відмінностей у продуктивності між підходами, визначення типових сценаріїв, в яких кожне рішення має перевагу, та демонстрація суттєвої різниці у поведінці на емуляторі та реальному девайсі. Значенням роботи та висновками є формування практичних рекомендацій щодо вибору стейт-менеджера залежно від типу додатка, а також підтвердження необхідності тестування продуктивності на фізичних пристроях для отримання релевантних результатів.

Пояснювальна записка складається з 7 розділів:

- вступ – у даному розділі описується необхідність проведення даного дослідження і його актуальність. Складається з 2 сторінок;
- теоретичні відомості – у даному розділі описується вибір способів управління станом, показники, за якими будуть проведені порівняння, тестові випадки, план експерименту та спосіб обробки отриманих даних. Складається з 12 сторінок;
- проєктування та розробка ПЗ для проведення експерименту – у даному розділі описується проєктування та процес розробки ПЗ для проведення експериментів. Складається з 20 сторінок;
- проведення експериментів – у даному розділі описується спосіб та процес проведення експериментів. Складається з 20 сторінок;

- обробка отриманих даних – у цьому розділі описується процес обробки експериментально отриманих даних. Складається з 59 сторінок;
- висновки – підбиваються підсумки порівняння способів управління станом та здійснюється оцінка проведеної роботи. Складається з 2 сторінок;
- список літератури – містить список використаної літератури та посилань. Складається з 1 сторінки;
- додатки – містять текст програми та скриптів для обробки даних, керівництво користувача та технічне завдання.

Кількість таблиць: 36 штук.

Кількість рисунків: 46 штук.

Ключові слова: спосіб управління станом, Flutter, додаток, експеримент, тестування, стан, Stateful, Provider, Bloc, Riverpod.

ЗМІСТ

ВСТУП

Кількість користувачів мобільних додатків у всьому світі невпинно росте, а разом з цим – і попит на їх розробку. Останні два роки принесли справжній “БУМ”, де додатки не лише множаться, а й активно доповнюються інструментами штучного інтелекту. Оскільки основних мобільних платформ дві – Android та iOS, вимога одночасної доступності на обидва ринки змушує бізнес шукати ефективні рішення. Тут на перший план виходять кросплатформені фреймворки, серед яких Flutter став найпопулярнішим інструментом, завдяки можливості створювати додатки для обох платформ майже за однією ціною. Однак швидкість розробки – лише одна сторона медалі. Кінцевого користувача цікавить плавність, відгук та стабільність інтерфейсу, які безпосередньо залежать від того, як у додатку організовано управління станом – тобто оновлення списків, полів форми, анімацій та реакції на дії.

Вибір підходу до управління станом – одне з ключових архітектурних рішень у Flutter. Існує кілька популярних бібліотек, таких як Bloc, Provider, Riverpod, а також вбудований StatefulWidget. Кожна з них пропонує свою філософію від простих до складних абстракцій. У спільноті виникають дискусії про те, який із способів «кращий», але ці суперечки часто ґрунтуються на суб’єктивному досвіді або тестах у ідеальних умовах емулятора. Реальна ж продуктивність на фізичному пристрої, особливо під навантаженням, залишається недостатньо дослідженою. Чи дійсно існує універсальний переможець? Чи є сенс використовувати легкий StatefulWidget для простих інтерфейсів? Чи винагороджуються великі витрати ресурсів у складних рішеннях типу Bloc у реальних умовах? На ці питання поки що немає науково підтверджених відповідей.

Саме ця прогалина й обумовлює актуальність даної роботи. Її мета – не просто порівняти різні способи управління станом, а зробити це на основі об’єктивних даних, зібраних у контрольованому тестовому середовищі, що імітує реальні сценарії: роботу з мережею, складні анімації, заповнення форм та

створення довгих списків. Тестування буде проведено на двох платформах – на емуляторі та на реальному смартфоні – щоб виявити можливу різницю в поведінці, яка часто ігнорується. Після збору великих масивів даних про частоту кадрів (FPS), час кадру та затримки відгуку інтерфейса, дані будуть піддані ретельній статистичній обробці. У роботі будуть застосовані непараметричні тести, щоб визначити, чи є виявлені відмінності статистично значущими, і буде оцінено їх практичну значущість за допомогою величини ефекту.

Таким чином, ця робота має чітку науково-практичну спрямованість. Її результати матимуть безпосереднє значення для розробників та архітекторів, надаючи їм емпірично підтверджену основу для вибору стейт-менеджера. Замість вибору на основі досвіду чи суб'єктивних порад, розробники зможуть керуватися результатами даного дослідження. Новизна роботи полягає саме в такому комплексному, статистично обґрунтованому підході до порівняння, що охоплює як різні бібліотеки, так і різні середовища виконання. Висновки та рекомендації, сформовані в результаті дослідження, стануть цінним внеском як для практики промислової розробки, так і для подальших академічних досліджень у цій галузі.

Результати даного дослідження були апробовані на XIX Міжнародній науково-практичній конференції «Сучасні інформаційні та комунікаційні технології на транспорті, в промисловості та освіті». Основні висновки та методологія порівняльного аналізу були представлені у вигляді тез доповіді, які успішно пройшли відбір та включені до збірника матеріалів конференції.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1 Аналіз сучасних рішень на тему способу контролю станом

Вибір оптимального підходу до управління станом залишається однією з найбільш суперечливих тем у Flutter-спільноті. Це підтверджують численні дискусії на платформах Stack Overflow, GitHub, Reddit та у фахових блогах, де розробники шукають відповідь на питання «який стейт-менеджер найкращий?» для конкретних сценаріїв [1, 2]. Однак, емпірична основа для таких рекомендацій часто є фрагментарною. Популярні відеоогляди та статті базуються переважно на суб'єктивному враженні від розробки (DX – Developer Experience), що не відображає реальну продуктивність складного додатку під навантаженням. Ця прогалина між суб’єктивними думками та об’єктивними даними створює реальну потребу у системному дослідженні. Наша робота є актуальною саме тому, що вона покликана замінити суперечки на точні дані, надаючи розробникам науково обґрунтований інструмент для прийняття рішень, що безпосередньо впливає на якість кінцевого продукту.

1.2 Критерії вибору способу управління станом

Перш ніж розпочати експеримент, необхідно обрати спосіб контролю стану. Критеріями для їх відбору було обрано:

- підхід до архітектури;
- популярність та підтримуваність у спільноті Flutter;
- розповсюдженість у production-додатках.

Способи контролю стану у Flutter можна поділити на два різновиди – локальний та глобальний. Під локальним контролем стану мається на увазі контроль стану в одному конкретному віджеті. Переважно для цих цілей використовують Stateful Widget [3]. Це зумовлюється трьома факторами: простотою реалізації, можливістю реалізації без сторонніх бібліотек і відсутністю альтернативних рішень, оскільки у своїй ніші Stateful Widget надає

дуже великий набір інструментів, що не мотивує розробників створювати нові власні рішення.

Глобальний контроль стану – це менеджмент даних та надання функцій, які знаходяться на вищому рівні абстракції ніж 1 віджет. Наприклад, ваш додаток це будильник. Ви вирішили, що для деяких користувачів буде зручніше користуватися 12-годинним варіантом відображення часу. Ви додали екран налаштувань, де можна переключатися між цими форматами. Але ж тепер треба якось ці дані передати на екран будильника. Так, можна зробити це все єдиним віджетом, і використовувати для контролю станом Stateful Widget, але тоді порушується Single-responsibility principle із SOLID. А у випадку більше ніж двох екранів, глобальний стан виносять на вищий рівень. Але тоді виникає необхідність в інструментах, якими можна передати його у будь-яку частину додатку. У фреймворку вже є нативне рішення – InheritedWidget [4]. Віджети можуть підписатися на його зміни, а сам він містить у собі потрібні для них значення, які будуть знаходитися поза життєвим циклом окремо взятого віджета. Цей спосіб уже є дієвим, але розробники вирішили його покращити, створивши різноманітні бібліотеки, які використовують у своїй базі InheritedWidget, але значно розширюють його функціонал.

Отже кандидат для управління локальним станом обраний за неможливістю розглянути альтернативи. А ось для вибору стейт-менеджерів глобального рівня потрібно звернутися до джерел, які покажуть найпопулярніші рішення.

На поточний стан ринку (2024-2025), на основі аналізу з офіційного сайту pub.dev [5] (офіційного менеджера пакетів Dart/Flutter), та GitHub [6] статистики можна виділити чотири найпопулярніші рішення для управління станом:

- Bloc - найбільш структурований та бізнес-орієнтований підхід;
- Provider – у минулому лідер, найближчий за структурою до нативного InheritedWidget;

- Riverpod – лідер за популярністю та сучасна альтернатива Provider з кращою типобезпекою;
- GetX – простий фреймворк із низьким порогом входу.

1.3 Обґрунтування вибору стейт-менеджерів

Для наукового дослідження було обрано **три найпопулярніші рішення** (рис. 1.1 – 1.4):

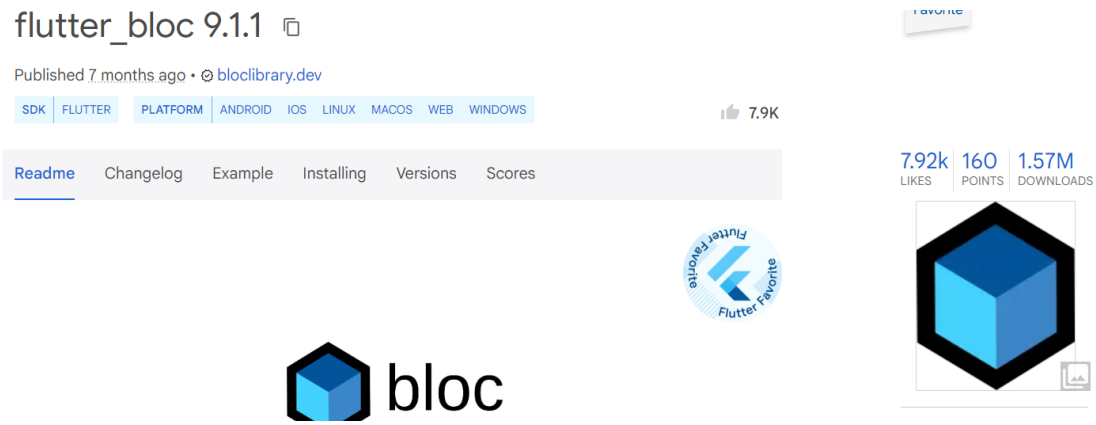

Рис. 1.1 – Статистика завантажень стейт-менеджеру Bloc на момент грудня 2025 р.

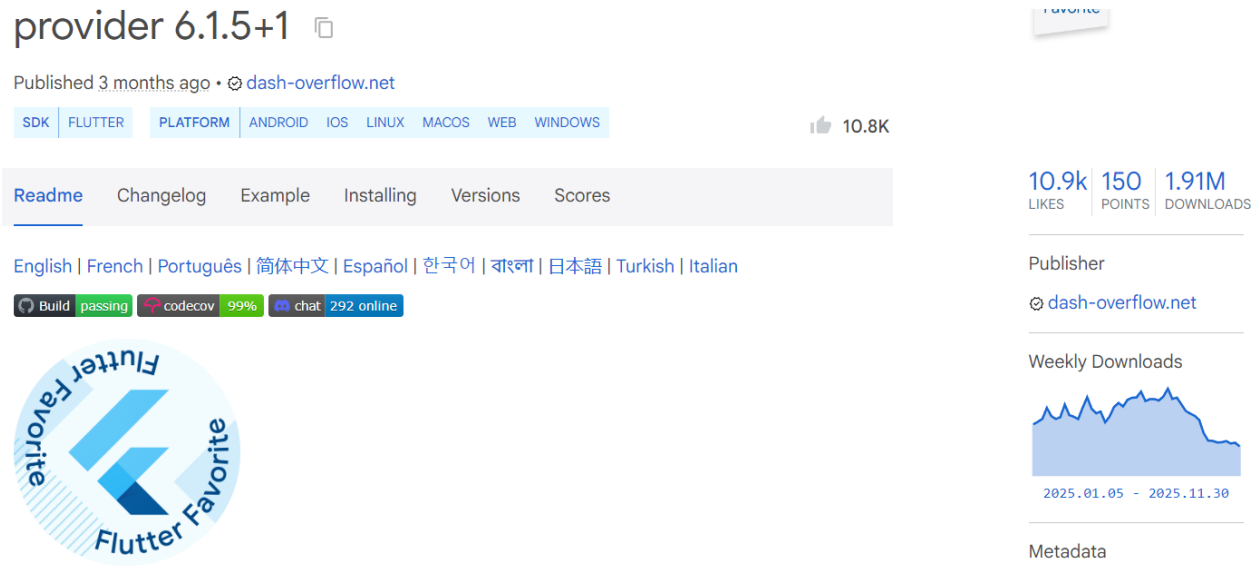

Рис. 1.2 – Статистика завантажень стейт-менеджеру Provider на момент грудня 2025 р.

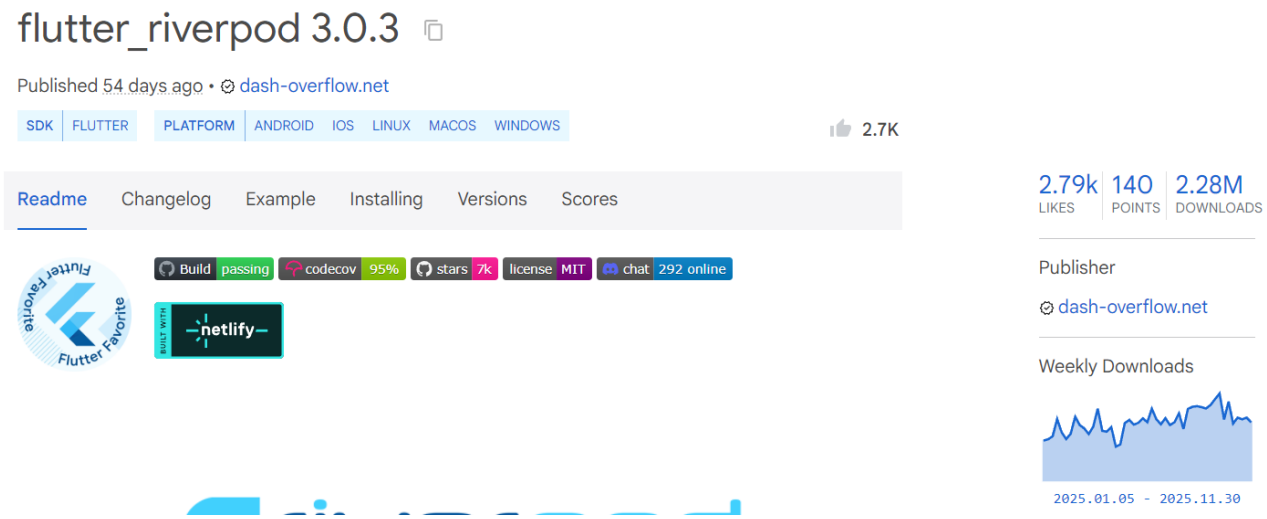

Рис. 1.3 – Статистика завантажень стейт-менеджеру Riverpod на момент грудня 2025 р.

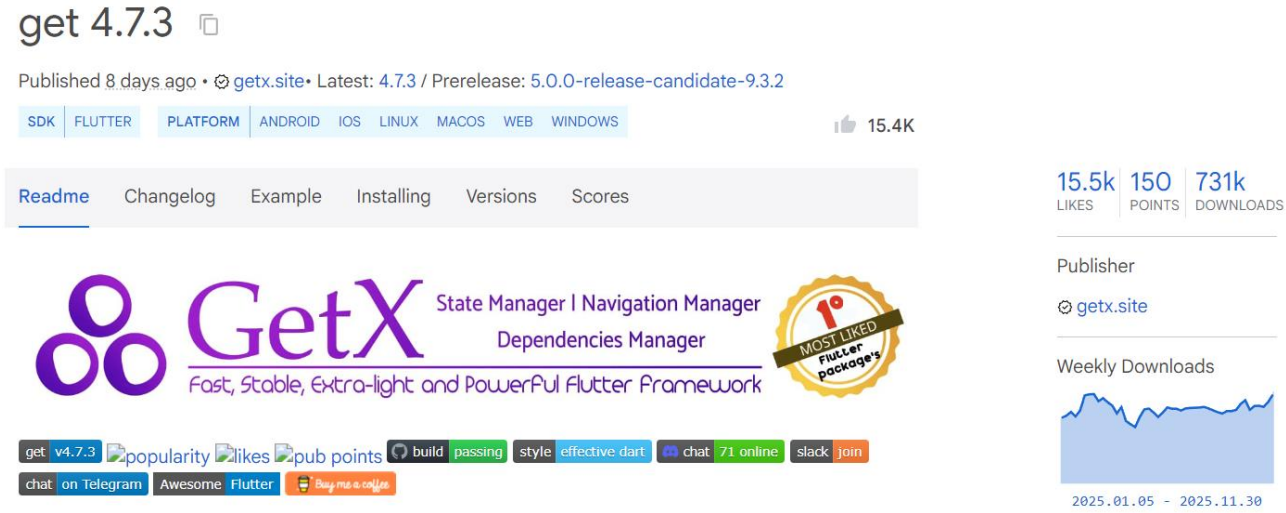

Рис. 1.4 – Статистика завантажень стейт-менеджеру GetX на момент грудня 2025 р.

1. Provider було включено до дослідження, зважаючи на наступні фактори:

- офіційна рекомендація Google для Flutter;
- найширша спільнота та кількість навчальних ресурсів;
- зріла екосистема з найбільшою кількістю сумісних бібліотек.

2. Bloc було включено, спираючись на такі аргументи:

- архітектурна чіткість (події, стани, редюсери);

- широке використання у корпоративних додатках;
- найкраща підтримка тестування та відлагодження;
- друга за популярністю серед серйозних проектів.

3. Riverpod було включено до дослідження, зважаючи на наступні фактори:

- технологічний прорив порівняно з Provider;
- compile-time безпека типів;
- швидке зростання популярності у 2022-2024 роках;
- сучасний підхід, що представляє майбутнє стейт-менеджменту.

4. GetX виключено з дослідження через такі недоліки:

- менша популярність порівняно з іншими рішеннями;
- архітектурні суперечності у спільноті;
- обмежене використання у великих проектах;
- основна аудиторія - початківці розробники.

Додаткові критерії відбору:

1. активність розробки: усі три обрані бібліотеки мають активну підтримку;
2. production readiness: підтверджені великими додатками у магазинах;
3. документація: наявність повноцінної документації та прикладів;
4. спільнота: активні форуми, чати та контрибутори.

Для оцінки тенденцій у спільноті були використані результати Flutter Developer Survey 2023, які демонструють наступний розподіл використання стейт-менеджерів:

- Provider використовують приблизно 45% розробників;

- Bloc використовують приблизно 25% розробників;
- Riverpod використовують приблизно 15% розробників, але він демонструє найшвидше зростання;
- GetX використовують приблизно 8% розробників, переважно у small-to-medium проектах.

Отже, можна зробити висновок, що вибірка з Provider, Bloc та Riverpod є репрезентативною, оскільки:

1. Покриває 85%+ ринку стейт-менеджерів Flutter;
2. Представляє різні архітектурні підходи;
3. Включає як лідера ринку, так і перспективні технології;
4. Відповідає реальній практиці комерційної розробки.

1.4 Показники для порівняння

Для порівняння способів контролю стану між собою потрібно виділити критерії, за якими можна буде обґрунтувати, який спосіб показує себе найкраще в одному або у декількох різних варіантах використання. Оскільки маємо справу із користувацьким інтерфейсом, основними показниками мають бути показники швидкості відклику UI на дії користувача, адже ефективний контроль стану має викликати якомога менше затримок у перемалюванні інтерфейсу та забезпечувати коректну реакцію на дії користувача. Але окрім цього є й інші групи показників, на які варто звернути увагу.

1.4.1 Продуктивність додатку

Перше, що бачить користувач, – це наскільки швидко та без затримок додаток реагує на його дії. Продуктивність – це один із найважливіших показників для бізнес-додатків, оскільки її високий рівень формує гарне враження та дає відчуття якості додатку, що викликає в користувача довіру та бажання ним користуватися. Основними показниками UI продуктивності є:

Час кадру — це кількість мілісекунд, яку пристрій витрачає на малювання одного кадру. Чим менше це число, тим краще. Якщо час кадру починає перевищувати 16 мілісекунд, людина вже може помічати підвищення.

Кількість кадрів за секунду (FPS) — це показник того, скільки окремих зображень (кадрів) відображається на екрані за одну секунду, створюючи ілюзію руху. Бажане цільове значення залежить від частоти оновлення екрану конкретного пристрою. Більшість сучасних екранів забезпечує частоту щонайменше 60 Гц, тому 60 кадрів на секунду – це стандартне рекомендоване значення FPS. У роботі досліджується, як різні способи управління станом впливають на значення цього показника.

Затримка реакції — це кількість мілісекунд що проходить між дією користувача (наприклад, натисканням кнопки) і моментом, коли додаток починає на неї реагувати. Навіть якщо додаток плавний, велика затримка створює відчуття «важкості» та неслухняності інтерфейсу.

1.4.2 Використання пам’яті

Важливим аспектом є ефективність використання додатком оперативної пам’яті (RAM) пристрою. В експериментах важливо відслідковувати не тільки середнє значення оперативної пам’яті, а й пікові значення, що дозволить виявляти ситуації, коли додаток раптово споживає дуже багато пам’яті. Також можна оцінити стабільність використання пам’яті: чи зростає споживання пам’яті з часом (наприклад, через витік пам’яті), чи залишається приблизно на одному рівні протягом тривалої роботи додатка.

1.4.3 Зручність для розробника

Третій блок показників важливий не для кінцевого користувача, а для розробників. Адже якщо рішення дуже швидке, але з ним незручно працювати, його все одно будуть використовувати рідко. Даний критерій є досить суб’єктивним, тому надалі він не буде використовуватися для порівняння, адже метою порівняння є саме дослідження впливу на ресурсоємність та швидкодію додатка, а не на зручність у використанні.

1.5 Тестові випадки для порівняння

Для порівняння стейт-менеджерів необхідно створити такі експериментальні умови, які покриють найбільшу частку можливих дій з інтерфейсом користувача, щоб отримані результати були корисними у більшості випадків. Для цього необхідно обрати найбільш поширені види UI віджетів та скомпонувати вибірку так, щоб вона симулювала найбільшу кількість типових дій з інтерфейсом (на кшталт скрол, свайп, клік та інші).

1.5.1 Тест зі списками

Майже кожен сучасний мобільний додаток використовує списки. Це може бути стрічка новин у соціальній мережі, перелік товарів у магазині, список контактів у телефоні або історія повідомлень у чаті.

У цьому тесті імітується саме така ситуація: додаток має довгий список елементів, який можна прокручувати, а окремі елементи можуть оновлюватися динамічно. Наприклад, коли надходять нові повідомлення або змінюється статус замовлення.

У даному тесті має перевірятися, як різні способи управління станом впливають на плавність прокручування та швидкість оновлення окремих елементів списку. Це критично важливо, бо користувачі дуже чутливі до «підвисань» при прокручуванні.

1.5.2 Тест з формами

Будь-який додаток, де потрібно вводити дані, має форми. Це може бути реєстрація, оформлення замовлення, заповнення анкети або налаштування параметрів.

Цей тест включає не просто статичну форму, а форму з динамічною валідацією. Наприклад, поле паролю показує помилку, поки ви не ввели достатньо символів, а поле email перевіряє правильність формату на льоту. Також було додано залежні поля - коли вибір у одному списку змінює доступні опції в іншому.

Такий тест показує, як швидко інтерфейс реагує на дії користувача, і як ефективно різні стейт-менеджери обробляють складну взаємодію між багатьма полями форми.

1.5.3 Тест з анімаціями

Сучасні додатки стають все більш інтерактивними та візуально привабливими. Анімації - це не просто прикраси, вони допомагають користувачеві зрозуміти, що відбувається в додатку. Плавні переходи між екранами, анімовані кнопки, ефекти завантаження - все це створює якісний користувацький досвід.

У цьому тесті перевіряється, як різні підходи до управління станом впливають на плавність анімацій. Особливо цікавлять ситуації, коли анімація відбувається одночасно з оновленням даних — наприклад, коли під час анімованого переходу на новий екран вже починають завантажуватися дані для цього екрана.

1.5.4 Тест з роботою з API

Майже жоден сучасний додаток не працює сам по собі. Він отримує дані з сервера — новини, повідомлення, інформацію про товари, погоду, курс валют. Цей тест перевіряє, як різні способи управління станом обробляють асинхронні операції: початок завантаження, очікування відповіді, обробка успішного результату або помилки.

Перевіряється не тільки швидкість отримання даних, але й те, як система веде себе при помилках мережі, як кешує отримані дані та як обробляє оновлення даних, які вже частково завантажені.

1.5.5 Тест з підписками на дані

Деякі додатки потребують не просто одноразового запиту даних, а постійного потоку оновлень. Наприклад, чат, де нові повідомлення з’являються в реальному часі, або фінансовий додаток, де курс акцій змінюється кожну секунду.

У цьому тесті імітується саме така ситуація: додаток підписується на потік даних і постійно отримує оновлення. Ціль перевірки – визначити як різні стейт-менеджери обробляють постійний потік змін, як ефективно вони оновлюють інтерфейс без зайвих перемальовувань і як себе поводять при високій частоті оновлень.

1.6 Проведення експерименту

Методологія експерименту дуже важлива, оскільки умови його проведення не ідеальні, і головна задача – отримати максимально достовірні дані. Для цього був розроблений спеціальний план експерименту.

1. Експеримент має проходити лише на двох девайсах – емулятор та реальний пристрій. Такий вибір зумовлений тим, що додатки в основному розробляють на емуляторі, але кінцевий користувач використовує його на реальному девайсі. Також у дослідженні необхідно порівняти стейт-менеджери між собою в однакових умовах, щоб різниця не виникала внаслідок різних платформ чи середовищ, і це не зіпсувало результати.
2. Прогрів Dart Virtual Machine (DVM). На цьому етапі необхідно, на девайсі, який буде використовуватися для подальшого тестування, запустити кожен вид тесту по декілька разів, щоб DVM оптимізувала навантаження. Без цього кроку перші тести можуть показати гірший результат, тому що графічній бібліотеці фреймворку необхідний час, щоб оптимізувати виконання коду. Цей етап повинен гарантувати виконання усього набору тестів у рівних умовах.
3. Тести необхідно проводити у змішаному порядку. Якщо завжди запускати тести в одному порядку, наприклад, спочатку всі тести з Provider, потім з Bloc, то може виникнути систематична похибка. Можливо, на початку тестування емулятор ще не прогрівся і працює повільніше, або навпаки, після тривалої роботи починає перегріватися. Тому необхідно запускати тести у випадковому порядку, щоб усереднити такі ефекти.

4. Кожен тестовий випадок має виконуватися 500 разів. Вибір кількості ітерацій для кожного тесту ґрунтується на методології послідовного аналізу, запропонованій Law та Carson [7]. Автори демонструють, що для досягнення стабільного стану (steady-state) в імітаційних експериментах необхідно визначати розмір вибірки на основі статистичної точності. Згідно з їхніми дослідженнями, збільшення кількості ітерацій призводить до кращої узгодженості результатів з теоретичними прогнозами.

Висновки до розділу 1

На основі даного розділу можна зробити наступні висновки щодо підготовчої частини дослідження.

По-перше, вибір методології порівняння був обґрунтований поєднанням об’єктивних та практичних критеріїв. У якості об’єктів дослідження обрано чотири підходи: Stateful Widget як обов’язковий, представлений у фреймворку, інструмент, для локального стану, а також три найпопулярніші бібліотеки для глобального стану – Provider, Bloc та Riverpod. Ця вибірка не є випадковою: вона охоплює понад 85% ринку стейт-менеджерів у Flutter, представляє різні архітектурні філософії (від мінімалістичних до чітко структурованих), і відображає реальну практику комерційної розробки. Відмову від GetX обґрунтовано його меншою поширеністю у великих production-проектах, що могло б знизити практичну цінність дослідження.

По-друге, розроблена система оцінювання є комплексною та орієнтованою на користувацький досвід. Замість абстрактних наборів віджетів, були сформовані п’ять реалістичних тестових сценаріїв (робота зі списками, формами, анімаціями, API та підписками на віджети), які моделюють переважну більшість взаємодій користувача з сучасним додатком. Такий підхід дозволяє оцінити вплив архітектурного рішення на кінцеве сприйняття якості додатку.

Нарешті, методологія проведення експерименту покликана мінімізувати сторонні фактори та забезпечити достовірність результатів. Ключовими елементами є: тестування у двох середовищах (емулятор та реальний девайс),

виконання попереднього прогріву віртуальної машини, виконання значної кількості ітерацій (по 500 на сценарій) для досягнення статистичної стійкості даних. Ці заходи створюють необхідні умови для того, щоб відслідкувати відмінності у продуктивності, які можна буде пов’язати саме з обраними способами управління станом, а не з артефактами процесу вимірювання.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ПЗ ДЛЯ ПРОВЕДЕННЯ ЕКСПЕРИМЕНТУ

2.1 Мета розробки спеціалізованого ПЗ

Метою розробки спеціалізованого додатку є створення необхідних тестових ситуацій та правильний запис показників. Даний додаток має надавати такі можливості:

- Використовувати усі 4 зазначених стейт-менеджера. Це означає що, експериментатор повинен мати змогу запустити будь-який тестовий випадок на будь-якому із обраних стейт-менеджерів.
- Окремий запуск тестового випадку та його стейт менеджеру. Це необхідно для мінімізації фонових процесів у додатку, щоб показники тесту отримувалися із найменшою похибкою.
- Чіткий вимір і запис потрібних показників. Оскільки тестування буде проводитися на емуляторі та реальному девайсі, є проблема з перенесенням результатів на ПК. Рішенням є можливість копіювати json результати як текст і самому створювати .json результуючий файл.
- Також є необхідність у створенні додаткового ПЗ, яке надасть можливість конвертувати json у зручний для подальшої обробки формат.

Для реалізації даної мети необхідно сформулювати архітектурні принципи додатку. Можна виділити 6 принципів які будуть використовуватися у проєктуванні:

Перший принцип — модульність. Програму необхідно побудувати так, щоб кожен її модуль був окремим компонентом. Це дозволить проводити додаткові налаштування способів контролю стану, тестових випадків та способів виміру показників, та розширювати набір інструментів у майбутньому.

Другий принцип — простота використання. Програма повинна бути зрозумілою навіть для розробника, який бачить її вперше. Кнопки «Почати тест», «Вибрати стейт-менеджер», «Переглянути результати» — все має бути на своєму місці і мати зрозумілі назви.

Третій принцип — надійність. Програма не повинна ламатися посеред тестування. Якщо щось піде не так (наприклад, закінчиться пам’ять), вона повинна акуратно записати помилку і продовжити з наступним тестом.

Четвертий принцип — повторюваність. Це означає, що якщо інший розробник запустить наші тести на своєму комп’ютері, він отримає такі самі результати (з невеликою статистичною похибкою). Програма не має допустити ситуації, щоб результат залежав від того, в який день тижня запускається тест, або скільки програм відкрито на комп’ютері.

П’ятий принцип — масштабованість. Програма повинна вміти обробляти як невеликі тести (кілька ітерацій), так і великі експерименти (сотні чи тисячі ітерацій), і робити це ефективно, не засмічуючи пам’ять і не займаючи комп’ютер на цілий день.

Шостий принцип — прозорість. Кожен, хто дивиться на код або результати, повинен розуміти, що і як ми робимо. Ніяких «чорних скриньок», ніяких магічних чисел, які з’являються нізвідки. Все має бути чітко, документовано і пояснено.

2.2 Flutter додаток для тестування

Програма поділена на наступні частини:

Таблиця 2.1 – Модулі програми

Назва модуля	**Класи, що входять**	**Функції модуля**
Глобальне управління	TestConfigCubit TestRunner	Даний модуль відповідає за управління станом в усьому додатку. Його класи відповідають за ведення процесу проведення тестування та запис результатів

Назва модуля	Класи, що входять	Функції модуля
Показники	FpsTracker, LatencyTracker, StatsRecorder, MemoryTracker, WidgetCounter	Даний модуль відповідає за запис конкретних показників та компоновку результатів у зручний формат.
Менеджери	BlocAdapter, ProviderAdapter, StateFulAdapter, RiverpodAdapter, StateManagerBase	Даний модуль відповідає за реалізацію способів контролю стану, які будуть порівнюватися між собою
Моделі даних	TestBase, TestResults	Цей модуль відповідає за структурні моделі даних у додатку.
Екрани	HomeScreen, ResultScreen, SelectManagerScreen, SelectTestScreen, TestRunScreen	Цей модуль є реалізацією інтерфейса користувача для роботи із додатком
Тестові випадки	ListTest, AnimationTest, ApiTest, SubscriptionTest, FormTest	Цей модуль є реалізацією тестових випадків у додатку

Взаємодія модулів представлена на рис. 2.1:

Рис. 2.1 – Діаграма взаємодії користувача із модулями програми

2.2.1 Реалізація програми

Для побудови додатка була обрана система features. Вона максимально гарно вбудовується у модульну модель описану вище, оскільки кожна feature і є окремим модулем. Уся логіка, а це - реалізація способів контролю стану, тестових випадків та виміру показників, побудована за принципом API. У кожного із цих блоків є кореневий інтерфейс для зв'язку з усією внутрішньою логікою. Усі дані можна скопіювати у вигляді json – тексту. Додаток був реалізований в IDE VS Code [8].

Для способів контролю стану таким інтерфейсом виступає StateManagerAdater. Він задає таку структуру: поле зі значенням імені менеджера та два методи – запуск способу управління станом та очистка. Усі класи реалізації базуються на цій структурі.

Але кожен спосіб контролю стану має свої особливості у роботі із цією реалізацією:

- Stateful – працює найпростіше. У реалізації метода запуску має звичайний setState, що якраз і дозволяє оновлювати стан;
- Provider – його реалізація нагадує роботу табло на вокзалі. У класі створюється окремий об'єкт постачальник даних, який зберігає стан. Коли

дані змінюються, об’єкт викликає метод триггер, який передає оновлені дані на інтерфейс, що дає йому команду змінитися;

- Bloc – його роботу можна описати конвеєром на фабриці. Bloc має два потоки даних – від інтерфейсу до bloc, та від bloc до інтерфейсу. Для збереження даних ця схема має клас стан, а усі зміни інтерфейсу мають відповідні класи events. Bloc у свою чергу підписаний на events, у собі обробляє надіслані дані та відправляє готовий стейт на UI;
- Riverpod – має дуже схожу структуру із провайдером.

Тестові випадки побудовані за схожою моделлю. У її основі також поля із назвою тесту та поле із кількістю ітерацій, а методів три – підготовка до тесту, виконання однієї ітерації та очищення після тесту. Важливу роль у результаті також відіграє реалізація збору показників:

2.2.1.1 FpsTracker

FpsTracker – трекер для вимірювання FPS та часу кадрів, який збирає дані згідно наступних принципів:

1. Механізм відстеження кадрів реалізується наступним чином:
   - Програма «підслуховує» систему Flutter через SchedulerBinding.addTimingsCallback;
   - Кожного разу, коли Flutter малює кадр, він генерує подію з часами:
     - buildDuration - час побудови віджетів;
     - rasterDuration - час малювання на екрані.
   - Сума цих двох значень дає загальний час кадру.
2. Використовуються два способи вимірювання:
   - Автоматичний (addTimingFrame): дані від Flutter Performance API;
   - Ручний (addManualFrame): програма самостійно заміряє час між кадрами.

Трекер обчислює наступні показники:

- Середній час кадру – сума всіх часів поділена на кількість кадрів;
- 95-й персентиль – час, який перевищують тільки 5% найгірших кадрів;
- Кількість «jank» кадрів – кадри, які тривають довше 33.33 мс (менше 30 FPS);
- Дані для побудови графіку емпіричної щільності розподілу FPS – границі «кошиків» значень та кількість значень у відповідному «кошику».

2.2.1.2 LatencyTracker

LatencyTracker – це трекер для вимірювання затримки відгуку інтерфейсу, який збирає дані згідно наступних принципів:

1. Момент початку вимірювання:
   – Коли користувач робить дію (натискає кнопку, скролить), програма запам’ятовує точний час;
2. Момент закінчення вимірювання:
   – Коли інтерфейс реагує (показує анімацію, оновлює дані), програма знову дивиться на час;
3. Розрахунок затримки є різницею значення часу на кінець вимірювання та на початку вимірювання, поділеним на тисячу.

Трекер обчислює наступні показники:

- Мінімальна затримка: найшвидша реакція програми;
- Максимальна затримка: найповільніша реакція;
- Середня затримка: усереднена за всі виміри;
- Розподіл затримок: як часто зустрічаються різні значення.

2.2.1.3 MemoryTracker

MemoryTracker – це трекер для вимірювання використання оперативної пам’яті, який збирає дані згідно наступних принципів:

1. Запит до операційної системи;
2. Перетворення у мегабайти;
3. Частота вимірювань:

– Пам’ять перевіряється після кожного кадру;

– Це дозволяє побачити динаміку використання.

Трекер обчислює наступні показники:

- Середнє використання: усереднене за час тесту;
- Пікове використання: максимальне значення за весь тест;
- Стандартне відхилення: наскільки сильно коливається використання пам’яті;
- Зразки у часі: 20 точок, рівномірно розподілених за часом тесту.

2.2.1.4 WidgetCounter

WidgetCounter – трекер для вимірювання перемальовувань віджетів, який збирає дані згідно наступних принципів:

1. Спеціальний контекст:

   – Створюється WidgetCounterScope, який «обгортає» весь додаток;

   – Всі віджети мають доступ до лічильника через цей контекст.

2. Механізм підрахунку:

```
// У кожному віджеті, який треба відстежувати:
Widget build(BuildContext context) {
    WidgetCounter.of(context).increment(); // +1 перемальовування
    return Container(...);
}
```

3. Міксин для спрощення:

   – WidgetCounterMixin автоматично додає підрахунок до build() методу.

Трекер обчислює наступні показники:

- Загальна кількість викликів build() методів;
- Це показує, наскільки «нервово» перемальовується інтерфейс.

2.2.2 Проєктування зовнішнього інтерфейсу

Інтерфейс додатку має мінімалістичний, навіть лаконічний стиль, для кращого розуміння і чіткої побудови процесу експерименту. Він має у своїй основі 4 екрани:

- Головний екран (рис. 2.2) виконує навігаційну функцію. Екран містить дві кнопки для переходу на наступні етапи, а саме: екран налаштувань експерименту та результуючий екран.
- Екран налаштувань експерименту (рис 2.3) необхідний для вибору стейт-менеджера, тестових випадків та кількості ітерацій. Екран надає можливість повернутися на екран назад, він містить випадаюче вікно для вибору способу контролю стану, список із можливістю вибрати тестовий випадок, випадаюче вікно для вибору кількості повторень кожного тесту та текстове поле для вводу кількості загальних ітерацій.
- Екран тестування (рис. 2.4 – 2.5) даний екран є місцем для реалізації UI частини тестових випадків. Саме на ньому запускаються компоненти, які підлягають аналізу. Екран має лічильник ітерацій та лінію прогресу, які вказують на те, як проходить процес, а також кнопку запуску/зупинки тестування. Нижче знаходиться поле для відмалювання тестових віджетів та навантаження способів контролю станів.
- Результуючий екран (рис. 2.6) відображає результати тестувань. Він має кнопки очищення та копіювання результатів. У верхній плашці відображено загальну статистику по усім представленим на екрані результатам. Нижче знаходиться результуюча таблиця, кожен рядок якої є конкретним тестовим випадком. Цей екран також має кнопку для повернення на головний екран.

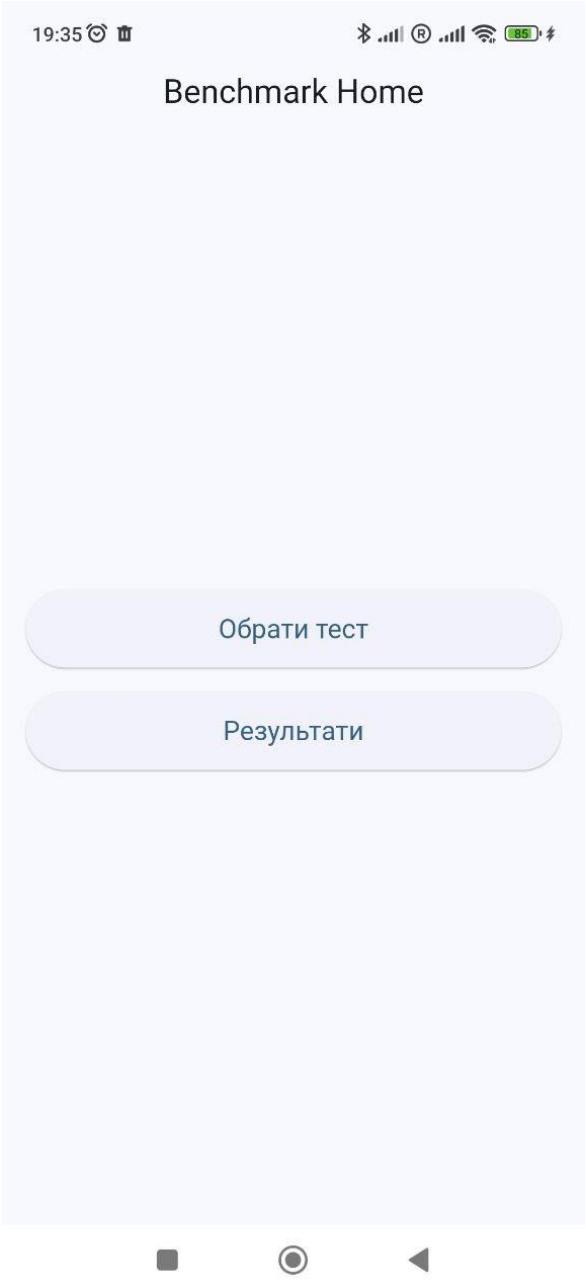

Рис. 2.2 – Головний екран додатку

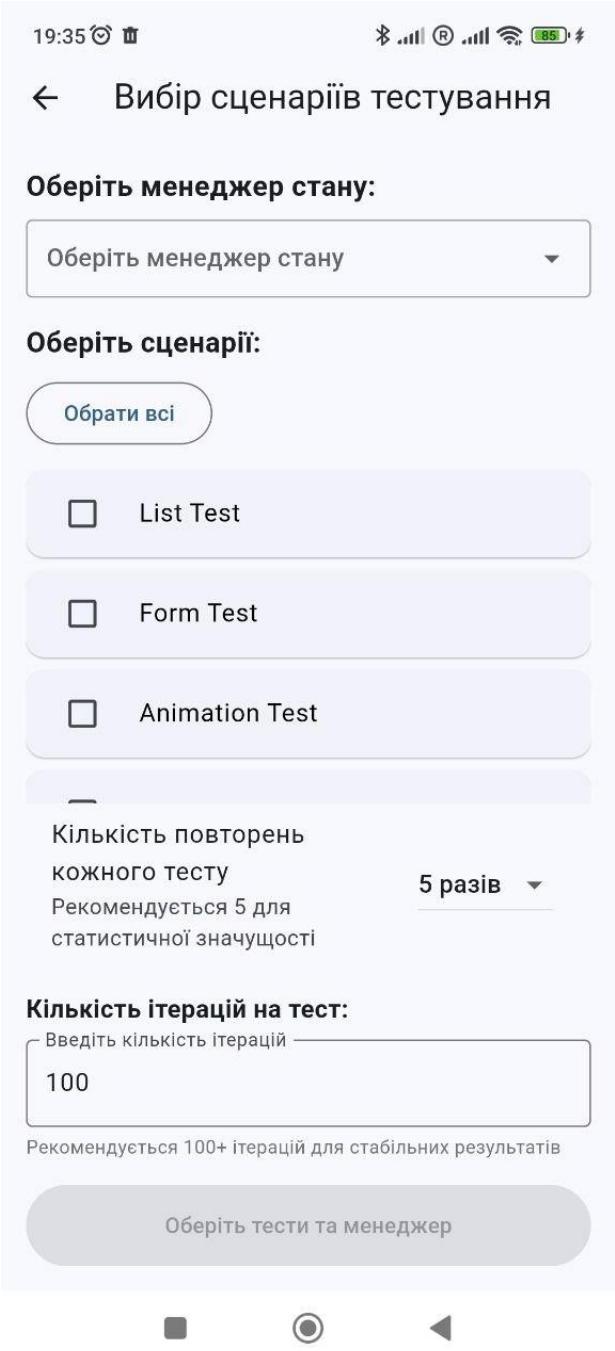

Рис. 2.3 – Екран налаштувань експерименту

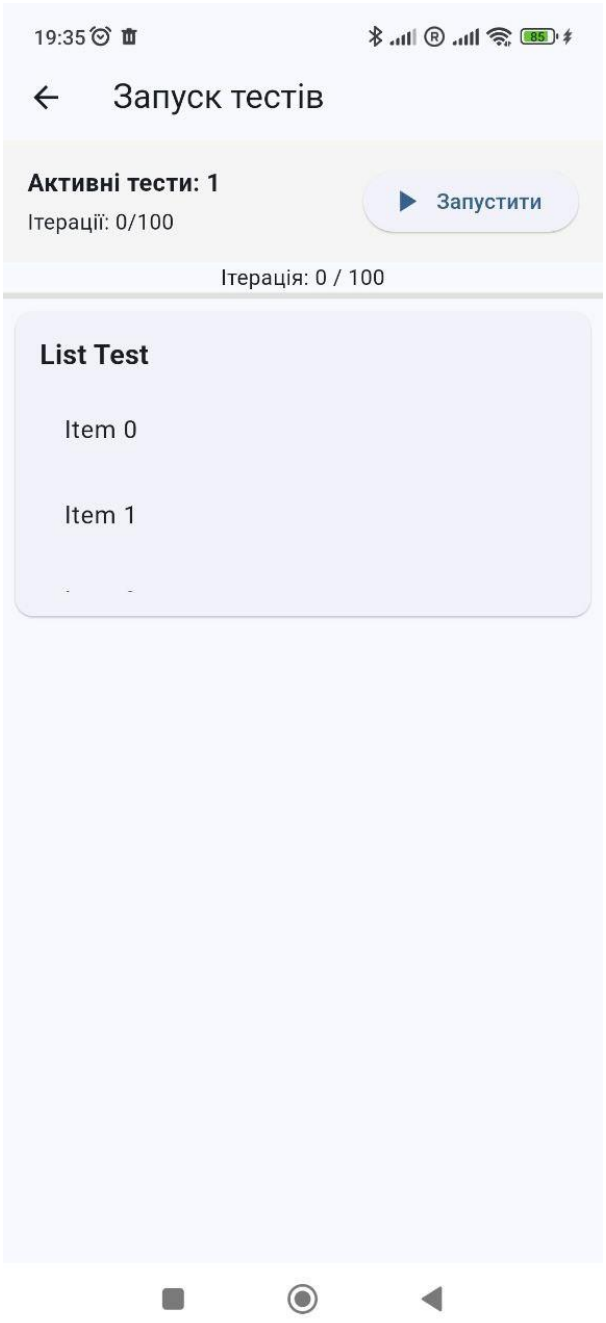

Рис. 2.4 – Екран тестування (тест не активовано, обраний тестовий випадок – список)

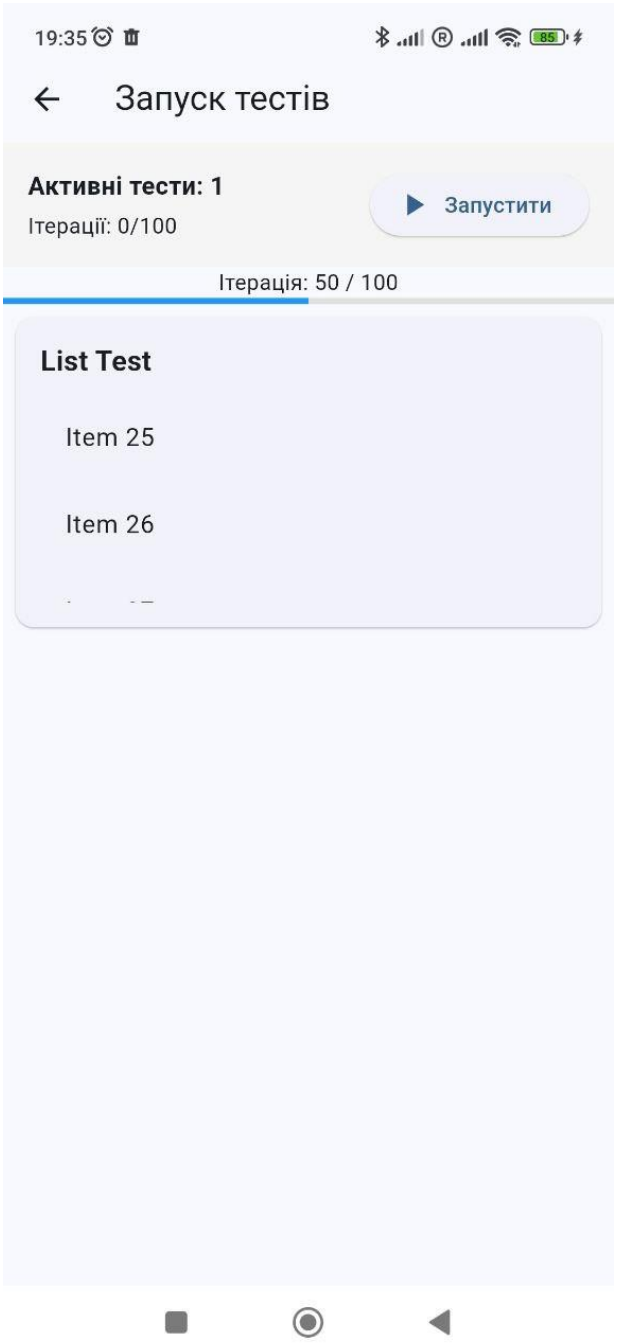

Рис. 2.5 – Екран тестування (тест активовано, обраний тестовий випадок – список)

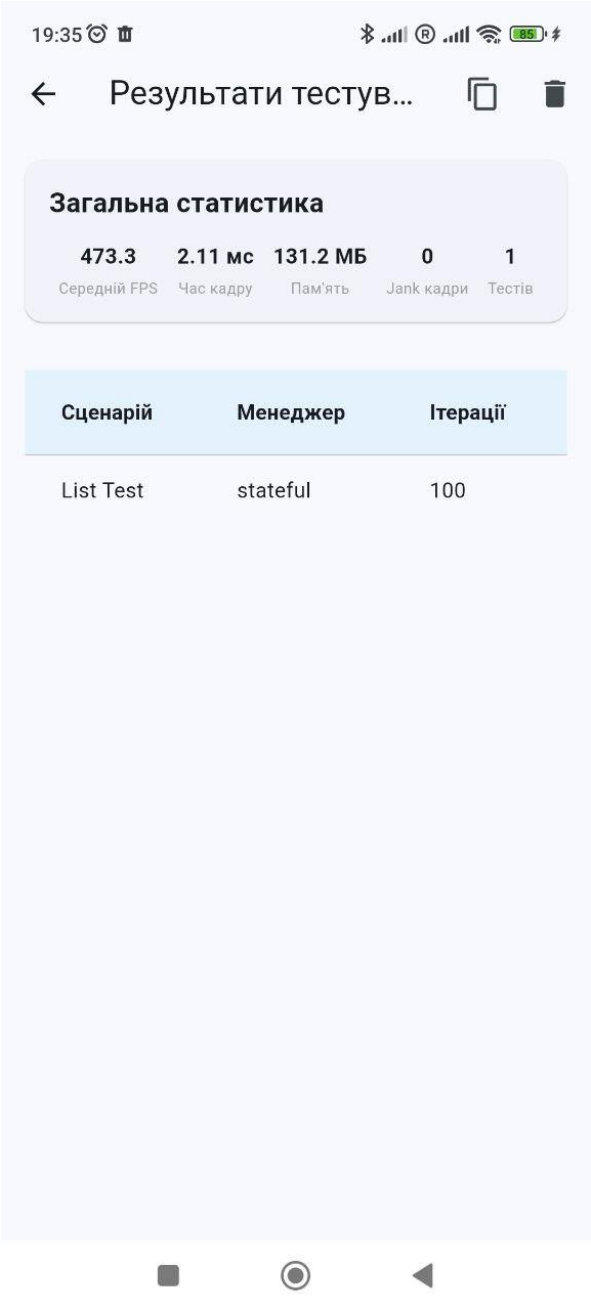

Рис. 2.6 – Результуючий екран

2.3 Скрипт для конвертування даних у Excel формат

Для обробки великих масивів експериментальних даних та їх подальшого аналізу було розроблено комплекс Python-скриптів. Основне призначення першого скрипта полягає в автоматизації процесу трансформації сирих даних у структуровану форму, придатну для статистичного аналізу та візуалізації.

Основні завдання скрипта:

1. імпорт та валідація даних з JSON-файлів, згенерованих Flutter-додатком;
2. трансформація вкладених структур у плоскі таблиці;
3. статистичний аналіз отриманих показників;
4. генерація звітів у форматі Excel та PDF;
5. візуалізація розподілів продуктивності для порівняльного аналізу.

2.3.1 Завантаження та обробка JSON-даних

Перший етап включає завантаження експериментальних даних з JSON-файлів. Кожен файл містить результати множинних тестових запусків з різними стейт-менеджерами та сценаріями.

Функція load_json_data виконує:

- перевірку існування файлу;
- валідацію JSON-структури;
- обробку помилок читання та декодування.

2.3.2 Розгортання розподілів у табличну форму

Експериментальні дані містять розподіли часу кадрів, FPS та затримок у формі словників, де ключами є значення метрик, а значеннями — кількість спостережень. Для подальшого аналізу необхідно трансформувати ці дані у табличний формат.

Алгоритм роботи функції:

- ітерація по всіх записах експериментів;
- екстракція сценарію та стейт-менеджера;
- розгортання словника розподілу на окремі рядки;

- створення колонок: Scenario, State Manager, Metric_Type, Bucket, Count.

2.3.3 Обробка часових рядів використання пам’яті

Для аналізу динаміки використання пам’яті реалізована окрема функція. Ця функція перетворює масив значень використання пам’яті, знятих через рівні проміжки часу, у таблицю з колонками: момент виміру та значення пам’яті.

2.3.4 Розрахунок щільності розподілів

Для порівняння розподілів між різними стейт-менеджерами необхідно нормалізувати дані шляхом обчислення щільності. Щільність розраховується як відношення кількості спостережень у кожному бакеті до загальної кількості спостережень для даної комбінації стейт-менеджера, сценарію та типу метрики.

2.3.5 Генерація Excel-звітів

Для структурованого представлення результатів створюється Excel-файл з окремими вкладками:

- Summary Metrics — основні числові показники;
- Frame Time Distribution — розподіл часу кадрів;
- FPS Distribution — розподіл кадрів за секунду;
- Latency Distribution — розподіл затримок;
- Memory Over Time — динаміка використання пам’яті;
- Pivot таблиці для порівняння середніх значень між стейт-менеджерами.

2.3.6 Створення графічних звітів у PDF

Для візуального аналізу генеруються гістограми розподілів та графіки динаміки пам’яті.

Особливості візуалізації:

- кожен стейт-менеджер представлений окремим стовпцем;
- використання дискретних кольорів для кращої диференціації;
- адаптивна ширина стовпців залежно від типу метрики;
- легенда з позиціонуванням поза областю графіка.

2.4 Скрипт для статистичного аналізу

Другий скрипт реалізує комплексну систему статистичного аналізу експериментальних даних з використанням непараметричного критерію Манна-Уітні. Основне призначення скрипта полягає у виявленні статистично значущих відмінностей у розподілах продуктивності між різними стейт-менеджерами для різних тестових сценаріїв.

Тест Манна-Уітні (також відомий як U-тест) [9] є непараметричним статистичним критерієм, який використовується для перевірки нульової гіпотези про те, що дві вибірки породжені однаковим розподілом. На відміну від параметричних тестів (наприклад, t-тест), Манна-Уітні не вимагає припущення про нормальність розподілу даних, що робить його особливо корисним у такому роді досліджень, оскільки у даному випадку немає гарантії отримання нормального розподілу даних.

2.4.1 Функція розгортання розподілів (expand_distribution)

Математична основа: Функція конвертує агреговані дані розподілів (де ключ — значення метрики, значення — кількість спостережень) у списки окремих спостережень. Це необхідно для застосування непараметричних статистичних тестів, які працюють з окремими спостереженнями.

2.4.2 Функція розрахунку статистик Манна-Уітні (calculate_mann_whitney_with_stats)

Математичні формули, що використовуються:

1. **U-статистика Манна-Уітні:**

- Обчислюється шляхом ранжування всіх спостережень обох вибірок
- Сума рангів для кожної вибірки

Формула: $U = n_1 n_2 + \frac{n_1(n_1+1)}{2} - R$ (2.1)

де: n1 - розмір першої вибірки, n2 - розмір другої вибірки, R – сума рангів вибірки.

2. Z-значення для великих вибірок:

$$z = \frac{U - \mu}{\sigma} \quad (2.2)$$

де: μ – це математичне очікування U-статистики, σ – стандартне відхилення.

3. Величина ефекту

$$r = \frac{|z|}{\sqrt{n_1+n_2}} \quad (2.3)$$

Класифікація величини ефекту:

- 0.10-0.30 — малий ефект;
- 0.30-0.50 — середній ефект;
- 0.50 — великий ефект.

2.4.3 Основна функція аналізу (perform_mann_whitney_tests)

Алгоритм роботи:

1. екстракція унікальних сценаріїв та стейт-менеджерів;
2. групування даних за сценаріями та показниками;
3. генерація всіх можливих пар стейт-менеджерів;
4. попарне тестування для кожної комбінації.

Статистичні гіпотези:

- Нульова гіпотеза H0: розподіли двох стейт-менеджерів ідентичні;
- Альтернативна гіпотеза H1: розподіли відрізняються;

- Рівень значущості: α=0.05.

2.4.4 Створення матриць порівняння (create_comparison_tables)

Типи матриць, що створюються:

1. матриця значущості — символи ***, **, *, н.з.;
2. матриця p-значень — числові значення ймовірностей;
3. матриця Z-значень — стандартизовані статистики;
4. матриця величин ефекту — практична значущість відмінностей.

Таблиця 2.2 – Приклад структури матриці величин ефекту

	Provider	Bloc	Riverpod	Stateful
Provider	-			
Bloc		-		
Riverpod			-	
Stateful				-

2.4.6 Обробка великих вибірок

Для великих вибірок (n1,n2>20) використовується нормальна апроксимація розподілу U-статистики. Це дозволяє обчислити точні p-значення за допомогою стандартного нормального розподілу.

2.4.7 Критерії статистичної значущості

Для інтерпретації результатів U-тесту Манна-Уітні використовуються стандартні статистичні критерії значущості, що базуються на p-значенні (ймовірності отримати спостережувані або більш екстремальні результати за умови істинності нульової гіпотези).

Згенеровані таблиці використовують наступну систему позначень:

- **н.з.** (незначуще): p≥0.05;

- ***: $p<0.05$;
- ****: $p<0.01$;
- *****: $p<0.001$.

Екстремальні значення (викиди) обробляються через рангову статистику, що робить тест Манна-Уітні стійким до впливу викидів.

2.4.8 Вихідні формати даних

Excel-звіт з основними результатами містить:

1. **Основні результати** — сценарій, показник, пари стейт-менеджерів, p-значення, значущість;
2. **Детальні статистики** — U-статистика, Z-значення, величини ефекту, середні, медіани;
3. **Статистика за сценаріями** — кількість тестів, значущих результатів, відсоток значущих.

Крім того створюються окремі файли з порівняльними таблицями для кожного показника:

- Frame Time;
- FPS;
- Latency.

Кожна матриця має розмір m×m, де m — кількість стейт-менеджерів.

Для інтерпретації статистичної значущості результатів використовується наступна шкала*:*

- $p<0.001$: дуже сильні докази проти нульової гіпотези;
- $p<0.01$: сильні докази проти нульової гіпотези;
- $p<0.05$: помірні докази проти нульової гіпотези;

- $p \geq 0.05$: недостатньо доказів для відхилення нульової гіпотези.

Для інтерпретації величини ефекту використовується наступна шкала:

- 0.0-0.1: незначний ефект;
- 0.1-0.3: малий ефект (практично незначущий);
- 0.3-0.5: середній ефект (помітний на практиці);
- 0.5 або більше: великий ефект (суттєвий на практиці).

Висновки до розділу 2

Розробка спеціалізованого програмного забезпечення стала ключовою складовою успішного проведення експерименту. Створений додаток реалізує всі необхідні функції для об’єктивного порівняння стейт-менеджерів: модульну архітектуру з чітким розділенням задач, точний збір показників продуктивності у реальному часі та інтуїтивний інтерфейс для управління тестуванням. Використання принципів модульності, надійності та повторюваності забезпечило стабільність експериментальної установки та можливість відтворення результатів.

Комплекс Python-скриптів для обробки даних значно автоматизував аналітичну частину дослідження. Перший скрипт ефективно трансформує сирі JSON-данні у структуровані таблиці Excel, що дозволяє здійснювати подальший статистичний аналіз. Другий скрипт реалізує науково обґрунтовану методологію статистичної перевірки на основі U-тесту Манна-Уітні, що є оптимальним вибором для даних продуктивності, які часто не відповідають нормальному розподілу. Використання величини ефекту Cohen’s r дозволяє оцінити не тільки статистичну значущість відмінностей, але й їх практичну важливість.

Інтегрована система збору, обробки та аналізу даних створює повний цикл експерименту — від виконання тестових сценаріїв до формування статистично обґрунтованих висновків. Ця методологія може бути успішно адаптована для подібних порівняльних досліджень у галузі розробки програмного забезпечення,

забезпечуючи високий рівень наукової достовірності та практичної значущості результатів.

РОЗДІЛ 3. ПРОВЕДЕННЯ ЕКСПЕРИМЕНТІВ

3.1 Методологія проведення експериментальних досліджень

Експериментальна методологія у дослідженні розроблялася шляхом послідовного вдосконалення на основі отриманих проміжних результатів та виявлених методологічних проблем. Замість одноразового проведення комплексного експерименту було обрано інкрементальний підхід, що дозволило поступово удосконалювати як технічні інструменти вимірювання, так і протоколи тестування.

Кожен наступний етап експериментальної роботи будувався на аналізі результатів попереднього етапу, що дозволило ідентифікувати та усунути систематичні похибки, вдосконалити процедури збору даних та забезпечити більш високу відтворюваність результатів. Такий підхід відповідає принципам наукового методу, де кожна ітерація сприяє підвищенню надійності експериментальних даних та валідності отриманих висновків.

Оскільки ця філософія проведення передбачала покращення процесу тестування, було проведено декілька ітерацій. Експериментальна робота була організована у вигляді трьох окремих сесій, кожна з яких мала чітко визначені цілі та завдання.

Перша експериментальна сесія мала за мету оцінити базову працездатність розробленої тестової платформи та отримати попереднє уявлення про характер відмінностей у продуктивності досліджуваних стейт-менеджерів. На цьому етапі основна увага приділялася технічній реалізації системи вимірювань та виявленню очевидних методологічних проблем. На цьому етапі і далі емулятор був створений за допомогою Android Studio [10].

Друга експериментальна сесія спрямовувалася на валідацію отриманих даних та перевірку їхньої статистичної значущості. Основне завдання цієї сесії полягало у верифікації того, що спостережувані відмінності в продуктивності не є випадковими артефактами або результатами методичних помилок.

Третя експериментальна сесія представляла фінальний етап експериментальної роботи, де проводилося комплексне тестування за вдосконаленою методологією зі статистично обґрунтованими параметрами. Ця сесія формувала основу для остаточних висновків щодо порівняльної ефективності різних підходів до управління станом.

Система збору даних організована на основі формалізованих протоколів, що забезпечують цілісність, повноту та консистентність зібраної інформації. Для кожного показника продуктивності визначено чіткі правила вимірювання, форматування та зберігання даних.

Всі вимірювання фіксуються з високою точністю, що дозволяє виявляти навіть незначні відмінності у продуктивності. Система автоматичного запису даних забезпечує реєстрацію не тільки самих значень показників, але й контекстної інформації: часу проведення тесту, його назви та інших параметрів середовища виконання.

Особлива увага приділялася мінімізації накладних витрат на збір даних, щоб процес вимірювання не впливав суттєво на результати тестування продуктивності. Інструменти збору метрик були вдосконалені для скорочення впливу на систему та забезпечення максимальної ефективності роботи.

3.2 Експериментальні умови

Системні характеристики емулятора:

Апаратні характеристики:

- процесор: емуляція ARM-архітектури з трансляцією інструкцій;
- кількість ядер: 4 віртуальних ядра;
- тактова частота: емулюється стабільна частота, незалежна від навантаження хостової системи;
- графічний процесор: програмна емуляція Adreno GPU з підтримкою OpenGL ES 3.0;

- оперативна пам’ять: 4 ГБ віртуальної RAM з детермінованим розподілом;
- внутрішня пам’ять: 10 ГБ емульованого сховища з фіксованою швидкістю доступу.

Програмне забезпечення:

- версія Android: Android 14 (API рівень 34);
- сервіси Google Play: відключені для уникнення фонової активності;
- системні додатки: мінімальний набір, залишені лише критичні системні компоненти;
- налаштування розробника: активовані режими для відладки та профілювання;
- дозволи: всі дозволи надані тестовому додатку за замовчуванням.

Параметри дисплея:

- роздільна здатність: 1080 × 2340 пікселів (Full HD+);
- щільність пікселів: 432 ppi (pixels per inch);
- частота оновлення: 60 Гц (стабільна, без адаптивної синхронізації);
- розмір діагоналі: 6.0 дюймів (віртуальний);
- співвідношення сторін: 19.5:9.

Системні характеристики реального девайсу:

Апаратні характеристики:

- процесор: Qualcomm Snapdragon 732G (SM7150-AC);
- кількість ядер: 2 продуктивні ядра - Kryo 470 Gold (Cortex-A76), 6 енергоефективних ядер - Kryo 470 Silver (Cortex-A55);

- тактова частота: в залежності від ядра – 2.8 ГГц для продуктивного, 1.8 ГГц для енергоефективного;
- графічний процесор: Qualcomm Adreno 618;
- оперативна пам’ять: 6 ГБ;
- внутрішня пам’ять: 128 ГБ.

Програмне забезпечення:

- версія Android: Android 14 (API рівень 34);
- сервіси Google Play: увімкнені;
- системні додатки: набір стандартних додатків Xiaomi;
- налаштування розробника: активовані режими для відладки та профілювання;
- дозволи: надані дозволи на використання сховища.

Параметри дисплея:

- роздільна здатність: 2400 × 1080 пікселів (Full HD+);
- щільність пікселів: 395 ppi (pixels per inch);
- частота оновлення: 60 Гц;
- розмір діагоналі: 6.67 дюймів;
- співвідношення сторін: 20:9.

3.3 Пілотне тестування

Початкові цілі та очікування:

Перший експеримент має бути пілотним. Основними його цілями є:

- Перевірка працездатності програми – як програма себе показує у кожному тестовому сценарії з кожним конкретним способом контролю стану;

- Перевірка якості збору та адекватність отриманих даних – подальша перевірка отриманих даних за допомогою методів статистичного аналізу, порівняння результатів між реальним девайсом та емулятором;

- Опрацювання методології проведення експерименту – оцінка на реальних випадках правильності обраної послідовності тестів, та способів мінімізації впливу сторонніх факторів на результати тестування.

Перший експеримент спочатку було проведено на емуляторі. Послідовність дій експерименту була така:

- запуск емулятора;

- завантаження на емулятор робочої версії product збірки додатку;

- відкриття додатку;

- проведення попереднього тестування у порядку (таблиця 3.1), на кожен тест 500 ітерацій:

Таблиця 3.1 – Порядок проведення експериментів

Номер тесту	**Стейт менеджер**	**Тестовий випадок**
1	Statefull	List View Test
2	Statefull	Form Test
3	Statefull	Animation Test
4	Statefull	API Test
5	Statefull	Subscription Test
6	Provider	List View Test
7	Provider	Form Test

Номер тесту	**Стейт менеджер**	**Тестовий випадок**
8	Provider	Animation Test
9	Provider	API Test
10	Provider	Subscription Test
11	Bloc	List View Test
12	Bloc	Form Test
13	Bloc	Animation Test
14	Bloc	API Test
15	Bloc	Subscription Test
16	Riverpod	List View Test
17	Riverpod	Form Test
18	Riverpod	Animation Test
19	Riverpod	API Test
20	Riverpod	Subscription Test

- очищення даних;
- повторне проведення тестів у тому ж порядку та по 500 ітерацій на тест;
- збереження результатів.

Далі був проведений експеримент на реальному девайсі. Послідовність дій експерименту була така:

- запуск девайсу;
- завантаження на девайс робочої версії product збірки додатку;
- відкриття додатку;
- проведення попереднього тестування у порядку (таблиця 3.1), на кожен тест 500 ітерацій;

- очищення даних;
- повторне проведення тестів у тому ж порядку та по 500 ітерацій на тест;
- збереження результатів.

Отримані дані:

Таблиця 3.2 – Дані експерименту 1 отримані на емуляторі

avgFrameTimeMs	avgLatencyMs	ramUsageMb	widgetRebuilds	stdDevFrameTimeMs	percentile95FrameTimeMs	jankFramesCount	jankFramesPercent	minLatencyMs	maxLatencyMs
5,64	0,05	266,01	1675,00	8,73	18,28	26,00	1,55	0,00	3,00
12,07	0,07	283,04	1737,00	13,91	31,69	57,00	3,28	0,00	10,00
3,37	0,12	285,40	1711,00	3,92	9,38	0,00	0,00	0,00	6,00
2,82	0,02	272,84	1668,00	3,24	7,25	0,00	0,00	0,00	4,00
2,80	0,02	275,88	1681,00	3,14	7,32	0,00	0,00	0,00	2,00
5,06	0,08	282,79	1682,00	6,58	17,19	6,00	0,36	0,00	8,00
11,90	0,08	286,64	1751,00	13,55	30,76	53,00	3,03	0,00	6,00
3,17	0,03	286,43	1697,00	3,75	8,51	0,00	0,00	0,00	5,00
2,84	0,05	276,24	1666,00	3,32	7,43	1,00	0,06	0,00	2,00
2,95	0,03	279,39	1680,00	4,81	7,51	2,00	0,12	0,00	2,00
4,82	0,07	273,91	1664,00	6,38	16,54	5,00	0,30	0,00	6,00
11,11	0,11	287,32	1652,00	13,48	30,80	54,00	3,27	0,00	8,00
3,18	0,11	289,87	1709,00	3,61	8,44	0,00	0,00	0,00	4,00
2,87	0,11	279,41	1710,00	3,04	7,09	0,00	0,00	0,00	9,00
2,66	0,07	279,90	1688,00	2,88	6,81	0,00	0,00	0,00	2,00
5,35	0,04	275,35	1683,00	7,91	17,90	18,00	1,07	0,00	4,00
11,41	0,11	293,58	1698,00	13,50	30,16	45,00	2,65	0,00	9,00
3,25	0,09	292,76	1702,00	3,91	8,39	2,00	0,12	0,00	9,00

avgFrameTimeMs	avgLatencyMs	ramUsageMb	widgetRebuilds	stdDevFrameTimeMs	percentile95FrameTimeMs	jankFramesCount	jankFramesPercent	minLatencyMs	maxLatencyMs
2,85	0,11	281,53	1669,00	3,22	7,26	0,00	0,00	0,00	2,00
2,76	0,05	283,19	1675,00	3,14	7,22	0,00	0,00	0,00	3,00

Таблиця 3.3 – Дані експерименту 1 отримані на реальному девайсі

avgFrameTimeMs	avgLatencyMs	ramUsageMb	widgetRebuilds	stdDevFrameTimeMs	percentile95FrameTimeMs	jankFramesCount	jankFramesPercent	minLatencyMs	maxLatencyMs
5,64	0,05	266,01	1675,00	8,73	18,28	26,00	1,55	0,00	3,00
12,07	0,07	283,04	1737,00	13,91	31,69	57,00	3,28	0,00	10,00
3,37	0,12	285,40	1711,00	3,92	9,38	0,00	0,00	0,00	6,00
2,82	0,02	272,84	1668,00	3,24	7,25	0,00	0,00	0,00	4,00
2,80	0,02	275,88	1681,00	3,14	7,32	0,00	0,00	0,00	2,00
5,06	0,08	282,79	1682,00	6,58	17,19	6,00	0,36	0,00	8,00
11,90	0,08	286,64	1751,00	13,55	30,76	53,00	3,03	0,00	6,00
3,17	0,03	286,43	1697,00	3,75	8,51	0,00	0,00	0,00	5,00
2,84	0,05	276,24	1666,00	3,32	7,43	1,00	0,06	0,00	2,00
2,95	0,03	279,39	1680,00	4,81	7,51	2,00	0,12	0,00	2,00
4,82	0,07	273,91	1664,00	6,38	16,54	5,00	0,30	0,00	6,00
11,11	0,11	287,32	1652,00	13,48	30,80	54,00	3,27	0,00	8,00
3,18	0,11	289,87	1709,00	3,61	8,44	0,00	0,00	0,00	4,00
2,87	0,11	279,41	1710,00	3,04	7,09	0,00	0,00	0,00	9,00
2,66	0,07	279,90	1688,00	2,88	6,81	0,00	0,00	0,00	2,00
5,35	0,04	275,35	1683,00	7,91	17,90	18,00	1,07	0,00	4,00
11,41	0,11	293,58	1698,00	13,50	30,16	45,00	2,65	0,00	9,00
3,25	0,09	292,76	1702,00	3,91	8,39	2,00	0,12	0,00	9,00
2,85	0,11	281,53	1669,00	3,22	7,26	0,00	0,00	0,00	2,00

avgFrameTimeMs	avgLatencyMs	ramUsageMb	widgetRebuilds	stdDevFrameTimeMs	percentile95FrameTimeMs	jankFramesCount	jankFramesPercent	minLatencyMs	maxLatencyMs
2,76	0,05	283,19	1675,00	3,14	7,22	0,00	0,00	0,00	3,00

Ці дані були проаналізовані та дали такі результати (таблиця 3.4):

Таблиця 3.4 – Аналіз даних першого експерименту

Metric	Value	Stateful	Riverpod	Provider	Bloc
FPS	Дисперсія	14437,66808	13172,271	13912,35274	14027,70487
	Оцінка	Висока	Висока	Висока	Висока
	Коеф варіації %	47,35593589	44,5413966	44,14641156	45,6932321
	Оцінка стабільності	Погана	Погана	Погана	Погана
Frame Time	Дисперсія	15,49791013	14,92892584	12,66331653	13,44983095
	Коеф варіації %	73,73847587	74,52107279	72,1894318	71,59944246
	Оцінка стабільності	Погана	Погана	Погана	Погана
StdDev Frame Time	Середнє	6,59	6,40	5,88	6,33
	Оцінка	Хорошо	Хорошо	Хорошо	Хорошо
Latency	Дисперсія	0,001597	0,0006523	0,0004922	0,0011077
Довірчі інтервали 95%	FPS ±	105,3201855	100,5989429	103,3863919	103,8141133
	Frame Time ±	3,450640044	3,386705002	3,11915402	3,214559744
	Latency ±	0,035028017	0,02238653	0,019446162	0,029172535

Ключовий критерій – стабільність (низька дисперсія, низький коефіцієнт варіації). Саме за цими показниками всі бібліотеки отримали оцінку «Погана».

- Високий коефіцієнт варіації (CV%) для FPS (44-47%) та Frame Time (71-74%):
  - Це означає, що цей показник демонструє, наскільки дані розсіяні відносно середнього значення. Значення понад 30-40% вважаються дуже високими для показників продуктивності;
  - Наслідок: Таке велике розсіювання означає нестабільну, трясучу роботу додатку. У певні моменти він може видавати ідеальні

60 FPS, а в інші – раптово падати до 30 FPS або нижче, що призводить до помітних «підвисань» (jank). Це прямо суперечить меті застосування стейт-менеджерів – організувати передбачуване та ефективне оновлення UI.

- Велика дисперсія Frame Time та широкі довірчі інтервали:
  – Frame Time ± 3.1-3.5 мс при середньому значенні близько 5-12 мс (з таблиць) – це дуже великий розкид. Це кількісне підтвердження нерівномірності рендерингу кадрів.
- Аналіз сирцевих даних (другі таблиці):
  – Чітко видно дві групи замірів: одна з низьким avgFrameTimeMs (~2.8-5.3 мс, що відповідає ~350-180 FPS) та друга з високим (~11-12 мс, що відповідає ~83-90 FPS). Це є прямим доказом неконсистентності;
  – Наявність jank-кадрів (особливо в «повільних» групах – до 57 jank-кадрів за тест, що становить 3.28%) підтверджує проблему з плавністю. Jank-кадр – це кадр, відмальований більше ніж за 33.33 мс (для 60 FPS), і він безпосередньо відчувається користувачем;
  – percentile95FrameTimeMs досягає 31.69 мс – це означає, що 5% найповільніших кадрів рендеряться за час, який вдвічі перевищує бюджет для 60 FPS. Це критично важливий показник для сприйняття плавності.

Висновок: Результати дивні. Їх дивність полягає не в абсолютно низькій середній швидкості (FPS високі), а в крайній нестабільності. Додаток працює «ривками», що є абсолютно неприйнятним для production-рішень.

Цей експеримент показав слабкі місця програми та сбору результатів, а саме:

- Програма працює нестабільно. Це означає що треба покращувати продуктивність додатку, краще працювати з ресурсами;
- Необхідно додати нові показники. Неможна порівнювати просто по середнім значенням. Потрібно отримати кожне точкове значення для трьох

найважливіших характеристик (кількість кадрів на секунду, час кадру, та затримка дії користувача). Необхідно розробити компактний спосіб збереження цих даних, та подальшої їх обробки.

3.4 Друге тестування

Покращення програми

Після проведення першого експерименту та отримання нечітких даних, було вирішено покращити програму, а саме внесені такі зміни:

- Класам, що відповідали за збір даних по кількості кадрів на секунду, швидкості кадру та затримки відклику інтерфейсу на дії користувача, необхідно збирати усі значення у виділений проміжок часу, для побудови по точкам графіків щільності розподілу.
- Також для цих показників було додано розрахунок дисперсії.
- Для показника використання оперативної пам’яті було додано відслідковування показника пікової пам’яті за сесію.
- Була покращена очистка пам’яті. До цього покращення показник, що відповідає за кількість перебудови віджетів нараховувався невірно, так як підпадав під дію процесів налагодження фреймворку. Також краща робота із пам’яттю краще вплинула на роботу додатку, що зменшило кількість підвисань додатку і стабілізувало роботу тестів.

Перша сесія:

Перша сесія була проведена по тому ж самому алгоритму (таблиця 3.1). Усі зміни відпрацювали добре, але експеримент було завершено за етапі проведення тестів на емуляторі. Причиною стало те, що кількість даних була занадто великою і дуже погано скомпонована. Усі точкові значення для кількості кадрів на секунду, швидкості кадру та затримки інтерфейсу призвели до неможливості їх обробляти. Тому одразу було прийняте рішення внести ще декілька покращень у процес збору та обробки даних:

- Необхідно змінити підхід для збору даних для побудови графіку. Для цього, замість збору кожного значення, був обраний метод частотних словників [11]. Дані у цьому методі збираються таким чином – кожне конкретне значення, якщо зустрічається уперше, стає одним із ключів цього словника, а при повторенні - просто інкрементуємо значення за цим ключем;
- Також необхідно розробити додаткове ПЗ для конвертації даних із .json у зрозумілий і структурований .xlsx формат.

Друга сесія

Ця сесія другого експерименту пройшла вдало. Використовуючи послідовність із таблиці 3.1, були проведені експерименти на емуляторі та реальному девайсі. До алгоритму проведення додалось 2 кінцеві кроки:

- Копіювання json даних у текстовий файл та конвертація у .json. У випадку із реальним девайсом необхідно було переносити інформацію на комп’ютер за допомогою хмарних сервісів;
- Обробка даних за допомогою спеціально розробленого python скрипта для переведення даних у ексель-формат для подальшого аналізу.

Після проведення експерименту та обробки даних було отримано декілька таблиць, включаючи таблиці середніх значень по кожному тестовому випадку для показників кількості кадрів на секунду, часу кадру та затримки інтерфейсу.

Оскільки експеримент був проведений по одному разу на емуляторі та реальному девайсі, велика розбіжність у даних (таблиці 3.5-3.6) здалася дуже дивною. Для порівняння також було створено діаграми (рис. 3.1 – 3.2):

Таблиця 3.5 – середні значення часу кадру (мс) на емуляторі

Scenario	bloc	provider	riverpod	stateful
API Test	2,558956	2,5783476	2,6981925	2,844963
Animation Test	5,329736	4,4707703	5,6319856	6,124715
Form Test	4,241514	3,8170996	4,3573711	4,266278
List Test	2,707842	2,7275803	2,7176079	2,832375
Subscription Test	2,645849	2,4922714	2,9389601	2,612155

Таблиця 3.6 – середні значення часу кадру (мс) на реальному девайсі

Scenario	bloc	provider	riverpod	stateful
API Test	2,558956	2,5783476	2,6981925	2,844963
Animation Test	5,329736	4,4707703	5,6319856	6,124715
Form Test	4,241514	3,8170996	4,3573711	4,266278
List Test	2,707842	2,7275803	2,7176079	2,832375
Subscription Test	2,645849	2,4922714	2,9389601	2,612155

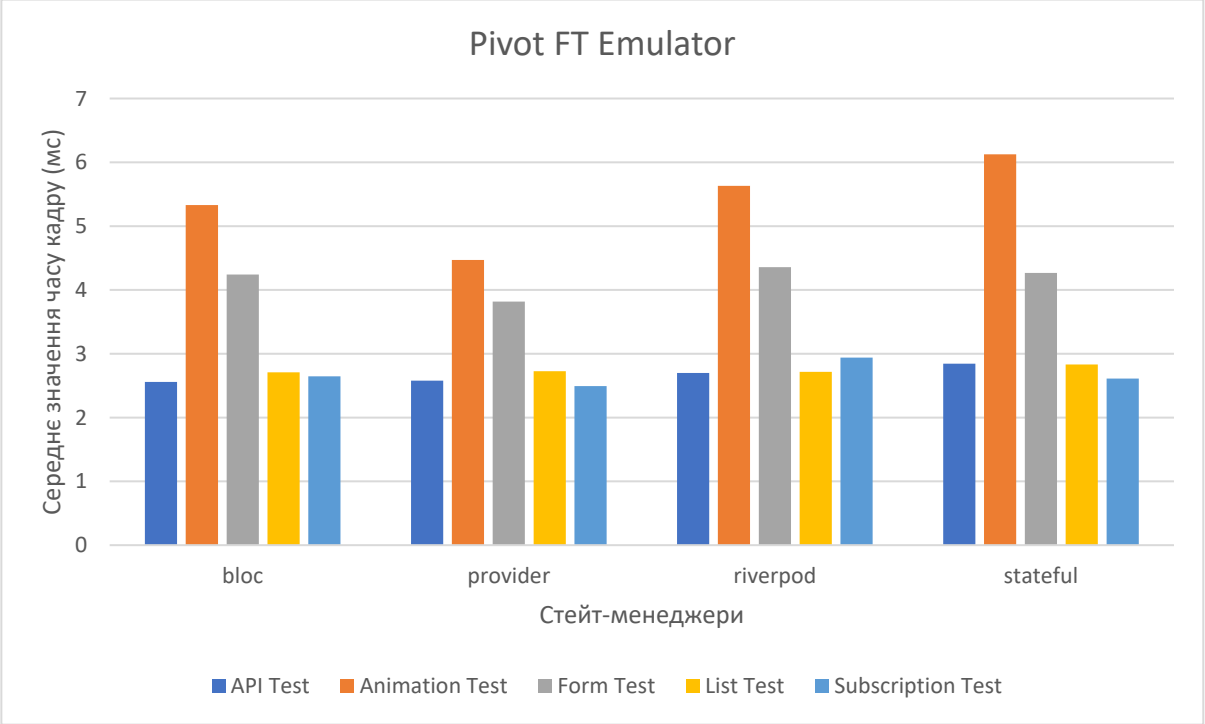

Рис. 3.1 – Діаграма середніх значень часу кадру на емуляторі

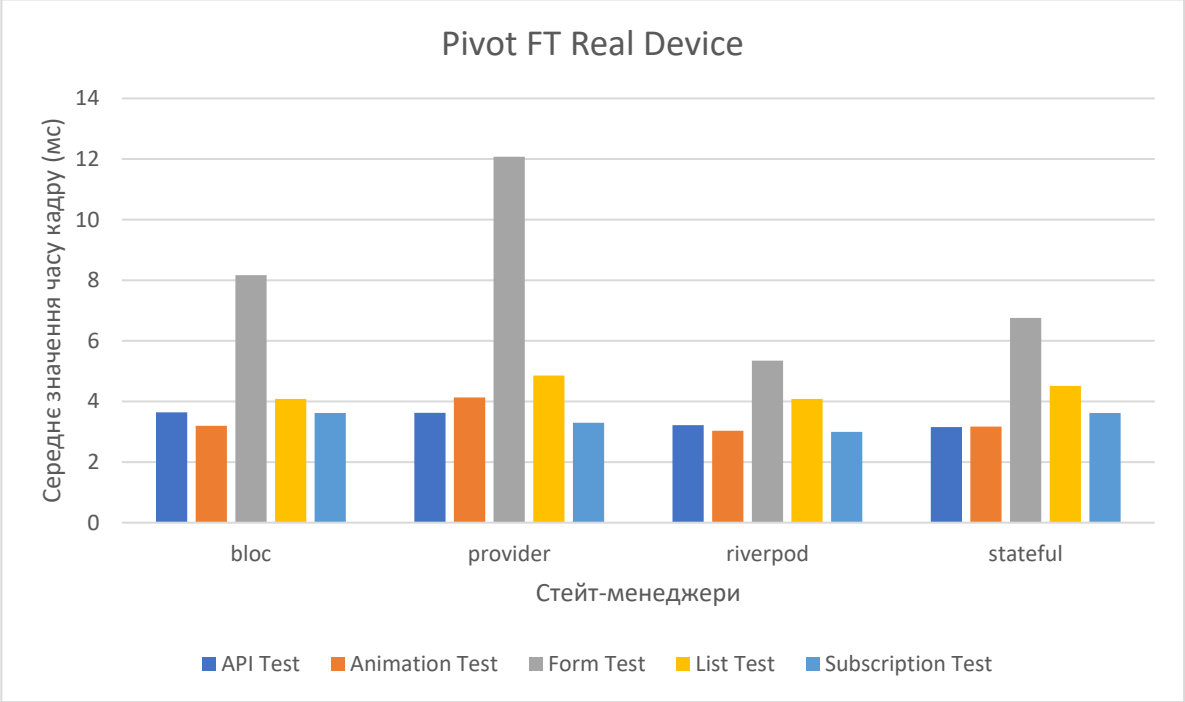

Рис. 3.2 – Діаграма середніх значень часу кадру на реальному девайсі

Така розбіжність зобов'язує перевірити дані на те, що вони не є випадковими. Така розбіжність може мати 3 причини:

- Різна поведінка програми на емуляторі та реальному девайсі. Це досить очевидна причина і тоді розбіжність не є помилкою, а є цікавим проявленням різниці в оптимізації різних підходів до контролю стану та їх взаємодії із зовнішніми процесами реального девайсу;
- Помилка у зборі даних. Збір даних не відображає реальної картини що може потребувати змін у способі збору даних;
- Нестабільність у роботі тестових випадків. Проблема не у способі збору даних, а у тому, що самі тести працюють не стабільно, і їх порівняння не є доцільним, бо є якийсь фактор, що дуже сильно впливає на роботу програми і може кардинально змінювати показники без видимої причини;
- Комбінація двох попередніх причин.

Для оцінки, наскільки дані є не випадковими, була проведена третя сесія експерименту.

Третя сесія

Очікувані результати:

План проведення цієї сесії будується на відсіканні помилок по черзі. Першим експериментом має бути проведення тестів на одному девайсі на вибір, із видаленням програми після проведення першої ітерації експерименту для очищення кешу на інструментів налагодження малювання UI фреймворку. Якщо ці дані не будуть мати кардинальних відмінностей, а будуть в одному діапазоні, це дозволить зробити висновки, що різниця є на рівні девайсів, а не на рівні одного девайсу, що не буде помилкою, а буде свідчити про різницю у роботі між реальним та емульованим середовищем.

Був повністю повторений алгоритм другої сесії, але для проведення двох послідовних експериментів, із видаленням програми між ними, було обрано 1 емулятор. Результати наведені у таблицях 3.7 – 3.10 та на відповідних діаграмах (рис. 3.3 – 3.6).

Таблиця 3.7 – середні значення FPS на емуляторі тест 1

Scenario	bloc	provider	riverpod	stateful
API Test	365,0188	393,38504	307,64185	397,2012
Animation Test	186,5186	255,84843	180,86571	217,6179
Form Test	312,3086	273,94949	301,66574	271,3058
List Test	380,2323	405,9845	370,76663	207,379
Subscription Test	396,8417	418,31499	337,32765	388,0253

Таблиця 3.8 – середні значення FPS на емуляторі тест 2

Scenario	bloc	provider	riverpod	stateful
API Test	397,633	331,45047	357,47586	419,084
Animation Test	181,211	170,30218	184,52722	168,6285
Form Test	254,3775	282,90743	248,71174	254,4074
List Test	363,642	329,52565	380,36656	319,5663
Subscription Test	409,584	374,36984	317,12036	355,5756

Таблиця 3.9 – середні значення часу кадру (мс) на емуляторі тест 1

Scenario	bloc	provider	riverpod	stateful
API Test	2,739585	2,5420387	3,250533	2,517616

Scenario	bloc	provider	riverpod	stateful
Animation Test	5,361395	3,9085642	5,5289641	4,595211
Form Test	3,201961	3,650308	3,3149273	3,685878
List Test	2,629971	2,4631482	2,6971143	4,822088
Subscription Test	2,519896	2,3905431	2,9644769	2,577152

Таблиця 3.10 – середні значення часу кадру (мс) на емуляторі тест 2

Scenario	bloc	provider	riverpod	stateful
API Test	2,514882	3,017042	2,7973917	2,386156
Animation Test	5,51843	5,8719155	5,4192546	5,930196
Form Test	3,931165	3,5347251	4,0207189	3,930703
List Test	2,749958	3,0346651	2,6290429	3,129241
Subscription Test	2,441502	2,6711554	3,1533768	2,812341

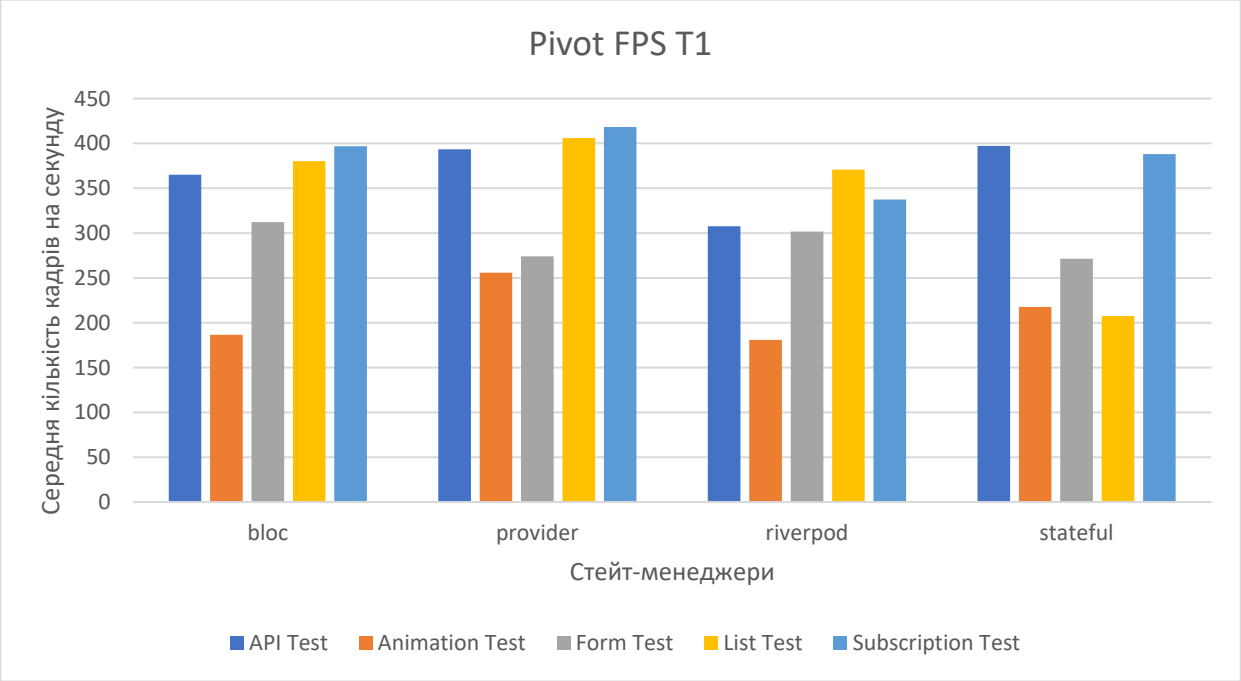

Рис. 3.3 – Діаграма середніх значень FPS на емуляторі тест 1

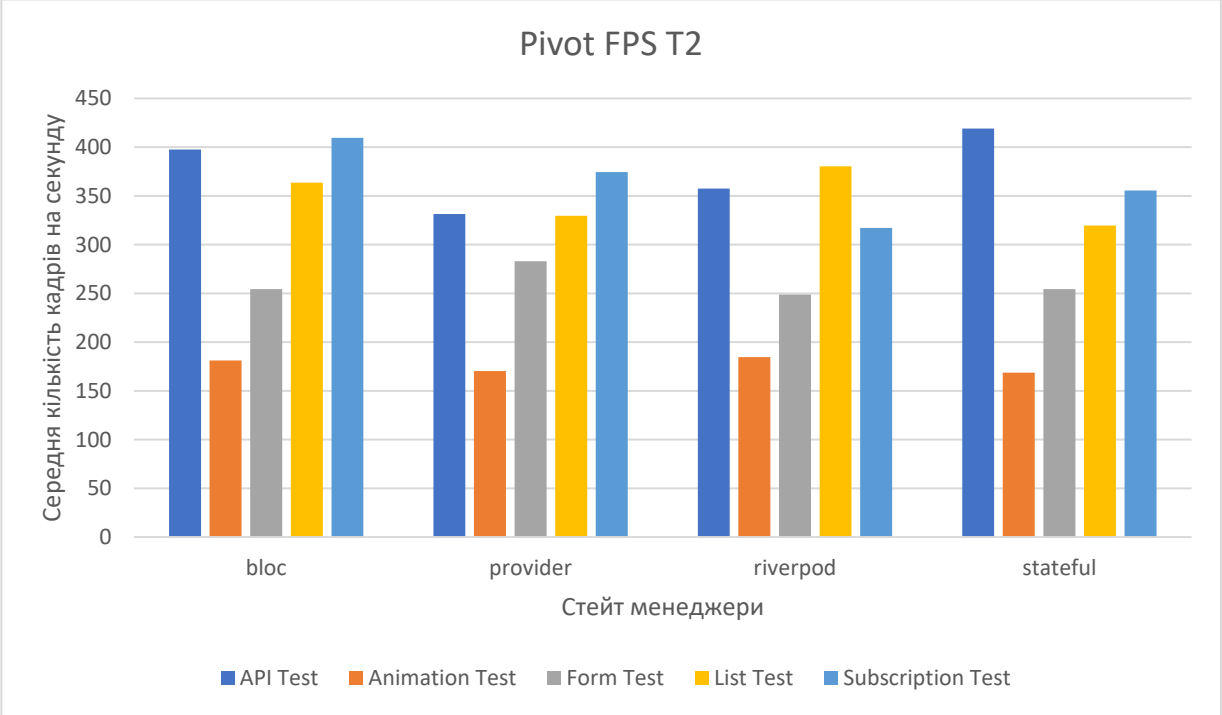

Рис. 3.4 – Діаграма середніх значень FPS на емуляторі тест 2

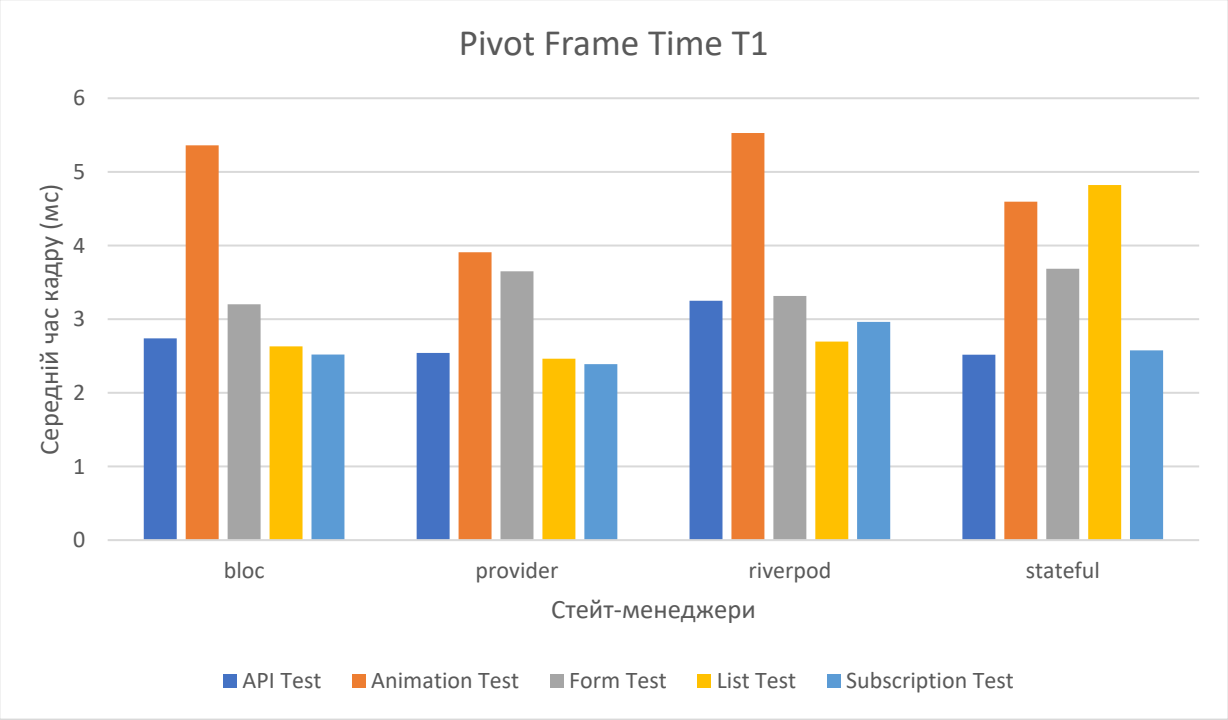

Рис. 3.5 – Діаграма середніх значень часу кадру на емуляторі тест 1

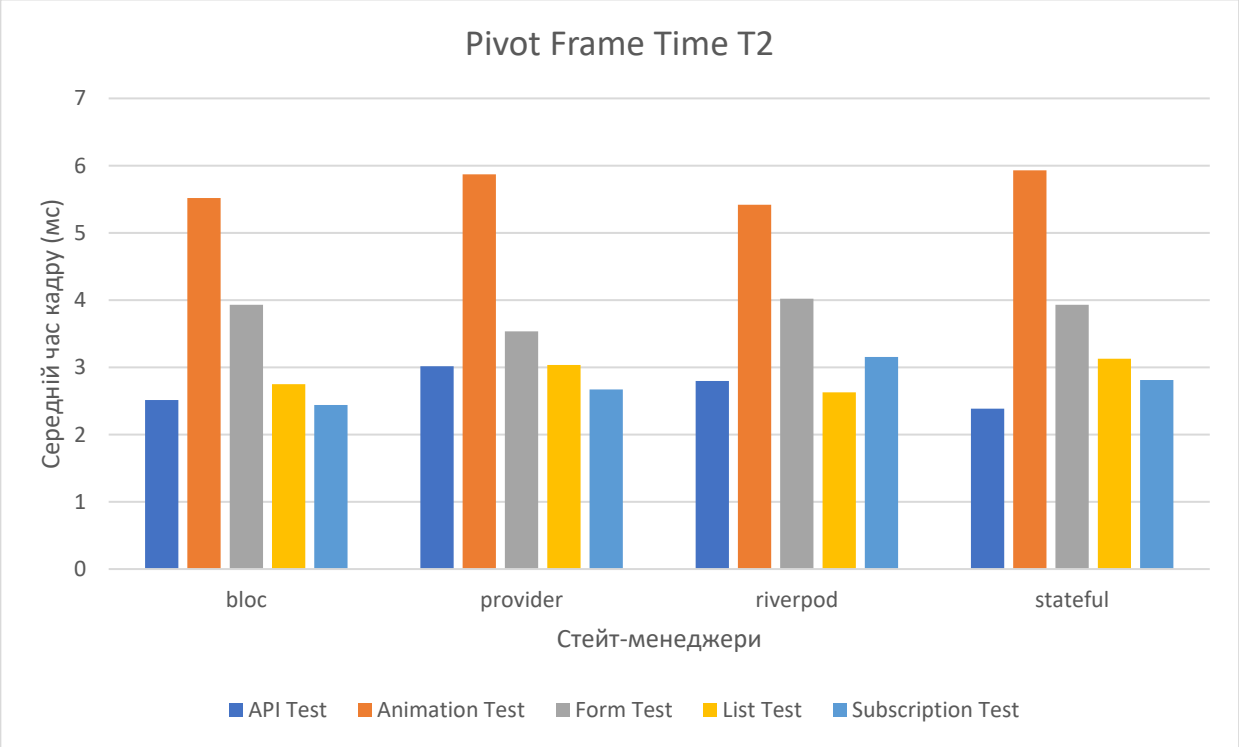

Рис. 3.6 – Діаграма середніх значень часу кадру на емуляторі тест 2

На основі проведеного контрольного експерименту можна зробити низку науково обґрунтованих висновків щодо природи розбіжностей між результатами на емуляторі та реальному девайсі.

- Отримані результати демонструють високу внутрішню узгодженість (internal consistency) у межах одного середовища виконання;
- Показники знаходяться в одному статистичному діапазоні, що свідчить про відтворюваність (reproducibility) експерименту;
- Відсутність статистично значущих розбіжностей між повторними вимірами підтверджує надійність (reliability) методики збору даних.

Згідно з принципом **відкидання гіпотез** (hypothesis rejection) у науковому методі, отримані дані дозволяють послідовно відкинути альтернативні пояснення:

Гіпотеза 1: Помилка у зборі даних

- **Передбачення:** Якщо розбіжності обумовлені артефактами вимірювання, то повторні експерименти в ідентичних умовах мали б демонструвати суттєву варіативність;
- **Спостереження:** Результати третьої серії показують високу консистентність;
- **Висновок:** Гіпотезу 1 можна відкинути з високою ймовірністю.

Гіпотеза 2: Нестабільність тестових випадків

- **Передбачення:** Якщо тестові сценарії мають стохастичну природу, повторні виконання мали б продукувати різнорідні результати;
- **Спостереження:** Подвійне виконання на одному емуляторі дало статистично еквівалентні результати;
- **Висновок:** Гіпотезу H_2 можна відкинути.

Залишається єдина неспростовна гіпотеза:

Гіпотеза 0: **Різна поведінка програми на емуляторі та реальному девайсі є справжньою та обумовленою фундаментальними відмінностями середовищ виконання.**

Ця гіпотеза узгоджується із відомою у комп'ютерних науках концепцією середовищної залежності продуктивності програмного забезпечення.

Отже далі можна переходити до обробки отриманих даних та порівняння роботи способів управління станом у кожному із тестових випадків та у різних тестових середовищах.

Висновок до розділу 3

Проведення експериментальної частини дослідження за інкрементальною методологією довело свою ефективність. Кожна з трьох послідовних сесій дозволила вдосконалювати як технічну інфраструктуру (програмне забезпечення для тестування та обробки даних), так і наукову методологію проведення вимірів. Початкове тестування виявило критичні недоліки у системі збору даних,

зокрема надмірну варіативність результатів та недостатню деталізацію показників. Це призвело до суттєвих покращень у програмному забезпеченні, зокрема, впровадження методу частотних словників для ефективного зберігання розподілів та розробки спеціалізованих Python-скриптів для автоматизованої обробки даних.

Найважливішим методологічним досягненням стала валідація отриманих результатів через контрольний експеримент третьої сесії. Порівняння двох незалежних вимірювань на одному емуляторі з повним скиданням стану між ними підтвердило високу відтворюваність та надійність системи вимірювань. Статистично незначні розбіжності між повторними тестами дозволили відкинути альтернативні гіпотези про помилки у зборі даних або нестабільність тестових сценаріїв. Таким чином, було науково обґрунтовано, що спостережувані суттєві відмінності між результатами на емуляторі та реальному пристрої є справжнім ефектом, зумовленим фундаментальними відмінностями цих середовищ виконання.

Отримані на цьому етапі дані створюють міцну основу для подальшого статистичного аналізу. Вони характеризуються високою внутрішньою узгодженістю, детальністю (включаючи повні розподіли ключових метрик продуктивності) та надійністю, що забезпечує їхню придатність для формування наукових висновків щодо порівняльної ефективності різних підходів до управління станом у Flutter-додатках.

РОЗДІЛ 4. ОБРОБКА ОТРИМАНИХ ДАНИХ

4.1 Первинний огляд даних

Після проведення експериментів, дані, що були отримані у результаті, були скомбіновані у таблиці, які дозволяють їх подальшу обробку. Основними показниками для аналізу будуть: кількість кадрів на секунду (FPS), час кадру (Frame Time) та затримка у відгуку UI (Latency). Це зумовлено тим, що саме ці показники є мірою якості роботи інтерфейсу користувача, і саме на них користувач у першу чергу зверне увагу.

Стовпчики мають таке значення:

- **Scenario** — тестовий сценарій (Animation, API, Form, List, Subscription), що відображає конкретний тип навантаження інтерфейсу;
- **State Manager** — підхід до управління станом (stateful, provider, bloc, riverpod);
- **Metric_Type** — тип метрики;
- **Bucket** — верхня межа інтервалу гістограми;
- **Count** — кількість кадрів, значення яких потрапили в даний інтервал;
- **Density** — відносна частка таких кадрів від загальної кількості вимірів.

Для наочності по таблицям були побудовані діаграми розподілів (рис. 4.1 – 4.20).

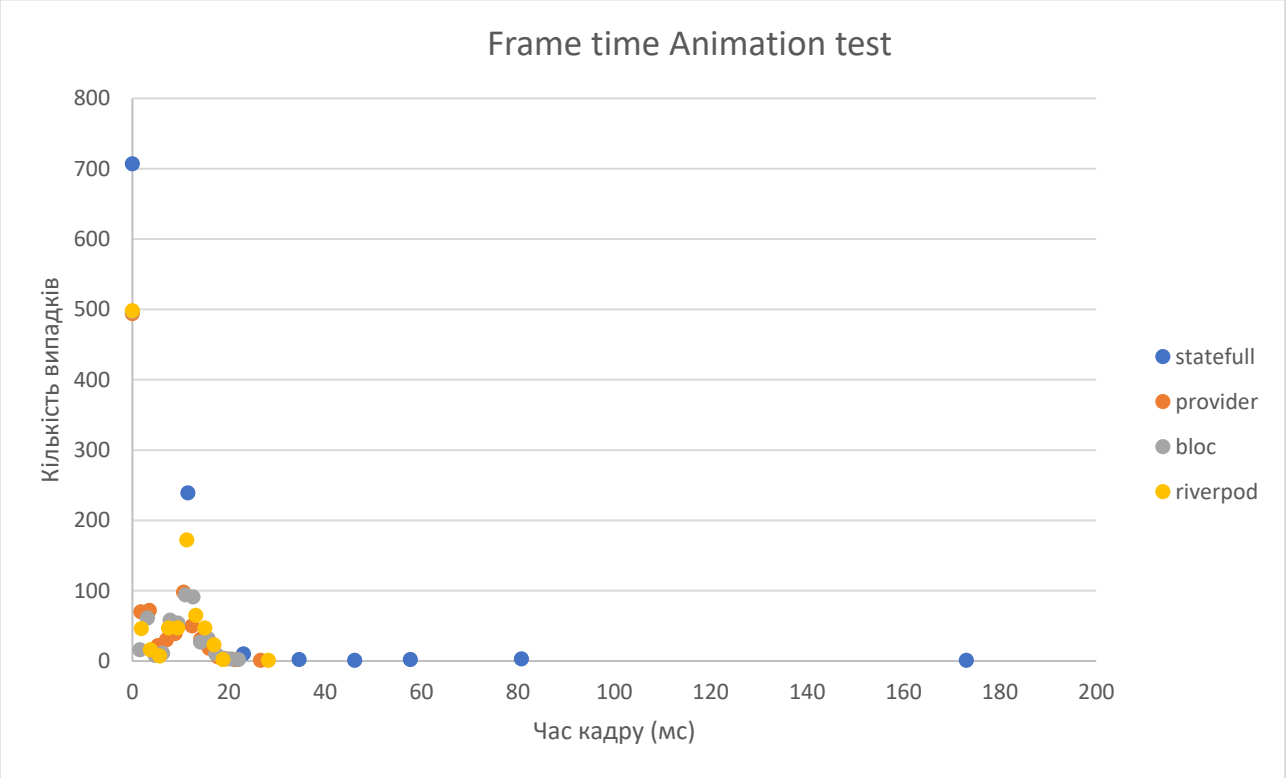

Рис. 4.1 – Діаграма розподілу Frame Time для Animation Test на емуляторі

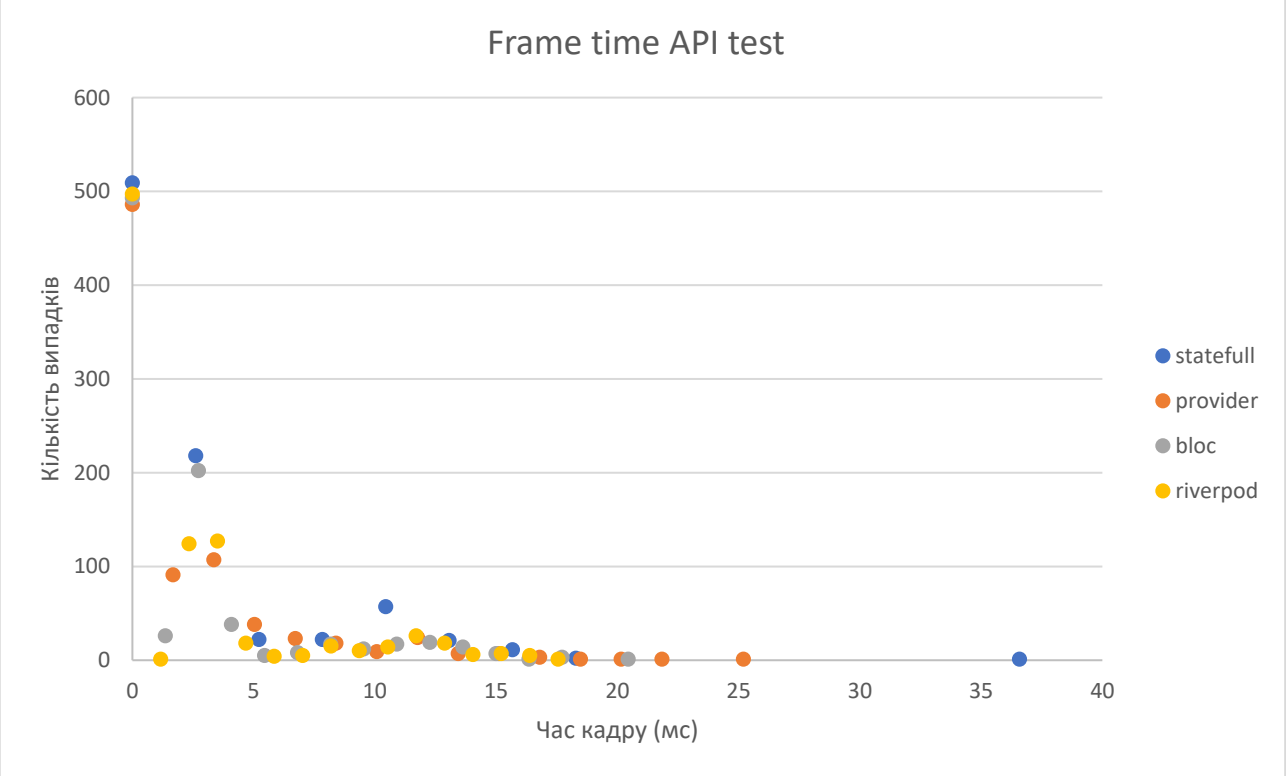

Рис. 4.2 – Діаграма розподілу Frame Time для API Test на емуляторі

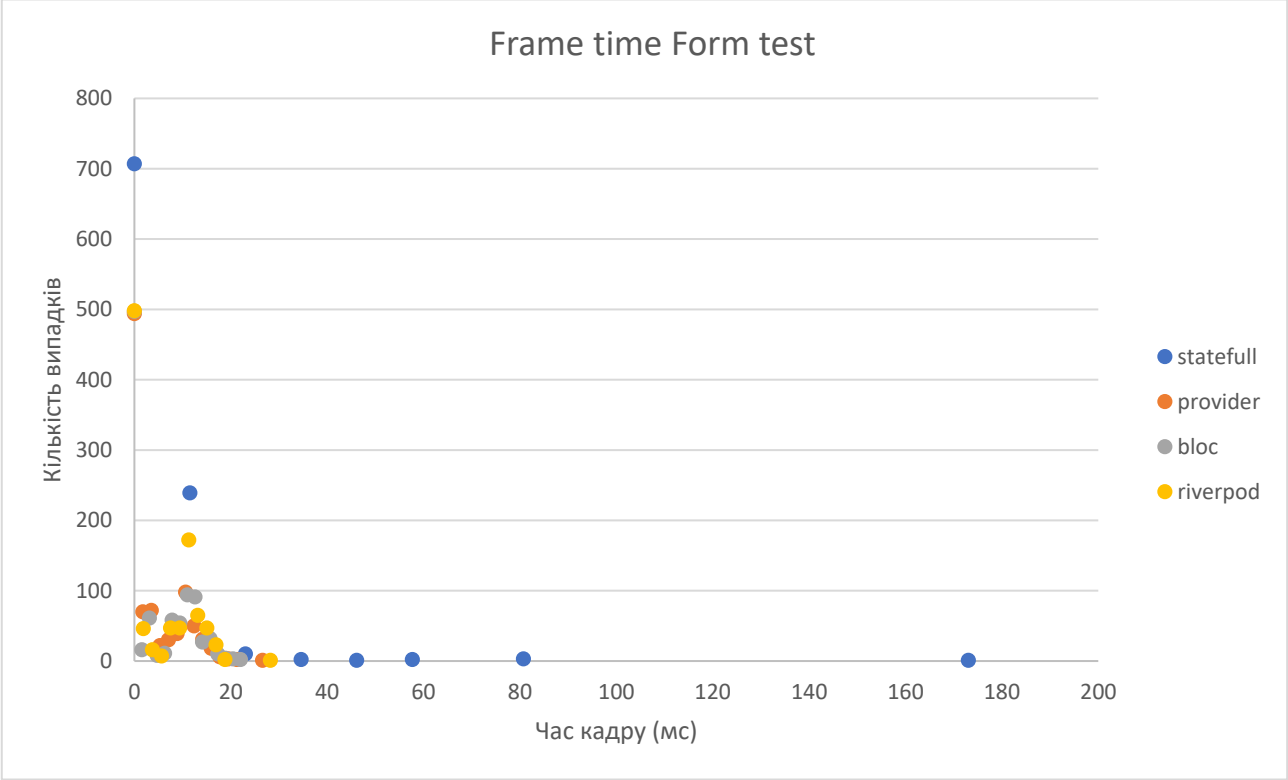

Рис. 4.3 – Діаграма розподілу Frame Time для Form Test на емуляторі

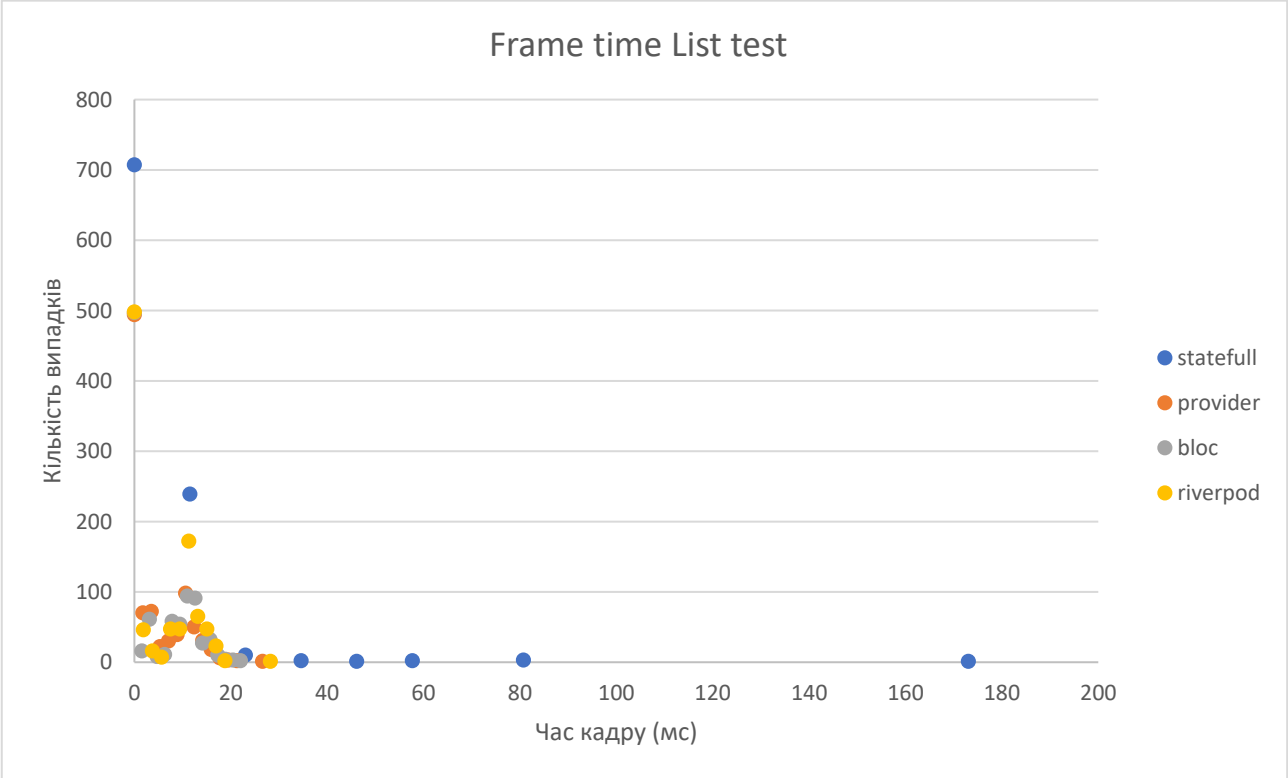

Рис. 4.4 – Діаграма розподілу Frame Time для List Test на емуляторі

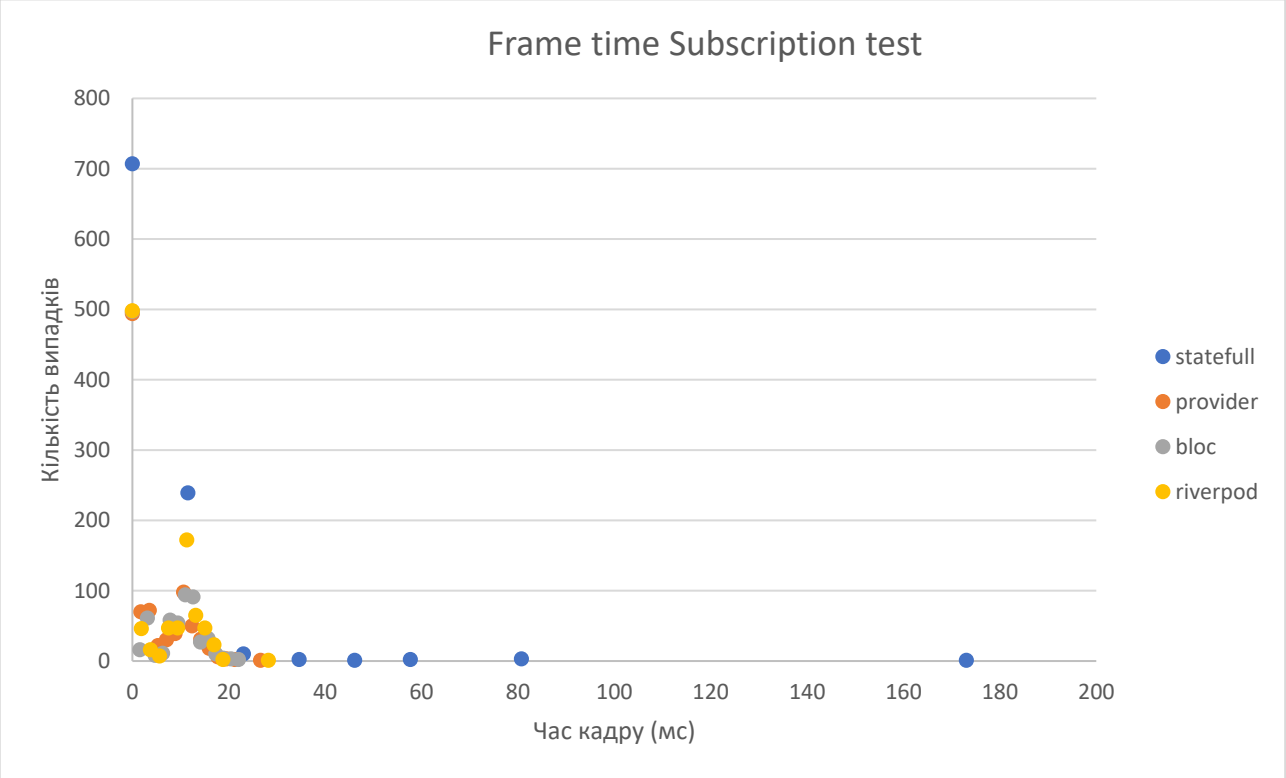

Рис. 4.5 – Діаграма розподілу Frame Time для Subscription Test на емуляторі

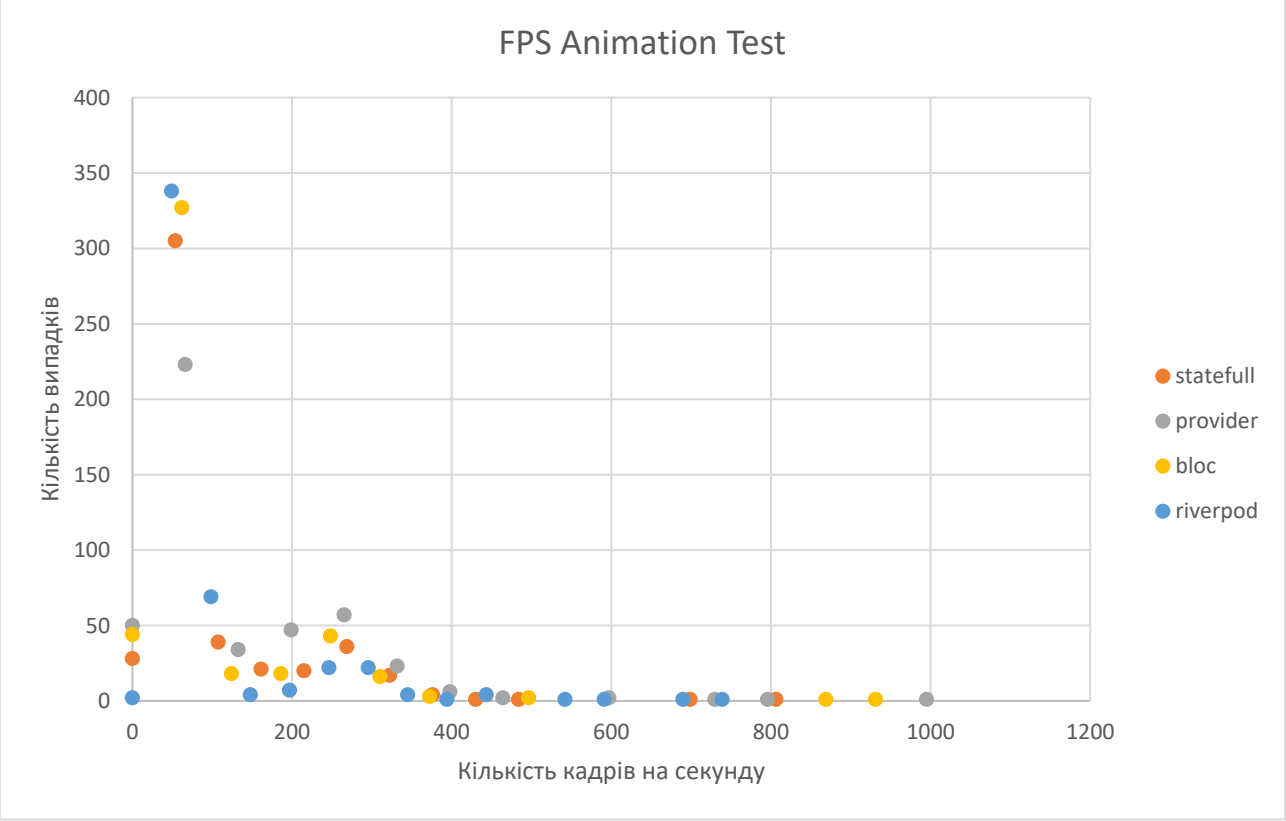

Рис. 4.6 – Діаграма розподілу FPS для Animation Test на емуляторі

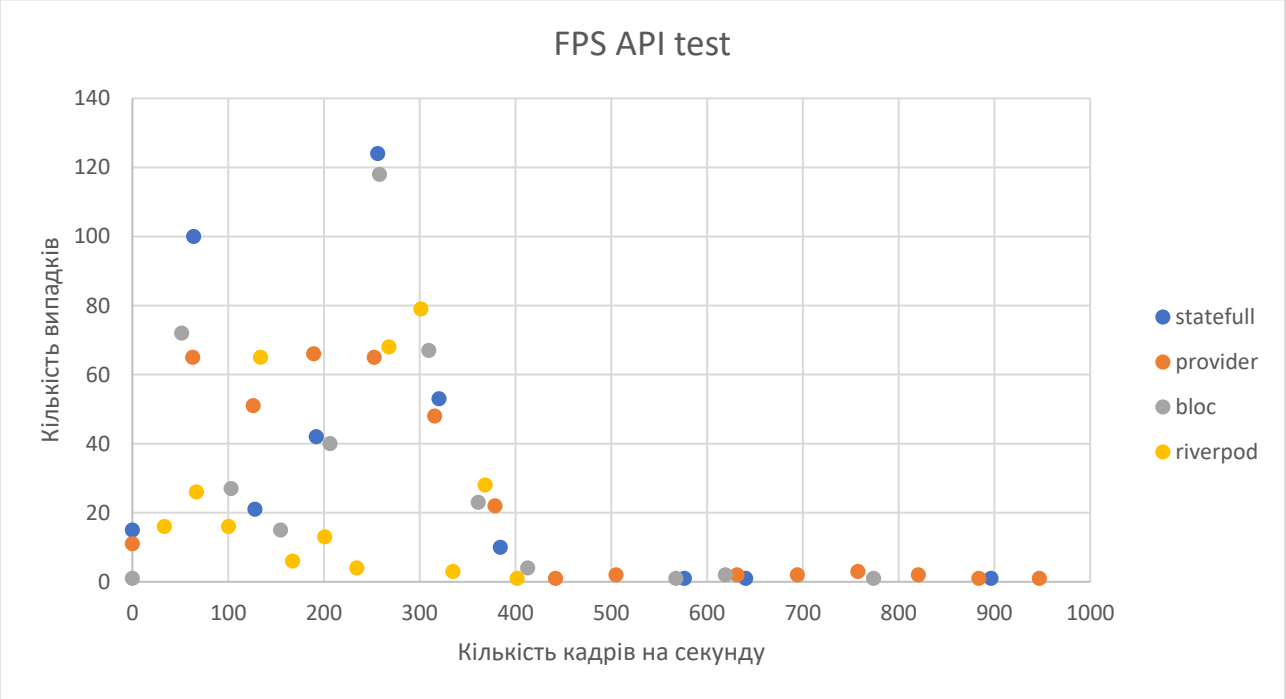

Рис. 4.7 – Діаграма розподілу FPS для API Test на емуляторі

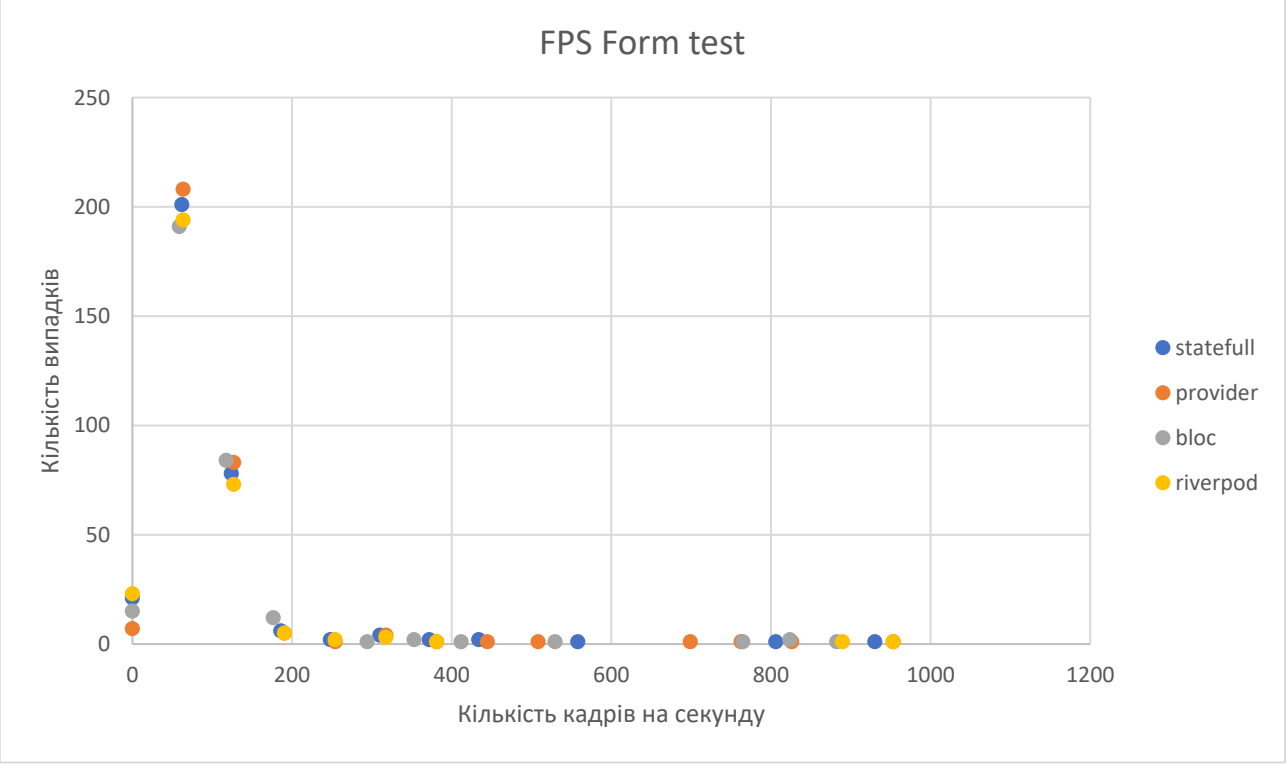

Рис. 4.8 – Діаграма розподілу FPS для Form Test на емуляторі

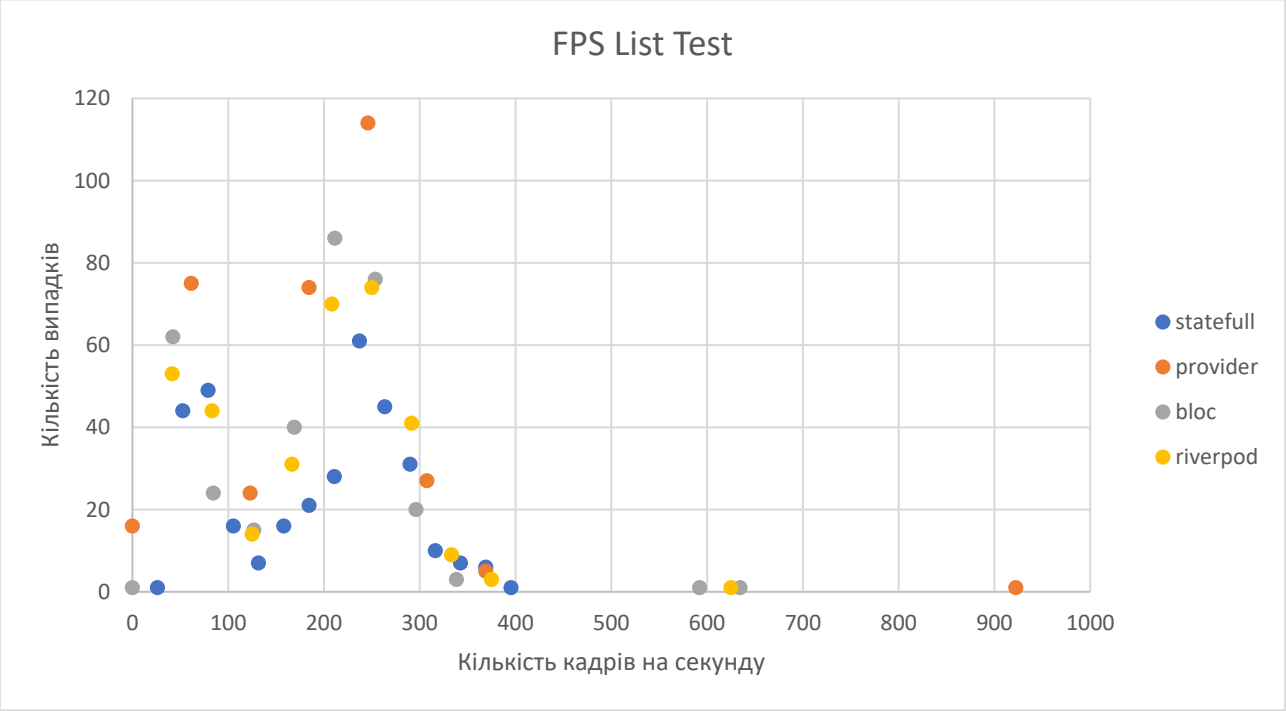

Рис. 4.9 – Діаграма розподілу FPS для List Test на емуляторі

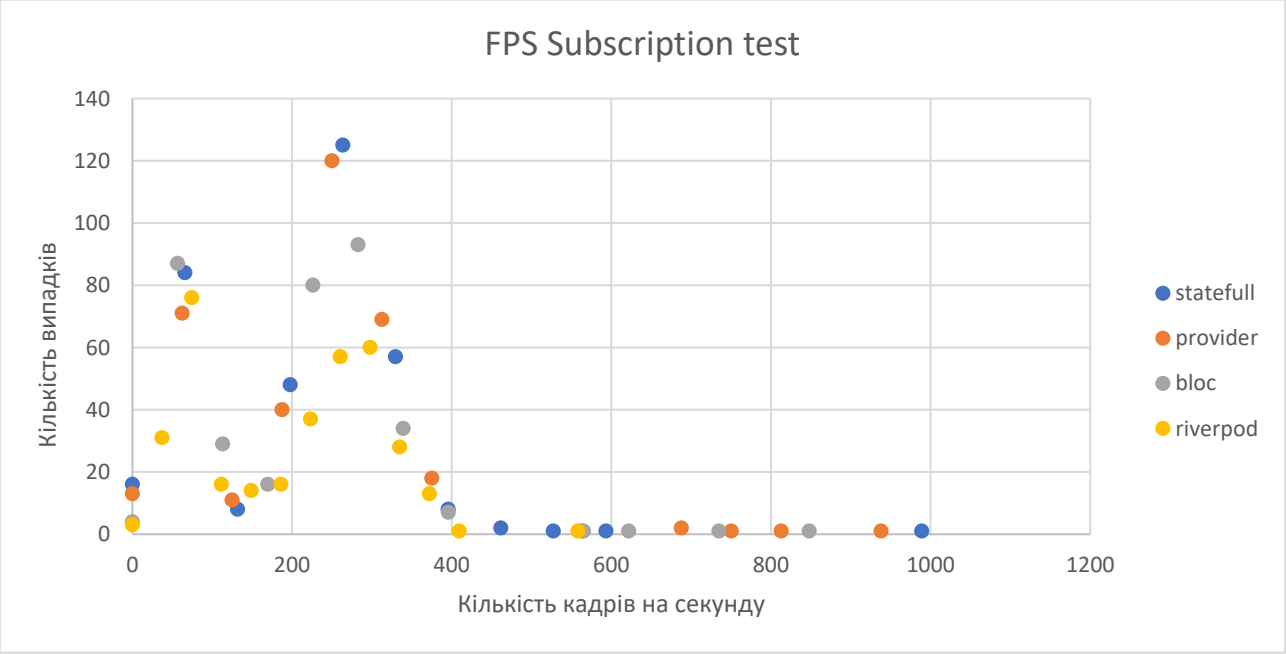

Рис. 4.10 – Діаграма розподілу FPS для Subscription Test на емуляторі

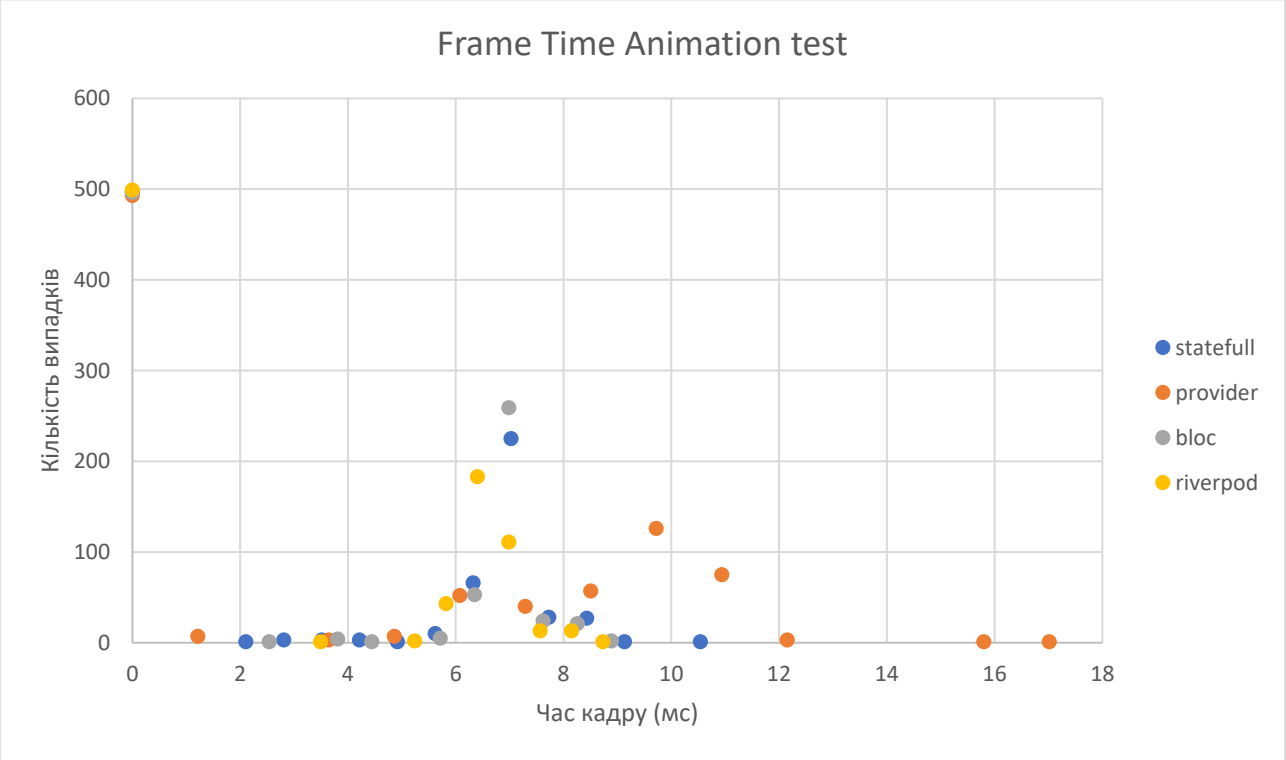

Рис. 4.11 – Діаграма розподілу Frame Time для Animation Test на реальному девайсі

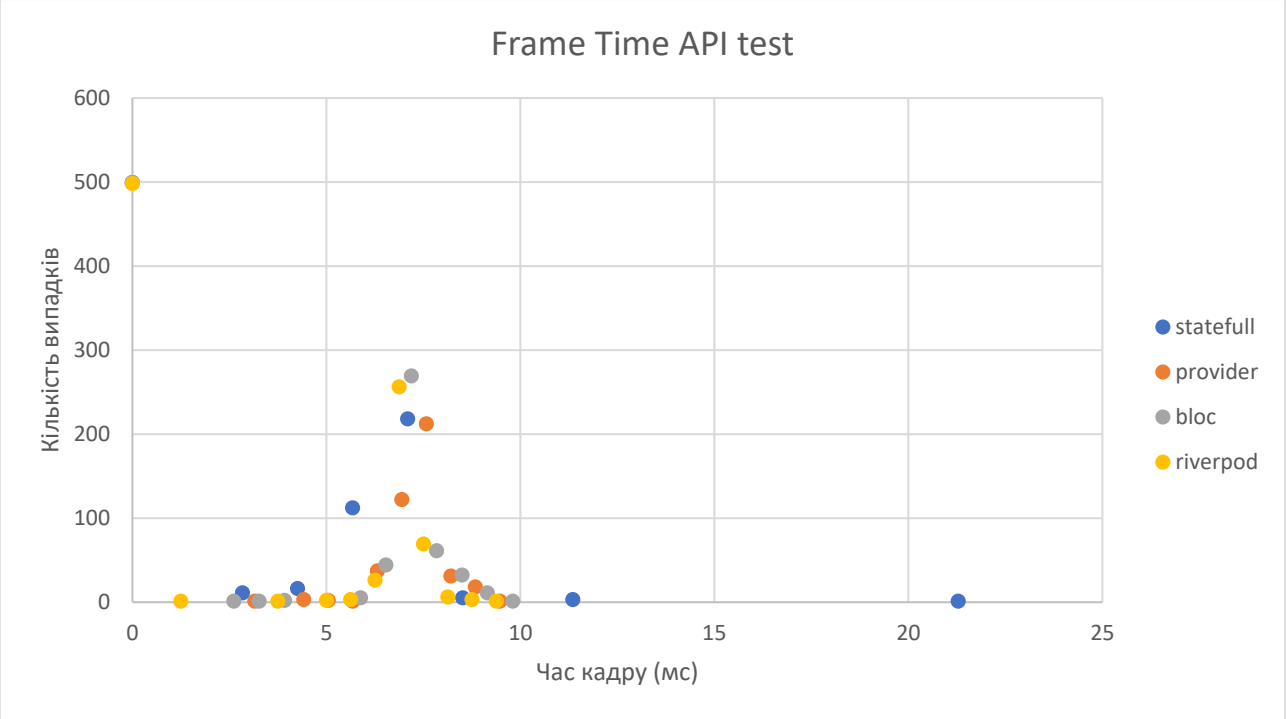

Рис. 4.12 – Діаграма розподілу Frame Time для API Test на реальному девайсі

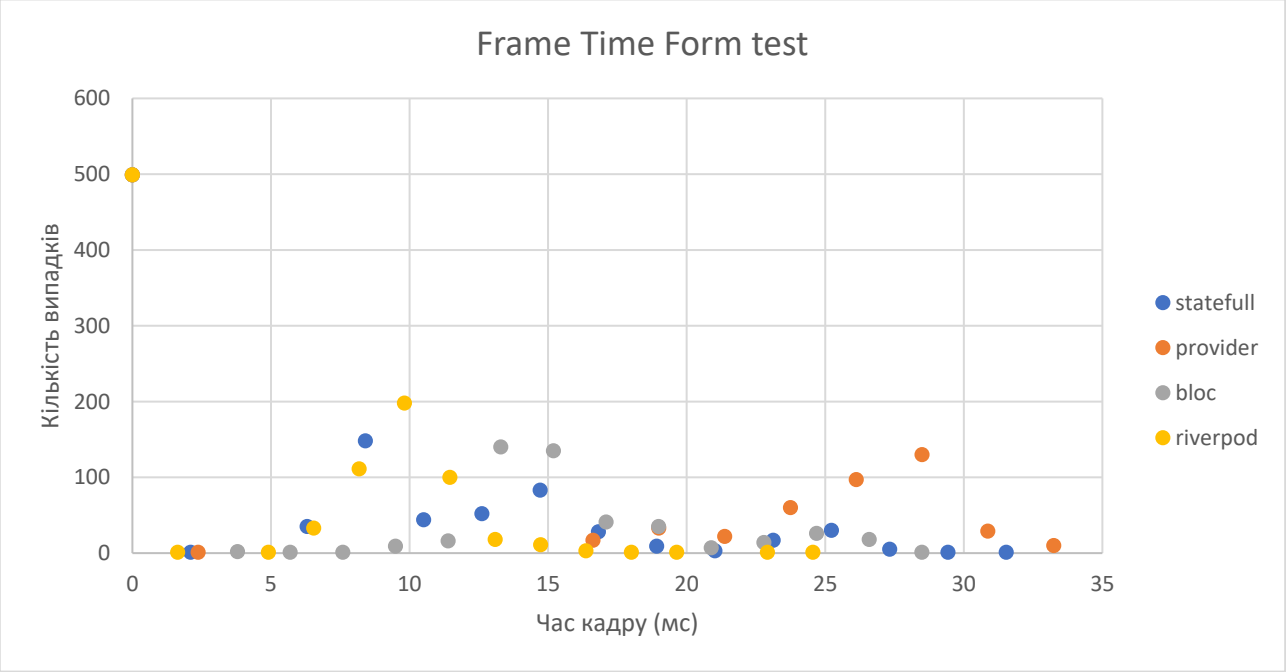

Рис. 4.13 – Діаграма розподілу Frame Time для Form Test на реальному девайсі

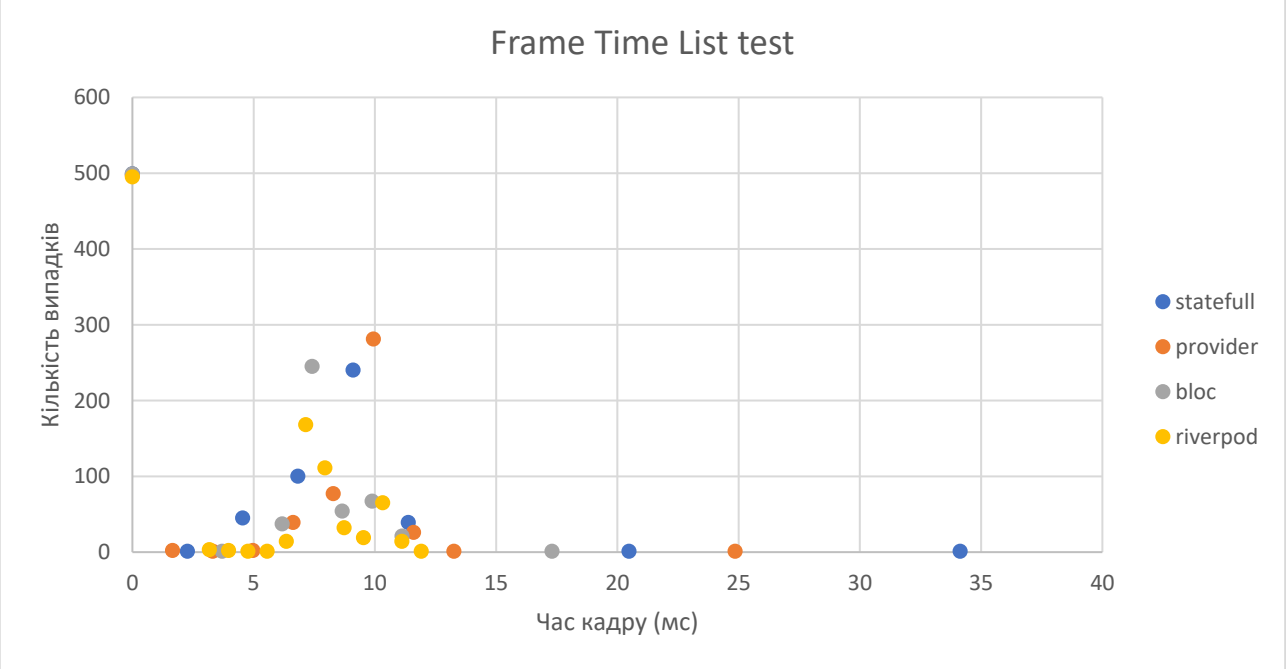

Рис. 4.14 – Діаграма розподілу Frame Time для List Test на реальному девайсі

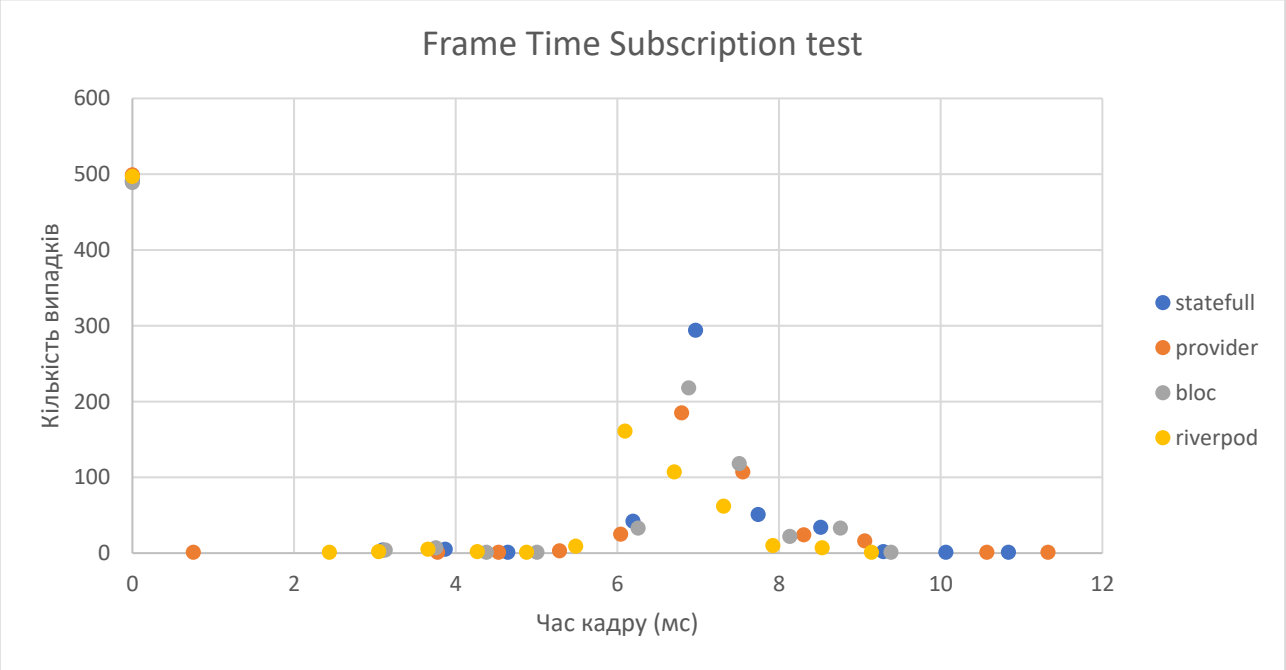

Рис. 4.15 – Діаграма розподілу Frame Time для Subscription Test на реальному девайсі

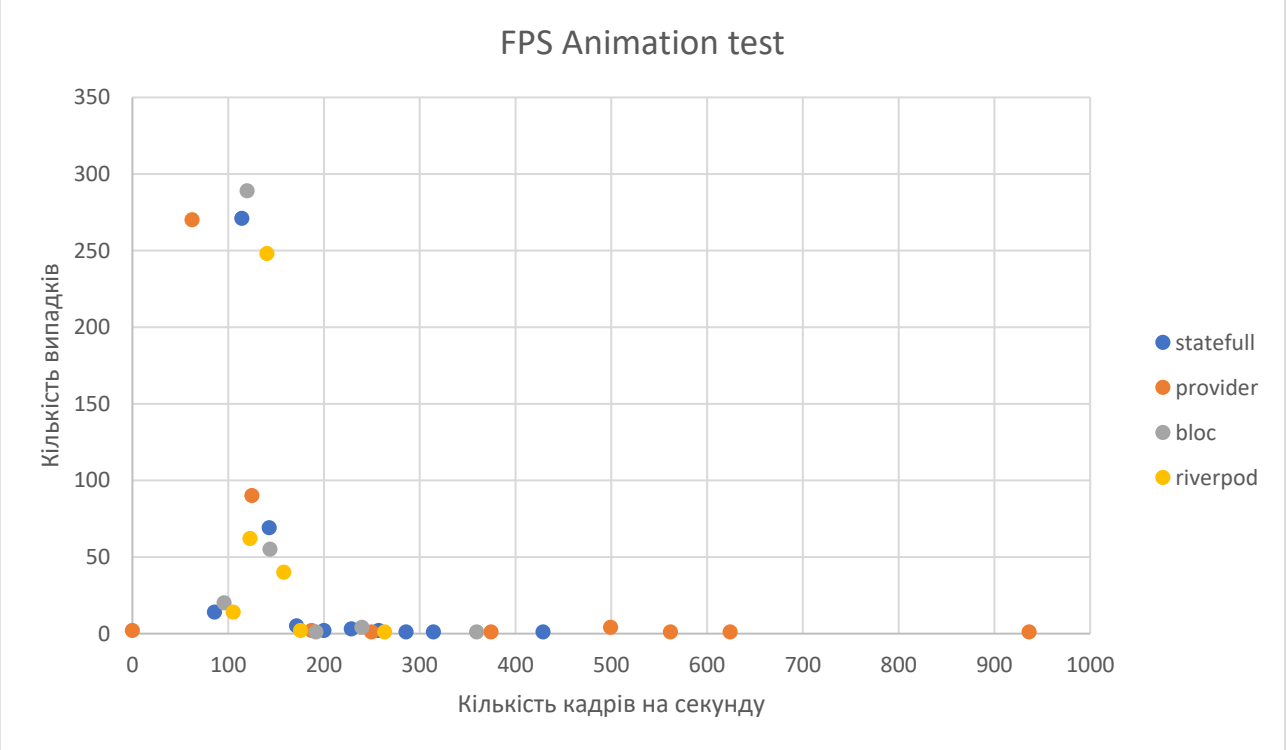

Рис. 4.16 – Діаграма розподілу FPS для Animation Test на реальному девайсі

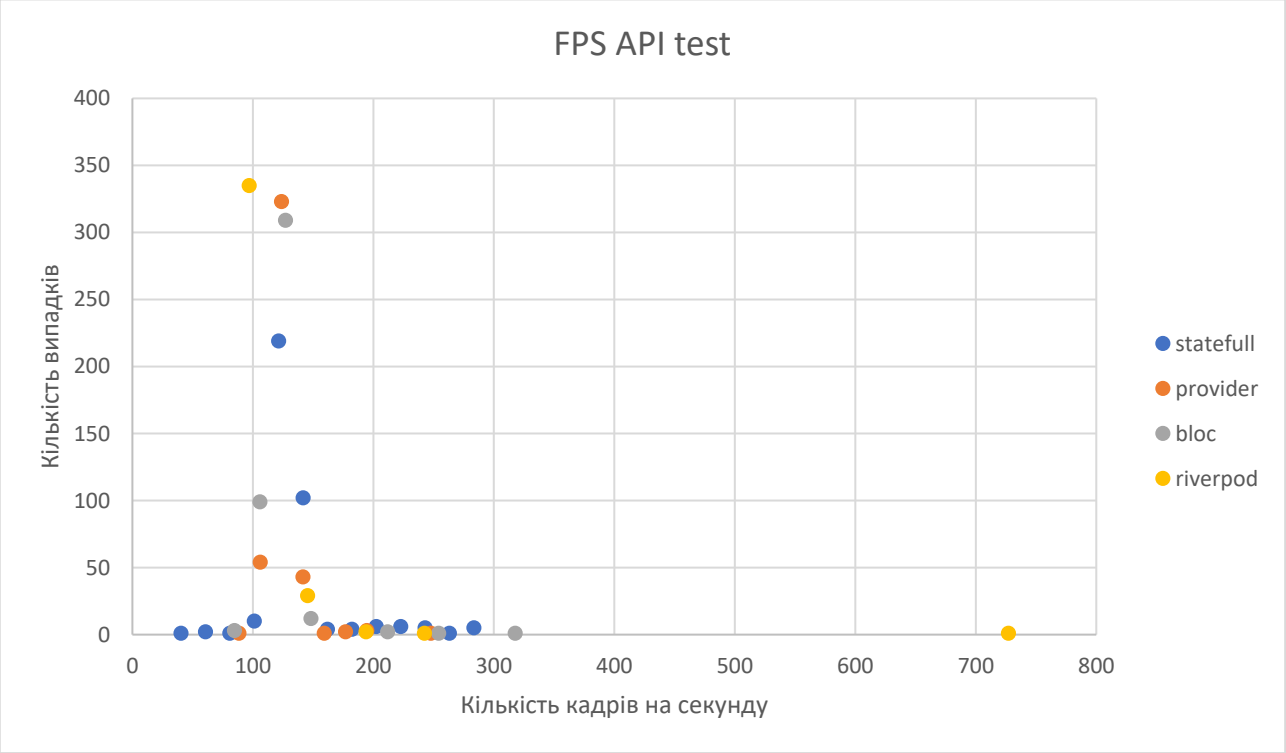

Рис. 4.17 – Діаграма розподілу FPS для API Test на реальному девайсі

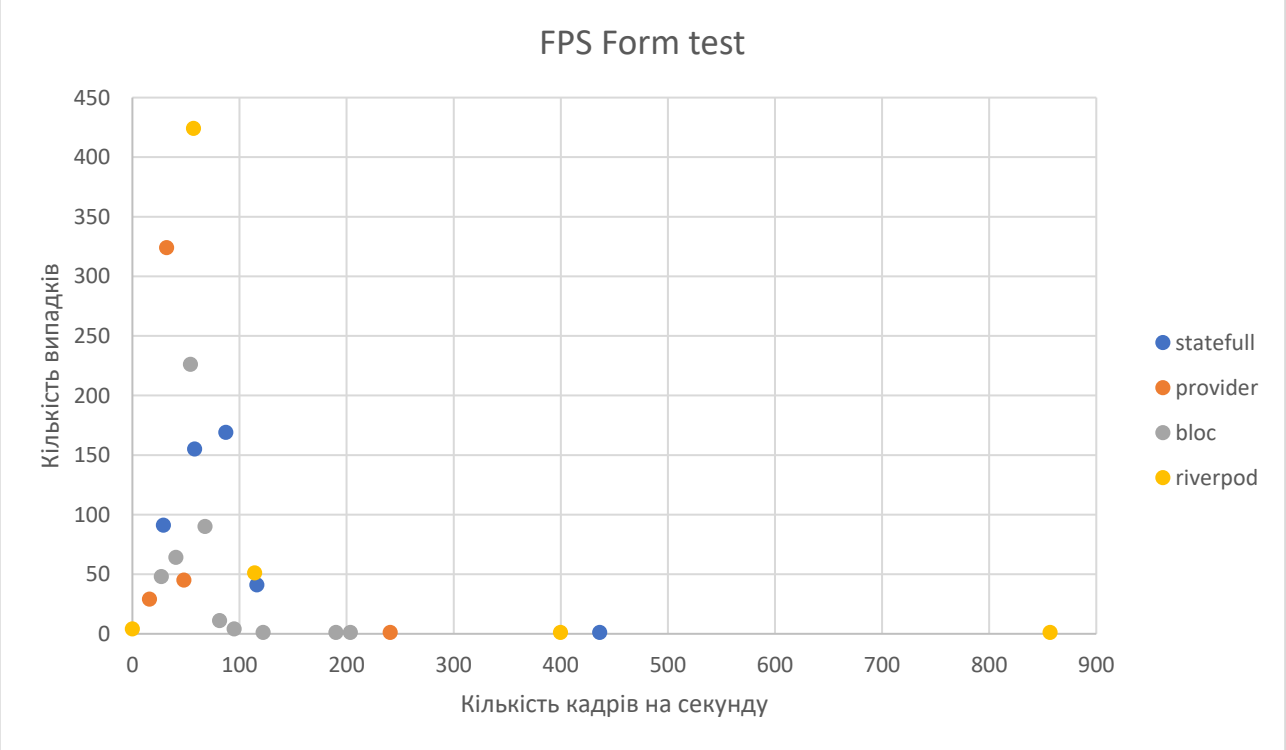

Рис. 4.18 – Діаграма розподілу FPS для Form Test на реальному девайсі

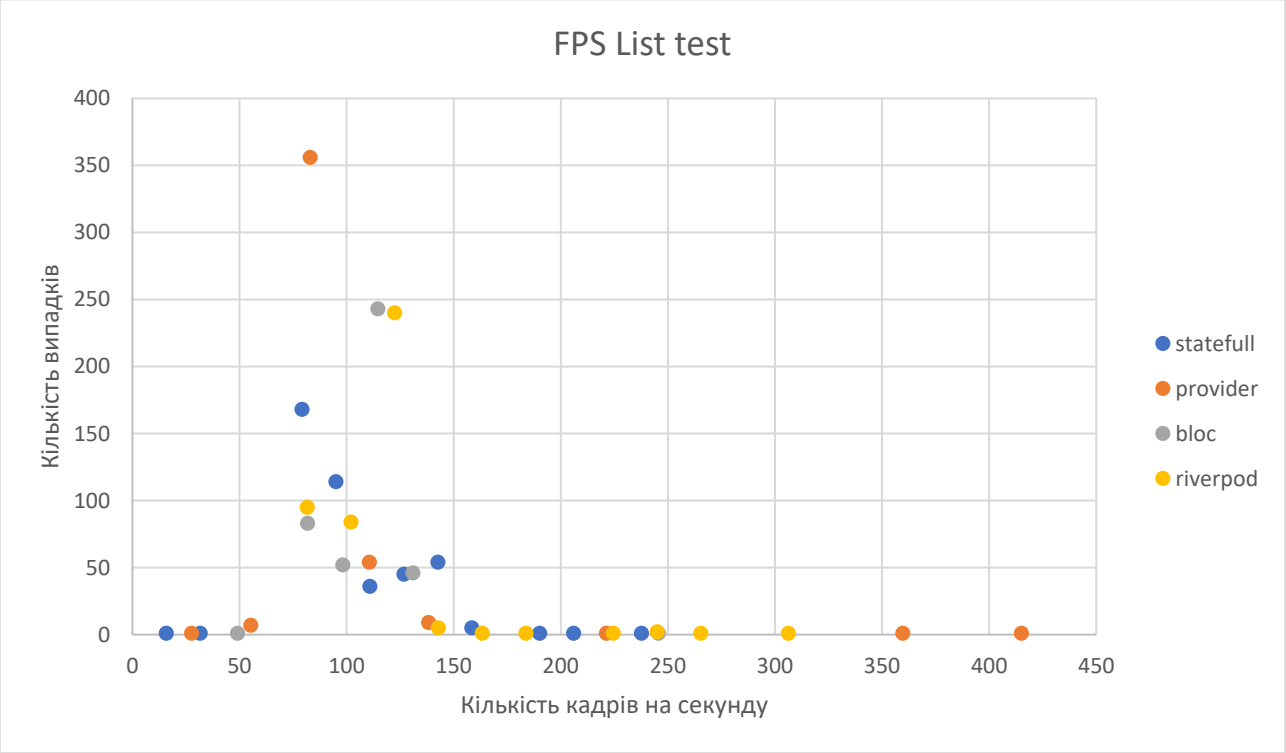

Рис. 4.19 – Діаграма розподілу FPS для List Test на реальному девайсі

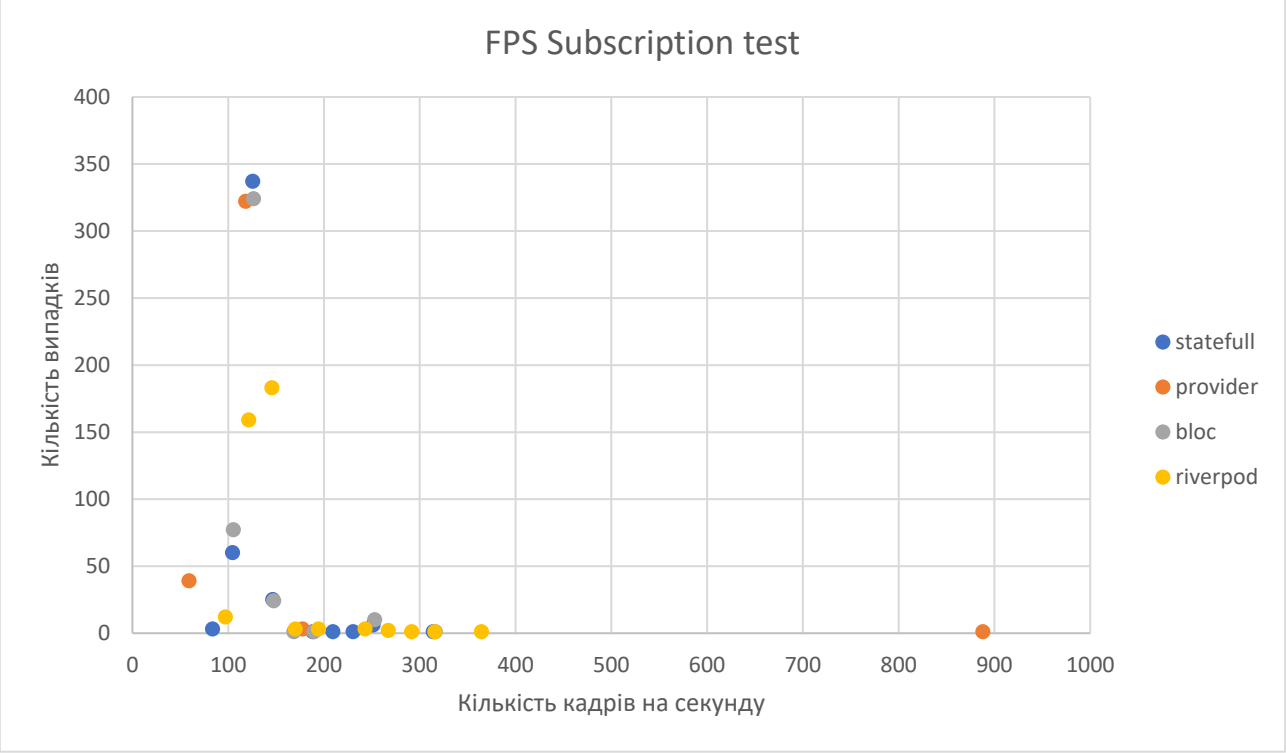

Рис. 4.20 – Діаграма розподілу FPS для Subscription Test на реальному девайсі

Загальні спостереження є такими:

Домінування нульового бакету

У всіх сценаріях та для всіх state manager спостерігається великий бакет із Bucket = 0, частка якого зазвичай складає:

- ~50–60% кадрів для frameTimeDistribution;
- ~4–11% кадрів для fpsDistribution.

Це означає, що значна частина кадрів:

- або відмальовується швидше за мінімальний крок гістограми;
- або агрегується в нульовий бакет через внутрішню квантизацію інструменту вимірювання.

Основна маса кадрів зосереджена в "робочому" діапазоні

Для frameTimeDistribution на емуляторі:

- більшість кадрів лежить у діапазоні 5–15 мс;
- поодинокі кадри виходять за 20–30 мс;
- екстремальні значення (80–170+ мс) мають щільність < 1%.

Це свідчить про:

- загальну стабільність рендерингу;
- наявність рідкісних, але відчутних "піків" (jank).

4.2 Порівняння state manager всередині одного сценарію (емулятор)

4.2.1 Animation Test

Аналіз показника Frame Time (рис. 4.1) виявив наступні тенденції:

1. **Stateful**
   - Найбільша концентрація кадрів у Bucket = 0 (~73%);
   - Але присутні довгі хвости до 173 мс;
   - Це свідчить про рідкісні, але дуже дорогі кадри.
2. **Provider**
   - Розподіл більш "розмазаний";
   - Основна маса кадрів у діапазоні 1–15 мс;
   - Менше екстремальних піків, але більша варіативність.

3. **Bloc**
    - Схожий на Provider, але:
        - трохи більше кадрів у середньому діапазоні (7–15 мс);
        - стабільніший хвіст.
4. **Riverpod**
    - Найбільш концентрований розподіл;
    - Переважна частина кадрів — 7–13 мс;
    - Мінімальна кількість екстремальних значень.

Аналіз показника FPS виявив наступні тенденції:

- У всіх менеджерів основна маса кадрів лежить у бакетах ~50–70 FPS;
- Riverpod має найбільшу щільність у найнижчому FPS-бакеті (≈70% у ~49 FPS);
- Stateful демонструє ширший розкид і більшу кількість високих FPS-піків (200+).

4.2.2 API test

Аналіз показника Frame Time (рис. 4.2) виявив наступні тенденції:

1. **Stateful**
    - Основна маса кадрів у діапазоні 2–8 мс;
    - Розподіл компактний, без довгих хвостів.
2. **Provider**
    - Дуже схожий профіль до Stateful;
    - Невелике збільшення варіативності в середньому діапазоні.
3. **Bloc**
    - Найбільш стабільний розподіл серед усіх;
    - Мінімальна кількість кадрів за межами основного діапазону.
4. **Riverpod**
    - Профіль практично ідентичний до Bloc;

- Відмінності між менеджерами мінімальні.

Аналіз показника FPS виявив наступні тенденції:

- Усі стейт-менеджери демонструють близькі FPS-розподіли;
- Основна маса кадрів знаходиться у стабільному високому FPS-діапазоні;
- Відмінності між менеджерами мають неструктурований характер.

4.2.3 Form Test

Аналіз показника Frame Time (рис. 4.3) виявив наступні тенденції:

1. **Stateful**
   - Найбільш широкий та асиметричний розподіл;
   - Присутні піки до 100+ мс;
   - Висока варіативність часу кадру.
2. **Provider**
   - Значна частина кадрів у діапазоні 6–15 мс;
   - Присутні довші хвости, ніж у Bloc та Riverpod.
3. **Bloc**
   - Більш компактний розподіл;
   - Менше екстремальних піків.
4. **Riverpod**
   - Найбільша концентрація кадрів у низьких значеннях;
   - Мінімальна кількість дорогих кадрів.

Аналіз показника FPS виявив наступні тенденції:

- Stateful демонструє найменшу стабільність FPS;
- Provider і Bloc мають помірну стабільність;
- Riverpod забезпечує найщільніший FPS-розподіл без різких спадів.

4.3 Узагальнення по реальному девайсу

Аналіз показника Frame Time виявив наступні тенденції:

- Розподіли для всіх state manager стають:
  - більш компактними;
  - менш асиметричними;
  - з коротшими хвостами.
- Екстремальні піки (>30–40 мс) майже зникають.

Аналіз показника FPS виявив наступні тенденції:

- FPS-розподіли стають більш стабільними;
- Різниця між state manager зменшується;
- Вплив апаратного забезпечення переважає над впливом архітектурного підходу.

Висновок:

Form Test залишається сценарієм, де вибір state manager впливає на стабільність навіть на реальному девайсі.

Дані спостереження не дають повної картини і можливості обрати найкращій із представлених варіантів. Наведений вище аналіз базується на візуальному та описовому порівнянні розподілів, однак:

- розподіли мають асиметричну форму;
- присутні викиди та довгі хвости;
- розміри вибірок між менеджерами не є однаковими;
- середні значення не відображають реальної поведінки даних.

У зв’язку з цим застосування параметричних тестів (наприклад, t-test) є некоректним.

Тому постає необхідність провести статистичний аналіз. Для цього був обраний критерій Манна – Уітні завдяки таким його особливостям:

- не вимагає нормального розподілу даних;
- дозволяє порівнювати дві незалежні вибірки;
- базується на рангах значень, а не на середніх;
- є стійким до викидів та асиметрії.

Таким чином, використання критерію Манна–Уітні дозволяє об’єктивно визначити, чи є спостережувані відмінності між стейт менеджером статистично значущими, а не результатом випадкових коливань.

4.4 Статистичний аналіз за критерієм Манна-Уітні

U-тест Манна-Уітні є непараметричним статистичним тестом для порівняння двох незалежних вибірок. На відміну від параметричних тестів, він не вимагає припущень про нормальність розподілу даних, що робить його особливо корисним для аналізу метрик продуктивності, які часто мають:

- асиметричні розподіли;
- викиди через системні затримки;
- гетерогенність дисперсій.

До переваг даного метода можна віднести:

- робастність до викидів: Ранги менш чутливі до екстремальних значень;
- збереження інформації про порядок: Враховує відносні, а не абсолютні значення;
- придатність для порядкових даних: Може застосовуватися до будь-яких даних, де можна визначити порядок.

Сама методологія аналізу описана у розділі 2.4. Для оцінки значущості результатів надалі будуть використовуватися три статистичних показника – рівні статистичної значущості (α), p – значення (ймовірнісне значення) та величина ефекту Cohen’s (r).

4.4.1 Рівні статистичної значущості (α)

У дослідженні застосовуються стандартні рівні значущості, що дозволяють класифікувати силу статистичних доказів:

- α = 0.05 (позначення: *): Рівень довіри 95%. Відповідає слабкій, але статистично значущій відмінності. Імовірність отримати спостережуваний результат випадково становить менше 5%;

- α = 0.01 (позначення: **): Рівень довіри 99%. Відповідає середній статистичній значущості. Імовірність випадкового результату менше 1%;
- α = 0.001 (позначення: ***): Рівень довіри 99.9%. Відповідає високій статистичній значущості. Імовірність випадкового результату менше 0.1%.

Визначення значущості: Якщо розраховане p-значення менше відповідного рівня α, відмінність вважається статистично значущою з відповідним рівнем довіри.

4.4.2 p - значення (ймовірнісне значення)

p - значення є ключовим показником у статистичному аналізі, який кількісно визначає:

Визначення: Ймовірність отримати результати, принаймні такі ж екстремальні, як спостережувані, за умови істинності нульової гіпотези.

Інтерпретація: Чим менше p-значення, тим сильніше докази проти нульової гіпотези (про відсутність відмінностей).

Практичне застосування: p-значення < 0.05 традиційно вважається статистично значущим у більшості наукових дисциплін.

4.4.3 Величина ефекту Cohen’s (r)

Величина ефекту доповнює статистичну значущість інформацією про практичну важливість відмінностей:

Визначення: Стандартизована міра розміру відмінності між групами, незалежна від одиниць вимірювання;

Розрахунок: Обчислюється як $r = z / \sqrt{N}$, де z — стандартизована U-статистика, N — сумарний розмір вибірок;

Шкала інтерпретації (за Cohen, 1988) [12]:

– $r < 0.1$: Незначний ефект (практично непомітний);
– $0.1 \leq r < 0.3$: Малий ефект (помітний лише при великих вибірках);
– $0.3 \leq r < 0.5$: Середній ефект (помітний при уважному спостереженні);

– $r \geq 0.5$: Великий ефект (помітний без статистичного аналізу).

4.5 Результати попарних тестів Манна-Уітні

Дані розрахунки були проведені окремо для емулятора (таблиці 4.1 – 4.11) та для реального девайсу (таблиці 4.12 – 4.22).

Таблиця 4.1 – Основні результати статистичного аналізу на емуляторі

Сценарій	Показник	Стейт-менеджер 1	Стейт-менеджер 2	P-значення	Значущість	Розмір вибірки 1	Розмір вибірки 2
API Test	Frame Time	bloc	provider	0,734799	н.з.	863	817
API Test	Frame Time	bloc	riverpod	0,65817	н.з.	863	878
API Test	Frame Time	bloc	stateful	0,00735	**	863	863
API Test	Frame Time	provider	riverpod	0,032522	*	817	878
API Test	Frame Time	provider	stateful	0,884397	н.з.	817	863
API Test	Frame Time	riverpod	stateful	0,462812	н.з.	878	863
API Test	FPS	bloc	provider	0,018437	*	371	342
API Test	FPS	bloc	riverpod	0,928251	н.з.	371	381
API Test	FPS	bloc	stateful	0,000029	***	371	368
API Test	FPS	provider	riverpod	0,014131	*	342	381
API Test	FPS	provider	stateful	0,003124	**	342	368
API Test	FPS	riverpod	stateful	0,000036	***	381	368
API Test	Latency	bloc	provider	0,007541	**	500	500
API Test	Latency	bloc	riverpod	0,0936	н.з.	500	500
API Test	Latency	bloc	stateful	0,554526	н.з.	500	500
API Test	Latency	provider	riverpod	0,000061	***	500	500
API Test	Latency	provider	stateful	0,001441	**	500	500
API Test	Latency	riverpod	stateful	0,256044	н.з.	500	500

Сценарій	Показник	Стейт-менеджер 1	Стейт-менеджер 2	P-значення	Значущість	Розмір вибірки 1	Розмір вибірки 2
Animation Test	Frame Time	bloc	provider	0,005344	**	968	936
Animation Test	Frame Time	bloc	riverpod	0,28683	н.з.	968	971
Animation Test	Frame Time	bloc	stateful	0	***	968	965
Animation Test	Frame Time	provider	riverpod	0,000215	***	936	971
Animation Test	Frame Time	provider	stateful	0	***	936	965
Animation Test	Frame Time	riverpod	stateful	0	***	971	965
Animation Test	FPS	bloc	provider	0	***	473	447
Animation Test	FPS	bloc	riverpod	0	***	473	477
Animation Test	FPS	bloc	stateful	0	***	473	474
Animation Test	FPS	provider	riverpod	0	***	447	477
Animation Test	FPS	provider	stateful	0	***	447	474
Animation Test	FPS	riverpod	stateful	0	***	477	474
Animation Test	Latency	bloc	provider	0,080864	н.з.	500	500
Animation Test	Latency	bloc	riverpod	0,483256	н.з.	500	500
Animation Test	Latency	bloc	stateful	0,128974	н.з.	500	500
Animation Test	Latency	provider	riverpod	0,27334	н.з.	500	500
Animation Test	Latency	provider	stateful	0,804202	н.з.	500	500
Animation Test	Latency	riverpod	stateful	0,395378	н.з.	500	500
Form Test	Frame Time	bloc	provider	0,951827	н.з.	801	803
Form Test	Frame Time	bloc	riverpod	0	***	801	795
Form Test	Frame Time	bloc	stateful	0,001463	**	801	805
Form Test	Frame Time	provider	riverpod	0	***	803	795
Form Test	Frame Time	provider	stateful	0,002113	**	803	805
Form Test	Frame Time	riverpod	stateful	0	***	795	805
Form Test	FPS	bloc	provider	0	***	311	315

Сценарій	Показник	Стейт-менеджер 1	Стейт-менеджер 2	P-значення	Значущість	Розмір вибірки 1	Розмір вибірки 2
Form Test	FPS	bloc	riverpod	0	***	311	303
Form Test	FPS	bloc	stateful	0	***	311	319
Form Test	FPS	provider	riverpod	0,067208	н.з.	315	303
Form Test	FPS	provider	stateful	0	***	315	319
Form Test	FPS	riverpod	stateful	0	***	303	319
Form Test	Latency	bloc	provider	0,795889	н.з.	500	500
Form Test	Latency	bloc	riverpod	0,408539	н.з.	500	500
Form Test	Latency	bloc	stateful	0,793626	н.з.	500	500
Form Test	Latency	provider	riverpod	0,279989	н.з.	500	500
Form Test	Latency	provider	stateful	0,995448	н.з.	500	500
Form Test	Latency	riverpod	stateful	0,279749	н.з.	500	500
List Test	Frame Time	bloc	provider	0,048225	*	825	834
List Test	Frame Time	bloc	riverpod	0,628614	н.з.	825	839
List Test	Frame Time	bloc	stateful	0,635887	н.з.	825	842
List Test	Frame Time	provider	riverpod	0,720476	н.з.	834	839
List Test	Frame Time	provider	stateful	0,728048	н.з.	834	842
List Test	Frame Time	riverpod	stateful	0,178056	н.з.	839	842
List Test	FPS	bloc	provider	0,614801	н.з.	329	336
List Test	FPS	bloc	riverpod	0,035827	*	329	340
List Test	FPS	bloc	stateful	0,047826	*	329	343
List Test	FPS	provider	riverpod	0,036082	*	336	340
List Test	FPS	provider	stateful	0,177677	н.з.	336	343
List Test	FPS	riverpod	stateful	0,373903	н.з.	340	343
List Test	Latency	bloc	provider	0,180295	н.з.	500	500
List Test	Latency	bloc	riverpod	0,178552	н.з.	500	500
List Test	Latency	bloc	stateful	0,180003	н.з.	500	500

Сценарій	Показник	Стейт-менеджер 1	Стейт-менеджер 2	P-значення	Значущість	Розмір вибірки 1	Розмір вибірки 2
List Test	Latency	provider	riverpod	1	н.з.	500	500
List Test	Latency	provider	stateful	1	н.з.	500	500
List Test	Latency	riverpod	stateful	1	н.з.	500	500
Subscription Test	Frame Time	bloc	provider	0,105469	н.з.	848	840
Subscription Test	Frame Time	bloc	riverpod	0,014042	*	848	852
Subscription Test	Frame Time	bloc	stateful	0,921966	н.з.	848	843
Subscription Test	Frame Time	provider	riverpod	0,082504	н.з.	840	852
Subscription Test	Frame Time	provider	stateful	0,733502	н.з.	840	843
Subscription Test	Frame Time	riverpod	stateful	0,975003	н.з.	852	843
Subscription Test	FPS	bloc	provider	0,007142	**	354	348
Subscription Test	FPS	bloc	riverpod	0,874367	н.з.	354	353
Subscription Test	FPS	bloc	stateful	0,42357	н.з.	354	351
Subscription Test	FPS	provider	riverpod	0,542869	н.з.	348	353
Subscription Test	FPS	provider	stateful	0,002165	**	348	351
Subscription Test	FPS	riverpod	stateful	0,277484	н.з.	353	351
Subscription Test	Latency	bloc	provider	0,763255	н.з.	500	500
Subscription Test	Latency	bloc	riverpod	0,101955	н.з.	500	500
Subscription Test	Latency	bloc	stateful	0,993917	н.з.	500	500
Subscription Test	Latency	provider	riverpod	0,058326	н.з.	500	500
Subscription Test	Latency	provider	stateful	0,77066	н.з.	500	500
Subscription Test	Latency	riverpod	stateful	0,10127	н.з.	500	500

Таблиця 4.2 – Статистика за сценаріями на емуляторі

Сценарій	Показник	Всього тестів	Значущих тестів	Відсоток значущих (%)
API Test	Frame Time	6	2	33,3
API Test	FPS	6	5	83,3
API Test	Latency	6	3	50
Animation Test	Frame Time	6	5	83,3
Animation Test	FPS	6	6	100

Сценарій	Показник	Всього тестів	Значущих тестів	Відсоток значущих (%)
Animation Test	Latency	6	0	0
Form Test	Frame Time	6	5	83,3
Form Test	FPS	6	5	83,3
Form Test	Latency	6	0	0
List Test	Frame Time	6	1	16,7
List Test	FPS	6	3	50
List Test	Latency	6	0	0
Subscription Test	Frame Time	6	1	16,7
Subscription Test	FPS	6	2	33,3
Subscription Test	Latency	6	0	0

Таблиця 4.3 – Frame Time значущість на емуляторі

State Manager	bloc	provider	riverpod	stateful
bloc		н.з.	н.з.	**
provider	н.з.		*	н.з.
riverpod	н.з.	*		н.з.
stateful	**	н.з.	н.з.	

Таблиця 4.4 – Frame Time p – значення на емуляторі

State Manager	bloc	provider	riverpod	stateful
bloc		0,734799	0,65817	0,00735
provider	0,734799		0,032522	0,884397
riverpod	0,65817	0,032522		0,462812
stateful	0,00735	0,884397	0,462812	

Таблиця 4.5 – Frame Time ефект на емуляторі

State Manager	bloc	provider	riverpod	stateful
bloc		0,007	0,01	0,058
provider	0,007		0,047	0,003
riverpod	0,01	0,047		0,016
stateful	0,058	0,003	0,016	

Таблиця 4.6 – FPS значущість на емуляторі

State Manager	bloc	provider	riverpod	stateful
bloc		*	н.з.	***
provider	*		*	**
riverpod	н.з.	*		***
stateful	***	**	***	

Таблиця 4.7 – FPS p – значення на емуляторі

State Manager	bloc	provider	riverpod	stateful
bloc		0,018437	0,928251	0,000029
provider	0,018437		0,014131	0,003124
riverpod	0,928251	0,014131		0,000036
stateful	0,000029	0,003124	0,000036	

Таблиця 4.8 – FPS ефект на емуляторі

State Manager	bloc	provider	riverpod	stateful
bloc		0,088	0,003	0,153
provider	0,088		0,091	0,11
riverpod	0,003	0,091		0,15
stateful	0,153	0,11	0,15	

Таблиця 4.9 – FPS значущість на емуляторі

State Manager	bloc	provider	riverpod	stateful
bloc		**	н.з.	н.з.
provider	**		***	**
riverpod	н.з.	***		н.з.
stateful	н.з.	**	н.з.	

Таблиця 4.10 – FPS p – значення на емуляторі

State Manager	bloc	provider	riverpod	stateful
bloc		0,007541	0,0936	0,554526
provider	0,007541		0,000061	0,001441
riverpod	0,0936	0,000061		0,256044
stateful	0,554526	0,001441	0,256044	

Таблиця 4.11 – FPS ефект на емуляторі

State Manager	bloc	provider	riverpod	stateful
bloc		0,024	0,009	0,004
provider	0,024		0,033	0,028
riverpod	0,009	0,033		0,005
stateful	0,004	0,028	0,005	

Таблиця 4.12 – Основні результати статистичного аналізу на реальному девайсі

Сценарій	Показник	Стейт-менеджер 1	Стейт-менеджер 2	P-значення	Значущість	Розмір вибірки 1	Розмір вибірки 2
API Test	Frame Time	bloc	provider	0,723755	н.з.	926	927
API Test	Frame Time	bloc	riverpod	0	***	926	866
API Test	Frame Time	bloc	stateful	0	***	926	865
API Test	Frame Time	provider	riverpod	0	***	927	866
API Test	Frame Time	provider	stateful	0	***	927	865
API Test	Frame Time	riverpod	stateful	0,922685	н.з.	866	865
API Test	FPS	bloc	provider	0	***	427	428
API Test	FPS	bloc	riverpod	0	***	427	368
API Test	FPS	bloc	stateful	0,82068	н.з.	427	366
API Test	FPS	provider	riverpod	0	***	428	368
API Test	FPS	provider	stateful	0,00099	***	428	366
API Test	FPS	riverpod	stateful	0	***	368	366
API Test	Latency	bloc	provider	1	н.з.	500	500
API Test	Latency	bloc	riverpod	0,564874	н.з.	500	500
API Test	Latency	bloc	stateful	1	н.з.	500	500
API Test	Latency	provider	riverpod	0,564874	н.з.	500	500
API Test	Latency	provider	stateful	1	н.з.	500	500
API Test	Latency	riverpod	stateful	0,564874	н.з.	500	500
Animation Test	Frame Time	bloc	provider	0,000004	***	866	865
Animation Test	Frame Time	bloc	riverpod	0,000012	***	866	866
Animation Test	Frame Time	bloc	stateful	0,007032	**	866	866
Animation Test	Frame Time	provider	riverpod	0,000001	***	865	866
Animation Test	Frame Time	provider	stateful	0,000004	***	865	866
Animation Test	Frame Time	riverpod	stateful	0,000377	***	866	866
Animation Test	FPS	bloc	provider	0	***	370	373
Animation Test	FPS	bloc	riverpod	0	***	370	367
Animation Test	FPS	bloc	stateful	0	***	370	369
Animation Test	FPS	provider	riverpod	0	***	373	367

Сценарій	Показник	Стейт-менеджер 1	Стейт-менеджер 2	P-значення	Значущість	Розмір вибірки 1	Розмір вибірки 2
Animation Test	FPS	provider	stateful	0	***	373	369
Animation Test	FPS	riverpod	stateful	0	***	367	369
Animation Test	Latency	bloc	provider	0,253483	н.з.	500	500
Animation Test	Latency	bloc	riverpod	0,179712	н.з.	500	500
Animation Test	Latency	bloc	stateful	0,706866	н.з.	500	500
Animation Test	Latency	provider	riverpod	0,019455	*	500	500
Animation Test	Latency	provider	stateful	0,133618	н.з.	500	500
Animation Test	Latency	riverpod	stateful	0,316584	н.з.	500	500
Form Test	Frame Time	bloc	provider	0	***	945	898
Form Test	Frame Time	bloc	riverpod	0	***	945	979
Form Test	Frame Time	bloc	stateful	0,000023	***	945	956
Form Test	Frame Time	provider	riverpod	0	***	898	979
Form Test	Frame Time	provider	stateful	0	***	898	956
Form Test	Frame Time	riverpod	stateful	0,014796	*	979	956
Form Test	FPS	bloc	provider	0	***	446	399
Form Test	FPS	bloc	riverpod	0	***	446	481
Form Test	FPS	bloc	stateful	0	***	446	457
Form Test	FPS	provider	riverpod	0	***	399	481
Form Test	FPS	provider	stateful	0	***	399	457
Form Test	FPS	riverpod	stateful	0	***	481	457
Form Test	Latency	bloc	provider	1	н.з.	500	500
Form Test	Latency	bloc	riverpod	0,565656	н.з.	500	500
Form Test	Latency	bloc	stateful	1	н.з.	500	500
Form Test	Latency	provider	riverpod	0,565656	н.з.	500	500
Form Test	Latency	provider	stateful	1	н.з.	500	500
Form Test	Latency	riverpod	stateful	0,564874	н.з.	500	500
List Test	Frame Time	bloc	provider	0	***	925	927
List Test	Frame Time	bloc	riverpod	0,861377	н.з.	925	926

Сценарій	Показник	Стейт-менеджер 1	Стейт-менеджер 2	P-значення	Значущість	Розмір вибірки 1	Розмір вибірки 2
List Test	Frame Time	bloc	stateful	0,276796	н.з.	925	926
List Test	Frame Time	provider	riverpod	0,000094	***	927	926
List Test	Frame Time	provider	stateful	0,000048	***	927	926
List Test	Frame Time	riverpod	stateful	0,585012	н.з.	926	926
List Test	FPS	bloc	provider	0	***	426	430
List Test	FPS	bloc	riverpod	0,000008	***	426	431
List Test	FPS	bloc	stateful	0	***	426	427
List Test	FPS	provider	riverpod	0	***	430	431
List Test	FPS	provider	stateful	0,001056	**	430	427
List Test	FPS	riverpod	stateful	0	***	431	427
List Test	Latency	bloc	provider	0,318038	н.з.	500	500
List Test	Latency	bloc	riverpod	0,102298	н.з.	500	500
List Test	Latency	bloc	stateful	1	н.з.	500	500
List Test	Latency	provider	riverpod	0,475321	н.з.	500	500
List Test	Latency	provider	stateful	0,318038	н.з.	500	500
List Test	Latency	riverpod	stateful	0,102298	н.з.	500	500
Subscription Test	Frame Time	bloc	provider	0,001044	**	927	864
Subscription Test	Frame Time	bloc	riverpod	0	***	927	865
Subscription Test	Frame Time	bloc	stateful	0,113619	н.з.	927	926
Subscription Test	Frame Time	provider	riverpod	0,000041	***	864	865
Subscription Test	Frame Time	provider	stateful	0,000595	***	864	926
Subscription Test	Frame Time	riverpod	stateful	0	***	865	926
Subscription Test	FPS	bloc	provider	0	***	438	365
Subscription Test	FPS	bloc	riverpod	0,000444	***	438	368
Subscription Test	FPS	bloc	stateful	0	***	438	435
Subscription Test	FPS	provider	riverpod	0	***	365	368

Сценарій	Показник	Стейт-менеджер 1	Стейт-менеджер 2	P-значення	Значущість	Розмір вибірки 1	Розмір вибірки 2
Subscription Test	FPS	provider	stateful	0	***	365	435
Subscription Test	FPS	riverpod	stateful	0,003095	**	368	435
Subscription Test	Latency	bloc	provider	0,003631	**	500	500
Subscription Test	Latency	bloc	riverpod	0,031429	*	500	500
Subscription Test	Latency	bloc	stateful	0,627096	н.з.	500	500
Subscription Test	Latency	provider	riverpod	0,315617	н.з.	500	500
Subscription Test	Latency	provider	stateful	0,010885	*	500	500
Subscription Test	Latency	riverpod	stateful	0,083833	н.з.	500	500

Таблиця 4.13 – Статистика за сценаріями на реальному девайсі

Сценарій	Показник	Всього тестів	Значущих тестів	Відсоток значущих (%)
API Test	Frame Time	6	4	66,7
API Test	FPS	6	5	83,3
API Test	Latency	6	0	0
Animation Test	Frame Time	6	6	100
Animation Test	FPS	6	6	100
Animation Test	Latency	6	1	16,7
Form Test	Frame Time	6	6	100
Form Test	FPS	6	6	100
Form Test	Latency	6	0	0
List Test	Frame Time	6	3	50
List Test	FPS	6	6	100
List Test	Latency	6	0	0
Subscription Test	Frame Time	6	5	83,3
Subscription Test	FPS	6	6	100
Subscription Test	Latency	6	3	50

Таблиця 4.14 – Frame Time значущість на реальному девайсі

State Manager	bloc	provider	riverpod	stateful
bloc		н.з.	***	***
provider	н.з.		***	***
riverpod	***	***		н.з.
stateful	***	***	н.з.	

Таблиця 4.15 – Frame Time p – значення на реальному девайсі

State Manager	bloc	provider	riverpod	stateful
bloc		0,723755	0	0
provider	0,723755		0	0
riverpod	0	0		0,922685
stateful	0	0	0,922685	

Таблиця 4.16 – Frame Time ефект на реальному девайсі

State Manager	bloc	provider	riverpod	stateful
bloc		0,008	0,115	0,167
provider	0,008		0,144	0,115
riverpod	0,115	0,144		0,002
stateful	0,167	0,115	0,002	

Таблиця 4.17 – FPS значущість на реальному девайсі

State Manager	bloc	provider	riverpod	stateful
bloc		***	***	н.з.
provider	***		***	***
riverpod	***	***		***
stateful	н.з.	***	***	

Таблиця 4.18 – FPS p – значення на реальному девайсі

State Manager	bloc	provider	riverpod	stateful
bloc		0	0	0,82068
provider	0		0	0,00099
riverpod	0	0		0
stateful	0,82068	0,00099	0	

Таблиця 4.19 – FPS ефект на реальному девайсі

State Manager	bloc	provider	riverpod	stateful
bloc		0,306	0,702	0,008
provider	0,306		0,707	0,111
riverpod	0,702	0,707		0,705
stateful	0,008	0,111	0,705	

Таблиця 4.20 – FPS значущість на реальному девайсі

State Manager	bloc	provider	riverpod	stateful
bloc		н.з.	н.з.	н.з.
provider	н.з.		н.з.	н.з.
riverpod	н.з.	н.з.		н.з.
stateful	н.з.	н.з.	н.з.	

Таблиця 4.21 – FPS p – значення на реальному девайсі

State Manager	bloc	provider	riverpod	stateful
bloc		1	0,564874	1
provider	1		0,564874	1
riverpod	0,564874	0,564874		0,564874
stateful	1	1	0,564874	

Таблиця 4.22 – FPS ефект на реальному девайсі

State Manager	bloc	provider	riverpod	stateful
bloc		0	0,002	0
provider	0		0,002	0
riverpod	0,002	0,002		0,002
stateful	0	0	0,002	

4.6 Аналіз результатів на емуляторі

Дані з таблиць 4.1 - 4.2 вказують на те, що показник FPS є найбільш диференціюючим між способами управління станом.

- **Animation Test:** 100% значущих відмінностей (6/6 тестів);
- **Form Test:** 83.3% значущих відмінностей (5/6);
- **API Test:** 83.3% значущих відмінностей (5/6).

Навпаки, Latency показує найменші відмінності:

- **Animation, Form, List, Subscription Tests:** 0% значущих відмінностей;
- **API Test:** лише 50% значущих відмінностей.
- **Найбільш виражені відмінності:** Animation Test (Frame Time: 83.3%, FPS: 100%);
- **Найменш виражені відмінності:** List Test (Frame Time: 16.7%, Latency: 0%).

4.6.1 Аналіз Frame Time (час рендерингу кадрів)

Аналіз таблиць 4.3 – 4.5 виявляє наступні ключові особливості:

Ключові особливості:

1. Stateful Widget демонструє статистично значущу відмінність від Bloc ($p = 0.00735$, **) з середнім ефектом ($r = 0.058$);
2. Provider vs Riverpod: Значуща, але слабка відмінність ($p = 0.032522$, *) з малим ефектом ($r = 0.047$);
3. Bloc vs Provider/Riverpod: Незначущі відмінності ($p > 0.05$).

Висновок: За часом рендерингу кадрів стейт-менеджери групуються на дві категорії:

- Група 1 (еквівалентні): Bloc, Provider, Riverpod;
- Група 2 (відмінний): Stateful Widget.

4.6.2 Аналіз FPS (частота кадрів)

Аналіз таблиць 4.6 – 4.8 показує, що частота кадрів є найбільш інформативним показником:

1. Stateful Widget значущо відрізняється від усіх інших:
   - vs Bloc: $p = 0.000029$ (***), $r = 0.153$ (середній ефект);
   - vs Provider: $p = 0.003124$ (**), $r = 0.11$ (середній ефект);
   - vs Riverpod: $p = 0.000036$ (***), $r = 0.15$ (середній ефект).
2. Provider показує системні відмінності:
   - vs Bloc: $p = 0.018437$ (*), $r = 0.088$ (малий ефект);
   - vs Riverpod: $p = 0.014131$ (*), $r = 0.091$ (малий ефект).

Висновок: FPS є найкращим індикатором для порівняння, оскільки:

- Має найвищі величини ефекту (до 0.153);
- Показує найбільшу кількість значущих відмінностей;
- Є найчутливішим показником до архітектурних відмінностей.

4.6.3 Latency (затримки)

Аналіз таблиць 4.9 – 4.11 показує, що це є найменш інформативний показник з найменшими ефектами:

1. Provider демонструє найкращі результати:
   - vs Bloc: $p = 0.007541$ (**), $r = 0.024$ (мінімальний ефект);

– vs Riverpod: p = 0.000061 (***), r = 0.033 (мінімальний ефект);

– vs Stateful: p = 0.001441 (**), r = 0.028 (мінімальний ефект).

2. Всі величини ефекту є мінімальними (r < 0.05), що свідчить про практичну незначущість відмінностей.

Висновок: Latency є найменш релевантною метрикою для порівняння стейт-менеджерів на емуляторі.

4.6.4 Порівняльна характеристика стейт-менеджерів на емуляторі

1. Bloc

Сильні сторони:

1. Стабільна продуктивність у List Test: Незначущі відмінності з іншими підходами за Frame Time (p > 0.05);

2. Добрі результати у Form Test: Незначущі відмінності з Provider за Frame Time (p = 0.951827);

3. Збалансована продуктивність: Не виділяється ні в позитивному, ні в негативному сенсі.

Слабкі сторони:

1. Найгірші показники FPS у API Test: Значущо поступається Stateful (**) *та Provider ()*;

2. Проблеми з латентністю: Поступається Provider у API Test (**).

Ресурсоємність: Середня.

2. Provider

Сильні сторони:

1. Найкраща латентність: Значущо кращі результати за всіма парами порівняння;

2. Стабільність у складних сценаріях: Добрі результати у Animation Test (Frame Time: *** з Riverpod та Stateful);

3. Консистентність: Системно відрізняється від інших підходів.

Слабкі сторони:

1. Проблеми з FPS у Form Test: Значущо поступається Stateful (**);

2. Неоптимальна продуктивність у List Test: Незначні переваги.

Ресурсоємність: Висока (найбільш ресурсоємний серед бібліотечних рішень).

3. Riverpod

Сильні сторони:

1. Оптимальний баланс у Form Test: Найкращі результати за Frame Time (*** vs усіх);

2. Конкурентоздатність у List Test: Незначущі відмінності за основними показниками;

3. Стабільність: Менше високо значущих відмінностей порівняно з Provider.

Слабкі сторони:

1. Найгірша латентність: Значущо поступається Provider (***);

2. Проблеми у Subscription Test: Значущо поступається Bloc за Frame Time (*).

Ресурсоємність: Нижча за Provider, вища за Stateful.

4. Stateful Widget

Сильні сторони:

1. Найкращі показники FPS: Значущо кращі результати за всіма парами порівняння (***);

2. Найкращі результати у Animation Test: Значущо кращі за Frame Time (***);

3. Мінімальна ресурсоємність: Найменша абстракція, найменші накладні витрати.

Слабкі сторони:

1. Проблеми з масштабуванням: Значущо поступається Riverpod у Form Test за Frame Time (***);

2. Нестабільність: Найбільші коливання продуктивності між сценаріями.

Ресурсоємність: Найнижча.

4.6.5 Рейтинг ефективності за типами операцій на емуляторі

1. **Анімації (Animation Test):**
    1. Stateful Widget (найкращий FPS);
    2. Riverpod/Bloc (еквівалентні);
    3. Provider (найгірший).
2. **Форми (Form Test):**
    1. Riverpod (найкращий Frame Time);
    2. Provider/Bloc (еквівалентні);
    3. Stateful Widget (найгірший).
3. **API запити (API Test):**
    1. Provider (найкраща латентність);
    2. Stateful Widget (найкращий FPS);
    3. Riverpod/Bloc (еквівалентні).
4. **Списки (List Test):**

Усі еквівалентні (мінімальні відмінності).

У підсумку для емулятора можна виділити:

1. Немає універсального найкращого рішення бо кожен підхід має свою нішу;
2. Stateful Widget є найефективнішим для рендерингу, але найменш масштабованим;
3. Provider є найстабільнішим для критичних операцій, але більш ресурсоємним;
4. Riverpod пропонує оптимальний баланс між продуктивністю та масштабуванням;
5. Bloc є найбільш прогнозованим, але не виділяється жодною характеристикою.

4.7 Аналіз результатів на реальному девайсі

З таблиць 4.12 – 4.13 видно, що на реальному девайсі відмінності значно посилюються:

- Frame Time: Від 33.3-83.3% на емуляторі до 50-100% на девайсі;
- FPS: Зберігає високі показники (83.3-100%);
- Latency: Зростає з 0-50% до 0-50%, але з більш вираженими ефектами.

Критичні зміни в конкретних сценаріях:

- Animation Test: Frame Time з 83.3% до 100% значущих відмінностей;
- Form Test: Frame Time з 83.3% до 100% значущих відмінностей;
- List Test: Різке зростання значущості FPS з 50% до 100%.

4.7.1 Аналіз Frame Time (час рендерингу кадрів)

Аналіз таблиць 4.14 – 4.16 показує:

1. Всі пари, крім Riverpod-Stateful, стали високозначущими (***);
2. Величини ефекту зросли в 2-10 разів:
   – Bloc vs Riverpod: r = 0.115 (середній ефект) vs 0.01 на емуляторі;
   – Bloc vs Stateful: r = 0.167 (великий ефект) vs 0.058 на емуляторі;
   – Provider vs Riverpod: r = 0.144 (середній ефект) vs 0.047 на емуляторі.

Ключове відкриття, що на реальному девайсі Riverpod і Stateful демонструють еквівалентну продуктивність (p = 0.922685, н.з.), тоді як на емуляторі вони значно відрізнялися.

4.7.2 Аналіз FPS (частота кадрів)

Аналіз таблиць 4.17 – 4.19 показує найбільш вражаючі відмінності:

1. Дуже великі величини ефекту:
   – Bloc vs Riverpod: r = 0.702 (дуже великий ефект);
   – Provider vs Riverpod: r = 0.707 (дуже великий ефект);

– Riverpod vs Stateful: r = 0.705 (дуже великий ефект).

2. Stateful Widget втрачає перевагу: На емуляторі він був найкращим, на девайсі поступається Riverpod з r = 0.705;

3. Bloc показує найгірші результати: Значущо поступається всім (***) з великими ефектами.

4.7.3 Аналіз Latency (затримки)

Аналіз таблиць 4.20 – 4.22 показує мінімальні зміни порівняно з емулятором:

1. Практично всі відмінності незначущі (p > 0.05);

2. Мінімальні величини ефекту (r ≤ 0.002).

3. Subscription Test - єдиний виняток: З’являються значущі відмінності.

4.7.4 Порівняльна характеристика стейт-менеджерів на реальному девайсі

1. Bloc

Різке погіршення продуктивності:

1. Найгірші показники FPS: Значущо поступається всім з дуже великими ефектами (r до 0.702);

2. Проблеми з Frame Time: Значущо поступається Riverpod (*, r = 0.115) та Stateful (*, r = 0.167);

3. Єдина позитивна зміна: Покращення відносно Provider у API Test (Frame Time: н.з. vs ** на емуляторі).

Висновок: Bloc найгірше адаптується до апаратних особливостей реальних пристроїв.

2. Provider

Стабільна, але не оптимальна продуктивність:

1. Консистентні відмінності: Значущо відрізняється від усіх за Frame Time (***);

2. Серйозні проблеми з FPS: Значущо поступається Riverpod (***, r = 0.707);

3. Покращення у List Test: З’являються значущі переваги над Riverpod та Stateful за Frame Time (***).

Висновок: Provider залишається стабільним, але не конкурує за продуктивність.

3. Riverpod

Абсолютний лідер на реальному девайсі:

1. Найкращі показники FPS: Значущо перевершує всіх з дуже великими ефектами (r = 0.702-0.707);

2. Оптимальний Frame Time: Еквівалентний Stateful (н.з.), значущо кращий за Bloc та Provider;

3. Консистентність: Високі результати у всіх сценаріях.

Ключове відкриття: Riverpod отримує максимальну вигоду від апаратних оптимізацій реальних пристроїв.

4. Stateful Widget

Змішана ефективність:

1. Втрата лідерства у FPS: Значущо поступається Riverpod (***, r = 0.705);

2. Покращення у Frame Time: Еквівалентний Riverpod (н.з.), значущо кращий за Bloc та Provider;

3. Нестабільність: Великі коливання між сценаріями.

Висновок: Stateful Widget втрачає переваги на реальному девайсі, особливо у порівнянні з Riverpod.

4.7.5 Рейтинг ефективності за типами операцій

Анімації (Animation Test):

1. Riverpod (найкращий FPS та Frame Time);
2. Stateful Widget;
3. Provider;
4. Bloc.

Форми (Form Test):

1. Riverpod (домінує за всіма показниками);
2. Stateful Widget;
3. Provider;
4. Bloc.

API запити (API Test):

1. Riverpod (найкращий FPS);
2. Stateful Widget (еквівалентний Riverpod за Frame Time);
3. Provider/Bloc (еквівалентні).

Списки (List Test):

1. Riverpod (найкращий FPS);
2. Stateful Widget;
3. Provider (кращий Frame Time);
4. Bloc.

На реальних пристроях Riverpod демонструє беззаперечну перевагу за всіма ключовими показниками продуктивності, роблячи його оптимальним вибором для сучасних Flutter-додатків.

4.8 Аналіз ресурсоємності стейт-менеджерів

Для аналізу ресурсоємності були узяті показники пікової пам’яті та середньої пам’яті у кожну окрему секунду перебігу тесту (таблиці 4.23-4.24 та рис. 4.21-4.22).

Таблиця 4.23 – Пікова пам’ять на емуляторі (Мб)

Scenario	State Manager	Iterations	Peak Memory (MB)
List Test	stateful	500	190,828125
Form Test	stateful	500	208,1835938
Animation Test	stateful	500	208,9648438
API Test	stateful	500	207,1992188
Subscription Test	stateful	500	208,8320313
List Test	provider	500	209,1210938
Form Test	provider	500	211,5234375
Animation Test	provider	500	203,6953125
API Test	provider	500	207,0703125
Subscription Test	provider	500	207,0390625
List Test	bloc	500	208,6445313
Form Test	bloc	500	213,6171875
Animation Test	bloc	500	211,21875
API Test	bloc	500	210,4296875
Subscription Test	bloc	500	206,9023438
List Test	riverpod	500	211,6601563
Form Test	riverpod	500	217,1601563
Animation Test	riverpod	500	212,4023438
API Test	riverpod	500	210,9726563
Subscription Test	riverpod	500	212,3984375

Таблиця 4.24 – Пікова пам’ять на реальному девайсі (Мб)

Scenario	State Manager	Iterations	Peak Memory (MB)
List Test	stateful	500	137,4726563
Form Test	stateful	500	153,1054688
Animation Test	stateful	500	149,0898438
API Test	stateful	500	146,7617188
Subscription Test	stateful	500	147,9453125
List Test	provider	500	151,2109375
Form Test	provider	500	170,3007813
Animation Test	provider	500	156,5234375
API Test	provider	500	149,9492188
Subscription Test	provider	500	153,1953125
List Test	bloc	500	156,5390625
Form Test	bloc	500	166,8945313

Scenario	State Manager	Iterations	Peak Memory (MB)
Animation Test	bloc	500	158,65625
API Test	bloc	500	151,484375
Subscription Test	bloc	500	152,6523438
List Test	riverpod	500	158,1210938
Form Test	riverpod	500	137,6210938
Animation Test	riverpod	500	139,4609375
API Test	riverpod	500	141,53125
Subscription Test	riverpod	500	141,8828125

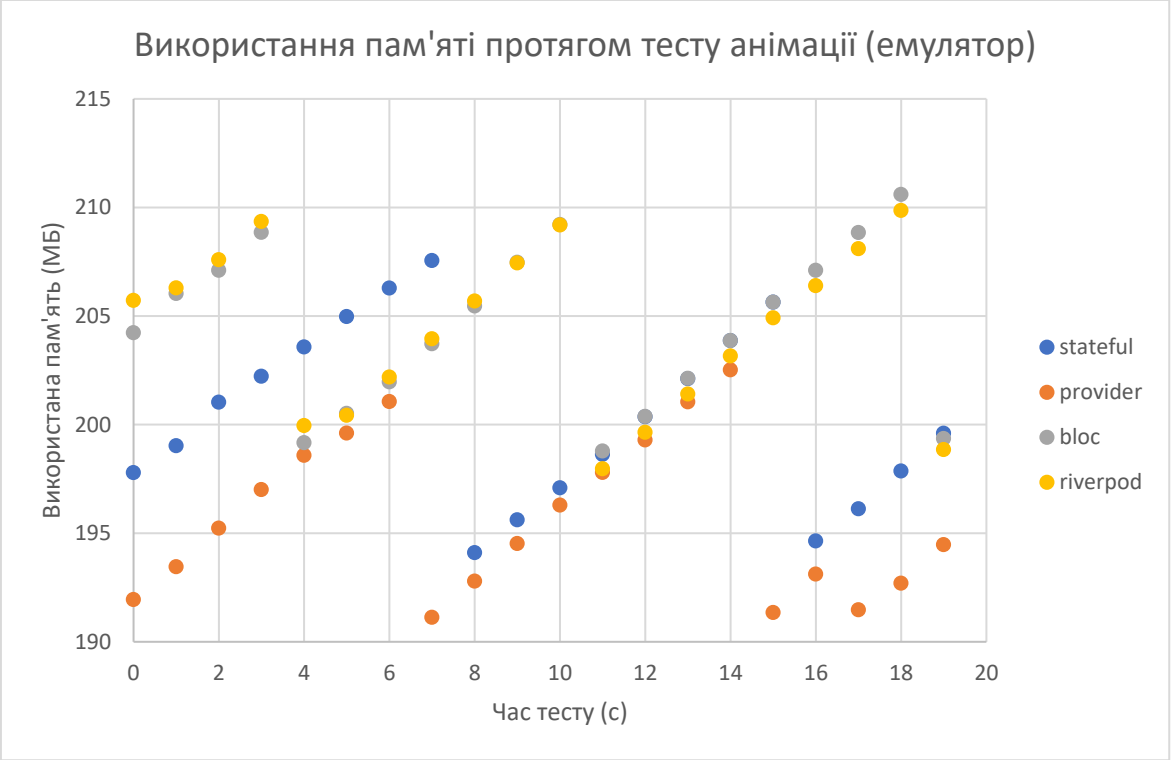

Рис. 4.21 – Використання пам’яті протягом анімаційного тесту на емуляторі (Мб)

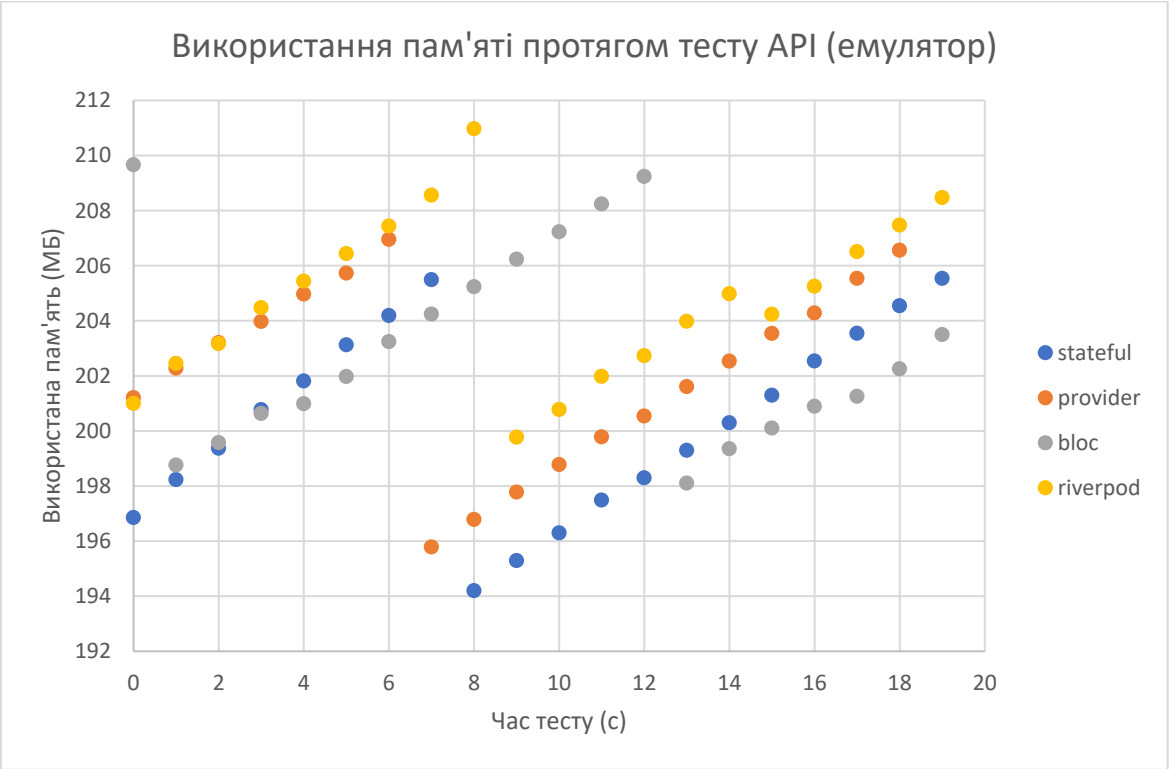

Рис. 4.22 – Використання пам’яті протягом API тесту на емуляторі (Мб)

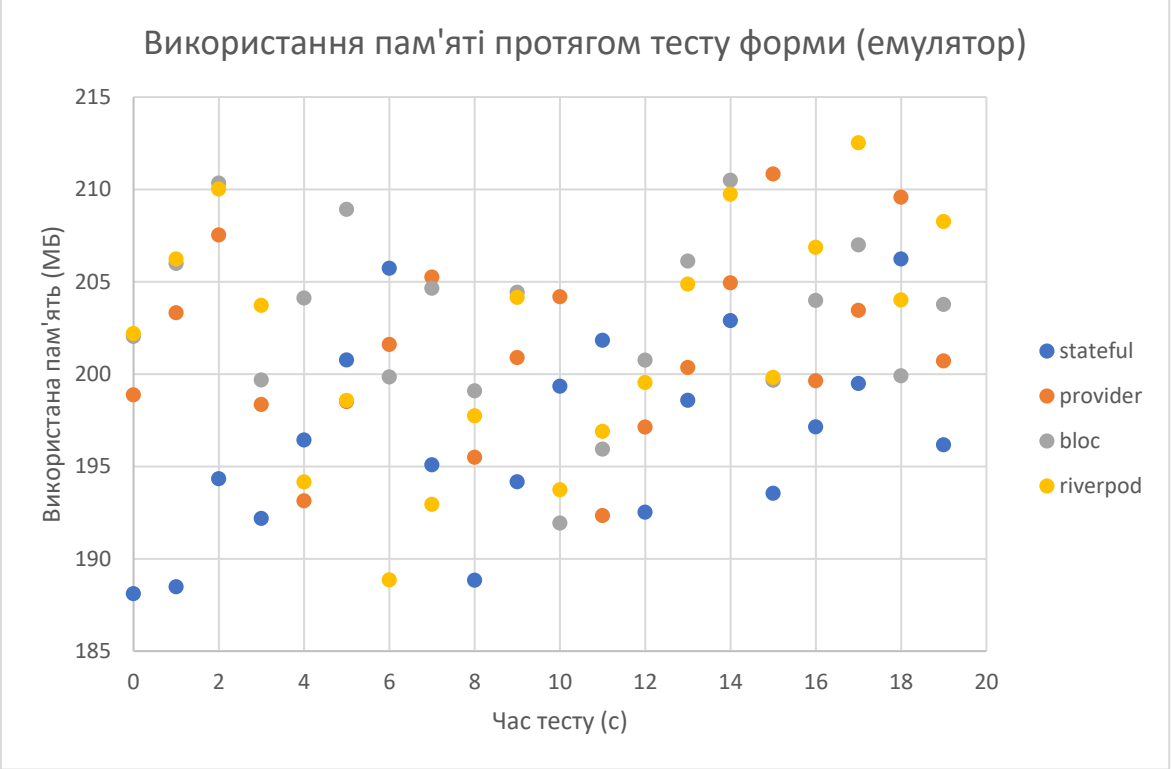

Рис. 4.23 – Використання пам’яті протягом тесту форми на емуляторі (Мб)

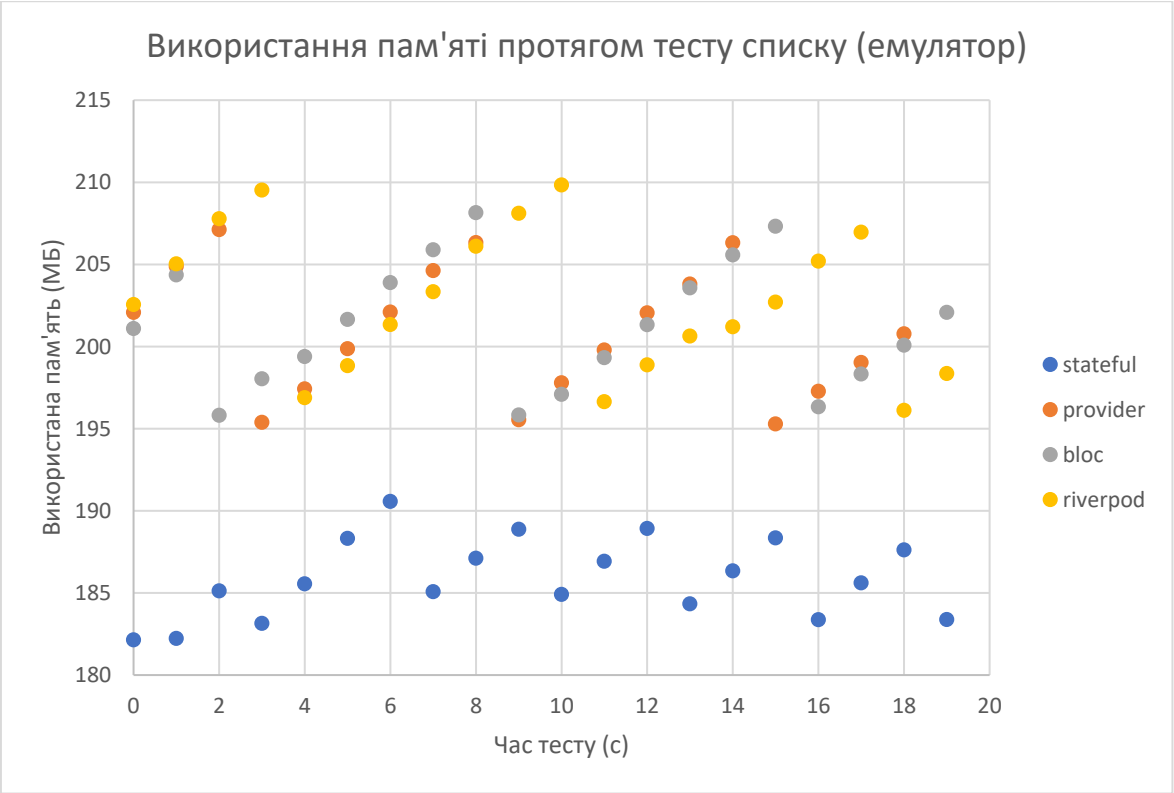

Рис. 4.24 – Використання пам’яті протягом тесту списку на емуляторі (Мб)

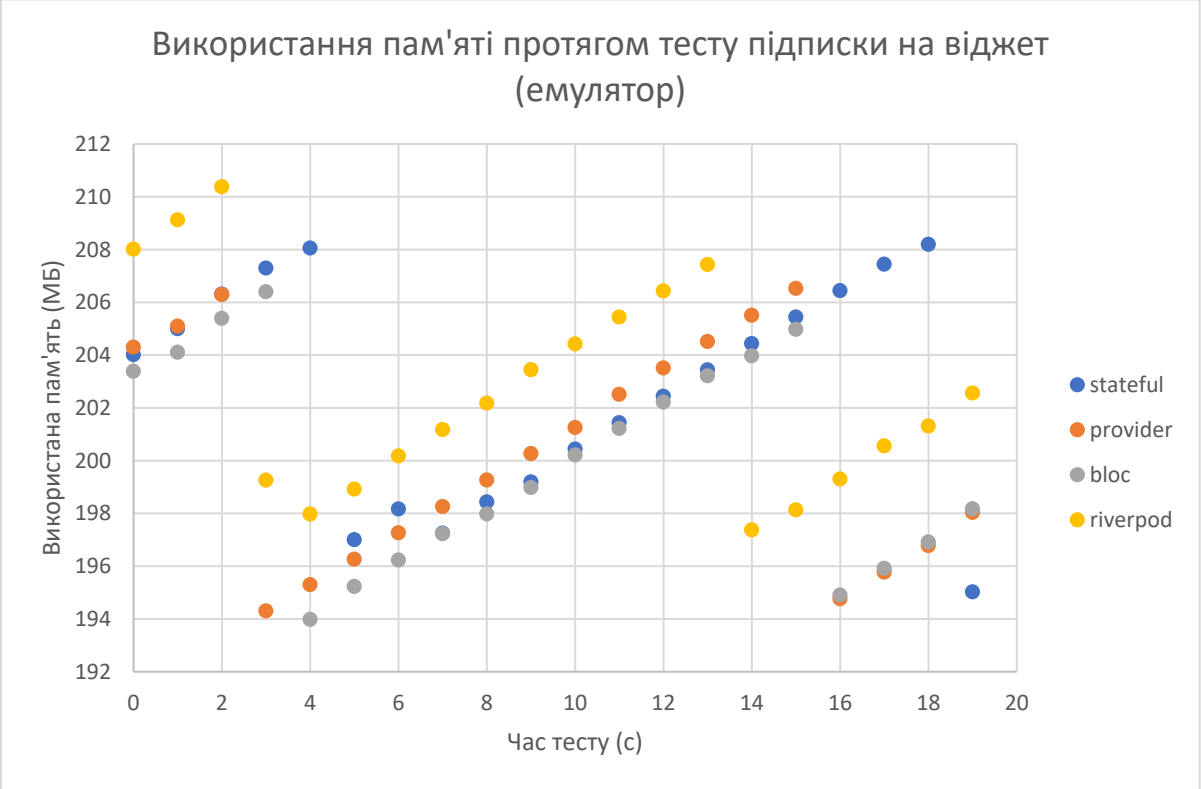

Рис. 4.25 – Використання пам’яті протягом тесту підписки на віджет на емуляторі (Мб)

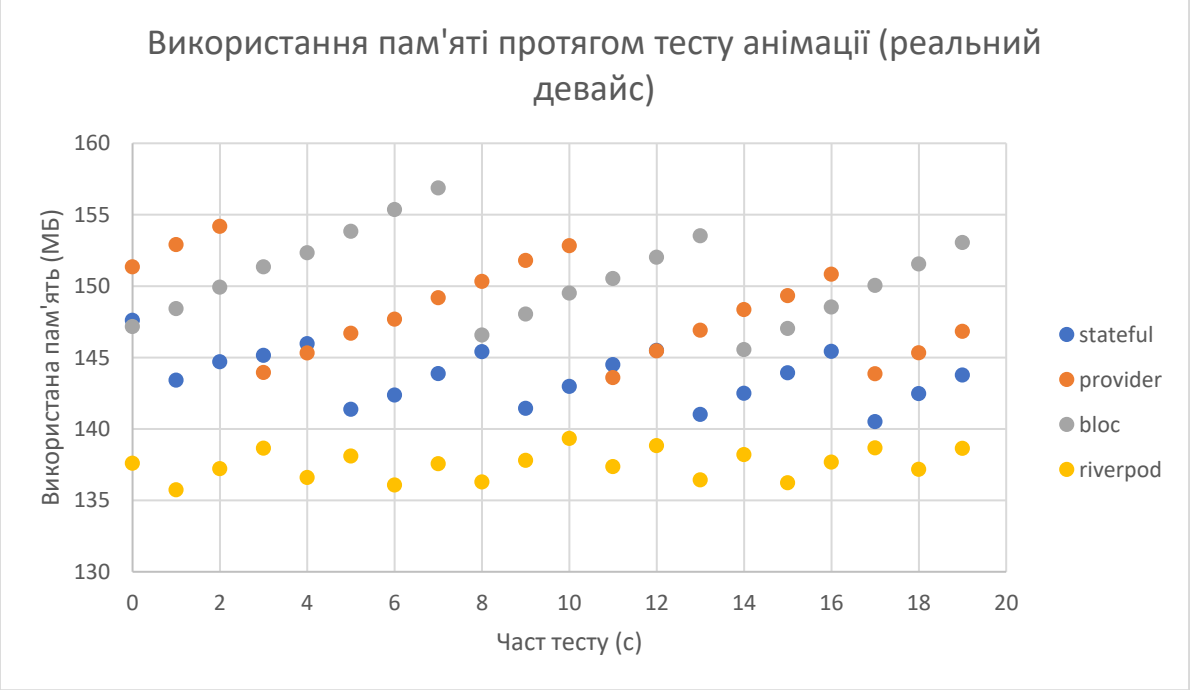

Рис. 4.26 – Використання пам’яті протягом анімаційного тесту на реальному девайсі (Мб)

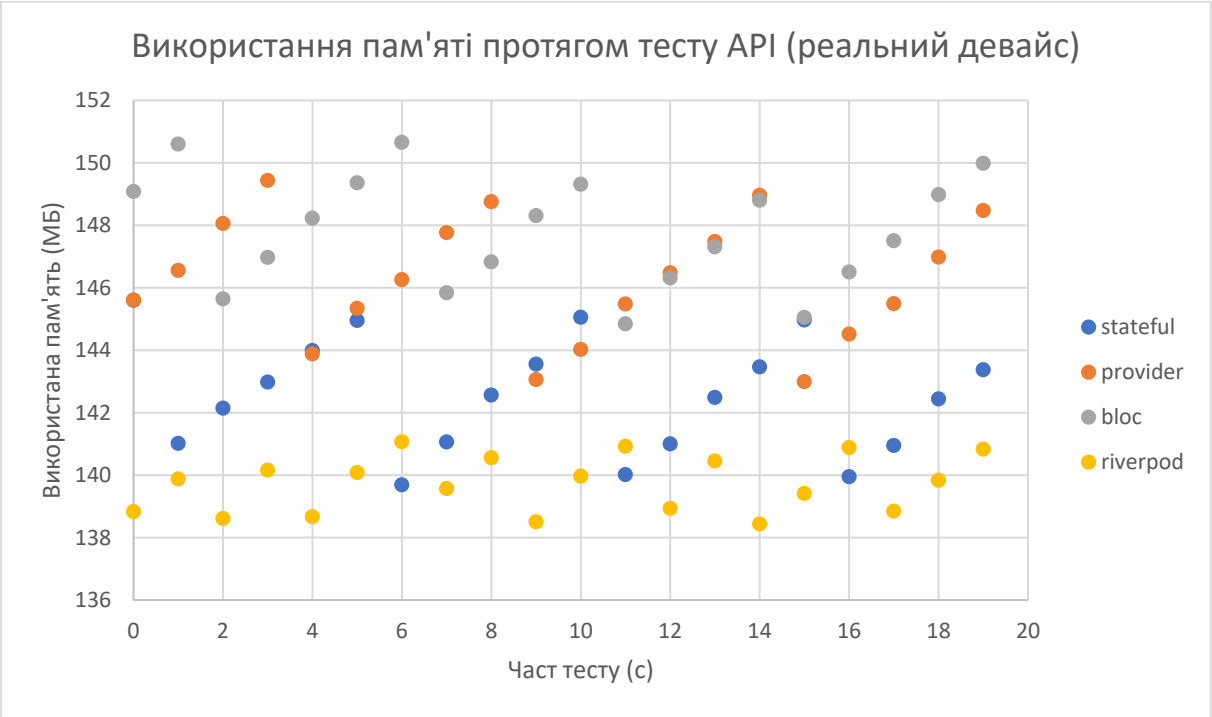

Рис. 4.27 – Використання пам’яті протягом API тесту на реальному девайсі (Мб)

Рис. 4.28 – Використання пам’яті протягом тесту форми на реальному девайсі (Мб)

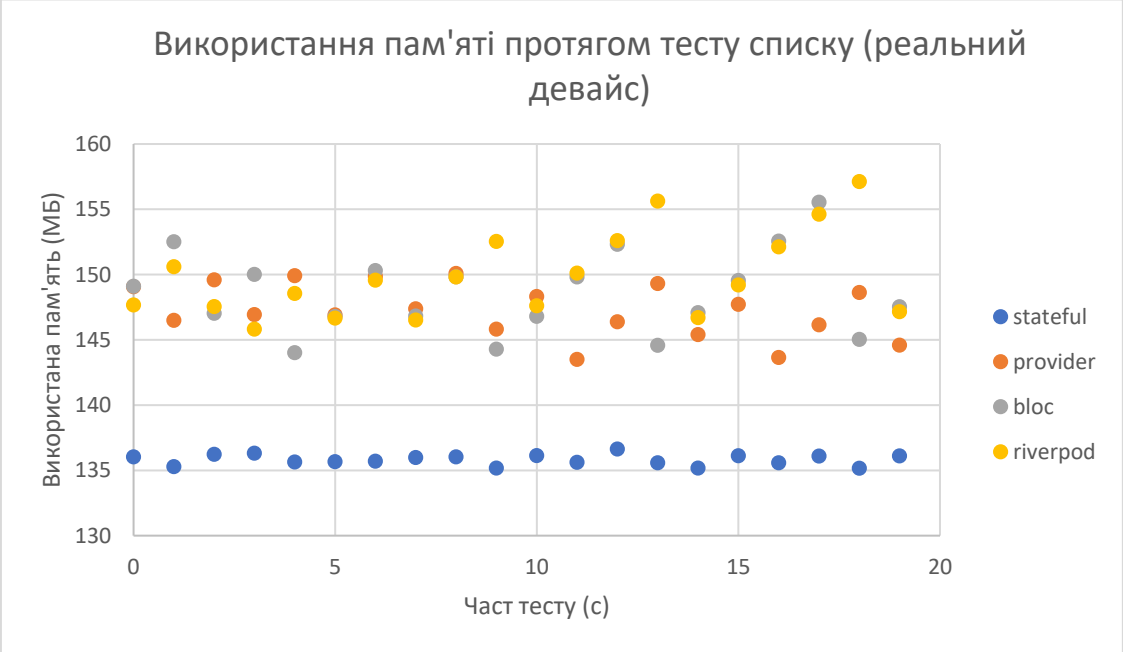

Рис. 4.29 – Використання пам’яті протягом тесту списку на реальному девайсі (Мб)

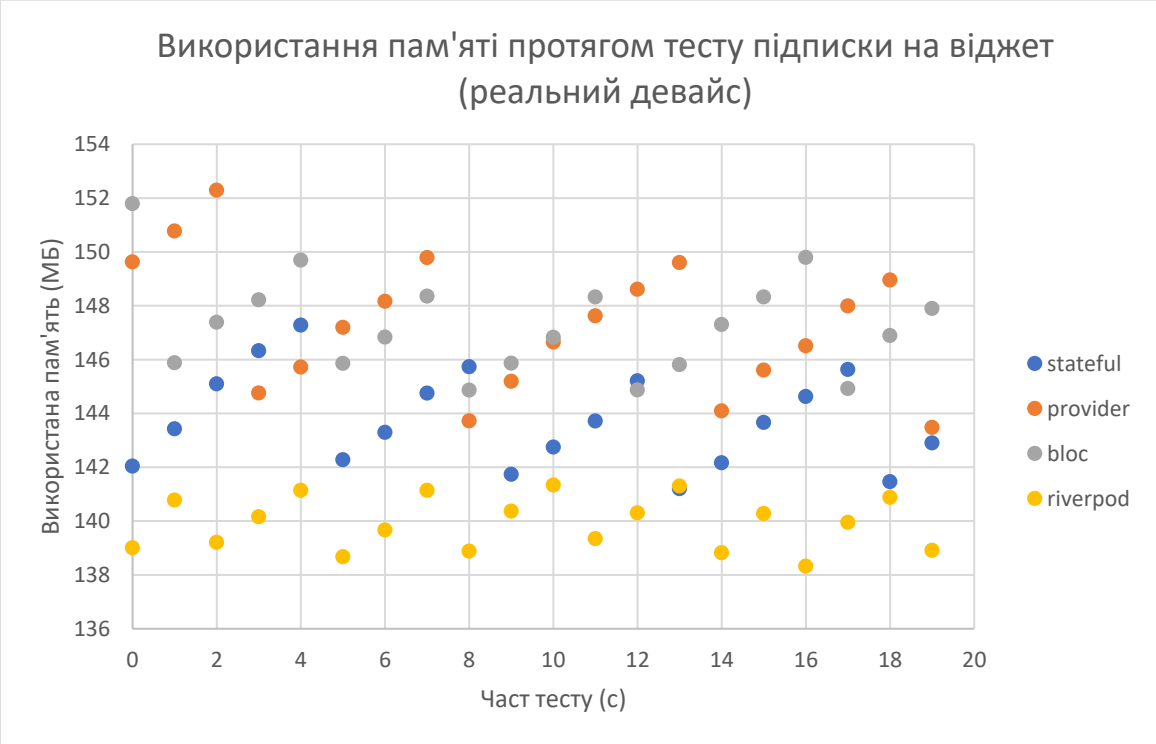

Рис. 4.30 – Використання пам’яті протягом тесту підписки на віджет на реальному девайсі (Мб)

На емуляторі (таблиця 4.23) спостерігається системно вище споживання пам’яті, що пояснюється додатковими накладними витратами на емуляцію ARM-архітектури. Stateful демонструє найнижчі показники у діапазоні 185–208 МБ, що обумовлено мінімалістичною архітектурою без додаткових абстракцій. Riverpod, навпаки, досягає максимальних значень до 217 МБ у Form Test, що може бути пов’язано з його складнішою системою залежностей та кешуванням. Provider та Bloc займають проміжне положення з показниками 191–213 МБ.

На реальному девайсі (таблиця 4.24) картина суттєво відрізняється. Загальне споживання пам’яті зменшується в середньому на 25–30% порівняно з емулятором. Riverpod демонструє найкращі результати з діапазоном 130–158 МБ, що свідчить про ефективну оптимізацію для апаратних платформ. Це особливо помітно у Form Test, де на реальному пристрої різниця між Riverpod (130 МБ) та Provider (170 МБ) становить 40 МБ — найбільший розрив серед усіх сценаріїв. Stateful зберігає стабільність з показниками 136–153 МБ.

Звернемо увагу на діаграми 4.21 – 4.30. На емуляторі чітко простежується тенденція до поступового зростання споживання пам'яті протягом виконання тестів. Наприклад, у Animation Test (рис. 4.21) усі стейт-менеджери демонструють збільшення від приблизно 197 до 208 МБ. Це може бути пов’язано з кумулятивним ефектом емуляції, де кожна ітерація тесту залишає певні сліди у пам’яті емулятора. Також, варто відзначити, що на емуляторі пам’ять рідше звільняється агресивно, оскільки ОС комп’ютера не застосовує таких же суворих обмежень, як мобільна ОС на реальному пристрої.

На реальному девайсі динаміка пам'яті має інший характер. Значення коливаються у діапазоні 130–150 МБ без чіткого тренду на зростання. Це свідчить про ефективну роботу механізмів керування пам’яттю в Android, які своєчасно звільняють невикористані ресурси. Особливо помітна різниця у поведінці між платформами у Form Test (рис. 4.28) — якщо на емуляторі

споживання пам'яті плавно зростає, то на реальному пристрої воно залишається відносно стабільним.

Таким чином, отримані дані підтверджують, що емулятор дає перебільшену оцінку споживання пам’яті — у середньому в 1.3–1.5 разів вищу за реальні показники. Це важливо враховувати при оцінці ресурсоємності додатків на етапі розробки.

Riverpod демонструє найкращу адаптацію до реальних апаратних умов. Незважаючи на високі показники на емуляторі, на фізичних пристроях він показує найнижче споживання пам’яті. Це робить його особливо привабливим для production-додатків, де ефективне використання ресурсів є критичним.

Stateful залишається найбільш стабільним рішенням з точки зору споживання пам’яті, що робить його добрим вибором для простих додатків або ситуацій, де мінімізація накладних витрат є пріоритетом.

Form Test є найбільш показовим сценарієм для порівняння ресурсоємності, оскільки саме в ньому різниця між різними підходами проявляється найяскравіше. Розробникам, які працюють зі складними формами, варто звернути особливу увагу на оптимізацію використання пам’яті.

4.9 Узагальнене порівняння

Проведене дослідження виявило суттєву контекстуальну залежність ефективності різних підходів до управління станом у Flutter-додатках. Аналіз даних з емулятора та реального пристрою демонструє, що не існує універсального «найкращого» рішення, а кожен підхід має свою нішу оптимального застосування, що визначається типом операцій, складністю додатка та цільовою платформою.

Stateful Widget, як найбільш мінімалістичний підхід, демонструє видатну продуктивність на емуляторі, особливо у сценаріях з інтенсивними анімаціями, де він забезпечує найвищу частоту кадрів. Однак на реальних пристроях ця перевага значно зменшується, особливо при порівнянні з сучасними бібліотечними рішеннями. Stateful Widget найкраще підходить для простих

додатків з обмеженою бізнес-логікою, прототипування або ситуацій, де мінімізація залежностей є пріоритетом. Його головний недолік — обмежена масштабованість та зростання складності коду при розширенні функціоналу.

Provider, будучи одним з найпопулярніших рішень, показує стабільну, але не видатну продуктивність на обох платформах. Його основна сила полягає в передбачуваності та широкому впровадженні в спільноті, що забезпечує велику кількість навчальних ресурсів та підтримку. Provider найкраще проявляє себе в додатках з акцентом на стабільність та підтримку, де критичною є простота навчання нових розробників. Однак для високонавантажених додатків з інтерактивними інтерфейсами його продуктивність може виявитись недостатньою порівняно з більш сучасними аналогами.

Bloc демонструє найбільш суперечливі результати. На емуляторі він показує себе як збалансоване рішення без явних переваг чи недоліків, але на реальних пристроях виявляються серйозні проблеми з продуктивністю, особливо у сценаріях з високими вимогами до частоти кадрів. Це робить Bloc оптимальним вибором для додатків зі складною бізнес-логікою, де переважають потоки даних та подій над інтерактивним UI, але нерекомендованим для додатків з інтенсивними анімаціями або високими вимогами до продуктивності рендерингу.

Riverpod виявляється найбільш адаптивним та ефективним рішенням, особливо на реальних пристроях. Якщо на емуляторі він показує хороші, але не видатні результати, то на апаратному обладнанні він демонструє абсолютну перевагу за ключовими показниками продуктивності. Riverpod оптимально підходить для сучасних додатків, що вимагають високої продуктивності, масштабування та підтримки складних залежностей. Його головна перевага — здатність максимально використовувати апаратні оптимізації реальних пристроїв, що робить його найкращим вибором для production-додатків з високими вимогами до користувацького досвіду.

Вибір конкретного підходу має ґрунтуватися на аналізі домінуючих операцій у додатку: для анімацій та інтерактивних інтерфейсів пріоритетом є Riverpod або

Stateful Widget; для стабільних бізнес-додатків — Provider або Riverpod; для складних потоків даних — Bloc або Riverpod. Критично важливим є тестування на реальних пристроях, оскільки результати на емуляторі можуть давати оманливу картину продуктивності, особливо для таких рішень як Bloc та Riverpod, ефективність яких суттєво залежить від апаратних оптимізацій.

Висновки до розділу 4

Комплексний аналіз експериментальних даних чітко продемонстрував контекстуальну залежність ефективності різних способів управління станом. Ключовим висновком є те, що не існує універсального «найкращого» стейт-менеджера — кожен підхід має свою нішу оптимального застосування, яка визначається типом операцій, складністю додатка та цільовою платформою. Це спростовує поширену в спільноті думку про те, що одне рішення може бути оптимальним для всіх сценаріїв.

На емуляторі найбільш виражені відмінності спостерігаються за показником FPS, де Stateful Widget демонструє беззаперечну перевагу, особливо у сценаріях з анімаціями. Однак ця перевага супроводжується обмеженою масштабованістю та нестабільністю у складніших сценаріях, таких як робота з формами. Provider, Bloc та Riverpod демонструють збалансовану, але не видатну продуктивність на емульованому середовищі.

На реальному пристрої картина кардинально змінюється. Riverpod перетворюється на абсолютного лідера, демонструючи дуже великі величини ефекту (r до 0.707) у порівнянні з іншими підходами. Це свідчить про те, що Riverpod дуже ефективно використовує апаратні оптимізації реальних пристроїв. Bloc, навпаки, показує найгірші результати на реальному девайсі, що робить його нерекомендованим для додатків з високими вимогами до продуктивності. Stateful Widget втрачає свої переваги, а Provider залишається стабільним, але неконкурентоспроможним за показниками FPS.

Найважливішим методологічним висновком є критична необхідність тестування на реальних пристроях. Результати на емуляторі дають оманливу

картину продуктивності, особливо для таких рішень як Bloc та Riverpod, ефективність яких суттєво залежить від апаратних оптимізацій. Це має важливі наслідки для практики розробки, оскільки багато команд обмежуються тестуванням на емуляторі, отримуючи неповну або навіть хибну інформацію про реальну продуктивність своїх додатків.

Статистичний аналіз на основі U-тесту Манна-Уітні підтвердив свою ефективність для порівняння продуктивності стейт-менеджерів. Комбінація p-значень та величини ефекту Cohen’s r дозволила не тільки виявити статистично значущі відмінності, але й оцінити їх практичну важливість. Ця методологія може бути успішно застосована для подібних порівняльних досліджень у галузі розробки програмного забезпечення.

ЗАГАЛЬНІ ВИСНОВКИ

В ході виконання роботи було проведене дослідження, метою якого було знайти, який із найпопулярніших способів керування станом Flutter-додатку показує найкращу швидкодію та ресурсоємність.

У ході дослідження було розроблене спеціальне програмне забезпечення, що складалось із спеціалізованого додатку написаного на мові Dart, який дозволяв на підготовленій вибірці із найпопулярніших інтерфейсних рішень проводити експерименти із використанням різних способів керування станом та коректно збирати дані для подальшого аналізу. Також, для роботи з даними були розроблені два скрипти мовою Python – один для перетворення даних із формату .json у скомпоновані Excel таблиці, другий для проведення статистичного аналізу методом Манна-Уітні.

Для дослідження знадобилася серія експериментів, оскільки у ході їх проведення виникала необхідність покращувати програмне забезпечення для виконання поставленої мети. Усі дані були зібрані та проаналізовані, і на основі цих результатів були розроблені рекомендації по використанню різних способів контролю станів у різних середовищах.

Дослідження показало, що єдиного переможця немає. Є стейт-менеджери що гарно показують себе у різних умовах, а є більш спеціалізовані. Кожен розробник має робити вибір в залежності від поставленої цілі. Дане дослідження надає змогу побачити, у яких випадках краще використовувати той чи інший варіант, адже за роки існування Flutter, не було винайдено єдиного найкращого рішення, яке могло б замінити усі існуючі аналоги.

Ця робота вносить вагомий внесок у розуміння того, як архітектурні рішення впливають на реальну продуктивність мобільних додатків. Розроблена методологія тестування та аналізу, що поєднує експериментальні виміри на різних платформах з науково обґрунтованою статистичною обробкою, може стати основою для подальших досліджень у цій галузі. Отримані результати мають безпосереднє практичне застосування — вони дозволяють розробникам

ухвалювати обґрунтовані рішення щодо вибору стейт-менеджера, уникаючи поширених помилок, коли тестування обмежується лише емулятором. Виявлення кардинальних відмінностей у поведінці одних і тих же рішень на різних середовищах виконання вказує на необхідність більш ретельного підходу до оцінки продуктивності в реальних умовах експлуатації.

Список літератури

1. Why everyone is talking about state management? [Електронний ресурс] / Режим доступу : URL : - https://www.reddit.com/r/FlutterDev/comments/1gzqvtv/why_everyone_is_talking_about_state_management/ – дата звернення 17.12.2025;

2. Struggling with State Management in Flutter: Which One to Choose? [Електронний ресурс] / Режим доступу : URL : - https://www.reddit.com/r/FlutterDev/comments/1dwiu95/struggling_with_state_management_in_flutter_which/ – дата звернення 17.12.2025;

3. StateFulWidget class doc [Електронний ресурс] / Режим доступу : URL : - https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html – дата звернення 17.12.2025;

4. InheritedWidget class doc [Електронний ресурс] / Режим доступу : URL : - https://api.flutter.dev/flutter/widgets/InheritedWidget-class.html – дата звернення 17.12.2025;

5. Pub dev збірка бібліотек Flutter [Електронний ресурс] / Режим доступу : URL : - https://pub.dev/ – дата звернення 17.12.2025;

6. Система контролю версій GitHub [Електронний ресурс] / Режим доступу : URL : - https://github.com/ – дата звернення 17.12.2025;

7. Estimating Sample Size in Computing Simulation Experiments (1979) [Електронний ресурс] / Режим доступу : URL : - https://www.jstor.org/stable/2629290 – дата звернення 17.12.2025;

8. VS Code [Електронний ресурс] / Режим доступу : URL : - https://code.visualstudio.com/docs – дата звернення 17.12.2025;

9. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other [Текст] / Mann, Henry B.; Whitney, Donald R. – 1947. – 50 - 60 с.

10. Android Studio [Електронний ресурс] / Режим доступу : URL : - https://developer.android.com/develop – дата звернення 17.12.2025;

11. «Efficient Algorithms for Mining Frequent Itemsets» [Текст] / Rakesh Agrawal – 1994. – 22с.;

12. «Statistical Power Analysis for the Behavioral Sciences» [Текст] / Jacob Cohen – 1988. – 77 - 83 с.

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ
16.09.25

ДОСЛІДЖЕННЯ ВПЛИВУ СПОСОБУ УПРАВЛІННЯ СТАНОМ FLUTTER ДОДАТКУ НА ЙОГО ШВИДКОДІЮ ТА РЕСУРСОЄМНІСТЬ

Технічне завдання
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.1525-01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
16.09.25
Керівник розробки
Віктор ШАРАВАРА
16.09.25
Виконавець
Юрій КРИВОНОСОВ
16.09.25
Нормоконтролер
Світлана ВОЛКОВА
16.09.25

2025

ЗАТВЕРДЖЕНО Додаток А

44165850.1525-01-ЛЗ

ДОСЛІДЖЕННЯ ВПЛИВУ СПОСОБУ УПРАВЛІННЯ СТАНОМ FLUTTER ДОДАТКУ НА ЙОГО ШВИДКОДІЮ ТА РЕСУРСОЄМНІСТЬ

Технічне завдання

44165850.1525-01

Листів 12

2025

ЗМІСТ

1. ВСТУП

Мобільний додаток розроблений на Flutter призначений для проведення експериментів з метою виявлення впливу конкретних способів управління станом на швидкодію та ресурсоємність Flutter додатків.

Ключові слова і терміни: додаток, спосіб управління станом, стан, стейт-менеджер, Flutter.

Область застосування: Програмне забезпечення розроблено тільки для експериментальних цілей, і його функціонал не несе більше ніякого фактичного значення.

Причини виникнення: Необхідність створити тестове середовище для проведення експериментів.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки програмного продукту є наказ №1401 ст. від 02.10.2025 затверджений ректором Українського державного університету науки та технологій щодо затвердження теми дипломного проекту. Тема дипломного проекту “Дослідження впливу способу управління станом Flutter додатку на його швидкодію та ресурсоємність”. Керівник Шаравара В. В.

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення: додаток розробляється для проведення експериментів на основі декількох тестових випадків та набору способів управління станом, показники яких стануть у подальшому об’єктом дослідження.

Експлуатаційне призначення: проведення тестувань або експериментів із доступним набором тестових випадків та способів управління станом.

4. ВИМОГИ ДО ПРОГРАМИ

4.1 Вимоги до функціональних характеристик

Програмний засіб має задовольняти наступні вимоги:

— можливість вибору конкретного способу управління станом для проведення експерименту. Наразі у списку мають бути: Stateful Widget, Provider, Bloc та Riverpod;

— можливість вибору конкретного тестового випадку із доступного списку віджетів, а саме: тестування списку з можливістю слайду, тестування текстового поля, тестування анімацій з використанням AnimatedContainer, тестування API виклику та тестування підписки на віджет;

— можливість задавати кількість ітерацій на кожний тестовий випадок;

— автоматичне проведення тестів;

— забезпечення надійного збору даних по таким показникам: час кадру, кількість кадрів на секунду, використання оперативної пам’яті, затримка відклику інтерфейсу;

— можливість копіювати результати у форматі json;

4.2 Вимоги до надійності

Вимоги до надійності наступні:

— вхідні дані вводяться у відповідному форматі в необхідне поле;

— наявність архівної копії тексту програми.

4.3 Умови експлуатації

Для забезпечення сталого функціонування програми користувачеві і програмісту необхідно дотримуватися таких умов:

— програма повинна використовуватись у приміщеннях, які відповідають умовам роботи із мобільним пристроєм [1];

З програмою може працювати людина, яка має базові навички роботи із мобільними пристроями та ознайомлена з керівництвом по використанню програми.

4.4 Вимоги до складу і параметрів технічних засобів

Мобільні пристрої:

— обсяг внутрішньої пам'яті: не менше 20 МБ вільного місця на пристрої;
— обсяг оперативної пам’яті не менше 2 ГБ;
— мінімальна тактова частота процесора не менше 1.6 ГГц.

4.5 Вимоги до інформаційної і програмної сумісності

Додаток має бути розроблений із використанням мови програмування Dart та фреймворку Flutter для проведення експериментів із даним фреймворком.

Для функціонування додатку необхідні наступні умови:

— операційна система Android версії 14.0 або вище;

4.6 Вимоги до маркування і упаковки

Упаковка програмного продукту, включаючи документацію, повинна бути захищена від пошкоджень. На упаковці повинна бути вказана назва продукту, розробник, контакти та рік розробки. На рис. 1 представлено приклад маркування.

Дослідження впливу способу управління станом Flutter додатку на його швидкодію та ресурсоємність

Розробник: Кривоносов Юрій

УДУНТ, кафедра КІТ

2025

Рис. 1. Приклад маркування

4.7 Вимоги до транспортування і зберігання

Транспортування повинне забезпечувати збереження програмного продукту, його цілісність і запобігання несанкціонованого доступу до нього. Програмний виріб міститься на оптичному носії даних типу CD-R і повинен мати відповідну упаковку для захисту від механічних ушкоджень та атмосферного впливу (пластиковий футляр). Також програма може бути передана у вигляді .aab файлу через захищене програмне сховище або по захищеному каналу зв’язку.

5. ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу програмної документації мають входити:

— текст програми;

— керівництво користувача.

Вся документація до програмного додатку повинна задовольняти вимогам до програмної документації [1].

6. СТАДІЇ І ЕТАПИ РОЗРОБКИ

Стадії та етапи розробки програми представлені у табл. 1.

Таблиця 1. Стадії та етапи розробки

Стадії розробки	Зміст робіт	Терміни виконання
1.Технічне завдання (ТЗ)	Постановка задачі	02.10.2025 – 03.10.2025
	Дослідження області способів управління станом та вибір тестових випадків	04.10.2025 – 06.10.2025
	Визначення вимог до програми.	07.10.2025 – 10.10.2025
	Узгодження та затвердження ТЗ	11.10.2025 – 14.10.2025
2. Робочий проект	Розробка та програмування логіки додатку	25.10.2025 –31.10.2025
	Розробка і реалізація інтерфейсу користувача	01.11.2025 – 05.11.2025
	Відлагодження програми додатку	06.11.2025 - 14.11.2025
	Узгодження програмної документації	15.11.2025 - 16.11.2025

7. ПОРЯДОК КОНТРОЛЮ І ПРИЙМАННЯ

Контроль за виконанням роботи здійснює розробник.

Прийом програми здійснює науковий керівник даного проекту.

Програма має пройти:

— тестування розробником;

— оцінку первинних результатів науковим керівником;

— можливе виправлення помилок та вдосконалення методів тестування для проведення основної роботи;

— затвердження проекту науковим керівником.

8. БІБЛІОГРАФІЧНИЙ СПИСОК

1. Івченко, Ю.М. Основи стандартизації програмних систем [Текст]: методичні вказівки до дипломного проектування та лабораторних робіт /уклад.: Ю. М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009. - 38 с.

ДОДАТОК Б

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ
16.09.25

ДОСЛІДЖЕННЯ ВПЛИВУ СПОСОБУ УПРАВЛІННЯ СТАНОМ FLUTTER ДОДАТКУ НА ЙОГО ШВИДКОДІЮ ТА РЕСУРСОЄМНІСТЬ

Керівництво користувача
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.1525-01 І3 01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
16.09.25
Керівник розробки
Віктор ШАРАВАРА
16.09.25
Виконавець
Юрій КРИВОНОСОВ
16.09.25
Нормоконтролер
Світлана ВОЛКОВА
16.09.25

2025

ЗАТВЕРДЖЕНО
44165850.1525-01 І3 01-ЛЗ

ДОДАТОК Б

ДОСЛІДЖЕННЯ ВПЛИВУ СПОСОБУ УПРАВЛІННЯ СТАНОМ FLUTTER ДОДАТКУ НА ЙОГО ШВИДКОДІЮ ТА РЕСУРСОЄМНІСТЬ

Керівництво користувача

44165850.1525-01 І3 01

Листів 15

2025

АННОТАЦІЯ

Документ 44165850.1525-01 І3 01 “Дослідження впливу способу управління станом Flutter додатку на його швидкодію та ресурсоємність” входить до складу програмної документації на мобільний додаток для проведення експерименту для дослідження впливу способу управління станом Flutter додатку на його швидкодію та ресурсоємність.

У даному документі представлено керівництво користувача.

ЗМІСТ

1. ВСТУП

Додаток розроблений для проведення експерименту для виявлення впливу використання конкретних способів управління стану на Flutter додаток.

Для виконання цієї задачі програма надає користувачу тестове середовище, у якому він може обрати конкретні тестові випадки та способи управління станом із представлених у наборі програми та провести автоматичне тестування. Усі вихідні дані доступні у форматі json.

Список показників, які виміряє програма є строго визначеним, і для їх зміни необхідна модернізація програмного забезпечення.

2. ПРИЗНАЧЕННЯ ТА УМОВИ ЗАСТОСУВАННЯ

Додаток призначений для проведення автоматичних тестів та збору показників швидкодії та ресурсоємності Flutter додатку в залежності від обраного способу управління станом на конкретних тестових випадках.

2.1. Вимоги до складу і параметрів технічних засобів

Мобільні пристрої:

— обсяг внутрішньої пам’яті: не менше 20 МБ вільного місця на пристрої;
— обсяг оперативної пам’яті не менше 2 ГБ;
— мінімальна тактова частота процесора не менше 1.6 ГГц;

2.2. Вимоги до інформаційної і програмної сумісності

Для функціонування додатку необхідні наступні умови:

— операційна система Android версії 14.0 або вище;

2.3. Вимоги до вхідних даних

Вибір тестових випадків і способів контролю стану відбувається через список доступних елементів. Для вводу кількості ітерацій необхідно ввести ціле число.

3. ПІДГОТОВКА ДО РОБОТИ

3.1. Склад і зміст дистрибутивного носія даних

— .aab файл із Android додатком;

— .docx файл із керівництвом користувача.

3.2. Порядок загрузки даних і програм

1) Прочитати керівництво користувача;
2) Завантажити .aab файл на девайс та відкрити його.
3) Відкрити додаток та потрапити на головний екран (рис. 1);

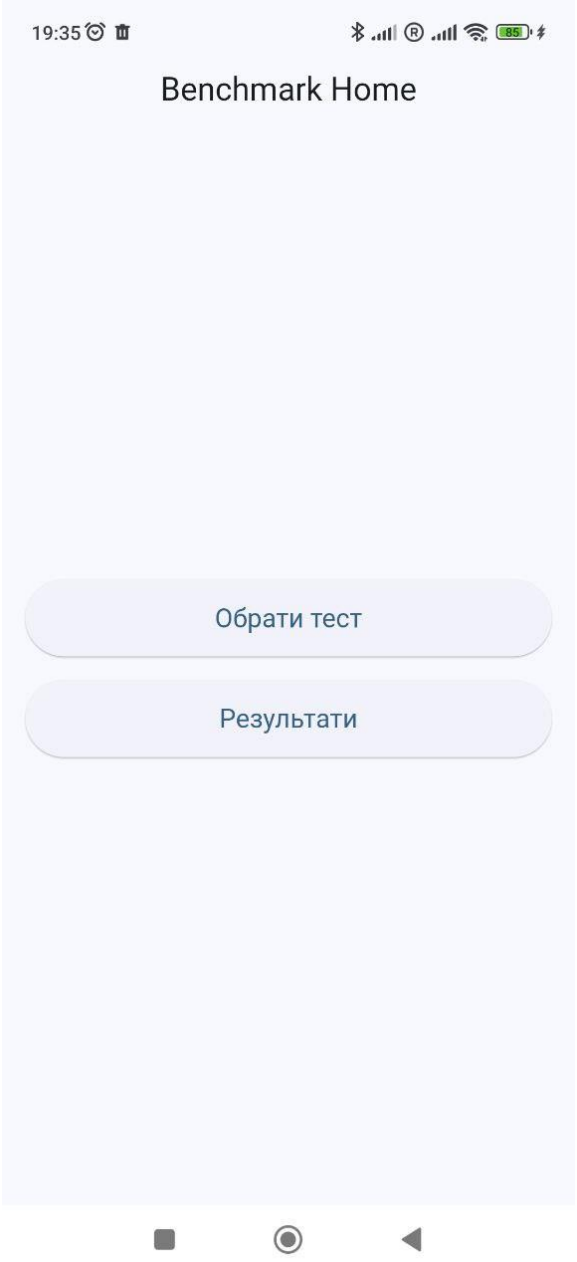

Рис. 1 – Головний екран

4) Натиснути на кнопку “Обрати тест”, що відкриє екран налаштувань тесту та обрати спосіб контролю станом, тестовий випадок та кількість ітерацій, та перейти на екран тестування, натиснувши кнопку у низу екрану (рис. 2);

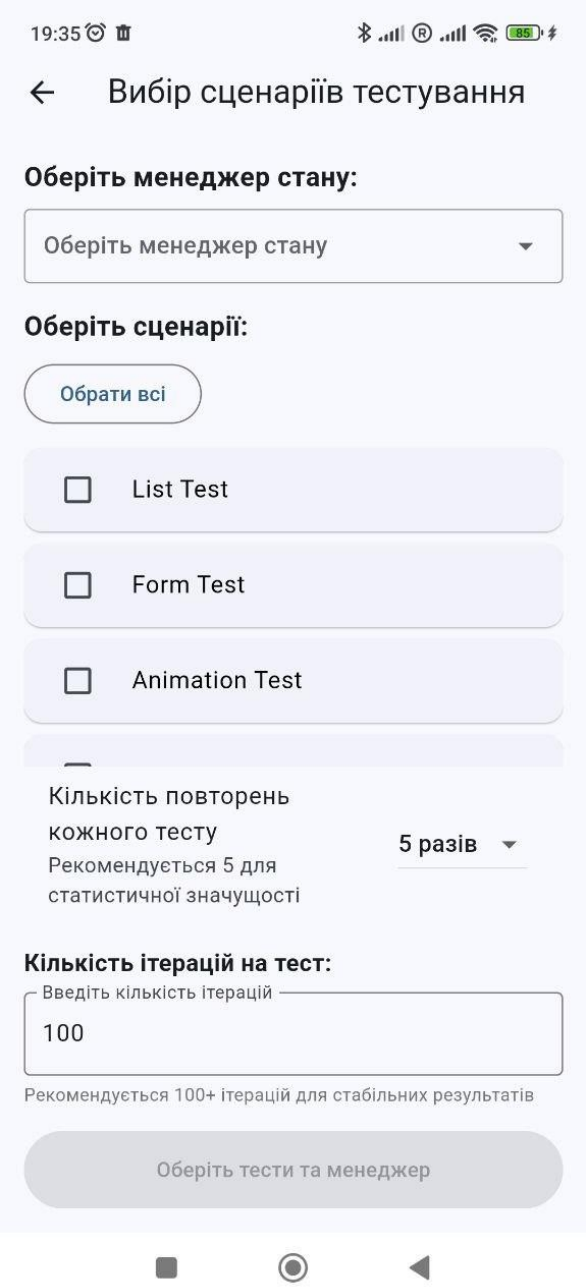

Рис. 2 – Екран налаштувань експерименту

5) Запустити тестування (рис. 3-4);

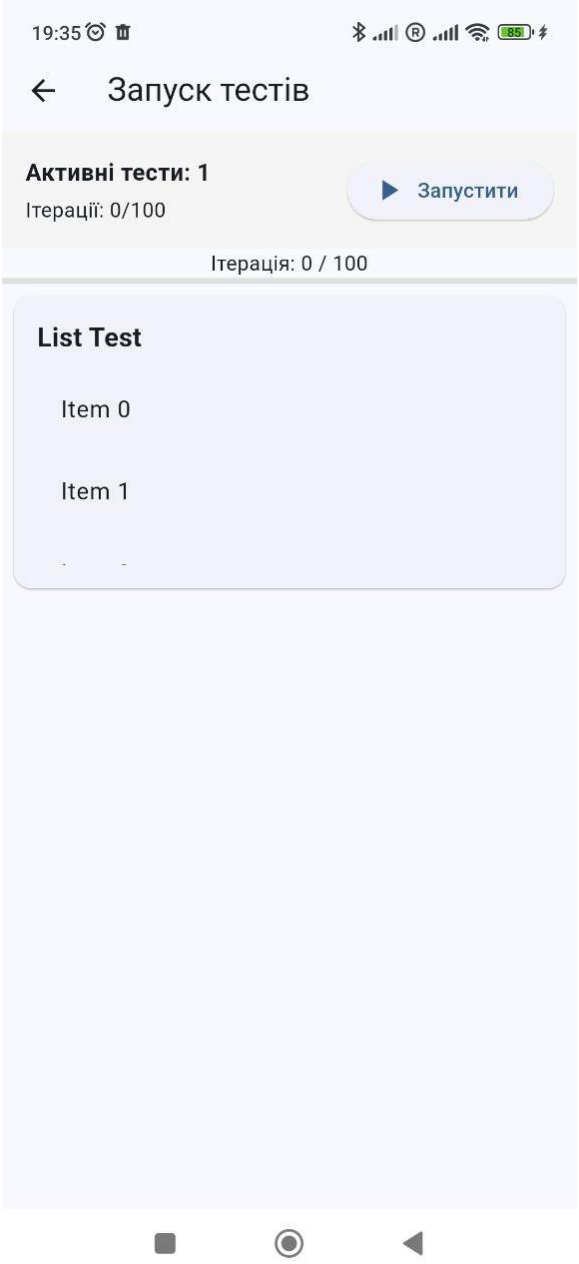

Рис. 3 – Екран тестувань у режимі очікування запуску тестів

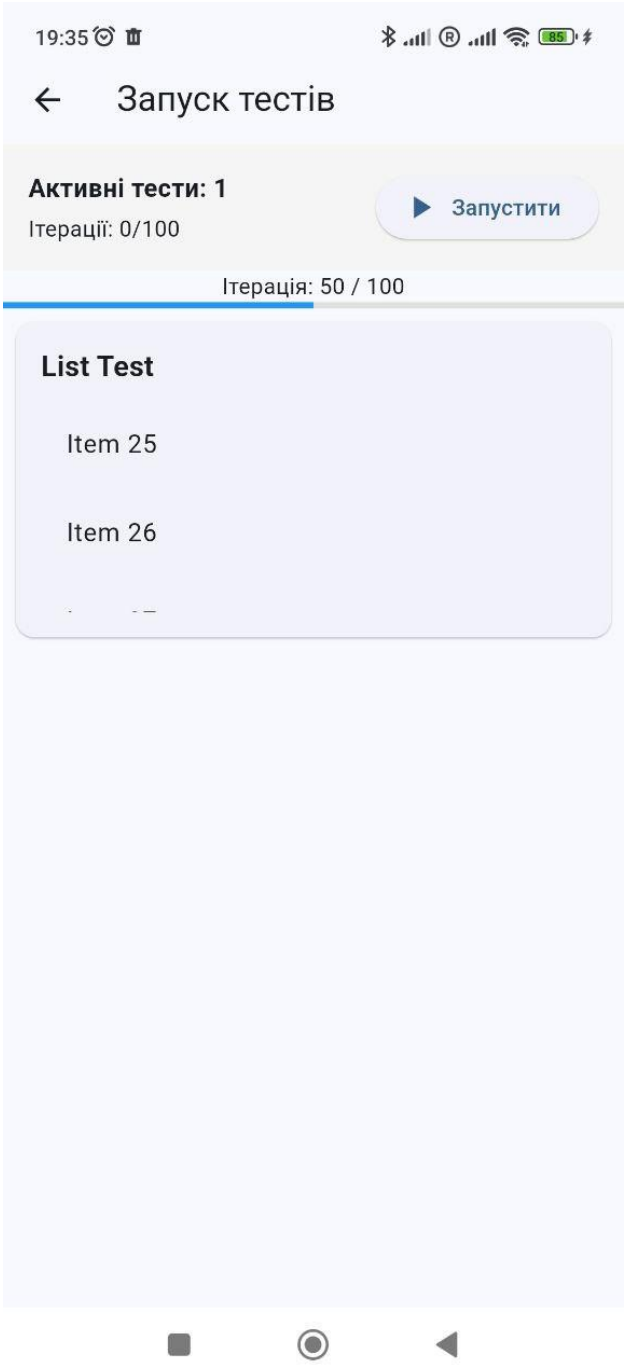

Рис. 4 – Екран тестувань у режимі тестування

6) Після проведення тестування відбудеться автоматичний перехід на результуючий екран, де можна скопіювати результати натисканням на кнопку копіювання у верхньому правому куті екрану (рис. 5);

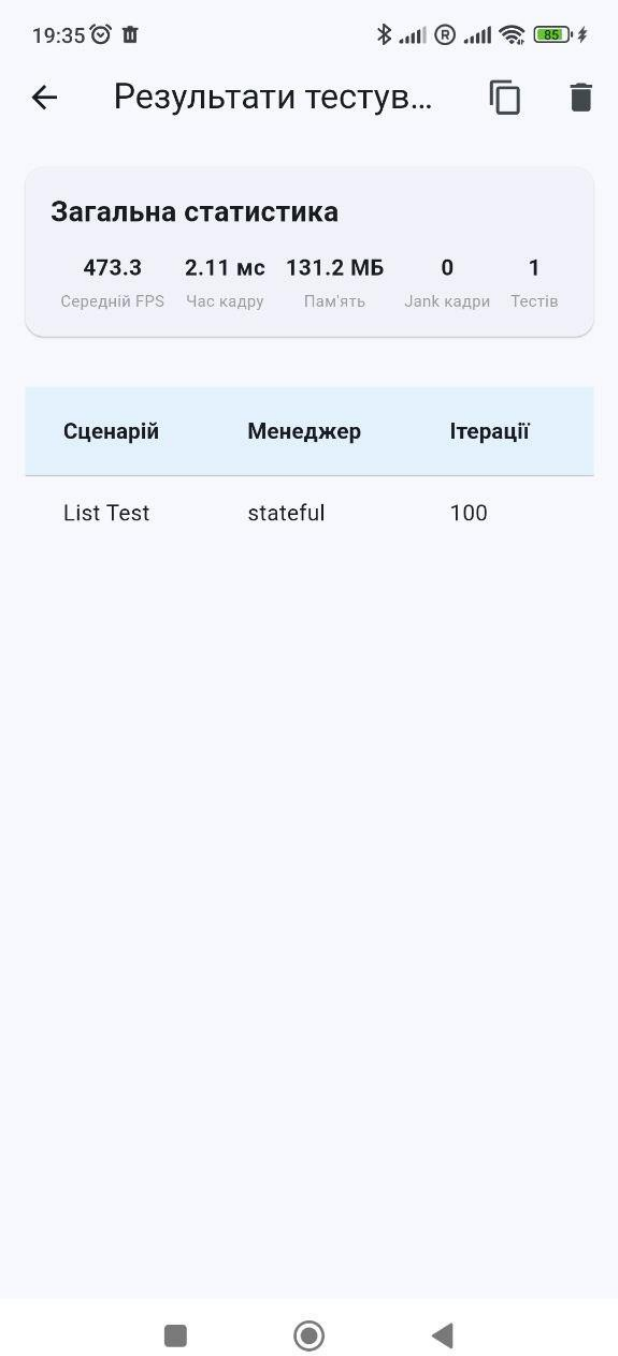

Рис. 5 – Результуючий екран

7) Повернутись на головний екран та повторити алгоритм для наступного експерименту, або ж вийти із додатку за допомогою навігаційних кнопок телефону.

4. ОПИС ОПЕРАЦІЙ

Після запуску програми (шлях до файлу з розширенням .aab) на екрані мобільного пристрою з’являється екран завантаження та головний екран.

Головний екран виконує лише навігаційні функції. На ньому є змога перейти на екран налаштувань тесту та екран результатів. Для цього необхідно натиснути на кнопки “Обрати тест” та “Результати” відповідно.

Екран налаштувань тесту має такі функції:

— Вибір способу керування станом – натискання на поле з вибором стейт-менеджерів та вибір із випадаючого списку;

— Вибір тестового випадку методом кліку на відповідний елемент слайдового списку;

— Вибір кількості ітерацій методом введення цілого числа ітерацій у текстове поле;

— Перехід до екрану тестування натисканням на кнопку внизу екрану (кнопка неактивна, доки не обрані усі налаштування із пунктів вище).

Екран автотестування має поле, на якому відображаються UI елементи тестових випадків. Воно не є інтерактивним, а є лише плацдармом для перебігу тестів. Також цей екран має кнопку запуску/зупинки тестів та стрічку прогресу, яка вказує скільки ітерацій вже виконано. Перехід на екран результатів відбувається автоматично після завершення автотестів.

Екран результатів показує дані у вигляді таблиці та спільної статистики, що окремо винесена у верхню частину екрану. Також він має функції повернення на головний екран, копіювання результатів у формат json та функцію очистки даних (їх видалення).

5. АВАРІЙНІ СИТУАЦІЇ

Додаток був розроблений таким чином, щоб бути максимально захищеним від аварійних ситуацій. На рівні введення даних неможливо ввести щось не так, оскільки списки і валідація не дозволять це зробити.

Але додаток є полем для експерименту, тому при деяких навантаженнях, на невідповідних платформах, додаток може зупинитись і аварійно завершити роботу. Це також є частиною експериментального процесу, оскільки аварійне завершення програми може вказувати на проблеми обраної платформи. При виникненні такої ситуації необхідно перезапустити додаток.

6. РЕКОМЕНДАЦІЇ ЩОДО ЗАСВОЄННЯ

Для початку роботи з додатком необхідно ознайомитися з даним керівництвом, відкрити додаток і спробувати налаштувати перший тест. Після чого ви маєте розробити чіткий алгоритм проведення експериментів та не забувати копіювати дані у його ході, щоб запобігати їх втраті.

Додаток В

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ
16.09.25

ДОСЛІДЖЕННЯ ВПЛИВУ СПОСОБУ УПРАВЛІННЯ СТАНОМ FLUTTER ДОДАТКУ НА ЙОГО ШВИДКОДІЮ ТА РЕСУРСОЄМНІСТЬ

Тези
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.1525-01 90-99 01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
16.09.25
Керівник розробки
Віктор ШАРАВАРА
16.09.25
Виконавець
Юрій КРИВОНОСОВ
16.09.25
Нормоконтролер
Світлана ВОЛКОВА
16.09.25

2025

ЗАТВЕРДЖЕНО
44165850.1525-01 90-99 01-ЛЗ

Додаток В

ДОСЛІДЖЕННЯ ВПЛИВУ СПОСОБУ УПРАВЛІННЯ СТАНОМ FLUTTER ДОДАТКУ НА ЙОГО ШВИДКОДІЮ ТА РЕСУРСОЄМНІСТЬ

Тези

44165850.1525-01 90-99 01
Листів 6

2025


МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УКРАЇНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ НАУКИ І ТЕХНОЛОГІЙ
ABSTRACTS
OF THE XIX INTERNATIONAL CONFERENCE
«MODERN INFORMATION AND COMMUNICATION TECHNOLOGIES ON A TRANSPORT, IN INDUSTRY AND EDUCATION»
18-19, December, 2025
СУЧАСНІ ІНФОРМАЦІЙНІ ТА КОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ НА ТРАНСПОРТІ, В ПРОМИСЛОВОСТІ І ОСВІТІ
ПРИСВЯЧЕНО ПАМ'ЯТІ ПРОФЕСОРА ІГОРЯ ЖУКОВИЦЬКОГО
ТЕЗИ
XIX МІЖНАРОДНОЇ НАУКОВО-ПРАКТИЧНОЇ КОНФЕРЕНЦІЇ
18-19 ГРУДНЯ 2025
ДНІПРО
2025

Міністерство освіти і науки України

Український державний університет науки і технологій

ТЕЗИ
XIX Міжнародної науково-практичної конференції
«СУЧАСНІ ІНФОРМАЦІЙНІ ТА КОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ НА ТРАНСПОРТІ, В ПРОМИСЛОВОСТІ І ОСВІТІ»
Присвячено пам'яті Ігоря ЖУКОВИЦЬКОГО

ABSTRACTS
of the XIX International Conference
«MODERN INFORMATION AND COMMUNICATION TECHNOLOGIES ON A TRANSPORT, IN INDUSTRY AND EDUCATION»
Dedicated to the memory of Igor ZHUKOVYTSKY

18.12.2025 – 19.12.2025

Дніпро
2025

УДК 658.512.2:681.3.06

Сучасні інформаційні та комунікаційні технології на транспорті, в промисловості і освіті: Тези XIX Міжнародної науково-практичної конференції (Дніпро, 18-19 грудня 2025 р.). – Д.: УДУНТ, 2025. – 172 с.

У збірнику представлені тези доповідей XIX Міжнародної науково-практичної конференції «Сучасні інформаційні та комунікаційні технології на транспорті, в промисловості і освіті», яка відбулася 18-19 грудня 2025 року в Українському державному університеті науки та технологій в онлайн форматі. Конференцію присвячено пам'яті Ігоря ЖУКОВИЦЬКОГО, доктора технічних наук, професора кафедри електронних обчислювальних машин (УДУНТ, м. Дніпро). Розглянуто результати теоретичних і експериментальних досліджень, а також проблемні питання функціонування та перспективи розвитку інформаційних технологій транспорту, промисловості й освіти.

Збірник призначений для науково-технічних працівників залізниць, підприємств транспорту, викладачів вищих навчальних закладів, докторантів, аспірантів і студентів.

Адреса редакційної колегії:
49010, м. Дніпро, вул. Лазаряна, 2, УДУНТ

Тези доповідей друкуються мовою оригіналу в редакції авторів.

Дослідження впливу способу управління станом Flutter додатку на його швидкодію та ресурсоємність

Кривоносов Ю. А., Шаравара В.В. Український державний університет науки і технологій. Україна.

У сучасній розробці мобільних додатків на Flutter вибір способу управління станом є критично важливим рішенням, яке безпосередньо впливає на продуктивність та стабільність застосунку. Існує широкий спектр бібліотек для управління станом - від простих StatefulWidget до складних рішень на кшталт BLoC, Provider та Riverpod. Однак практично відсутні систематичні дослідження, які б кількісно оцінювали вплив цих рішень на реальну продуктивність додатку. Це дослідження спрямоване на заповнення цієї прогалини шляхом об'єктивного вимірювання ключових показників продуктивності для різних підходів до управління станом.

Мета дослідження полягає у комплексному аналізі продуктивності різних способів управління станом у Flutter додатках. Конкретними завданнями є: розробка методології тестування, створення тестового середовища, проведення експериментів з чотирма типами стейт-менеджерів (Stateful, Provider, BLoC, Riverpod), аналіз отриманих даних та формування практичних рекомендацій для розробників. Результати дослідження дозволять обґрунтовано вибирати оптимальні рішення для різних типів застосунків.

Для проведення експерименту було розроблено спеціалізоване програмне забезпечення. Flutter додаток здатний генерувати тестові сценарії з використанням будь-якого з чотирьох реалізованих стейт-менеджерів та автоматично збирати показники продуктивності: кількість кадрів на секунду (FPS), кількість оперативної пам'яті, час відтворення кадрів та затримки відтворення кадрів. Додатково створено Python скрипт для автоматизованої обробки даних, який генерує зведені таблиці у Excel та порівняльні діаграми у форматі PDF на основі JSON-даних, що експортуються з тестового додатку.

Методологія експерименту передбачала проведення 500 ітерацій кожного тестового сценарію для кожного стейт-менеджера. Тестування проводилося у релізній збірці на емуляторі та реальному пристрої для забезпечення репрезентативності результатів. Дослідження охопило п'ять типових сценаріїв роботи додатку: робота зі списками, обробка форм, анімації, взаємодія з API та робота з підписками. Такий обсяг вибірки забезпечує статистичну значущість отриманих результатів.

Результати дослідження виявили статистично значущі відмінності у продуктивності між різними підходами до управління станом. Застосування непараметричних статистичних критеріїв підтвердило наявність систематичних відмінностей у розподілах ключових показників продуктивності між досліджуваними стейт-менеджерами. Аналіз розподілів часу відтворення кадрів та частоти оновлення екрану виявив характерні паттерни продуктивності для кожного з рішень, при цьому було встановлено, що деякі підходи демонструють більш стабільну поведінку з меншою дисперсією показників, тоді як інші характеризуються значними коливаннями продуктивності.

Висновки дослідження дозволяють сформулювати практичні рекомендації для розробників Flutter додатків. Отримані результати свідчать про те, що вибір стейт-менеджера повинен ґрунтуватися на конкретних вимогах до продуктивності та типі взаємодії з користувачем. Дослідження також підтвердило, що жоден з аналізованих підходів не є універсальним рішенням, а оптимальний вибір залежить від специфіки застосунку та пріоритетних вимог до продуктивності, що підкреслює необхідність індивідуального підходу до архітектури кожного проєкту.

Додаток Г

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ
16.09.25

ДОСЛІДЖЕННЯ ВПЛИВУ СПОСОБУ УПРАВЛІННЯ СТАНОМ FLUTTER ДОДАТКУ НА ЙОГО ШВИДКОДІЮ ТА РЕСУРСОЄМНІСТЬ

Текст програми
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.1525-01 12-01 ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
16.09.25
Керівник розробки
Віктор ШАРАВАРА
16.09.25
Виконавець
Юрій КРИВОНОСОВ
16.09.25
Нормоконтролер
Світлана ВОЛКОВА
16.09.25

2025

ЗАТВЕРДЖЕНО

Додаток Г

44165850.1525-01 12-01 ЛЗ

ДОСЛІДЖЕННЯ ВПЛИВУ СПОСОБУ УПРАВЛІННЯ СТАНОМ FLUTTER ДОДАТКУ НА ЙОГО ШВИДКОДІЮ ТА РЕСУРСОЄМНІСТЬ

Текст програми

44165850.1525-01 12-01

Листів 38

2025

АННОТАЦІЯ

Документ 44165850.1525-01 12-01 “ Дослідження впливу способу управління станом Flutter додатку на його швидкодію та ресурсоємність. Текст програми” входить до складу програмної документації на мобільний додаток для проведення експериментів за даною темою.

У даному документі представлено текст програми.

ЗМІСТ

1. Зміст файлу main.dart

```
// Розробник: Кривоносов Ю.А.
// Точка входу у програму
import 'package:flutter/material.dart';
import 'package:magi_work/app.dart';

void main() {
  runApp(const MainApp());
}
```

2. Зміст файлу app.dart

```
// Точка входу у додаток
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:magi_work/blocs/test_config_cubit.dart';
import 'package:magi_work/screens/result_screen.dart';

import 'screens/home_screen.dart';
import 'screens/select_manager_screen.dart';
import 'screens/select_test_screen.dart';

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

  @override
  Widget build(BuildContext context) {
    return BlocProvider(
      create: (_) => TestConfigCubit(),
      child: MaterialApp(
        debugShowCheckedModeBanner: false,
        title: 'State Managers Benchmark',
        theme: ThemeData(
          useMaterial3: true,
                                        colorScheme: ColorScheme.fromSeed(seedColor: Colors.blue),
        ),
        initialRoute: '/',
        routes: {
          '/': (context) => const HomeScreen(),
            '/select_manager': (context) => const SelectManagerScreen(),
              '/select_tests': (context) => const SelectTestScreen(),
                '/results': (context) => const ResultsScreen(),
        },
      ),
    );
  }
}
```

3. Зміст файлу test_config_cubit.dart

```
// Управління станом обраних тестів та менеджерів
```

```
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:magi_work/blocs/test_runner.dart';
import 'package:magi_work/models/test_results.dart';

enum StateManagerType { stateful, provider, bloc, riverpod }

// Новий класс для конфигурации теста
class TestConfig {
  final String scenario;
  final String manager;

  const TestConfig(this.scenario, this.manager);
}

// Новий статус тестирования
enum TestStatus { idle, running, completed }

class TestConfigState {
  final StateManagerType? selectedManager;
  final List<String> selectedTests;
  final int iterations;
  final int repetitions;
  final List<TestResult> results;
  final TestStatus testStatus;
  final double progress;
  const TestConfigState({
    this.selectedManager,
    this.selectedTests = const [],
    this.iterations = 100,
    this.repetitions = 5,
    this.results = const [],
    this.testStatus = TestStatus.idle,
    this.progress = 0.0,
  });

  TestConfigState copyWith({
    StateManagerType? selectedManager,
    List<String>? selectedTests,
    int? iterations,
    int? repetitions,
    List<TestResult>? results,
    TestStatus? testStatus,
    double? progress,
  }) {
    return TestConfigState(
      selectedManager: selectedManager ?? this.selectedManager,
      selectedTests: selectedTests ?? this.selectedTests,
      iterations: iterations ?? this.iterations,
      repetitions: repetitions ?? this.repetitions,
      results: results ?? this.results,
      testStatus: testStatus ?? this.testStatus,
      progress: progress ?? this.progress,
    );
  }
}

class TestConfigCubit extends Cubit<TestConfigState> {
  TestConfigCubit() : super(const TestConfigState());

  static const List<String> allManagers = [
    'stateful',
    'provider',
```

```
    'bloc',
    'riverpod'
  ];

  static const List<String> allTests = [
    'List Test',
    'Form Test',
    'Animation Test',
    'API Test',
    'Subscription Test'
  ];

    void selectManager(StateManagerType manager) {
        emit(state.copyWith(selectedManager: manager));
  }

  void toggleTest(String testName) {
            final updated = List<String>.from(state.selectedTests);
    if (updated.contains(testName)) {
      updated.remove(testName);
    } else {
      updated.add(testName);
    }
        emit(state.copyWith(selectedTests: updated));
  }

  void setIterations(int count) {
    emit(state.copyWith(iterations: count));
  }

  void setRepetitions(int count) {
    emit(state.copyWith(repetitions: count));
  }

  void selectAllManagers() {}

  void selectAllTests() {
        emit(state.copyWith(selectedTests: List<String>.from(allTests)));
  }

  void addResult(TestResult result) {
            final updated = List<TestResult>.from(state.results)..add(result);
    emit(state.copyWith(results: updated));
  }

  void clearResults() {
    emit(state.copyWith(results: []));
  }

  void reset() {
    emit(const TestConfigState());
  }

  Future<void> runAllTests() async {
    if (state.selectedTests.isEmpty) {
      throw Exception('Оберіть хоча б один тест');
    }

        final selectedManagers = _getSelectedManagers();
    if (selectedManagers.isEmpty) {
      throw Exception('Оберіть хоча б один менеджер стану');
```

```
    }

    emit(state.copyWith(
      testStatus: TestStatus.running,
      progress: 0.0,
      results: [],
    ));

    final results = <TestResult>[];
    final testRunner = TestRunner();

    final testCombinations = _generateTestCombinations(selectedManagers)
      ..shuffle();
    final totalTests = testCombinations.length * state.repetitions;
    var completedTests = 0;

    try {
      for (final config in testCombinations) {
        for (int repetition = 0; repetition < state.repetitions; repetition++) {
          final result = await testRunner.runSingleTest(
            scenario: config.scenario,
            manager: config.manager,
            iterations: state.iterations,
          );

          results.add(result);
          completedTests++;

          final progress = completedTests / totalTests;
          emit(state.copyWith(
            results: List<TestResult>.from(results),
            progress: progress,
          ));

          if (repetition < state.repetitions - 1) {
            await Future.delayed(const Duration(seconds: 2));
          }
        }

        await Future.delayed(const Duration(seconds: 1));
      }

      emit(state.copyWith(
        testStatus: TestStatus.completed,
        progress: 1.0,
      ));
    } catch (e) {
      emit(state.copyWith(
        testStatus: TestStatus.idle,
      ));
      rethrow;
    }
  }

  List<String> _getSelectedManagers() {
    if (state.selectedManager == null) return [];

    switch (state.selectedManager!) {
      case StateManagerType.stateful:
        return ['stateful'];
      case StateManagerType.provider:
        return ['provider'];
      case StateManagerType.bloc:
```

```
      return ['bloc'];
    case StateManagerType.riverpod:
      return ['riverpod'];
  }
 }

                                List<TestConfig>
_generateTestCombinations(List<String>
managers) {
   return [
    for (final scenario in state.selectedTests)
          for (final manager in managers)
TestConfig(scenario, manager),
   ];
 }

 void stopTests() {
   emit(state.copyWith(
     testStatus: TestStatus.idle,
   ));
 }
}
```

4. Зміст файлу test_runner.dart

```
// Модуль вибору налаштувань тесту
import
'package:magi_work/core/metrics/stats_recor
der.dart';
import
'package:magi_work/core/metrics/widget_cou
nter.dart';
import
'package:magi_work/models/test_results.dart'
;

class TestRunner {
 static const int WARM_UP_ITERATIONS =
20;
 static const int MEASURED_ITERATIONS =
100;

 Future<TestResult> runSingleTest({
   required String scenario,
   required String manager,
   required int iterations,
 }) async {

              await Future.delayed(const
Duration(seconds: 1));

   final recorder = StatsRecorder();
   recorder.startListening();

   try {

    for (int i = 0; i < WARM_UP_ITERATIONS;
i++) {
      await _runScenario(scenario, manager,
recorder.widgetCounter);
                 await Future.delayed(const
Duration(milliseconds: 16));
    }

    for (int i = 0; i < iterations; i++) {
      recorder.startFrame();
      await _runScenario(scenario, manager,
recorder.widgetCounter);
      recorder.endFrame();
                 await Future.delayed(const
Duration(milliseconds: 16));
    }
```

```
        return recorder.buildResult(scenario, manager, iterations);
    } finally {
      recorder.stopListening();
    }
  }

  Future<void> _runScenario(String scenario, String manager, WidgetCounter counter) async {}
}
```

5. Зміст файлу fps_tracker.arb

```
/// Трекер для заміру FPS з детальними даними для графіків
import 'dart:math';

class FpsTracker {
  final List<double> _timingFrameMs = [];
  final List<double> _manualFrameMs = [];
  final List<double> _allFrameTimes = [];

  void addTimingFrame(double ms) {
    _timingFrameMs.add(ms);
    _allFrameTimes.add(ms);
  }

  void addManualFrame(double ms) {
    _manualFrameMs.add(ms);
    _allFrameTimes.add(ms);
  }

  void clear() {
    _timingFrameMs.clear();
    _manualFrameMs.clear();
    _allFrameTimes.clear();
  }

    List<double> get allFrameMs => [..._timingFrameMs, ..._manualFrameMs];

  // РОЗПОДІЛ ЧАСУ КАДРІВ
        Map<double, int> getAggregatedFrameTimes(int buckets) {
    if (_allFrameTimes.isEmpty) return {};

    final maxTime = _allFrameTimes.reduce((a, b) => a > b ? a : b);
    final bucketSize = maxTime / buckets;

    Map<double, int> distribution = {};
    for (double time in _allFrameTimes) {
      double bucket = (time / bucketSize).floor() * bucketSize;
      distribution[bucket] = (distribution[bucket] ?? 0) + 1;
    }

    return distribution;
  }

  // НОВИЙ МЕТОД - РОЗПОДІЛ FPS
  Map<double, int> getAggregatedFps(int buckets) {
    if (_allFrameTimes.isEmpty) return {};

    // Конвертуємо часи кадрів у FPS і приводимо до double
    final allFps = _allFrameTimes.map((ms) => ms > 0 ? 1000.0 / ms : 0.0).toList();
```

```
  // Фільтруємо аномально високі значення FPS
  final validFps = allFps.where((fps) => fps <= 1000.0).toList();
  if (validFps.isEmpty) return {};

  final maxFps = validFps.reduce((a, b) => a > b ? a : b);
  final bucketSize = maxFps / buckets;

  Map<double, int> distribution = {};
  for (double fps in validFps) {
    double bucket = (fps / bucketSize).floor() * bucketSize;
    distribution[bucket] = (distribution[bucket] ?? 0) + 1;
  }

  return distribution;
}

  //  ОСНОВНІ МЕТРИКИ
  double get avgFrameTimeMs {
    final all = allFrameMs;
    if (all.isEmpty) return 0.0;
    return all.reduce((a, b) => a + b) / all.length;
  }

  double get avgFps {
    final frameMs = avgFrameTimeMs;
    if (frameMs <= 0) return 0.0;
    return 1000.0 / frameMs;
  }

  //  ВИПРАВЛЕНА ДИСПЕРСІЯ FPS
  double get varianceFps {
    final allFps = _allFrameTimes.map((ms) => ms > 0 ? 1000.0 / ms : 0).toList();
    if (allFps.isEmpty) return 0.0;

    // Фільтруємо аномальні значення
    final validFps = allFps.where((fps) => fps <= 1000).toList();
    if (validFps.isEmpty) return 0.0;

    final mean = validFps.reduce((a, b) => a + b) / validFps.length;
    return validFps.map((fps) => pow(fps - mean, 2)).reduce((a, b) => a + b) / validFps.length;
  }

  //  СТАНДАРТНЕ ВІДХИЛЕННЯ FPS
  double get stdDevFps => sqrt(varianceFps);

  //  ДИСПЕРСІЯ ЧАСУ КАДРІВ
  double get varianceFrameTimeMs {
    final all = allFrameMs;
    if (all.isEmpty) return 0.0;
    final mean = avgFrameTimeMs;
    return all.map((x) => pow(x - mean, 2)).reduce((a, b) => a + b) / all.length;
  }

  double get stdDevFrameTimeMs => sqrt(varianceFrameTimeMs);

  double get percentile95FrameTimeMs {
    final sorted = List<double>.from(allFrameMs)..sort();
    if (sorted.isEmpty) return 0.0;
    final index = (sorted.length * 0.95).floor();
```

```
    return sorted[index];
  }

      int get jankFramesCount => allFrameMs.where((ms) => ms > 33.33).length;

  double get jankFramesPercent {
    final all = allFrameMs;
    if (all.isEmpty) return 0.0;
    return (jankFramesCount / all.length) * 100;
  }
}
```

6. Зміст файлу latency_tracker.arb

```
import 'dart:math';

/// Трекер затримки кадрів у секундах з детальними даними

class LatencyTracker {
  final List<double> _latMs = [];

  void recordLatencyMs(double ms) {
    _latMs.add(ms);
  }

  void clear() {
    _latMs.clear();
  }

  double get avgLatencyMs {
    if (_latMs.isEmpty) return 0.0;
    return _latMs.reduce((a, b) => a + b) / _latMs.length;
  }

  /// Дисперсія затримки
  double get varianceLatencyMs {
    if (_latMs.isEmpty) return 0.0;
    final mean = avgLatencyMs;
    return _latMs.map((x) => pow(x - mean, 2)).reduce((a, b) => a + b) / _latMs.length;
  }

  /// Стандартне відхилення затримки
    double get stdDevLatencyMs => sqrt(varianceLatencyMs);

  double get minLatencyMs => _latMs.isEmpty ? 0.0 : _latMs.reduce((a, b) => a < b ? a : b);
    double get maxLatencyMs => _latMs.isEmpty ? 0.0 : _latMs.reduce((a, b) => a > b ? a : b);

  //НОВИЙ МЕТОД ДЛЯ АГРЕГОВАНИХ ДАНИХ
          Map<double, int> getAggregatedLatencies(int buckets) {
    if (_latMs.isEmpty) return {};

    final maxLatency = _latMs.reduce((a, b) => a > b ? a : b);
    final bucketSize = maxLatency / buckets;

    Map<double, int> distribution = {};
    for (double latency in _latMs) {
          double bucket = (latency / bucketSize).floor() * bucketSize;
      distribution[bucket] = (distribution[bucket] ?? 0) + 1;
```

```
    }

    return distribution;
  }
}
```

7. Зміст файлу memory_tracker.dart

```
// Трекер оперативної пам'яті з піковими значеннями
import 'dart:io';
import 'dart:math';

class MemoryTracker {
  final List<double> memoryMb = [];

  double _peakMemoryMb = 0.0;

  void record() {
    try {
      final usedBytes = ProcessInfo.currentRss;
      final usedMb = usedBytes / (1024 * 1024);
      memoryMb.add(usedMb);

      if (usedMb > _peakMemoryMb) {
        _peakMemoryMb = usedMb;
      }
    } catch (e) {
      memoryMb.add(-1.0);
    }
  }

  void clear() {
    memoryMb.clear();
    _peakMemoryMb = 0.0;
  }

  double get avgMemoryMb {
    final valid = memoryMb.where((v) => v >= 0).toList();
    if (valid.isEmpty) return 0.0;
    return valid.reduce((a, b) => a + b) / valid.length;
  }

  double get peakMemoryMb => _peakMemoryMb;

  double get stdDevMemoryMb {
    final valid = memoryMb.where((v) => v >= 0).toList();
    if (valid.isEmpty) return 0.0;
    final mean = avgMemoryMb;
```

```
    final variance = valid.map((x) => pow(x - mean, 2)).reduce((a, b) => a + b) / valid.length;
    return sqrt(variance);
  }

  // НОВИЙ МЕТОД ДЛЯ АГРЕГОВАНИХ ДАНИХ
  List<double> getMemorySamples(int sampleCount) {
    final valid = memoryMb.where((v) => v >= 0).toList();
    if (valid.isEmpty) return [];

    List<double> samples = [];
    final step = valid.length ~/ sampleCount;

    for (int i = 0; i < valid.length; i += step) {
      if (i < valid.length) {
        samples.add(valid[i]);
      }
      if (samples.length >= sampleCount) break;
    }

    // Додаємо останнє значення, якщо ще є місце
    if (samples.length < sampleCount && valid.isNotEmpty) {
      samples.add(valid.last);
    }

    return samples;
  }
}
```

8. Зміст файлу stats_recorder.dart

```
// Універсальний рекордер параметрів з детальними даними
import 'package:flutter/scheduler.dart';
import 'package:magi_work/models/test_results.dart';
import 'fps_tracker.dart';
import 'latency_tracker.dart';
import 'memory_tracker.dart';
import 'widget_counter.dart';

class StatsRecorder {
  final FpsTracker fpsTracker;
  final LatencyTracker latencyTracker;
  final MemoryTracker memoryTracker;
  final WidgetCounter widgetCounter;

  int? _iterationStartMicros;

  StatsRecorder()
      : fpsTracker = FpsTracker(),
```

```
      latencyTracker = LatencyTracker(),
      memoryTracker = MemoryTracker(),
      widgetCounter = WidgetCounter() {
  reset();
 }


     void _onTimings(List<FrameTiming> timings) {
  for (final t in timings) {
          final ms = (t.buildDuration + t.rasterDuration).inMicroseconds / 1000.0;
    fpsTracker.addTimingFrame(ms);
  }
 }


 void startListening() {
  try {
   SchedulerBinding.instance.addTimingsCallback(_onTimings);
  } catch (e) {
   // Обробка помилок
  }
 }


 void stopListening() {
  try {
   SchedulerBinding.instance.removeTimingsCallback(_onTimings);
  } catch (e) {}
 }


 void reset() {
  fpsTracker.clear();
  latencyTracker.clear();
  memoryTracker.clear();
  widgetCounter.reset();
  _iterationStartMicros = null;
 }


 void startFrame() {
          _iterationStartMicros = DateTime.now().microsecondsSinceEpoch;
 }


 void endFrame() {
          final now = DateTime.now().microsecondsSinceEpoch;
  if (_iterationStartMicros != null) {
   final ms = (now - _iterationStartMicros!) / 1000.0;
    fpsTracker.addManualFrame(ms);
    _iterationStartMicros = null;
  }


  memoryTracker.record();
 }


  void recordLatencyMs(double ms) => latencyTracker.recordLatencyMs(ms);


  TestResult buildResult(String scenario, String manager, int iterations) {
  return TestResult(
    scenarioName: scenario,
    stateManager: manager,
    iterations: iterations,


    // ОСНОВНІ МЕТРИКИ
    avgFps: fpsTracker.avgFps,
                avgFrameTimeMs: fpsTracker.avgFrameTimeMs,
```

```
                avgLatencyMs: latencyTracker.avgLatencyMs,
                ramUsageMb: memoryTracker.avgMemoryMb,
                widgetRebuilds: widgetCounter.rebuilds,

                // СТАТИСТИЧНІ МЕТРИКИ
                stdDevFrameTimeMs: fpsTracker.stdDevFrameTimeMs,
                percentile95FrameTimeMs: fpsTracker.percentile95FrameTimeMs,
                jankFramesCount: fpsTracker.jankFramesCount,
                jankFramesPercent: fpsTracker.jankFramesPercent,
                minLatencyMs: latencyTracker.minLatencyMs,
                maxLatencyMs: latencyTracker.maxLatencyMs,

                // ДИСПЕРСІЇ ТА СТАНДАРТНІ ВІДХИЛЕННЯ
                varianceFrameTimeMs: fpsTracker.varianceFrameTimeMs,
                varianceLatencyMs: latencyTracker.varianceLatencyMs,
                stdDevLatencyMs: latencyTracker.stdDevLatencyMs,
                varianceFps: fpsTracker.varianceFps,
                stdDevFps: fpsTracker.stdDevFps,

                // ПАМ'ЯТЬ
                peakMemoryMb: memoryTracker.peakMemoryMb,
                stdDevMemoryMb: memoryTracker.stdDevMemoryMb,

                // АГРЕГОВАНІ ДАНІ ДЛЯ ГРАФІКІВ
                frameTimeDistribution: fpsTracker.getAggregatedFrameTimes(15),
                fpsDistribution: fpsTracker.getAggregatedFps(15),
                latencyDistribution: latencyTracker.getAggregatedLatencies(15),
                memoryUsageOverTime: memoryTracker.getMemorySamples(20),
            );
        }
    }
```

9. Зміст файлу widget_counter.dart

```
// Міксин для отримання кількості збудованих виджетів
import 'package:flutter/material.dart';

class WidgetCounter {
  int _rebuilds = 0;

  WidgetCounter();

  int get rebuilds => _rebuilds;

  void increment() => _rebuilds++;

  void reset() => _rebuilds = 0;
}

class WidgetCounterScope extends InheritedWidget {
  final WidgetCounter counter;

  const WidgetCounterScope({
    required this.counter,
```

```
    required super.child,
    super.key,
  });

    static WidgetCounter of(BuildContext context) {
                final result = context.dependOnInheritedWidgetOfExactType<WidgetCounterScope>();
            assert(result != null, 'No WidgetCounterScope found in context');
    return result!.counter;
  }

  @override
                                        bool updateShouldNotify(WidgetCounterScope oldWidget) => false;
}

mixin WidgetCounterMixin<T extends StatefulWidget> on State<T> {
  void incrementRebuild() {
                final counter = WidgetCounterScope.of(context);
    counter.increment();
  }
}
```

10. Зміст файлу bloc_adapter.dart

```
// Адаптер для менеджеру Bloc
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'state_manager_base.dart';

/// Проста подія
class UpdateEvent {}

/// Стейт BLoC
class CounterState {
  final int value;
  CounterState(this.value);
}

/// BLoC
class TestBloc extends Bloc<UpdateEvent, CounterState> {
  TestBloc() : super(CounterState(0)) {
        on<UpdateEvent>((event, emit) => emit(CounterState(state.value + 1)));
  }

  void trigger() => add(UpdateEvent());
}

/// Адаптер
class BlocAdapter extends StatelessWidget implements StateManagerAdapter {
   final Widget Function(BuildContext, int) builder;
  late final TestBloc bloc;

      BlocAdapter({Key? key, required this.builder}) : super(key: key) {
    bloc = TestBloc();
  }

  @override
  Widget build(BuildContext context) {
    return BlocProvider.value(
```

```
      value: bloc,
                  child: BlocBuilder<TestBloc, CounterState>(
                  builder: (context, state) => builder(context, state.value),
      ),
    );
  }

  @override
  String get name => "BLoC";

  @override
  void bindScenario(dynamic scenario) {
    scenario.onAction = bloc.trigger;
  }

  @override
  void dispose() {
    bloc.close();
  }
}
```

11. Зміст файлу provider_adapter.dart

```
// Адаптер для менеджеру Provider
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'state_manager_base.dart';

class ProviderNotifier extends ChangeNotifier {
  void trigger() {
    notifyListeners();
  }
}

class ProviderAdapter extends StatelessWidget implements StateManagerAdapter {
  final Widget Function() builder;
      final ProviderNotifier notifier = ProviderNotifier();

    ProviderAdapter({Key? key, required this.builder}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return ChangeNotifierProvider.value(
      value: notifier,
      child: Consumer<ProviderNotifier>(
        builder: (_, __, ___) => builder(),
      ),
    );
  }

  @override
  String get name => "Provider";

  @override
  void bindScenario(dynamic scenario) {
    scenario.onAction = notifier.trigger;
  }

  @override
  void dispose() {
    notifier.dispose();
```

```
  }
}
```

12. Зміст файлу riverpod_adapter.dart

```
// Адаптер для менеджеру Riverpod
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'state_manager_base.dart';

final counterProvider = StateProvider<int>((ref) => 0);

class RiverpodAdapter extends ConsumerStatefulWidget {
  final Widget Function(WidgetRef, int) builder;

  const RiverpodAdapter({Key? key, required this.builder}) : super(key: key);

  @override
  ConsumerState<RiverpodAdapter> createState() => _RiverpodAdapterState();
}

class _RiverpodAdapterState extends ConsumerState<RiverpodAdapter>
    implements StateManagerAdapter {
  @override
  Widget build(BuildContext context) {
    final value = ref.watch(counterProvider);
    return widget.builder(ref, value);
  }

  @override
  String get name => "Riverpod";

  @override
  void bindScenario(dynamic scenario) {
    scenario.onAction = () {
      ref.read(counterProvider.notifier).state++;
    };
  }

  @override
  void dispose() {
    super.dispose();
  }
}
```

13. Зміст файлу state_manager_base.dart

```
// Контрактор для стейт менеджерів
abstract class StateManagerAdapter {
  /// Назва менеджеру
  String get name;

  /// Запуск: підключення сценарію
  void bindScenario(dynamic scenario);

  /// Очищення після тесту
  void dispose();
}
```

14. Зміст файлу stateful_adapter.dart

```
// Адаптер для менеджеру Stateful
import 'package:flutter/material.dart';
import 'state_manager_base.dart';

class StatefulAdapter extends StatefulWidget {
  final Widget Function(void Function()) builder;

  const StatefulAdapter({Key? key, required this.builder}) : super(key: key);

  @override
  State<StatefulAdapter> createState() => _StatefulAdapterState();
}

class _StatefulAdapterState extends State<StatefulAdapter> implements StateManagerAdapter {
  late VoidCallback _rebuild;

  @override
  void initState() {
    super.initState();
    _rebuild = () => setState(() {});
  }

  @override
  Widget build(BuildContext context) {
    return widget.builder(_rebuild);
  }

  @override
  String get name => "StatefulWidget";

  @override
  void bindScenario(dynamic scenario) {
    scenario.onAction = _rebuild;
  }

  @override
  void dispose() {
    super.dispose();
  }
}
```

15. Зміст файлу test_base.dart

```
// Класс універсальної моделі тесту
abstract class LoadTest {
  final String name;
  final int iterations;

  LoadTest(this.name, {this.iterations = 100});

  /// підготовка до тесту
  Future<void> setup();

  /// Виконання одної ітерації тесту
  Future<void> runIteration();

  /// завершення тесту, видалення ресурсів
  Future<void> teardown();
}
```

16. Зміст файлу test_results.dart

```
// Модель результату тесту з детальними даними для графіків
class TestResult {
  final String scenarioName;
  final String stateManager;
  final int iterations;

  // Основні метрики
  final double avgFps;
  final double avgFrameTimeMs;
  final double avgLatencyMs;
  final double ramUsageMb;
  final int widgetRebuilds;

  // Статистичні метрики
  final double stdDevFrameTimeMs;
  final double percentile95FrameTimeMs;
  final int jankFramesCount;
  final double jankFramesPercent;
  final double minLatencyMs;
  final double maxLatencyMs;

  // Дисперсії
  final double varianceFrameTimeMs;
  final double varianceLatencyMs;
  final double stdDevLatencyMs;
  final double varianceFps;
  final double stdDevFps;

  // Пам'ять
  final double peakMemoryMb;
  final double stdDevMemoryMb;

  // АГРЕГОВАНІ ДАНІ ДЛЯ ГРАФІКІВ
  final Map<double, int> frameTimeDistribution;
  final Map<double, int> fpsDistribution;
  final Map<double, int> latencyDistribution;
  final List<double> memoryUsageOverTime;

  final DateTime timestamp;

  TestResult({
    required this.scenarioName,
    required this.stateManager,
    required this.iterations,
    required this.avgFps,
    required this.avgFrameTimeMs,
    required this.avgLatencyMs,
    required this.ramUsageMb,
    required this.widgetRebuilds,
    required this.stdDevFrameTimeMs,
    required this.percentile95FrameTimeMs,
    required this.jankFramesCount,
    required this.jankFramesPercent,
    required this.minLatencyMs,
    required this.maxLatencyMs,
    required this.varianceFrameTimeMs,
    required this.varianceLatencyMs,
    required this.stdDevLatencyMs,
    required this.varianceFps,
    required this.stdDevFps,
    required this.peakMemoryMb,
    required this.stdDevMemoryMb,
    required this.frameTimeDistribution,
```

```
    required this.fpsDistribution,
    required this.latencyDistribution,
    required this.memoryUsageOverTime,
    DateTime? timestamp,
  }) : timestamp = timestamp ?? DateTime.now();

  factory TestResult.fromJson(Map<String, dynamic> json) {
    return TestResult(
      scenarioName: json["scenarioName"] as String,
      stateManager: json["stateManager"] as String,
      iterations: json["iterations"] as int,
      avgFps: (json["avgFps"] as num).toDouble(),
      avgFrameTimeMs: (json["avgFrameTimeMs"] as num).toDouble(),
      avgLatencyMs: (json["avgLatencyMs"] as num).toDouble(),
      ramUsageMb: (json["ramUsageMb"] as num).toDouble(),
      widgetRebuilds: json["widgetRebuilds"] as int,
      stdDevFrameTimeMs: (json["stdDevFrameTimeMs"] as num).toDouble(),
      percentile95FrameTimeMs: (json["percentile95FrameTimeMs"] as num).toDouble(),
      jankFramesCount: json["jankFramesCount"] as int,
      jankFramesPercent: (json["jankFramesPercent"] as num).toDouble(),
      minLatencyMs: (json["minLatencyMs"] as num).toDouble(),
      maxLatencyMs: (json["maxLatencyMs"] as num).toDouble(),
      varianceFrameTimeMs: (json["varianceFrameTimeMs"] as num).toDouble(),
      varianceLatencyMs: (json["varianceLatencyMs"] as num).toDouble(),
      stdDevLatencyMs: (json["stdDevLatencyMs"] as num).toDouble(),
      varianceFps: (json["varianceFps"] as num).toDouble(),
      stdDevFps: (json["stdDevFps"] as num).toDouble(),
      peakMemoryMb: (json["peakMemoryMb"] as num).toDouble(),
      stdDevMemoryMb: (json["stdDevMemoryMb"] as num).toDouble(),
      frameTimeDistribution: _parseDistribution(json["frameTimeDistribution"]),
      fpsDistribution: _parseDistribution(json["fpsDistribution"]),
      latencyDistribution: _parseDistribution(json["latencyDistribution"]),
      memoryUsageOverTime: List<double>.from(json["memoryUsageOverTime"] ?? []),
      timestamp: DateTime.parse(json["timestamp"] as String),
    );
  }

  Map<String, dynamic> toJson() => {
    "scenarioName": scenarioName,
    "stateManager": stateManager,
    "iterations": iterations,
    "avgFps": avgFps,
    "avgFrameTimeMs": avgFrameTimeMs,
    "avgLatencyMs": avgLatencyMs,
    "ramUsageMb": ramUsageMb,
```

```
      "widgetRebuilds": widgetRebuilds,
                        "stdDevFrameTimeMs": stdDevFrameTimeMs,
                    "percentile95FrameTimeMs": percentile95FrameTimeMs,
      "jankFramesCount": jankFramesCount,
                        "jankFramesPercent": jankFramesPercent,
      "minLatencyMs": minLatencyMs,
      "maxLatencyMs": maxLatencyMs,
                    "varianceFrameTimeMs": varianceFrameTimeMs,
                        "varianceLatencyMs": varianceLatencyMs,
      "stdDevLatencyMs": stdDevLatencyMs,
      "varianceFps": varianceFps,
      "stdDevFps": stdDevFps,
      "peakMemoryMb": peakMemoryMb,
      "stdDevMemoryMb": stdDevMemoryMb,
                    "frameTimeDistribution": _distributionToJson(frameTimeDistribution),
                        "fpsDistribution": _distributionToJson(fpsDistribution),
                        "latencyDistribution": _distributionToJson(latencyDistribution),
                    "memoryUsageOverTime": memoryUsageOverTime,
                        "timestamp": timestamp.toIso8601String(),
    };

  // Допоміжні методи залишаються незмінними
      static Map<double, int> _parseDistribution(dynamic jsonData) {
  if (jsonData == null) return {};

  final Map<double, int> result = {};
    final Map<String, dynamic> data = Map<String, dynamic>.from(jsonData);

  data.forEach((key, value) {
    result[double.parse(key)] = value as int;
  });

  return result;
 }

      static Map<String, int> _distributionToJson(Map<double, int> distribution) {
  final Map<String, int> result = {};
  distribution.forEach((key, value) {
    result[key.toString()] = value;
  });
  return result;
 }
}
```

17. Зміст файлу home_screen.dart

```
// Головний екран
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:magi_work/screens/result_screen.dart';
import 'package:magi_work/screens/select_test_screen.dart';
import 'package:magi_work/blocs/test_config_cubit.dart';
```

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

  @override
  Widget build(BuildContext context) {
    return BlocProvider(
      create: (_) => TestConfigCubit(),
      child: Scaffold(
        appBar: AppBar(
          title: const Text("Benchmark Home"),
          centerTitle: true,
        ),
        body: Padding(
          padding: const EdgeInsets.all(16),
          child: Column(
                                    mainAxisAlignment:
MainAxisAlignment.center,
            children: [
              BuildButton(
                label: "Обрати тест",
                onTap: () => Navigator.push(
                  context,
                  MaterialPageRoute(
                    builder: (_) => BlocProvider.value(
                                                    value:
context.read<TestConfigCubit>(),
                      child: const SelectTestScreen(),
                    ),
                  ),
                ),
              ),
              const SizedBox(height: 16),
              BuildButton(
                label: "Резултати",
                onTap: () => Navigator.push(
                  context,
                  MaterialPageRoute(
                                builder: (_) => const
ResultsScreen(),
                  ),
                ),
              ),
            ],
          ),
        ),
      ),
    );
  }
}


class BuildButton extends StatelessWidget {
  final String label;
  final VoidCallback onTap;
    const BuildButton({super.key, required
this.label, required this.onTap});


  @override
  Widget build(BuildContext context) {
    return SizedBox(
      width: double.infinity,
      child: ElevatedButton(
        onPressed: onTap,
        style: ElevatedButton.styleFrom(
                                  padding: const
EdgeInsets.symmetric(vertical: 16),
          textStyle: const TextStyle(fontSize: 18),
        ),
        child: Text(label),
      ),
```

```
    );
  }
}
```

18. Зміст файлу result_screen.dart

```
// Екран результатів
import 'dart:convert';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:magi_work/models/test_results.dart';
import 'package:magi_work/blocs/test_config_cubit.dart';
import 'package:magi_work/screens/home_screen.dart';

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("Результати тестування"),
        leading: IconButton(
          icon: const Icon(Icons.arrow_back),
          onPressed: () {
            Navigator.of(context).pushAndRemoveUntil(
              MaterialPageRoute(
                builder: (context) =>
                  const HomeScreen()),
              (Route<dynamic> route) => false,
            );
          },
          tooltip: 'Повернутися на головний екран',
        ),
        actions: [
          IconButton(
            icon: const Icon(Icons.copy),
            onPressed: () {
              _copyResultsToClipboard(context);
            },
            tooltip: 'Копіювати результати в буфер обміну',
          ),
          IconButton(
            icon: const Icon(Icons.delete),
            onPressed: () {
              context.read<TestConfigCubit>().clearResults();
            },
            tooltip: 'Очистити результати',
          ),
        ],
      ),
      body: BlocBuilder<TestConfigCubit, TestConfigState>(
        builder: (context, state) {
          if (state.results.isEmpty) {
            return const Center(
              child: Column(
                mainAxisAlignment: MainAxisAlignment.center,
```

```
            children: [
              Icon(Icons.assessment, size: 64, color: Colors.grey),
              SizedBox(height: 16),
              Text(
                "Немає результатів для відображення",
              style: TextStyle(fontSize: 18, color: Colors.grey),
              ),
            ],
          ),
        );
      }


      return Column(
        children: [
          // Сводная статистика
          _buildSummaryCard(state.results),
          const SizedBox(height: 16),


          // Детальная таблица
          Expanded(
            child: Padding(
                padding: const EdgeInsets.symmetric(horizontal: 16),
              child: SingleChildScrollView(
                scrollDirection: Axis.vertical,
                child: SingleChildScrollView(
                  scrollDirection: Axis.horizontal,
                  child: DataTable(
                    headingRowColor: MaterialStateProperty.resolveWith(
                      (states) => Colors.blue[50],
                    ),
                    columns: const [
                      DataColumn(
                        label: Text("Сценарій",
                          style:
                            TextStyle(fontWeight: FontWeight.bold))),
                      DataColumn(
                        label: Text("Менеджер",
                          style:
                            TextStyle(fontWeight: FontWeight.bold))),
                      DataColumn(
                        label: Text("Ітерації",
                          style:
                            TextStyle(fontWeight: FontWeight.bold))),
                      DataColumn(
                        label: Text("FPS",
                          style:
                            TextStyle(fontWeight: FontWeight.bold))),
                      DataColumn(
                          label: Text("Frame Time (мс)",
                          style:
                            TextStyle(fontWeight: FontWeight.bold))),
                      DataColumn(
                       label: Text("±σ Frame Time",
                          style:
                            TextStyle(fontWeight: FontWeight.bold))),
                      DataColumn(
                          label: Text("95% Frame Time",
                          style:
                            TextStyle(fontWeight: FontWeight.bold))),
```

```
                DataColumn(
                  label: Text("Jank %",
                    style:
                      TextStyle(fontWeight:
FontWeight.bold))),
                DataColumn(
                  label: Text("Latency (мс)",
                    style:
                      TextStyle(fontWeight:
FontWeight.bold))),
                DataColumn(
                    label:  Text("Min/Max
Latency",
                    style:
                      TextStyle(fontWeight:
FontWeight.bold))),
                DataColumn(
                  label: Text("RAM (МБ)",
                    style:
                      TextStyle(fontWeight:
FontWeight.bold))),
                DataColumn(
                  label: Text("Rebuilds",
                    style:
                      TextStyle(fontWeight:
FontWeight.bold))),
              ],
                                    rows:
state.results.map((TestResult result) {
                return DataRow(
                  cells: [
                    DataCell(Text(result.scena
rioName)),
                     DataCell(Text(result.state
Manager)),
                     DataCell(Text(result.iterati
ons.toString())),
                    DataCell(Text(result.avgFp
s.toStringAsFixed(1))),
                        DataCell(Text(
                          result.avgFrameTimeMs
.toStringAsFixed(2))),
                        DataCell(Text(
                          "±${result.stdDevFrame
TimeMs.toStringAsFixed(2)}")),
                         DataCell(Text(result.perce
ntile95FrameTimeMs
                          .toStringAsFixed(2))),
                        DataCell(Text(
                          "${result.jankFramesPer
cent.toStringAsFixed(1)}%")),
                        DataCell(
                           Text(result.avgLatency
Ms.toStringAsFixed(2))),
                        DataCell(Text(
                          "${result.minLatencyMs.
toStringAsFixed(2)}/${result.maxLatencyMs.t
oStringAsFixed(2)}")),
                        DataCell(
                           Text(result.ramUsageM
b.toStringAsFixed(1))),
                         DataCell(Text(result.widge
tRebuilds.toString())),
                      ],
                    );
                  }).toList(),
                ),
              ),
            ),
          ),
          const SizedBox(height: 16),
        ],
      );
    },
```

```
    ),
   );
 }


 void _copyResultsToClipboard(BuildContext context) {
                final state = context.read<TestConfigCubit>().state;
   if (state.results.isEmpty) {
    ScaffoldMessenger.of(context).showSnackBar(
       const SnackBar(content: Text('Немає результатів для копіювання')),
    );
    return;
   }

   try {
    // Создаем структурированные данные для экспорта
    final exportData = {
                                'exportedAt': DateTime.now().toIso8601String(),
     'totalTests': state.results.length,
                                 'summary': _calculateSummary(state.results),
      'results': state.results.map((result) => result.toJson()).toList(),
    };

    // Конвертируем в красивый JSON
                const encoder = JsonEncoder.withIndent('  ');
                final jsonString = encoder.convert(exportData);

    // Копируем в буфер обмена
      Clipboard.setData(ClipboardData(text: jsonString));

    ScaffoldMessenger.of(context).showSnackBar(
                  const SnackBar(content: Text('Результати скопійовано в буфер обміну')),
    );
   } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('Помилка при копіюванні: $e')),
    );
   }
 }

          Map<String, dynamic> _calculateSummary(List<TestResult> results) {
   final avgFps =
     results.map((r) => r.avgFps).reduce((a, b) => a + b) / results.length;
   final avgFrameTime =
                  results.map((r) => r.avgFrameTimeMs).reduce((a, b) => a + b) /
       results.length;
     final avgRam = results.map((r) => r.ramUsageMb).reduce((a, b) => a + b) /
      results.length;
   final avgLatency =
                  results.map((r) => r.avgLatencyMs).reduce((a, b) => a + b) /
       results.length;
   final avgStdDev =
                  results.map((r) => r.stdDevFrameTimeMs).reduce((a, b) => a + b) /
       results.length;
```

```
    final totalJankFrames =
                                results.map((r)       =>
r.jankFramesCount).reduce((a, b) => a + b);


    return {
      'averageFps': avgFps,
      'averageFrameTimeMs': avgFrameTime,
      'averageRamUsageMb': avgRam,
      'averageLatencyMs': avgLatency,
                    'averageStdDevFrameTimeMs':
avgStdDev,
      'totalJankFrames': totalJankFrames,
      'totalWidgetRebuilds':
                                 results.map((r)      =>
r.widgetRebuilds).reduce((a, b) => a + b),
    };
  }


                                                 Widget
_buildSummaryCard(List<TestResult>
results) {
    final avgFps =
       results.map((r) => r.avgFps).reduce((a, b)
=> a + b) / results.length;
    final avgFrameTime =
                                results.map((r)       =>
r.avgFrameTimeMs).reduce((a, b) => a + b) /
           results.length;
        final avgRam = results.map((r) =>
r.ramUsageMb).reduce((a, b) => a + b) /
        results.length;
    final totalJankFrames =
                                results.map((r)       =>
r.jankFramesCount).reduce((a, b) => a + b);


    return Card(
      margin: const EdgeInsets.all(16),
      child: Padding(
        padding: const EdgeInsets.all(16),
        child: Column(
                                    crossAxisAlignment:
CrossAxisAlignment.start,
          children: [
            const Text(
              "Загальна статистика",
                      style:  TextStyle(fontSize:  18,
fontWeight: FontWeight.bold),
            ),
            const SizedBox(height: 12),
            Row(
                                    mainAxisAlignment:
MainAxisAlignment.spaceAround,
              children: [
                  _buildStatItem("Середній FPS",
avgFps.toStringAsFixed(1)),
                _buildStatItem(
                                        "Час кадру",
"${avgFrameTime.toStringAsFixed(2)} мс"),
                             _buildStatItem("Пам'ять",
"${avgRam.toStringAsFixed(1)} МБ"),
                    _buildStatItem("Jank кадри",
totalJankFrames.toString()),
                                _buildStatItem("Тестів",
results.length.toString()),
              ],
            ),
          ],
        ),
      ),
    );
  }


  Widget _buildStatItem(String label, String
value) {
```

```
    return Column(
      children: [
        Text(
          value,
              style: const TextStyle(fontSize: 14, fontWeight: FontWeight.bold),
        ),
        const SizedBox(height: 4),
        Text(
          label,
         style: const TextStyle(fontSize: 10, color: Colors.grey),
          textAlign: TextAlign.center,
        ),
      ],
    );
  }
}
```

19. Зміст файлу select_manager_screen.dart

```
// Екран вибору менеджера стану
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:magi_work/blocs/test_config_cubit.dart';

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
          title: const Text("Вибір менеджера стану"),
        centerTitle: true,
      ),
      body: Padding(
        padding: const EdgeInsets.all(16),
        child: Column(
                                mainAxisAlignment: MainAxisAlignment.center,
          children: const [
            ManagerButton(
              label: "StatefulWidget",
                                managerType: StateManagerType.stateful,
            ),
            SizedBox(height: 16),
            ManagerButton(
              label: "Provider",
                                managerType: StateManagerType.provider,
            ),
            SizedBox(height: 16),
            ManagerButton(
              label: "Bloc",
                                managerType: StateManagerType.bloc,
            ),
            SizedBox(height: 16),
            ManagerButton(
              label: "Riverpod",
                                managerType: StateManagerType.riverpod,
            ),
          ],
```

```
        ),
      ),
    );
  }
}


class ManagerButton extends StatelessWidget {
  final String label;
  final StateManagerType managerType;


  const ManagerButton({
    super.key,
    required this.label,
    required this.managerType,
  });


  @override
  Widget build(BuildContext context) {
          return BlocBuilder<TestConfigCubit, TestConfigState>(
      builder: (context, state) {
        final isSelected = state.selectedManager == managerType;


        return SizedBox(
          width: double.infinity,
          child: ElevatedButton(
            onPressed: () {
              context.read<TestConfigCubit>().selectManager(managerType);
              ScaffoldMessenger.of(context).showSnackBar(
                SnackBar(
                  content: Text("Вибрано: $label"),
                  duration: const Duration(seconds: 1),
                ),
              );
            },
            style: ElevatedButton.styleFrom(
                                padding: const EdgeInsets.symmetric(vertical: 16),
              textStyle: const TextStyle(fontSize: 18),
              backgroundColor:
                  isSelected ? Colors.blueAccent : const Color.fromARGB(255, 242, 242, 242),
            ),
            child: Text(label),
          ),
        );
      },
    );
  }
}
```

20. Зміст файлу select_test_screen.dart

```
// Екран вибору тестів
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:magi_work/blocs/test_config_cubit.dart';
import 'package:magi_work/screens/test_run_screen.dart';
```

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("Вибір сценаріїв тестування"),
        centerTitle: true,
      ),
      body: const Padding(
        padding: EdgeInsets.all(16),
        child: _TestSelectionForm(),
      ),
    );
  }
}

class _TestSelectionForm extends StatelessWidget {
  const _TestSelectionForm();

  @override
  Widget build(BuildContext context) {
    return BlocBuilder<TestConfigCubit, TestConfigState>(
      builder: (context, state) {
        return Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [

            const Text(
              "Оберіть менеджер стану:",
              style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold),
            ),
            const SizedBox(height: 8),
            const _ManagerDropdown(),
            const SizedBox(height: 16),



            const Text(
              "Оберіть сценарії:",
              style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold),
            ),
            const SizedBox(height: 8),
            const _SelectAllButton(),
            const SizedBox(height: 8),
            Expanded(
```

```
            child: ListView(children: const [
              TestCheckbox(label: "List Test", testKey: "ListTest"),
              TestCheckbox(label: "Form Test", testKey: "FormTest"),
              TestCheckbox(label: "Animation Test", testKey: "AnimationTest"),
              TestCheckbox(label: "API Test", testKey: "ApiTest"),
              TestCheckbox(
                label: "Subscription Test", testKey: "SubscriptionTest"),
            ]),
          ),

          const _RepetitionsSelector(),

          const SizedBox(height: 16),

          const IterationsInput(),

          const SizedBox(height: 16),

          const _StartTestButton(),
        ],
      );
    },
  );
 }
}

class _ManagerDropdown extends StatelessWidget {
 const _ManagerDropdown();

 @override
 Widget build(BuildContext context) {
  return BlocBuilder<TestConfigCubit, TestConfigState>(
    builder: (context, state) {
      return Container(
        padding: const EdgeInsets.symmetric(horizontal: 12),
        decoration: BoxDecoration(
          border: Border.all(color: Colors.grey),
          borderRadius: BorderRadius.circular(4),
        ),
        child: DropdownButton<StateManagerType>(
          value: state.selectedManager,
          isExpanded: true,
          underline: const SizedBox(),
```

```
        hint: const Text('Оберіть менеджер стану'),
        onChanged: (value) {
          if (value != null) {
            context.read<TestConfigCubit>().selectManager(value);
          }
        },
        items: const [
          DropdownMenuItem(
            value: StateManagerType.stateful,
            child: Text('StatefulWidget'),
          ),
          DropdownMenuItem(
            value: StateManagerType.provider,
            child: Text('Provider'),
          ),
          DropdownMenuItem(
            value: StateManagerType.bloc,
            child: Text('Bloc'),
          ),
          DropdownMenuItem(
            value: StateManagerType.riverpod,
            child: Text('Riverpod'),
          ),
        ],
      ),
    );
  },
 );
 }
}

class _SelectAllButton extends StatelessWidget {
 const _SelectAllButton();

 @override
 Widget build(BuildContext context) {
  return BlocBuilder<TestConfigCubit, TestConfigState>(
   builder: (context, state) {
    final allSelected = state.selectedTests.length == 5;

    return OutlinedButton(
     onPressed: () {
      final allTests = const ["ListTest", "FormTest", "AnimationTest", "ApiTest", "SubscriptionTest"];

      if (allSelected) {

       for (final test in allTests) {
```

```
            if
(state.selectedTests.contains(test)) {
              context.read<TestConfigCubit>().t
oggleTest(test);
            }
          }
        } else {

          for (final test in allTests) {
            if
(!state.selectedTests.contains(test)) {
              context.read<TestConfigCubit>().t
oggleTest(test);
            }
          }
        },
        child: Text(allSelected ? 'Зняти всі' :
'Обрати всі'),
      );
    },
  );
 }
}

class _RepetitionsSelector extends
StatelessWidget {
 const _RepetitionsSelector();

 String _getRepetitionText(int count) {
  if (count == 1) return '';
  if (count >= 2 && count <= 4) return 'и';
  return 'ів';
 }

 @override
 Widget build(BuildContext context) {
  return BlocBuilder<TestConfigCubit,
TestConfigState>(
    builder: (context, state) {
      return ListTile(
        title: const Text('Кількість повторень
кожного тесту'),
        subtitle: const Text('Рекомендується 5
для статистичної значущості'),
        trailing: DropdownButton<int>(
          value: state.repetitions,
          onChanged: (value) {
            if (value != null) {
              context.read<TestConfigCubit>().se
tRepetitions(value);
            }
          },
          items: [1, 3, 5, 10]
              .map((e) => DropdownMenuItem(
```

```
                    value: e,
                    child: Text('$e раз${_getRepetitionText(e)}'),
                  ))
              .toList(),
          ),
        );
      },
    );
  }
}

class _TestInfoCard extends StatelessWidget {
  const _TestInfoCard();

  @override
  Widget build(BuildContext context) {
    return BlocBuilder<TestConfigCubit, TestConfigState>(
      builder: (context, state) {
        final totalTests = state.selectedTests.length;
        final totalRuns = totalTests * state.repetitions;
        final estimatedTime =
            (totalRuns * state.iterations * 0.05 / 60).toStringAsFixed(1);

        return Card(
          color: Colors.blue[50],
          child: Padding(
            padding: const EdgeInsets.all(12),
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                const Text(
                  "Інформація про тестування:",
                  style: TextStyle(fontWeight: FontWeight.bold),
                ),
                const SizedBox(height: 8),
                Text("• Тестів: $totalTests"),
                Text("• Повторень на тест: ${state.repetitions}"),
                Text("• Ітерацій на запуск: ${state.iterations}"),
                Text("• Всього запусків: $totalRuns"),
                Text("• Приблизний час: ~${estimatedTime} хв"),
              ],
            ),
          ),
        );
```

```
    },
   );
  }
}


class _StartTestButton extends StatelessWidget {
  const _StartTestButton();


  @override
  Widget build(BuildContext context) {
    return BlocBuilder<TestConfigCubit, TestConfigState>(
      builder: (context, state) {
        final isEnabled =
            state.selectedTests.isNotEmpty && state.selectedManager != null;


        return SizedBox(
          width: double.infinity,
          child: ElevatedButton(
            onPressed: isEnabled
                ? () {
                    Navigator.of(context).push(
                      MaterialPageRoute(
                        builder: (context) => BlocProvider.value(
                          value: context.read<TestConfigCubit>(),
                          child: const TestRunnerScreen(),
                        ),
                      ),
                    );
                  }
                : null,
            style: ElevatedButton.styleFrom(
              padding: const EdgeInsets.symmetric(vertical: 16),
              backgroundColor: isEnabled ? Colors.blue : Colors.grey,
            ),
            child: isEnabled
                ? const Text(
                    "Запустити тести",
                    style: TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
                  )
                : const Text("Оберіть тести та менеджер"),
          ),
        );
      },
    );
  }
```

```
}


class TestCheckbox extends StatelessWidget
{
  final String label;
  final String testKey;


  const TestCheckbox({
    super.key,
    required this.label,
    required this.testKey,
  });


  @override
  Widget build(BuildContext context) {
    return BlocBuilder<TestConfigCubit,
TestConfigState>(
      builder: (context, state) {
        final isSelected =
state.selectedTests.contains(testKey);


        return Card(
          margin: const
EdgeInsets.symmetric(vertical: 4),
          child: CheckboxListTile(
            value: isSelected,
            title: Text(
              label,
              style: TextStyle(
                fontWeight: isSelected ?
FontWeight.bold : FontWeight.normal,
                color: isSelected ? Colors.blue :
Colors.black,
              ),
            ),
            controlAffinity:
ListTileControlAffinity.leading,
            onChanged: (_) {
              context.read<TestConfigCubit>().tog
gleTest(testKey);
            },
          ),
        );
      },
    );
  }
}


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


  @override
  Widget build(BuildContext context) {
```

```
   return BlocBuilder<TestConfigCubit, TestConfigState>(
     builder: (context, state) {
       return Column(
         crossAxisAlignment: CrossAxisAlignment.start,
         children: [
           const Text(
             "Кількість ітерацій на тест:",
             style: TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
           ),
           const SizedBox(height: 8),
           TextField(
             controller:
                 TextEditingController(text: state.iterations.toString()),
             keyboardType: TextInputType.number,
             decoration: const InputDecoration(
               border: OutlineInputBorder(),
               labelText: "Введіть кількість ітерацій",
               hintText: "100",
             ),
             onSubmitted: (value) {
               final intValue = int.tryParse(value) ?? 100;
               if (intValue < 1) return;
               context.read<TestConfigCubit>().setIterations(intValue);
             },
           ),
           const SizedBox(height: 4),
           Text(
             "Рекомендується 100+ ітерацій для стабільних результатів",
             style: TextStyle(
               fontSize: 12,
               color: Colors.grey[600],
             ),
           ),
         ],
       );
     },
   );
 }
}
```

21. Зміст файлу test_run_screen.dart

```
// Екран запуску тестів
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
```

```
import 'package:magi_work/core/utils/file_export.dart';
import 'package:magi_work/models/test_base.dart';
import 'package:magi_work/blocs/test_config_cubit.dart';
import 'package:magi_work/screens/result_screen.dart';
import 'package:magi_work/ui_tests/animation_test_widget.dart';
import 'package:magi_work/ui_tests/api_test_widget.dart';
import 'package:magi_work/ui_tests/form_test_widget.dart';
import 'package:magi_work/ui_tests/list_test_widget.dart';
import 'package:magi_work/ui_tests/subscription_test_widget.dart';
import '../tests/animation_test.dart';
import '../tests/api_test.dart';
import '../tests/form_test.dart';
import '../tests/list_test.dart';
import '../tests/subscription_test.dart';
import '../core/metrics/stats_recorder.dart';
import '../core/metrics/widget_counter.dart';

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

  @override
  State<TestRunnerScreen> createState() => _TestRunnerScreenState();
}

class _TestRunnerScreenState extends State<TestRunnerScreen> {
  late List<LoadTest> activeTests = [];
  bool isRunning = false;
  int currentIteration = 0;
    final Map<String, StatsRecorder> _testRecorders = {};

  @override
  void initState() {
    super.initState();
    WidgetsBinding.instance.addPostFrameCallback((_) {
      _initializeTests();
    });
  }

  @override
  void dispose() {
    for (final recorder in _testRecorders.values) {
      recorder.stopListening();
    }
    super.dispose();
  }

  void _initializeTests() {
                final state = context.read<TestConfigCubit>().state;
    activeTests.clear();
    _testRecorders.clear();

    for (final testKey in state.selectedTests) {
      switch (testKey) {
```

```
        case 'AnimationTest':
          final test = AnimationTest(
            elementsCount: 5,
            iterations: state.iterations,
          );
          activeTests.add(test);
                    _testRecorders[test.name]    =
StatsRecorder();
          break;
        case 'ApiTest':
                  final   test   =   ApiTest(iterations:
state.iterations);
          activeTests.add(test);
                    _testRecorders[test.name]    =
StatsRecorder();
          break;
        case 'FormTest':
          final test = FormTest(
            fieldsCount: 5,
            iterations: state.iterations,
          );
          activeTests.add(test);
                    _testRecorders[test.name]    =
StatsRecorder();
          break;
        case 'ListTest':
          final test = ListTest(
            initialCount: 100,
            iterations: state.iterations,
          );
          activeTests.add(test);
                    _testRecorders[test.name]    =
StatsRecorder();
          break;
        case 'SubscriptionTest':
          final test = SubscriptionTest(
            updatesPerSecond: 10,
            iterations: state.iterations,
          );
          activeTests.add(test);
                    _testRecorders[test.name]    =
StatsRecorder();
          break;
      }
    }
    setState(() {});
  }

  Future<void> _runTests() async {
  if (isRunning || activeTests.isEmpty) return;

  setState(() {
    isRunning = true;
    currentIteration = 0;
  });



  for (final recorder in _testRecorders.values) {
    recorder.reset();
    recorder.startListening();
  }

  for (final test in activeTests) {
    await test.setup();
  }



  final totalIterations = activeTests.isNotEmpty
? activeTests.first.iterations : 0;

  for (int i = 0; i < totalIterations; i++) {
```

```
    if (!mounted) break;

    setState(() {
      currentIteration = i + 1;
    });


    for (final test in activeTests) {
      final recorder = _testRecorders[test.name];
      if (recorder != null) {

        recorder.startFrame();
        final testStart = DateTime.now();

        await test.runIteration();

        final testEnd = DateTime.now();
                final latencyMs = testEnd.difference(testStart).inMilliseconds.toDouble();
        recorder.recordLatencyMs(latencyMs);


        recorder.endFrame();
      }
    }

                await Future.delayed(const Duration(milliseconds: 10));
  }

  for (final test in activeTests) {
    await test.teardown();
  }
```

```
  for (final recorder in _testRecorders.values) {
    recorder.stopListening();
  }

  if (mounted) {
    setState(() {
      isRunning = false;
    });


    await _saveResults();
    _navigateToResults();
  }
}

  Future<void> _saveResults() async {
                final cubit = context.read<TestConfigCubit>();
    final state = cubit.state;


    for (final test in activeTests) {
      final recorder = _testRecorders[test.name];
      if (recorder != null) {
        final result = recorder.buildResult(
          test.name,
          state.selectedManager?.toString().split('.').last ?? 'unknown',
          state.iterations,
        );


        cubit.addResult(result);

      }
```

```
  }
 }

 void _navigateToResults() {
  Navigator.of(context).push(
   MaterialPageRoute(
    builder: (context) => BlocProvider.value(
     value: context.read<TestConfigCubit>(),
     child: const ResultsScreen(),
    ),
   ),
  );
 }

 void _stopTests() {
  setState(() {
   isRunning = false;
  });

  for (final recorder in _testRecorders.values)
{
   recorder.stopListening();
  }
 }

 void _goBack() {
  if (isRunning) {
   _stopTests();
  }
  Navigator.of(context).pop();
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: const Text('Запуск тестів'),
    leading: IconButton(
     icon: const Icon(Icons.arrow_back),
     onPressed: _goBack,
    ),
   ),
   body: Column(
    children: [

     const _ControlPanel(),


     _ProgressBar(
      isRunning: isRunning,
      currentIteration: currentIteration,
      activeTests: activeTests,
     ),


     _TestWidgetsList(
      activeTests: activeTests,
      testRecorders: _testRecorders,
     ),
    ],
   ),
  );
 }
}

class _ControlPanel extends StatelessWidget
{
 const _ControlPanel();
```

```
  @override
  Widget build(BuildContext context) {
        return BlocBuilder<TestConfigCubit, TestConfigState>(
    builder: (context, state) {
                    final screenState = context.findAncestorStateOfType<_TestRunnerScreenState>();


      return Container(
        padding: const EdgeInsets.all(16),
        color: Colors.grey[100],
        child: Row(
                              mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
            Column(
                              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Text(
                                  'Активні тести: ${state.selectedTests.length}',
                    style: const TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
                ),
                const SizedBox(height: 4),
                Text(
                                      'Ітерації: ${screenState?.currentIteration ?? 0}/${state.iterations}',
                    style: const TextStyle(fontSize: 14),
                ),
              ],
            ),
            Row(
              children: [
                if (screenState?.isRunning != true)
                  ElevatedButton.icon(
                                          onPressed: screenState?._runTests,
                                      icon: const Icon(Icons.play_arrow),
                    label: const Text('Запустити'),
                  )
                else
                  ElevatedButton.icon(
                                          onPressed: screenState?._stopTests,
                    icon: const Icon(Icons.stop),
                    label: const Text('Зупинити'),
                    style: ElevatedButton.styleFrom(
                      backgroundColor: Colors.red,
                    ),
                  ),
              ],
            ),
          ],
        ),
      );
    },
  );
  }
}


class _ProgressBar extends StatelessWidget {
  final bool isRunning;
  final int currentIteration;
  final List<LoadTest> activeTests;
```

```
  const _ProgressBar({
    required this.isRunning,
    required this.currentIteration,
    required this.activeTests,
  });

  @override
  Widget build(BuildContext context) {
    if (activeTests.isEmpty) {
      return const SizedBox();
    }

    final totalIterations = activeTests.isNotEmpty ? activeTests.first.iterations : 0;
    final progress = totalIterations > 0 ? currentIteration / totalIterations : 0.0;

    return Column(
      children: [
        Padding(
          padding: const EdgeInsets.symmetric(horizontal: 16),
          child: Text(
            'Ітерація: $currentIteration / $totalIterations',
            textAlign: TextAlign.center,
          ),
        ),
        LinearProgressIndicator(
          value: progress,
          backgroundColor: Colors.grey[300],
          valueColor: AlwaysStoppedAnimation<Color>(
            isRunning ? Colors.blue : Colors.green,
          ),
        ),
      ],
    );
  }
}

class _TestWidgetsList extends StatelessWidget {
  final List<LoadTest> activeTests;
  final Map<String, StatsRecorder> testRecorders;

  const _TestWidgetsList({
    required this.activeTests,
    required this.testRecorders,
  });

  @override
  Widget build(BuildContext context) {
    if (activeTests.isEmpty) {
      return const Expanded(
        child: Center(
          child: Text('Немає активних тестів'),
        ),
      );
    }

    return Expanded(
      child: ListView(
        children: activeTests.map((test) {
          final recorder = testRecorders[test.name];
          if (recorder == null) return const SizedBox();
```

```
        return Card(
          margin: const EdgeInsets.all(8),
          child: SizedBox(
            height: 200,
            child: Padding(
              padding: const EdgeInsets.all(16),
              child: Column(
                                crossAxisAlignment:
CrossAxisAlignment.start,
                children: [
                  Text(
                    test.name,
                    style: const TextStyle(
                      fontSize: 18,
                      fontWeight: FontWeight.bold,
                    ),
                  ),
                  const SizedBox(height: 8),
                  Expanded(
                    child: WidgetCounterScope(
                                                counter:
recorder.widgetCounter,
                      child: _TestWidget(test: test),
                    ),
                  ),
                ],
              ),
            ),
          ),
        );
      }).toList(),
    ),
  );
 }
}
```

```
class _TestWidget extends StatelessWidget {
 final LoadTest test;

 const _TestWidget({required this.test});

 @override
 Widget build(BuildContext context) {
  switch (test.runtimeType) {
    case AnimationTest:
      return  AnimationTestWidget(scenario:
test as AnimationTest);
    case ApiTest:
      return ApiTestWidget(scenario: test as
ApiTest);
    case FormTest:
      return FormTestWidget(scenario: test as
FormTest);
    case ListTest:
      return ListTestWidget(scenario: test as
ListTest);
    case SubscriptionTest:
      return SubscriptionTestWidget(scenario:
test as SubscriptionTest);
    default:
      return Center(
        child: Text('Не підтримуваний тип
тесту: ${test.runtimeType}'),
      );
  }
 }
}
```

22. Зміст файлу animation_test.dart

```
// Модель для тесту анімації
import '../models/test_base.dart';

class AnimationTest extends LoadTest {
  final int elementsCount;
  final int iterations;
  void Function()? onAction;

    AnimationTest({this.elementsCount = 5, this.iterations = 300}) : super("Animation Test");

  @override
  Future<void> setup() async {}

  @override
  Future<void> runIteration() async {
    onAction?.call();
  }

  @override
  Future<void> teardown() async {}
}
```

23. Зміст файлу api_test.dart

```
// Модель для тесту API
import '../models/test_base.dart';

class ApiTest extends LoadTest {
  final int iterations;
  void Function()? onAction;

   ApiTest({this.iterations = 100}) : super("API Test");

  @override
  Future<void> setup() async {}

  @override
  Future<void> runIteration() async {
    // імітація "нового запроса"
    onAction?.call();
  }

  @override
  Future<void> teardown() async {}
}
```

24. Зміст файлу form_test.dart

```
// Модель для тесту форм
import '../models/test_base.dart';

class FormTest extends LoadTest {
  final int fieldsCount;
  final int iterations;
  void Function()? onAction;

  FormTest({this.fieldsCount = 5, this.iterations = 200}) : super("Form Test");
```

```
  @override
  Future<void> setup() async {}

  @override
  Future<void> runIteration() async {
    onAction?.call();
  }

  @override
  Future<void> teardown() async {}
}
```

25. Зміст файлу list_test.dart

```
// Модель для тесту списку
import '../models/test_base.dart';

class ListTest extends LoadTest {
  final int initialCount;
  final int iterations;
  void Function()? onAction;

  ListTest({this.initialCount = 100, this.iterations = 500}) : super("List Test");

  @override
  Future<void> setup() async {}

  @override
  Future<void> runIteration() async {
    // Масова модифікація списку
    onAction?.call();
  }

  @override
  Future<void> teardown() async {}
}
```

26. Зміст файлу subscription_test.dart

```
// Модель для тесту підписки
import '../models/test_base.dart';

class SubscriptionTest extends LoadTest {
  final int updatesPerSecond;
  final int iterations;
  void Function()? onAction;

  SubscriptionTest({this.updatesPerSecond = 10, this.iterations = 200}) : super("Subscription Test");

  @override
  Future<void> setup() async {}

  @override
  Future<void> runIteration() async {
    onAction?.call();
  }
```

```
  @override
  Future<void> teardown() async {}
}
```

27. Зміст файлу animation_test_widget.dart

```
// Віджет моделі тесту з анімації
import 'package:flutter/material.dart';
import 'package:magi_work/core/metrics/widget_counter.dart';
import '../tests/animation_test.dart';

class AnimationTestWidget extends StatefulWidget {
  final AnimationTest scenario;

    const AnimationTestWidget({Key? key, required this.scenario}) : super(key: key);

  @override
   State<AnimationTestWidget> createState() => _AnimationTestWidgetState();
}

class _AnimationTestWidgetState extends State<AnimationTestWidget> with WidgetCounterMixin {
  bool toggled = false;

  @override
  void initState() {
    super.initState();
                widget.scenario.onAction = _toggleAnimation;
  }

  void _toggleAnimation() {
    setState(() {
      toggled = !toggled;
      incrementRebuild(); // Считаем ребилд
    });
  }

  @override
  Widget build(BuildContext context) {
     incrementRebuild();
    return Center(
      child: Wrap(
        spacing: 10,
        runSpacing: 10,
        children: List.generate(
          widget.scenario.elementsCount,
          (i) => AnimatedContainer(
            duration: const Duration(milliseconds: 500),
            width: toggled ? 50 : 100,
            height: toggled ? 100 : 50,
                    color: Colors.primaries[i % Colors.primaries.length],
          ),
        ),
```

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

28. Зміст файлу api_test_widget.dart

```
// Віджет моделі тесту з API
import 'package:flutter/material.dart';
import 'package:magi_work/core/metrics/widget_counter.dart';
import '../tests/api_test.dart';

class ApiTestWidget extends StatefulWidget {
  final ApiTest scenario;

   const ApiTestWidget({Key? key, required this.scenario}) : super(key: key);

  @override
   State<ApiTestWidget> createState() => _ApiTestWidgetState();
}

class _ApiTestWidgetState extends State<ApiTestWidget> with WidgetCounterMixin {
  String data = "No data";

  @override
  void initState() {
   super.initState();
   widget.scenario.onAction = _loadData;
  }

  void _loadData() async {
            await Future.delayed(const Duration(milliseconds: 50));
   setState(() {
            data = "Data updated at ${DateTime.now().toIso8601String()}";
    incrementRebuild();
   });
  }

  @override
  Widget build(BuildContext context) {
       incrementRebuild();
   return Center(child: Text(data));
  }
}
```

29. Зміст файлу form_test_widget.dart

```
// Віджет моделі тесту з формами
import 'package:flutter/material.dart';
import 'package:magi_work/core/metrics/widget_counter.dart';
import '../tests/form_test.dart';
```

```
class FormTestWidget extends StatefulWidget {
  final FormTest scenario;

  const FormTestWidget({Key? key, required this.scenario}) : super(key: key);

  @override
  State<FormTestWidget> createState() => _FormTestWidgetState();
}

class _FormTestWidgetState extends State<FormTestWidget> with WidgetCounterMixin {
  late List<TextEditingController> controllers;

  @override
  void initState() {
    super.initState();
    controllers = List.generate(widget.scenario.fieldsCount, (_) => TextEditingController());
    widget.scenario.onAction = _simulateInput;
  }

  void _simulateInput() {
    setState(() {
      for (int i = 0; i < controllers.length; i++) {
        controllers[i].text += String.fromCharCode(97 + (i % 26));
      }
      incrementRebuild();
    });
  }

  @override
  Widget build(BuildContext context) {
    incrementRebuild();
    return ListView.builder(
      shrinkWrap: true,
      physics: const AlwaysScrollableScrollPhysics(),
      itemCount: controllers.length,
      itemBuilder: (context, index) {
        return Padding(
          padding: const EdgeInsets.all(8.0),
          child: TextField(controller: controllers[index]),
        );
      },
    );
  }
}
```

30. Зміст файлу list_test_widget.dart

```
// Віджет моделі тесту з списками
import 'package:flutter/material.dart';
import 'package:magi_work/core/metrics/widget_cou
```

```
nter.dart';

import '../tests/list_test.dart';


class ListTestWidget extends StatefulWidget {

  final ListTest scenario;


   const ListTestWidget({super.key, required this.scenario});


  @override

    State<ListTestWidget> createState() => _ListTestWidgetState();

}


class _ListTestWidgetState extends State<ListTestWidget> with WidgetCounterMixin {

  late List<int> items;


  @override

  void initState() {

    super.initState();

                                items = List.generate(widget.scenario.initialCount, (i) => i);

    widget.scenario.onAction = _modifyList;

  }


  void _modifyList() {

    setState(() {

      if (items.length % 2 == 0) {

        items.add(items.length + 1);

      } else {

        if (items.isNotEmpty) items.removeAt(0);

      }

      incrementRebuild();

    });

  }


  @override

  Widget build(BuildContext context) {

        incrementRebuild();

    return ListView.builder(

      shrinkWrap: true,

                                physics: const AlwaysScrollableScrollPhysics(),

      itemCount: items.length,

           itemBuilder: (_, i) => ListTile(title: Text("Item ${items[i]}")),

    );

  }

}
```

31. Зміст файлу subscription_test_widget.dart

```
// Віджет моделі тесту з підписками

import 'package:flutter/material.dart';

import 'package:magi_work/core/metrics/widget_counter.dart';

import '../tests/subscription_test.dart';
```

```
class SubscriptionTestWidget extends StatefulWidget {

  final SubscriptionTest scenario;


    const SubscriptionTestWidget({Key? key, required this.scenario}) : super(key: key);


  @override

                  State<SubscriptionTestWidget> createState() => _SubscriptionTestWidgetState();

}


class _SubscriptionTestWidgetState extends State<SubscriptionTestWidget> with WidgetCounterMixin {

  int counter = 0;


  @override

  void initState() {

    super.initState();

                  widget.scenario.onAction = _updateCounter;

  }


  void _updateCounter() {

    setState(() {

      counter++;

      incrementRebuild();

    });

  }

  @override

  Widget build(BuildContext context) {

        incrementRebuild();

    return Center(

        child: Text("Updates: $counter", style: const TextStyle(fontSize: 24)),

    );

  }

}
```