**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

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

**КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

# Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему **Автоматизація CI/CD для мобільних додатків на Expo та React Native**

Виконав: студент 2 курсу, групи 8.1214-іпз-2
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
А. А. Хлівний
(ініціали та прізвище )
Керівник доцент А.І. Безверхий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент Директор ТОВ «Дісітел»
П.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
**2025**

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

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

**ЗАТВЕРДЖУЮ**

Завідувач кафедри _____Т.В. Критська_
"_01_" __квітня_____2025_року

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

Хлівному Андрію Анатолійовичу
(прізвище, ім'я, по батькові)

1.Тема роботи __Автоматизація CI/CD для мобільних додатків на Expo та React Native_____

керівник роботи _____доцент Безверхий Анатолій Ігорьович_____
( прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від _28.03. 2025 р. № 361-е__
2. Строк подання студентом кваліфікаційної роботи_____01.12.2025 р.___
3. Вихідні дані магістерської роботи
1. комплект нормативних документів ;
2. технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
3. Огляд та збір літератури стосовно теми кваліфікаційної роботи;
4. Огляд та аналіз існуючих рішень та аналогів;
5. Дослідження проблеми автоматизації CI/CD для мобільних додатків на Expo та React Native та розробка методів її вирішення;
6. Створення програмного продукту та його опис;
7. Перелік вимог для роботи програми;
8. Дослідження поставленої проблеми та розробка висновків та пропозицій.

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

6. Консультанти розділів магістерської роботи

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

7. Дата видачі завдання 01.04.2025

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів магістерської роботи | Строк виконання етапів магістерської роботи | Примітка |
|---|---|---|---|
| 1 | Аналіз предметної області | 05.04 – 25.04.25 | виконано |
| 2 | Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником | 01.05 – 13.05.25 | виконано |
| 3 | Аналіз існуючих методів рішення | 23.05 – 17.06.25 | виконано |
| 4 | Огляд концепцій CI/CD та інтеграції у SDLC, роль DevOps | 25.06 – 13.07.25 | виконано |
| 5 | Огляд платформи React Native/Expo, нової архітектури RN (JSI/Fabric/TurboModules) | 28.07 – 21.08.25 | виконано |
| 6 | Вибір інструментів CI/CD (GitHub Actions, EAS, Fastlane), узгодження з керівником | 23.08 – 28.08.25 | виконано |
| 7 | Розробка пайплайнів: лінтинг, модульні та інтеграційні тести (Jest/RTL) | 04.09 – 26.09.25 | виконано |
| 8 | Налаштування E2E-тестів (Detox/Maestro) на емуляторах/симуляторах | 28.09 – 16.10.25 | виконано |
| 9 | Автоматизовані збірки Android/iOS через EAS Build, керування артефактами | 17.10 – 19.10.25 | виконано |
| 10 | Впровадження деплою: TestFlight, Google Play Console; OTA через EAS Update | 20.10 – 09.11.25 | виконано |
| 11 | Експериментальна оцінка конвеєра: час, стабільність, покриття, DORA-метрики | 10.10 – 17.10.25 | виконано |
| 12 | Порівняння до/після оптимізацій, рекомендації та стабілізація пайплайнів | 18.10 – 31.10.25 | виконано |
| 13 | Оформлення пояснювальної записки | 01.11 – 29.11.25 | виконано |

Студент _____ А.А. Хпівний
( підпис ) (прізвище та ініціали)

Керівник роботи _____ А.І. Безверхий
( підпис ) (прізвище та ініціали)

**Нормоконтроль пройдено**

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

# АНОТАЦІЯ

Сторінок: 87

Рисунків: 7

Таблиць: 1

Джерел: 33

Хлівний А. А. Автоматизація CI/CD для мобільних додатків на Expo та React Native: кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник А. І. Безверхий. Запоріжжя : ЗНУ, 2025. 87 с.

Мета дослідження полягає у розробці, реалізації та оцінці ефективності повністю автоматизованого CI/CD конвеєра, орієнтованого на прискорення процесу інтеграції, тестування та розгортання мобільних додатків, розроблених із застосуванням фреймворків Expo та React Native.

Реалізувати автоматичну збірку (EAS Build), тестування та доставку (EAS Submit) у магазини додатків; оцінити отриману економію часу та підвищення частоти деплою. У процесі дослідження була визначена проблема: велика ресурсомісткість, тривалість та схильність до помилок ручних операцій зі збірки, підписання коду та публікації мобільних додатків у App Store та Google Play, що суттєво уповільнює цикл розробки та доставку оновлень.

Ключові слова: CI/CD, Автоматизація, Expo, React Native, Неперервна Інтеграція, Неперервна Поставка, EAS (Expo Application Services), Мобільний Додаток, Розгортання, OTA (Over-The-Air) Updates.

# SUMMARY

Pages: 87

Figures: 7

Tables: 1

Sources: 33

Khlivnyi A. A. CI/CD Automation for Mobile Applications on Expo and React Native: Master's qualifying thesis in specialty 121 «Software Engineering» / Scientific supervisor A. I. Bezverkhyi. Zaporizhzhia : ZNU, 2025. 78 p.

The goal of the research is to develop, implement, and evaluate the effectiveness of a fully automated CI/CD pipeline aimed at accelerating the process of integration, testing, and deployment of mobile applications developed using the Expo and React Native frameworks. The objectives include implementing automatic builds (EAS Build), testing, and delivery (EAS Submit) to app stores; evaluating the resulting time savings and increased deployment frequency. During the research, the problem of high resource consumption, duration, and error-proneness of manual operations for building, code signing, and publishing mobile applications to the App Store and Google Play was identified, which significantly slows down the development cycle and update delivery.

Keywords: CI/CD, Automation, Expo, React Native, Continuous Integration, Continuous Delivery, EAS (Expo Application Services), Mobile Application, Deployment, OTA (Over-The-Air) Updates.

**ЗМІСТ**

ВСТУП ..... 9
РОЗДІЛ 1 ТЕОРЕТИЧНЕ ОБҐРУНТУВАННЯ ТА ОГЛЯД ТЕХНОЛОГІЧНИХ ЗАСОБІВ АВТОМАТИЗАЦІЇ CI/CD ..... 15
1.1 Обґрунтування актуальності та концепції CI/CD ..... 15
1.2 Інтеграція CI/CD у життєвий цикл розробки (SDLC) ..... 18
1.2.1 Концептуальна Роль DevOps у Модернізації Життєвого Циклу Програмного Забезпечення ..... 22
1.3 Теоретичні Основи Оцінки Продуктивності: Метрики DORA ..... 23
1.4 Висновки з розділу 1 ..... 27
РОЗДІЛ 2 ОГЛЯД ТЕХНОЛОГІЧНОЇ ПЛАТФОРМИ REACT NATIVE, ІНСТРУМЕНТАРІЮ ТА СПЕЦИФІКИ АВТОМАТИЗАЦІЇ ..... 29
2.1 Архітектурний Аналіз React Native, Expo та Механізму Неперервної Поставки ..... 29
2.1.2 Нова Архітектура React Native: Принципи JSI, Fabric та TurboModules ..... 31
2.1.3 Роль EAS у Спрощенні Нативної Збірки та Оптимізації Доставки через OTA-оновлення ..... 33
2.2 Огляд Інструментарію CI/CD та Специфіка Конфігурації Конвеєрів ..... 35
2.2.1 Pipeline as Code: управління CI/CD як кодом ..... 40
2.3 Автоматизоване тестування у CI/CD для React Native ..... 42
2.3.1 Модульне тестування в екосистемі React Native: підходи, інструменти та інтеграція у CI ..... 43
2.3.2 Інтеграційне тестування та валідація бізнес-логіки ..... 43
2.3.3 E2E-тестування мобільних застосунків: Detox та Maestro ..... 44
2.3.4 Метрики якості тестів та покриття коду ..... 44

2.3.5 Інтеграція тестування у CI/CD-конвеєри ..... 45

2.3.6 Забезпечення тестованості та оптимізація продуктивності ..... 45

2.4 Висновки з розділу 2 ..... 45

РОЗДІЛ 3 РОЗРОБКА СИСТЕМИ АВТОМАТИЗАЦІЇ CI/CD ДЛЯ МОБІЛЬНИХ ДОДАТКІВ НА EXPO ТА REACT NATIVE ..... 48

3.1 Планування та підготовка середовища для автоматизації CI/CD .... 48

3.1.1 Вибір інструментів та сервісів для CI/CD (GitHub Actions, EAS, Fastlane) ..... 48

3.1.2 Налаштування локального середовища розробника та тестового проєкту ..... 49

3.1.3 Підготовка проєкту React Native та Expo для інтеграції з CI/CD ..... 51

3.1.4 Опис тестового мобільного застосунку та вимоги до нього ..... 52

3.2 Розробка та конфігурація CI/CD-пайплайнів ..... 53

3.2.1 Структура та логіка пайплайна для автоматичного тестування ..... 54

3.2.2 Конфігурація модульного та інтеграційного тестування ..... 56

3.2.3 Налаштування E2E-тестів та запуск на емуляторах/реальних пристроях ..... 59

3.2.4 Автоматизовані збірки під Android та iOS ..... 62

3.2.5 Інтеграція артефактів та контроль якості релізів ..... 65

3.3 Впровадження автоматичного деплою та OTA-оновлень ..... 67

3.3.1 Налаштування EAS Update для Over-the-Air оновлень ..... 67

3.3.2 Публікація додатків у TestFlight та Google Play Console ..... 69

3.3.3 Створення автоматичних релізів та моніторинг стабільності .. 72

3.4 Висновки з розділу 3 ..... 75

РОЗДІЛ 4 ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ ВПРОВАДЖЕНОЇ СИСТЕМИ АВТОМАТИЗАЦІЇ CI/CD ДЛЯ МОБІЛЬНИХ ДОДАТКІВ НА EXPO ТА REACT NATIVE ..... 77
4.1 Методологія оцінювання та експериментальний стенд ..... 77
4.1.1 Опис оточення: GitHub Actions, EAS, емулятори/реальні пристрої ..... 77
4.1.2 Набір метрик: час збірки, стабільність тестів, DORA ..... 77
4.1.3 Процедури вимірювання та реплікації експериментів ..... 78
4.2 Аналіз продуктивності конвеєра та якості тестування ..... 79
4.2.1 Час виконання джобів, кешування та паралелізація ..... 79
4.2.2 Якість тестів: покриття, тривалість, частота флейків (Detox/Maestro) ..... 79
4.2.3 Надійність збірок iOS/Android та управління артефактами ..... 80
4.3 Оцінка релізних процесів і OTA-оновлень ..... 80
4.3.1 Метрики DORA: частота деплою, lead time, MTTR, change failure rate ..... 80
4.3.2 Ефективність EAS Update: швидкість доставлення та вплив на UX ..... 80
4.3.3 Порівняння до/після впровадження CI/CD та рекомендації щодо покращень ..... 81
4.5 Висновки з розділу 4 ..... 81
ВИСНОВКИ ..... 82
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ..... 84

# ВСТУП

## Актуальність Теми

В епоху домінування мобільних платформ на ринку програмного забезпечення, здатність компаній швидко, надійно та економічно ефективно доставляти оновлення користувачам є критичним фактором конкурентоспроможності. Сучасні методології гнучкої розробки (Agile, Scrum) вимагають високої частоти релізів, що вступає в протиріччя зі складністю традиційного процесу розгортання мобільних додатків. Специфіка Гібридних Фреймворків: Використання React Native та платформи Expo значно прискорює крос-платформенну розробку, але водночас вимагає ефективного керування як JavaScript-кодом, так і нативними конфігураціями (особливо для iOS та Android). Ручне управління цими процесами, включаючи підписання коду, управління сертифікатами та генерацію бінарних файлів, є витратним, тривалим та схильним до помилок. Потреба в Автоматизації: Впровадження принципів CI/CD (Continuous Integration/Continuous Delivery) є прямим рішенням цієї проблеми. Автоматизація дозволяє не лише скоротити час між розробкою та випуском (Time-to-Market), але й підвищити якість продукту за рахунок інтеграції автоматичного тестування на ранніх етапах. Виклики Платформи Expo: Хоча Expo Application Services (EAS) надає потужні інструменти, інтеграція їх у повноцінний, надійний та масштабований CI/CD конвеєр, здатний ефективно використовувати OTA-оновлення та керувати різними середовищами (staging, production), вимагає глибокого аналізу та інженерної розробки.

Таким чином, розробка та впровадження ефективного, повністю автоматизованого CI/CD конвеєра для мобільних додатків на Expo та React Native є не просто технічним завданням, а актуальною науково-практичною проблемою, вирішення якої забезпечує значне зниження операційних витрат, підвищення швидкості доставки оновлень та загальної надійності програмного продукту.

**Мета Дослідження**

Мета дослідження полягає у розробці, реалізації та оцінці ефективності повністю автоматизованого CI/CD конвеєра, орієнтованого на прискорення процесу інтеграції, тестування та розгортання мобільних додатків, розроблених із застосуванням фреймворків Expo та React Native.

**Завдання**

Вивчити особливості технологій Expo та React Native у контексті автоматизації збірки; проаналізувати сучасні інструменти CI/CD (EAS, Fastlane, GitHub Actions); розробити архітектуру CI/CD конвеєра.

**Об’єкт дослідження**

Процеси життєвого циклу розробки програмного забезпечення (SDLC) мобільних додатків, побудованих на платформі Expo та фреймворку React Native.

**Предмет дослідження**

Методи, алгоритми та інструментальні засоби для автоматизації неперервної інтеграції та неперервної поставки (CI/CD) мобільних додатків.

**Методи дослідження**

Для вирішення поставленої задачі використовуються наступні методи дослідження:

1. Провести аналіз архітектурних особливостей Expo та React Native, які впливають на процеси автоматизованої збірки та розгортання.
2. Дослідити сучасні інструменти CI/CD, зокрема Expo Application Services (EAS), та обґрунтувати вибір платформи для реалізації конвеєра.
3. Розробити детальну архітектуру CI/CD конвеєра, що охоплює стадії інтеграції, тестування, стейджингу та продуктивного розгортання.

4. Практично реалізувати автоматичну збірку (EAS Build), тестування та доставку (EAS Submit) додатку в магазини App Store та Google Play.
5. Впровадити механізм OTA-оновлень для миттєвої доставки змін JavaScript-коду.
6. Здійснити кількісну оцінку ефективності впровадженої системи, порівнюючи ключові метрики CI/CD (наприклад, Deployment Frequency та Lead Time for Change) з ручним процесом.

**Наукова новизна**

Наукова новизна одержаних результатів дослідження полягає у тому, що для вирішення проблеми ресурсомісткого та повільного розгортання мобільних додатків на гібридних фреймворках був використаний новий та ефективний інженерний підхід, а саме: розроблено та теоретично обґрунтовано модель комплексного CI/CD конвеєра, спеціально оптимізованого для екосистеми Expo Application Services (EAS) та React Native, яка забезпечує безшовну автоматизацію всіх етапів: від інтеграції коду до фінальної доставки в App Store та Google Play; синтезовано інтегрований алгоритм керування версіями та конфігураціями, який поєднує можливості CI-сервера (наприклад, GitHub Actions) з інструментарієм EAS Build та EAS Submit, дозволяючи автоматично генерувати нативні бінарні файли та керувати підписанням коду без необхідності ручного втручання. Запропоновано та апробовано метод оптимізованого використання OTA (Over-The-Air) Updates у складі CI/CD конвеєра, що дозволяє миттєво доставляти оновлення JavaScript-коду, значно скорочуючи час реакції на критичні фікси, при цьому забезпечуючи необхідний механізм відкату.

**Практичне значення одержаних результатів**

Практичне значення одержаних результатів дослідження полягає у тому, що була розроблена та реалізована працездатна модель CI/CD системи, яка повністю автоматизує процес розгортання мобільних додатків на Expo та React Native. Впроваджено практичний інструментарій, який мінімізує час виходу продукту на ринок (Time-to-Market) шляхом скорочення циклу збірки та публікації, що є критично важливим для комерційних проектів. Доведена економічна ефективність підходу: система дозволяє суттєво зменшити операційні витрати, пов'язані з ручним керуванням нативними збірками, сертифікатами та взаємодією з магазинами додатків. Розроблена модель є масштабованим та гнучким рішенням, яке може бути використане в якості типового шаблону (best-practice blueprint) для команд розробників, що працюють з Expo та React Native, забезпечуючи високу частоту деплою та надійність кінцевого продукту. Продемонстрована можливість інтеграції автоматичного тестування та моніторингу в конвеєр, що підвищує загальну якість та стабільність мобільних додатків. З цього можна зробити висновок, що представлений у роботі підхід може використовуватись в якості ефективного та стандартизованого рішення для прискорення доставки мобільних додатків у сучасному середовищі розробки.

**Апробація одержаних результатів**

Результати дослідження були представлені на науково-практичній конференції студентів, аспірантів, докторантів і молодих вчених Запорізького національного університету «Молода наука-2025» [1].

**Глосарій**

*CI/CD* (*англ.Continuous Integration / Continuous Delivery*) — це набір практик, спрямованих на автоматизацію та моніторинг усіх етапів розробки програмного забезпечення: від інтеграції змін до випуску продукту.

*Неперервна Інтеграція (англ.Continuous Integration (CI))* — це практика, за якої розробники часто інтегрують свій код у спільну гілку, де він

автоматично перевіряється (збирається та тестується) для раннього виявлення помилок.

*Неперервна Поставка (англ.Continuous Delivery (CD))* — це практика, за якої код, що пройшов усі тести, автоматично готується до випуску, але розгортання у виробниче середовище може бути запущене вручну.

*Неперервне Розгортання (англ.Continuous Deployment)* — це практика, за якої кожна зміна, що пройшла всі етапи тестування, автоматично випускається для кінцевих користувачів без ручного підтвердження.

*React Native (англ.React Native)* — це фреймворк з відкритим вихідним кодом, що дозволяє створювати крос-платформенні мобільні додатки, використовуючи JavaScript.

*Expo* — це набір інструментів і сервісів, що спрощує розробку, збірку та розгортання додатків на базі React Native.

*EAS (англ.Expo Application Services)* — це хмарні сервіси від Expo для автоматизованої збірки, подачі в стори та оновлень.

*EAS Build* — це хмарний сервіс для віддаленої збірки нативних бінарних файлів (.apk, .ipa) для додатків Expo.

*EAS Submit* — це інструмент для автоматичного завантаження зібраних бінарних файлів у магазини додатків (App Store Connect та Google Play Console).

*OTA-оновлення (англ.Over-The-Air (OTA) Updates)* — це технологія, що дозволяє доставляти оновлення JavaScript-коду та асетів користувачам безпосередньо, минаючи модерацію в магазинах.

*Підписування Коду (англ.Code Signing)* — це процес цифрового підпису мобільного додатку сертифікатами розробника, необхідний для підтвердження джерела та запобігання зміні коду.

*Fastlane* — це набір інструментів з відкритим вихідним кодом для автоматизації рутинних завдань випуску мобільних додатків.

*Артефакт Збірки ( англ.Build Artifact)* — це кінцевий файл (бінарний файл або пакет), створений у результаті процесу збірки, готовий до розгортання.

*CI-сервер* — Інструмент (наприклад, GitHub Actions, GitLab CI), який керує та запускає автоматизовані конвеєри CI/CD.

*Конвеєр / Пайплайн (англ.Pipeline)* — це послідовність автоматизованих кроків (стадій), що виконуються для доставки нової версії програмного забезпечення.

*Юніт-тестування (англ.Unit Testing)* — це тестування найменших, ізольованих частин коду (функцій, компонентів) для перевірки їхньої коректної роботи.

*Наскрізне тестування (E2E) (англ.End-to-End Testing)* — це тестування функціональності додатку від початку до кінця, імітуючи дії реального користувача у різних середовищах.

*Стейджинг (англ.Staging)* — це середовище, що максимально імітує виробниче (production) середовище, використовується для фінального тестування перед випуском.

*Відкат (англ.Rollback***)** — це механізм або дія повернення системи (або оновлення) до попереднього, стабільного стану у разі виявлення критичних помилок після розгортання.

*Управління версіями (англ.Versioning)* — це процес присвоєння унікальних ідентифікаторів (номерів версій) різним станам розробки додатку (наприклад, 1.0.0 або build number).

# РОЗДІЛ 1 ТЕОРЕТИЧНЕ ОБҐРУНТУВАННЯ ТА ОГЛЯД ТЕХНОЛОГІЧНИХ ЗАСОБІВ АВТОМАТИЗАЦІЇ CI/CD

## 1.1 Обґрунтування актуальності та концепції CI/CD

Сучасний світ характеризується постійною технологічною трансформацією, де програмне забезпечення виступає ключовим каталізатором прискорення та масштабування організацій усіх типів. Рух DevOps, із якого виникли сучасні практики, є відповіддю на інженерний виклик створення безпечних, стійких та швидко еволюціонуючих розподілених систем у великому масштабі. Оскільки необхідність цифрової трансформації визнана імперативом (згідно з дослідженням Gartner, 47% генеральних директорів відчувають тиск щодо її реалізації, Panetta 2017), організації повинні освоїти ці можливості, щоб зберегти конкурентоспроможність.

Незважаючи на успіхи веб-гігантів (Amazon, Google) та традиційних великих компаній, звіт Forrester (Stroud et al. 2017) вказує на те, що значна частина галузі (близько 31%) досі не використовує критично важливі принципи, такі як неперервна інтеграція та неперервна поставка (CI/CD). Цей розрив, а також схильність керівників до переоцінки власної зрілості DevOps (Klavens et al. 2017), підкреслює критичну необхідність точного вимірювання цих можливостей та їхнього подальшого впровадження [5].

Впровадження CI/CD (Continuous Integration/Continuous Delivery) є фундаментальним для сучасного інженерного процесу. Воно забезпечує надійність, гнучкість, оптимізацію витрат та ефективність. CI/CD побудований на п'яти основних принципах:

1. Автоматизація.
2. Безперервне розгортання.
3. Автоматизоване тестування.
4. Безперервний моніторинг.

5. Петлі зворотного зв'язку.

Сучасна розробка мобільних та веб-додатків вимагає високої швидкості інтеграції змін та оперативного доставки нових функцій користувачам. У цьому контексті підхід Continuous Integration / Continuous Delivery (CI/CD) стає ключовим елементом ефективного процесу розробки [14]. Дослідження показують, що застосування CI/CD дозволяє скоротити час інтеграції, підвищити якість коду та знизити ризик помилок при випуску нових версій.

Актуальність CI/CD зростає особливо для крос-платформених проектів, таких як додатки на React Native або Flutter, де одночасна підтримка різних платформ підвищує складність розробки та збірки [19]. Використання CI/CD дозволяє автоматизувати збірку, тестування, підписування та розгортання додатків, що забезпечує стабільність і відтворюваність релізів.

Концептуально CI/CD включає два основні процеси:

1. Continuous Integration (CI) — безперервна інтеграція змін коду у спільний репозиторій із автоматизованим тестуванням, що дозволяє виявляти дефекти на ранніх етапах розробки. Це підвищує якість коду та скорочує час відгуку на виявлені помилки [3].
2. Continuous Delivery / Continuous Deployment (CD) — автоматизація процесу доставки та розгортання додатку в тестові або продукційні середовища. CD дозволяє швидко доставляти новий функціонал користувачам та забезпечує стабільність роботи сервісу, знижуючи ризики при релізах багатофункціональних оновлень [20].

Наукові дослідження демонструють, що впровадження CI/CD у малих та великих командах має істотний вплив на продуктивність розробки та якість програмного продукту. Наприклад, систематичний аналіз практик CI/CD показав, що автоматизація конвеєрів сприяє кращому управлінню змінами, полегшує масштабування команд і дозволяє швидко реагувати на зміни вимог [20].

Ключові Аспекти CD: Цільові Середовища (Таргети): Як цільовий об'єкт (таргет) виступає будь-яке середовище, де код буде використовуватися за призначенням. У мобільній розробці це включає:

- Середовища для QA-спеціалістів (наприклад, тестові канали TestFlight або Google Play Beta Tracks).
- Проміжні Deployment Environment (Staging) для фінальної перевірки.
- Магазини додатків (App Store та Google Play) для кінцевих користувачів.

Процес доставки, включно з необхідними кроками (наприклад, підписання коду, завантаження артефактів), є повністю автоматизованим, але фінальне рішення про випуск у виробниче середовище може прийматися людиною (в цьому різниця між CD та Continuous Deployment). Створення централізованого середовища для збірки та доставки забезпечує уніфікований та повторюваний процес, гарантуючи, що збірка, протестована на Staging, ідентична збірці, що постачається в Production. Тож,CD є мостом між розробкою та користувачем, забезпечуючи швидку, надійну та безперервну поставку перевіреного програмного забезпечення.

Автоматизований процес CI/CD функціонує на основі чотирьох ключових технічних компонентів:

- тригер (trigger),
- конвеєр (pipeline),
- завдання (job)
- крок (step).

Тригери є рушіями, що ініціюють конвеєр, і поділяються на ручні (запуск через інтерфейс, наприклад, для повторного тестування), часозалежні (запуск за розкладом, наприклад, нічні збірки) та Git-залежні (запуск при подіях у репозиторії, як-от створення Pull Request або новий commit). Конвеєр (Pipeline) є інструкцією, що описує весь процес автоматизації (збірка, тестування, деплой), і зазвичай визначається у конфігураційному файлі у форматі YAML (.yml), розміщеному в корені

проєкту. Кожен конвеєр складається з одного або декількох завдань (Jobs), які виконуються послідовно або паралельно, а кожне завдання, своєю чергою, включає низку кроків (Steps) — мінімальних одиниць виконання (наприклад, виконання команди лінтингу чи збірки) [2].

Отже, важливим аспектом є також узгодження CI/CD із практиками DevOps та загальним життєвим циклом розробки (SDLC). Інтеграція автоматизованих конвеєрів на різних етапах SDLC дозволяє:

- забезпечити постійне тестування та контроль якості на ранніх стадіях розробки;
- прискорити доставку функціоналу від етапу розробки до продакшн-середовища;
- зменшити людський фактор і ризик помилок при ручному розгортанні;
- забезпечити відстеження ефективності розробки та зворотний зв’язок у реальному часі.

Таким чином, концепція CI/CD не тільки підвищує ефективність процесу розробки, але й є невід’ємною складовою інтеграції в SDLC, забезпечуючи безперервність процесів тестування, збірки та доставки програмного продукту. Це плавно підводить до наступного розділу, де розглядається інтеграція CI/CD у життєвий цикл розробки програмного забезпечення та її практичне застосування в мобільних проєктах.

## 1.2 Інтеграція CI/CD у життєвий цикл розробки (SDLC)

Життєвий Цикл Розробки Програмного Забезпечення (Software Development Life Cycle, SDLC) є методологічною основою, яка забезпечує структурований та послідовний підхід до процесу створення, розгортання та підтримки програмного продукту. SDLC являє собою каркас, що розбиває складний процес розробки на дискретні, керовані етапи, сприяючи злагодженій та ефективній роботі команди. Для забезпечення Неперервної

Поставки (CD), як центральної теми даного дослідження, необхідно, щоб фази реалізації, тестування та розгортання у рамках SDLC були максимально автоматизовані за допомогою інструментів CI/CD.

Розуміння та ефективне застосування SDLC є фундаментальним для комерційної розробки з кількох ключових причин:

1. Структурування та управління: SDLC надає чітку структуру для роботи над проєктом, дозволяючи розбити його на конкретні фази (наприклад, планування, аналіз, дизайн, реалізація, тестування, розгортання (див.рис. 2). Це значно спрощує контроль за ходом виконання робіт та управління проєктними ризиками.

2. Підвищення якості продукту: кожен етап життєвого циклу інтегрує процеси перевірки та валідації якості (QA). Систематичне тестування та огляд, вбудовані в SDLC, гарантують, що дефекти виявляються на ранніх стадіях, що безпосередньо призводить до підвищення кінцевої якості програмного забезпечення.

3. Оптимізація ресурсів: методологія допомагає оптимізувати використання часових та фінансових ресурсів. Чітке визначення вимог і завдань на кожному етапі запобігає непередбачуваним змінам і перевитратам, що є критично важливим для комерційної успішності проєкту.

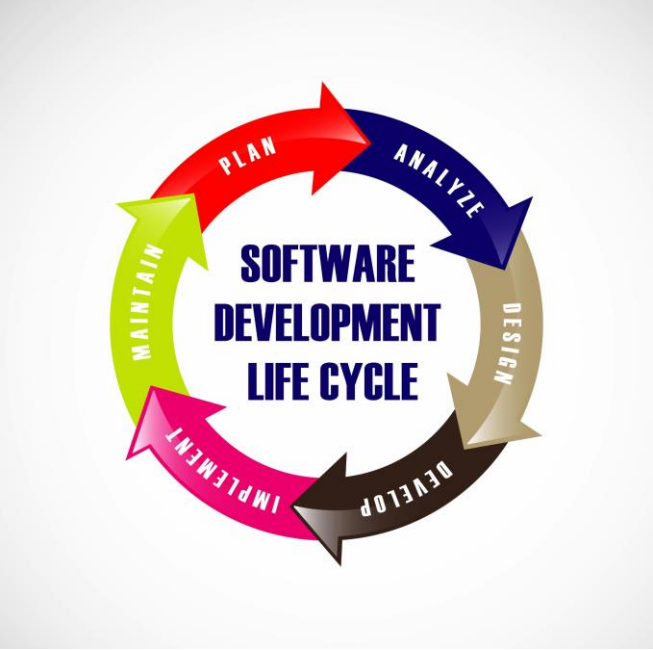


Рисунок 1 – Схема процесу SDLC

Процес SDLC зазвичай складається з декількох ключових фаз, кожна з яких робить внесок в успішну розробку програмного забезпечення. Основні фази SDLC включають планування, реалізацію, тестування та розгортання, але цим список не обмежується. Кожна фаза відіграє вирішальну роль в ефективному проєктуванні програмного забезпечення, задоволенні потреб користувачів та забезпеченні своєчасної доставки продукту.

Фаза 1: Планування (Planning) є фундаментом проєкту, де збираються та документуються цілі, завдання та вимоги до майбутнього продукту. Стейкхолдери спільно визначають обсяг проєкту, встановлюють терміни та розподіляють необхідні ресурси. Планування визначає чіткий напрямок роботи, забезпечуючи однакове розуміння цілей усіма учасниками.

Фаза 2: Аналіз Доцільності (Feasibility Analysis).На цій фазі команда оцінює, чи є проєкт технічно та фінансово життєздатним для реалізації. Проводиться оцінка технічних вимог, кошторис витрат та обов'язковий аналіз ризиків. Це допомагає виявити потенційні виклики та визначити, чи варто продовжувати інвестиції у проєкт.

Фаза 3: Проєктування Системи (System Design) включає створення архітектури та детального технічного плану програмного забезпечення. Розробляються специфікації дизайну, зокрема інтерфейсу користувача, та визначаються вимоги сумісності. Метою є створення чіткого креслення (blueprint), яке описує, як система функціонуватиме.

Фаза 4: Реалізація (Implementation), або розробки, перетворює проєктні специфікації на функціональний додаток. Розробники пишуть код, суворо дотримуючись стандартів кодування та найкращих практик. Результатом має бути ефективний, безпечний та зручний для подальшого обслуговування програмний продукт.

Фаза 5: Тестування (Testing) є критичною фазою, спрямованою на виявлення дефектів, помилок та збоїв. Використовуються різноманітні типи

тестування, зокрема юніт-тестування та автоматизоване тестування. Мета — гарантувати, що програмне забезпечення працює належним чином відповідно до початкових вимог.

Фаза 6: Розгортання (Deployment).Після успішного тестування рішення розгортається для кінцевих користувачів, часто починаючи з бета-тестування чи пілотного запуску. Розгортання може здійснюватися локально або у хмарному середовищі. Вибір стратегії доставки визначає простоту доступу користувачів до програмного забезпечення.

Фаза 7: Обслуговування (Maintenance).Обслуговування є завершальною, але постійною фазою SDLC, яка триває після випуску продукту. Вона охоплює вирішення виявлених проблем, застосування оновлень та додавання нових функцій. Безперервне обслуговування необхідне для забезпечення актуальності та функціональності програмного забезпечення протягом усього його життєвого циклу [3].

### 1.2.1 Концептуальна Роль DevOps у Модернізації Життєвого Циклу Програмного Забезпечення

DevOps слугує потужним каталізатором у прискоренні життєвого циклу розробки програмного забезпечення (SDLC). Шляхом просування автоматизації тестування, розгортання та управління інфраструктурою, DevOps значно скорочує цикли розробки. Це прискорення забезпечує постачання висококласного, безпомилкового програмного забезпечення, одночасно зменшуючи час виходу на ринок (time-to-market). У сучасному динамічному ландшафті розробки програмного забезпечення обов'язковим є впровадження стратегій, що підвищують продуктивність та співпрацю. DevOps виступає як трансформаційна сила, яка переформатовує традиційний SDLC. Завдяки культивуванню культури, зосередженої на неперервній інтеграції, розгортанні та співпраці, DevOps ефективно долає розрив між розробкою та операціями [6].

На проєкті розробки мобільного додатку DevOps-інженер може використовувати широкий спектр інструментів для автоматизації, моніторингу та управління процесами розробки:

1. Системи управління конфігурацією. Інструменти, такі як Ansible або Puppet, застосовуються для автоматизації налаштування серверних середовищ та інфраструктури, забезпечуючи стандартизацію та відтворюваність середовищ розробки, тестування та продакшну.
2. Інструменти CI/CD. Системи Jenkins, CircleCI, GitLab CI/CD дозволяють автоматизувати процеси збірки, тестування та розгортання мобільних додатків, скорочуючи час інтеграції змін та підвищуючи стабільність релізів.
3. Контейнерні технології. Платформи Docker та Kubernetes забезпечують портативність і швидке розгортання додатків, дозволяють створювати ізольовані середовища та масштабувати сервіси за потребою.
4. Моніторингові системи. Інструменти на кшталт Prometheus або Nagios використовуються для відстеження продуктивності додатків та інфраструктури в реальному часі, виявлення потенційних проблем і забезпечення стабільної роботи системи.
5. Системи управління версіями коду. Інструменти Git або SVN дозволяють організувати спільну роботу над кодом, вести історію змін, контролювати інтеграцію нових функцій та підтримувати паралельну розробку в команді [21].

Наведемо приклад використання DevOps платформою Spotify:

Spotify застосовує DevOps для забезпечення швидкої розробки та стабільного доступу до свого сервісу. Компанія використовує контейнеризацію (Docker) для пакування додатків і залежностей, що дозволяє легко розгортати та масштабувати сервіси. Для автоматизації збирання, тестування та розгортання коду застосовуються інструменти CI/CD, такі як Jenkins та CircleCI, що забезпечує безперервну інтеграцію та доставку нових функцій.

Spotify також активно використовує моніторинг і логування (Prometheus, ELK Stack) для оперативного виявлення та усунення проблем у реальному часі. Інтеграція автоматизованого тестування дозволяє підвищувати якість коду та знижувати ризики помилок у багатофункціональних оновленнях.

Таким чином, DevOps-практики дозволяють Spotify швидко адаптуватися до змін на ринку і підтримувати безперебійну роботу додатку, забезпечуючи користувачам доступ до улюблених треків та плейлистів у будь-який час і в будь-якому місці.

### 1.3 Теоретичні Основи Оцінки Продуктивності: Метрики DORA

Існує низка наукових робіт, що обговорюють значний вплив практик DevOps на SDLC. Неперервна інтеграція/неперервна поставка (CI/CD), автоматизоване тестування та покращена якість — це лише деякі з процесів, які DevOps використовує для допомоги командам доставляти програмне забезпечення швидше та якісніше. Тим не менш, опір змінам, інтеграція інструментів та брак навичок продовжують створювати великі перешкоди, які запобігають широкому впровадженню. Деякі сфери потребують подальших досліджень для виявлення стратегій подолання цих викликів у міру розвитку DevOps [11].

Сучасні команди розробників потребують не просто високої швидкості, але й надійних механізмів доставки коду, що забезпечує реальну цінність. Метрики CI/CD — це кількісні показники, які використовуються для оцінки продуктивності та надійності конвеєрів неперервної інтеграції та розгортання. Вони вимірюють увесь процес доставки, від частоти випуску коду до швидкості усунення проблем. Ці дані є критично важливими, оскільки вони виявляють вузькі місця у робочому процесі, вказують на потенційні причини збоїв під час розгортання та допомагають зрозуміти, як команда може підвищити свою ефективність. Відстежуючи частоту розгортання, час циклу

та швидкість відновлення, команди можуть приймати обґрунтовані рішення щодо оптимізації своїх процесів [4].

Ефективне використання метрик CI/CD допомагає узгодити цілі розробки з бізнес-завданнями, демонструючи, чи постачається якісний код із належною швидкістю та чи задовольняються потреби користувачів. Кожен показник — від успішності збірки до часу розгортання — створює чітку картину стану доставки програмного забезпечення. Ці метрики слугують ключовим інструментом для постійного вдосконалення, забезпечуючи надійність і передбачуваність процесів розгортання як для невеликих команд, так і для великих корпоративних систем.

Наукова система Метрик DORA (DevOps Research and Assessment), запроваджена Forsgren et al. (2018), слугує фундаментальною основою для кількісної оцінки ефективності DevOps-практик. Масштабні галузеві звіти від провідних гравців, таких як Atlassian, Gitlab та GitHub, протягом останніх років емпірично довели, що систематичне застосування DORA корелює з підвищенням організаційної зрілості, покращенням операційної стійкості та прискоренням інкрементальної доставки змін.

Вимірювання Впливу DevOps. Метрики продуктивності є критично важливими для оцінки ефективності практик DevOps. До загальновживаних метрик належать (див рис. 2):

1. Частота Розгортання (Deployment Frequency): Кількість релізів, розгорнутих протягом певного часового проміжку.
2. Час Виконання Змін (Lead Time for Changes): Час від коміту коду до його розгортання у виробничому середовищі (production).
3. Коефіцієнт Невдач Змін (Change Failure Rate): Відсоток змін, які призводять до збоїв у виробничому середовищі.
4. Середній Час Відновлення (Mean Time to Recovery, MTTR): Час, необхідний для відновлення сервісу після збою [9].

| DORA Metric | Elite | High | Medium |
|---|---|---|---|
| Deployment Frequency | Multiple per day | Once per week to once per month | Once per month to twice per year |
| Lead Time for Changes | Less than one hour | One day to one week | One to six months |
| Mean Time to Recover | Less than one hour | Less than one day | One day to one week |
| Change Failure Rate | 0%–15% | 15%–30% | 15%–30% |

Рисунок 2 — DORA Metrics Performance Benchmarks (Wilkes et al., 2023)

Додатково, дослідницькі підрозділи Google Cloud та ThoughtWorks підкреслили необхідність застосування DORA як стандарту архітектурного вимірювання для прямого покращення комерційних результатів (Google Cloud, 2024; Thoughtworks, 2024), що надає цим метрикам найвищу релевантність для цілей даної роботи.

Розглянемо детальніше кожну з метрик спираючись на дослідження [8, с.11-12].

Deployment Frequency, Cloud Security. Ключовим показником, що відображає здатність команди часто та успішно випускати релізи у виробниче середовище. У контексті хмарної безпеки це має прямий позитивний вплив:

1. Зменшення Часу Вразливості: Часті розгортання мінімізують проміжок часу, протягом якого програмні застосунки залишаються вразливими, дозволяючи оперативніше впроваджувати патчі та виправлення.
2. Пом'якшення Ризиків: Обсяг інкрементальних розгортань дає змогу швидше інтегрувати виправлення безпеки, суттєво зменшуючи площу атаки.

Команди з найвищою продуктивністю здійснюють розгортання кілька разів на день, що сприяє безпечним практикам розробки завдяки створенню малих, інкрементальних релізів, які забезпечують кращу трасованість та значно спрощують відкат змін у разі виявлення дефекту чи критичної вразливості (Forsgren et al., 2017). Дослідження Sallin et al. (2021) підтвердило, що саме автоматизоване розгортання дозволяє командам швидко виявляти та

реагувати на проблеми у продакшені, що одночасно підсилює як швидкість доставки, так і загальну кіберстійкість системи.

Lead Time for Change and Cloud Security. LTFC є критично важливим показником у сфері кібербезпеки та хмарних операцій. Організація, яка має низький LTFC, демонструє здатність швидко реагувати на нові загрози безпеки, оперативно впроваджуючи необхідні патчі, оновлення конфігурацій та усуваючи вразливості, тим самим ефективно мінімізуючи загальні ризики.

Швидкість реагування команди на вразливості є запорукою успішного пом'якшення експлойтів нульового дня (Zero-Day Exploits) та кібератак. Автоматизація вимірювання LTFC в операціях безпеки створює швидкий цикл зворотного зв'язку (Forsgren et al., 2017), що повністю узгоджується з принципами розробки програмного забезпечення та допомагає команді забезпечувати постійну відповідність вимогам безпеки. Таким чином, низький LTFC є ключовою метрикою для ефективного реагування на інциденти, швидкого відновлення після помилкових конфігурацій інфраструктури або експлойтів.

Mean Time to Recover (MTTR) and Cloud Security. В організаціях, що працюють у сфері хмарної безпеки, низький показник MTTR свідчить про високу операційну стійкість. Команди з низьким MTTR демонструють здатність швидко виявляти, стримувати та відновлюватися після інцидентів безпеки, спричинених помилками, збоями чи кібератаками. Це зміцнює довіру стейкхолдерів, забезпечуючи мінімізацію перебоїв у роботі сервісів та підтримання безперервності бізнесу.

Інтеграція DevSecOps у робочі процеси покращує моніторинг інцидентів у реальному часі та підвищує ефективність відновлення. Ключова перевага автоматизованого конвеєра DevSecOps полягає в тому, що реалізація заходів безпеки (наприклад, патчів) виконується один раз, а розгортання здійснюється узгоджено у багатьох середовищах (Wilkes et al., 2023). Ця ефективність забезпечує швидше зменшення ризиків та узгоджені патчі безпеки, у порівнянні з традиційними розділеними підходами.

Отже, метрики DORA є ключовим інструментом для об'єктивної оцінки ефективності DevOps. Ці чотири показники є результатом масштабних емпіричних досліджень (Forsgren et al., 2018) і слугують не лише технічними індикаторами, а й мостом, що поєднує інженерну діяльність із бізнес-результатами. Вимірюючи пропускну здатність та кіберстійкість, DORA створює необхідний кількісний фреймворк.

### 1.4 Висновки з розділу 1

Перший розділ роботи присвячено теоретичному обґрунтуванню та визначенню концептуальних засад автоматизації CI/CD. Було проведено аналіз еволюції методологій розробки, починаючи від традиційного життєвого циклу розробки програмного забезпечення (SDLC) до сучасних DevOps-практик. Розділ встановив, що неперервна інтеграція та поставка (CI/CD) є ключовим інструментом для подолання недоліків послідовних моделей та забезпечення автоматизованої, швидкої та надійної доставки програмного забезпечення.

Було деталізовано, що CI/CD трансформує життєвий цикл розробки, фокусуючись на автоматизації критичних фаз — реалізації, тестування та розгортання. Ця автоматизація функціонує на основі принципу "Конвеєр як код" (Pipeline as Code), використовуючи декларативні файли (YAML) для визначення технічних компонентів: тригерів, конвеєрів, завдань та кроків. Вибір відповідної Git-стратегії, як-от Feature Branching або Trunk-based Development, є критичною передумовою для ефективного використання цих автоматизованих процесів.

Найважливіше, було науково обґрунтовано, що оцінка успішності впровадження CI/CD повинна ґрунтуватися на об'єктивних кількісних показниках. Для цієї мети обрано метрики DORA, які забезпечують комплексний погляд, балансуючи між швидкістю (частота розгортання та час

виконання змін) та стабільністю (коефіцієнт невдач змін та час відновлення сервісу).

Отже, теоретичний аналіз підтверджує, що впровадження DevOps та його ключових практик неперервної інтеграції та поставки (CI/CD) є необхідним для підтримки конкурентоспроможності в сучасній цифровій економіці.

Таким чином, Розділ 1 заклав міцний теоретичний фундамент, визначивши ключові концепції, принципи функціонування та критерії оцінки. Ця теоретична база є необхідною для обґрунтованого переходу до Розділу 2, де буде здійснено вибір та детальний огляд конкретних технологічних засобів (React Native, Expo, GitHub Actions) для реалізації практичного CI/CD конвеєра.

# РОЗДІЛ 2 ОГЛЯД ТЕХНОЛОГІЧНОЇ ПЛАТФОРМИ REACT NATIVE, ІНСТРУМЕНТАРІЮ ТА СПЕЦИФІКИ АВТОМАТИЗАЦІЇ

## 2.1 Архітектурний Аналіз React Native, Expo та Механізму Неперервної Поставки

Сучасні реалії, що стосуються проєктів мобільних розробок, повинні бути масштабовані, гнучкі та підтримувані. Саме фреймворк React Native дозволяє швидко розробляти мобільні застосунки, хоча великі проєкти часто страждають від монолітної структури, що суттєво ускладнює їхню підтримку, імплементацію нових модулів, тестування та подальший розвиток.

Фреймворк з відкритим вихідним кодом React Native (RN), представлений у 2015 році, набув значної актуальності завдяки комбінованій парадигмі, яка синтезує гнучкість середовища Node.js з принципами нативної мобільної розробки.

Основний архітектурний принцип React Native полягає у наданні крос-платформних API JavaScript (див. рис. 3), що забезпечують можливість виклику платформо-специфічних функцій (зокрема, засобів Objective-C/Swift для iOS або Java/Kotlin для Android) для безпосереднього використання нативних компонентів операційних систем. Цей механізм гарантує створення застосунків із єдиною кодовою базою JavaScript, зберігаючи при цьому нативну продуктивність та високу відповідність цільовим інтерфейсам.

Для відображення логіки React на цільовій хост-платформі (iOS або Android), рендерер React Native виконує послідовну низку дій, відому як Конвеєр Рендерингу. Цей конвеєр активується як при початковому відображенні, так і при кожному оновленні стану користувацького інтерфейсу, і може бути розділений на три основні фази: Рендеринг, Фіксація (Комміт) та Монтування (див. табл.1).

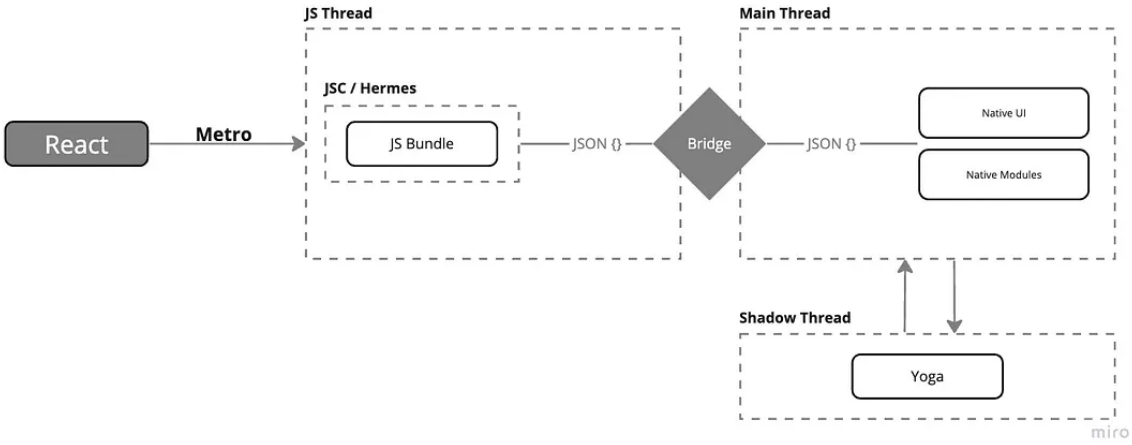



Рисунок 3 — Міст архітектури React Native

Таблиця 1— Конвеєр Рендерингу

| **Фаза** | **Середовище** | **Опис Процесу** | **Результат / Ключові Структури** |
|---|---|---|---|
| Рендеринг (Render) | JavaScrip | React виконує логіку програми (компоненти), яка генерує Дерево елементів React на JavaScript. На основі цього дерева рендерер створює Тіньове дерево React. | Відображення нативних компонентів (Native Views) на екрані пристрою. |
| Фіксація (Commit) | Рендерер (C++) | Після повного створення Тіньового дерева React ініціюється фіксація. Цей етап визначає Дерево елементів React та щойно створене Тіньове дерево React як «наступне дерево» для відображення. | Запуск розрахунку макета (Layout calculation) та підготовка до монтування. |
| Монтування (Mount) | Хост-платформа | Тіньове дерево React, тепер з результатами розрахунку макета, остаточно перетворюється на Дерево хост-подання (Host View Tree). | Відображення нативних компонентів (Native Views) на екрані пристрою |

Це послідовне розділення відповідальності між середовищем JavaScript, рівнем C++ та нативною хост-платформою є ключовим архітектурним

елементом React Native. Воно, однак, підкреслює складність втручання у нативний процес (фаза 3), що додатково обґрунтовує необхідність використання уніфікованих CI/CD рішень, таких як EAS, для керування нативними залежностями та збіркою [10].

### 2.1.2 Нова Архітектура React Native: Принципи JSI, Fabric та TurboModules

Версія React Native 0.82 стала важливим кроком — вона запускається виключно на «Новій архітектурі» (New Architecture) (див.рис. 4) і більше не підтримує стару архітектуру, що дозволяє зменшити розмір установки та спростити кодову базу. Додано експериментальну підтримку Hermes V1, оновлено React до версії 19.1.1, а також додано підтримку API вузлів DOM-подібного інтерфейсу у компонентах React Native.

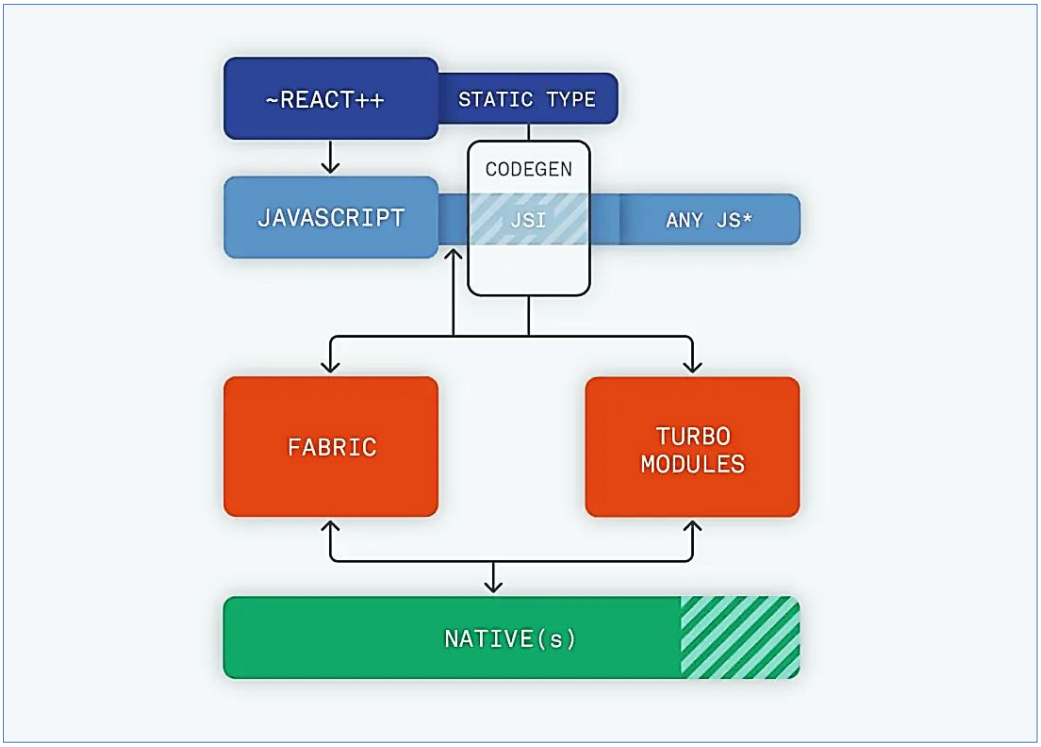



Рисунок 4 — Нова архітектура React Native 0.82

Серед інших важливих змін — введення DOM-Node API для React Native, що дає можливість використовувати методи на кшталт parentNode, childNodes,

getBoundingClientRect та ін., наближаючи інтерфейс до веб-моделі. Додані веб-орієнтовані API продуктивності (Performance APIs) та тип збірки «debugOptimized» для Android, який дозволяє відладчику JS працювати, але при цьому використовувати оптимізації C++ і досягати ~60fps у режимі відладки. А необроблені проміси тепер будуть виводитися через console.error, а також наводиться список застарілих API та змін для iOS/Android/C++ [12].

JavaScript Interface (JSI) у новій архітектурі React Native замінює традиційний Bridge і забезпечує прямий зв’язок між JavaScript та нативними модулями, мінімізуючи накладні витрати на пересилку повідомлень; методи JSI можуть виконуватись як синхронно, так і асинхронно. Fabric, як нова система рендерингу, дозволяє керувати деревом компонентів безпосередньо через JSI, усуваючи необхідність дублювання його у JS та синхронізації через Bridge, що забезпечує пріоритетність обробки UI-задач над асинхронними операціями. Turbo Modules вирішують проблему повільного старту додатків, дозволяючи завантажувати нативні модулі лише за потребою та інтегрувати їх з JavaScript через JSI. CodeGen автоматизує генерацію нативних інтерфейсів для Turbo Modules та Fabric, створюючи C++ код на основі опису інтерфейсів у Flow або TypeScript. Комплексне застосування цих технологій забезпечує підвищення продуктивності, оптимізацію взаємодії між JavaScript та нативними платформами, а також спрощує розробку масштабованих кросплатформених додатків [13].

Нова архітектура React Native, що включає JSI (JavaScript Interface), Fabric та TurboModules, забезпечує значне підвищення продуктивності додатків та оптимізацію взаємодії між JavaScript і нативними модулями. JSI дозволяє безпосередньо викликати нативні API без накладних витрат bridge, Fabric оптимізує рендеринг UI та пріоритезує обробку інтерфейсних задач, а TurboModules забезпечує динамічне завантаження нативних модулів за потребою, що скорочує час старту додатку та підвищує масштабованість системи.

### 2.1.3 Роль EAS у Спрощенні Нативної Збірки та Оптимізації Доставки через OTA-оновлення

У сучасній розробці мобільних додатків особливе значення набуває швидкість та ефективність процесу збірки, особливо для крос-платформних рішень. Expo Application Services (EAS) виступає як комплексний набір сервісів, що спрощує створення, підписання та публікацію нативних бінарників для платформ iOS та Android, інтегруючись із CI/CD-процесами та автоматизуючи керування конфігураціями додатків. Крім того, EAS підтримує механізми доставки Over-the-Air (OTA) оновлень, що дозволяє оперативно розгортати зміни у JavaScript та ресурсах без потреби публікації нових версій у App Store чи Google Play. Важливим аспектом є те, що ефективність і гнучкість EAS значною мірою реалізуються завдяки новій архітектурі React Native, яка включає JSI, Fabric, Turbo Modules та CodeGen. Завдяки цим компонентам забезпечується швидка взаємодія між JavaScript і нативними модулями, синхронне оновлення UI та оптимізоване підвантаження нативних функціональних модулів, що робить процес збірки та оновлення додатків більш надійним і масштабованим.

EAS Build є хостинг-сервісом для створення нативних бінарних файлів додатків у проєктах на Expo та React Native. Він забезпечує спрощену та автоматизовану збірку, надаючи налаштування за замовчуванням, що оптимально підходять для таких проєктів, а також керує обліковими даними для підпису додатків у разі потреби. Сервіс полегшує внутрішній розподіл збірок серед команди та інтегрується з EAS Submit для публікації додатків у магазинах, забезпечуючи підтримку бібліотеки expo-updates для оперативного оновлення контенту. EAS Build призначений для використання з будь-якими нативними проєктами та дозволяє значно скоротити час від ініціалізації нового проєкту до його розповсюдження у магазинах додатків [18].

Сучасна розробка мобільних додатків потребує ефективних підходів для прискорення збірки та оптимізації процесів оновлення, особливо для крос-платформних проєктів. Expo Application Services (EAS) надає комплекс

інструментів, що дозволяє автоматизувати збірку, підписання та публікацію нативних бінарників для iOS та Android, інтегруючись із CI/CD-процесами та мінімізуючи ручні операції. Нова архітектура React Native включає JavaScript Interface (JSI), Fabric, Turbo Modules та CodeGen. JSI забезпечує прямий доступ JavaScript до нативних модулів, усуваючи накладні витрати на пересилку повідомлень через традиційний Bridge, що знижує latency та підвищує швидкодію. Fabric оптимізує рендеринг інтерфейсу, роблячи його синхронним та пріоритезуючи обробку UI-задач, що покращує плавність роботи додатку. Turbo Modules дозволяють динамічно підвантажувати нативні модулі лише за потребою, скорочуючи час старту додатків і підвищуючи їх продуктивність. CodeGen автоматизує створення нативних інтерфейсів для Turbo Modules і Fabric на основі Flow або TypeScript, спрощуючи інтеграцію нових функцій [14,15].

EAS Build дозволяє автоматизувати процес створення нативних бінарників у хмарі, управляти ключами підпису, а також інтегруватися з CI/CD-процесами для прискорення випуску версій. Використання таких сервісів підвищує стабільність і відтворюваність збірок, зменшує ручну роботу розробників і оптимізує доставку додатків для тестування та публікації у магазинах додатків. Сервіс EAS Update підтримує механізм Over-The-Air (OTA) updates, що дозволяє швидко доставляти зміни у JavaScript-код, стилі та ресурси без необхідності випуску нової нативної версії. OTA-оновлення використовують канали та гілки (channels / branches) для поетапного rollout, а також забезпечують можливість rollback у разі помилок. Безпека доставки забезпечується перевіркою цілісності та підписом пакетів [16].

Поєднання нової архітектури React Native та сервісів EAS забезпечує:

- значне підвищення продуктивності додатків (швидкий запуск, низька latency, оптимізований рендеринг UI);
- ефективну автоматизацію збірки та релізу через CI/CD;
- швидке та безпечне оновлення функціоналу без потреби випуску нових нативних версій ;

- підвищення масштабованості та надійності крос-платформених додатків, що скорочує час виходу продукту на ринок.

Таким чином, інтеграція EAS Build та Update з новою архітектурою React Native забезпечує сучасний підхід до крос-платформної розробки мобільних додатків, який поєднує продуктивність, надійність і гнучкість доставки оновлень [17].

Отже, інтеграція Expo Application Services (EAS) з новою архітектурою React Native забезпечує ефективне та надійне рішення для крос-платформної розробки мобільних додатків. Використання JSI, Fabric та Turbo Modules підвищує продуктивність, скорочує час запуску та оптимізує рендеринг інтерфейсу, тоді як EAS Build дозволяє автоматизувати збірку та інтегрувати її у CI/CD-процеси, зменшуючи ручну роботу та підвищуючи стабільність релізів. Механізм OTA-оновлень через EAS Update забезпечує швидку та безпечну доставку змін без необхідності випуску нових нативних версій, підтримуючи поетапний rollout і можливість rollback.

Застосування цих технологій у комплексі дозволяє значно скоротити час виходу продукту на ринок, підвищити масштабованість та надійність додатків, а також ефективно управляти процесом розробки та підтримки мобільних рішень. Таким чином, поєднання EAS та нової архітектури React Native формує сучасний і науково обґрунтований підхід до розробки високопродуктивних крос-платформених мобільних застосунків.

## 2.2 Огляд Інструментарію CI/CD та Специфіка Конфігурації Конвеєрів

Ефективна реалізація практик CI/CD, обґрунтованих у Розділі 1, вимагає використання спеціалізованих інструментів — CI-серверів, які здатні автоматично відстежувати зміни в репозиторії та виконувати послідовність завдань. При виборі інструментарію вирішальне значення має його відповідність сучасній парадигмі управління конвеєрами.

Сучасна розробка програмного забезпечення, особливо мобільних додатків, вимагає швидкої інтеграції змін та безперервної доставки функціоналу користувачам. CI/CD є ключовим інструментом для автоматизації цих процесів, що дозволяє командам скорочувати час від написання коду до його релізу, підвищувати стабільність додатків та зменшувати ризики помилок [22].

Серед найбільш поширених інструментів CI/CD можна виділити:

1. GitHub Actions є сервісом CI/CD, який інтегровано безпосередньо в GitHub. Конфігурація pipeline здійснюється через YML-файли, розміщені в каталозі .github/workflows. Сервіс дозволяє гнучко налаштовувати кожен етап CI/CD за допомогою директив job та step. Тригери можуть реагувати на різні події, такі як пуші в репозиторій, pull request або випуск релізу. Додатково, GitHub Actions надає доступ до широкого набору готових дій (pre-built actions) з GitHub Marketplace, які можна легко інтегрувати у workflow. Завдяки цьому GitHub Actions підходить як для невеликих проектів, так і для великих команд, що потребують адаптованих пайплайнів.

2. GitLab CI/CD є повністю інтегрованим сервісом у систему контролю версій GitLab, але його також можна використовувати для проектів на GitHub. Конфігурація pipeline здійснюється через файл .gitlab-ci.yml, де можна визначати стандартні етапи, такі як before_script, script та after_script, а також створювати кастомні job-и для більшої гнучкості. GitLab CI/CD підтримує детальне визначення етапів та стадій (stages), що дозволяє побудувати складні workflow для великих або багатомодульних проектів. Завдяки інтеграції з GitHub, розробники можуть поєднувати зручність GitHub для контролю версій із функціональністю GitLab для автоматизації CI/CD [24].

Лістинг 1 демонструє приклад конфігурації CI/CD для Android-проєкту з використанням GitHub Actions. Файл конфігурації визначає пайплайн для автоматичного збірки, тестування та розгортання додатку. Workflow запускається на події push або pull_request. Конфігурація містить етапи

налаштування середовища, встановлення залежностей, збірки, генерації APK, тестування, публікації у Google Play та збору логів з повідомленням у Slack.

Лістинг 1 — Sample YML configuration file (GitHub Actions)

```
1  name: Android CI
2  on: [push, pull_request]
3  jobs:
4    build:
5      runs-on: ubuntu-latest  # (1) Environment Setup
6      steps:
7        - name: Checkout code
8          uses: actions/checkout@v2
9        - name: Set up JDK and SDK
10         run: |
11           actions/setup-java@v2 --java-version '11'
12           sdkmanager "platforms;android-30" "build-
tools;30.0.3"
13
14       - name: Install dependencies  # (2) Dependency
Management
15         run: ./gradlew dependencies
16
17       - name: Build  # (3) Building
18         run: ./gradlew build assembleDebug
19
20       - name: Generate APK  # (4) Packaging
21         if: success()
22         uses: actions/upload-artifact@v3
23         with:
24           name: app.apk
25           path: app/build/apk/app.apk
26
27       - name: Test  # (5) Testing
28         run: ./gradlew test
29
30       - name: Deploy to Google Play  # (6)
Deployment
31         run: |
32           echo $ANDROID_KEYSTORE | base64 --decode >
keystore.jks
33           ./gradlew publish --no-daemon
34
35       - name: Collect Logs and Notify  # (7)
Feedback
36         run: |
37           ./gradlew collectLogs
```

```
    38           curl -X POST -H 'Content-type:
application/json' --data \
    39           '{"text":"Completed successfully."}'
$SLACK_WEBHOOK_URL
```

Пояснення кроків:

1. Environment Setup – налаштування ОС, JDK та Android SDK.
2. Dependency Management – встановлення залежностей через Gradle.
3. Building – збірка проекту та формування debug-білду.
4. Packaging – завантаження APK як артефакту для подальшого використання.
5. Testing – запуск unit-тестів для перевірки працездатності коду.
6. Deployment – публікація додатку у Google Play із розкодуванням Keystore.
7. Feedback – збір логів та повідомлення у Slack про завершення процесу.

Сучасна розробка мобільних та крос-платформних додатків вимагає високого рівня автоматизації процесів збірки, тестування та розгортання. Для цього застосовуються різноманітні інструменти CI/CD, які дозволяють командам скоротити час виходу продукту на ринок, підвищити якість коду та забезпечити стабільну роботу додатків. Серед таких рішень особливу увагу варто приділити Jenkins, CircleCI та Bitrise, які пропонують різні підходи до автоматизації залежно від потреб команди та специфіки проекту.

Jenkins є одним із найпопулярніших серверів автоматизації для CI/CD, що забезпечує повну гнучкість у створенні кастомізованих конвеєрів (pipeline). Його ключові характеристики:

- Плагіни та інтеграції: Jenkins має понад 1800 плагінів, які дозволяють інтегрувати систему з різними SCM (Git, SVN), системами тестування, хмарними платформами та інструментами для моніторингу.
- Declarative та Scripted Pipeline: забезпечує два підходи до конфігурації конвеєра, що дозволяє вибрати баланс між простотою та гнучкістю [25].

- Підтримка контейнеризації: можна запускати job-и у Docker-контейнерах, що забезпечує ізоляцію середовищ і повторюваність збірок.
- Недоліки: Jenkins потребує значних зусиль на підтримку, оновлення плагінів та управління інфраструктурою сервера, особливо при масштабних проектах [27].

Jenkins підходить для проєктів із високою складністю пайплайнів та специфічними вимогами до збірки та тестування, де SaaS-рішення можуть бути обмежені.

CircleCI – це хмарне SaaS-рішення для автоматизації CI/CD, яке підтримує як контейнеризовані, так і віртуальні середовища:

- Швидке розгортання та масштабування: завдяки хмарній архітектурі CircleCI дозволяє швидко запускати паралельні job-и, що скорочує загальний час виконання pipeline.
- Мобільна інтеграція: підтримує Android та iOS проєкти, інтегрується з Gradle, Fastlane та CocoaPods, що спрощує збірку та деплой мобільних додатків.
- Контейнеризація та кешування: забезпечує використання Docker-контейнерів та кешування залежностей для пришвидшення повторних збірок .

CircleCI підходить для команд, які хочуть швидко інтегрувати CI/CD без підтримки власного сервера та масштабувати пайплайни відповідно до навантаження.

Bitrise – SaaS-платформа, орієнтована на мобільну розробку, що надає готові workflow-шаблони для Android, iOS, Flutter та React Native, [30,31]:

- Простота налаштування: містить візуальний редактор workflow, що дозволяє легко створювати та змінювати конвеєри без ручного написання YML.

- Підтримка OTA та мобільних інструментів: інтеграція з TestFlight, Firebase App Distribution та Expo дозволяє автоматизувати тестування і доставку оновлень.
- Паралельне тестування та контейнеризація: використовує хмарні build-агенти та Docker-контейнери для прискорення виконання job-ів .

Bitrise підходить для невеликих та середніх команд мобільних розробників, яким потрібна готова хмарна платформа з мінімальним адміністративним навантаженням.

Огляд сучасних інструментів CI/CD показав, що ефективна автоматизація процесів збірки, тестування та деплойменту мобільних додатків забезпечується як гнучкими серверними рішеннями (Jenkins), так і хмарними SaaS-сервісами (GitHub Actions, CircleCI). Jenkins дозволяє створювати кастомізовані конвеєри з великою кількістю плагінів, проте потребує додаткової підтримки та конфігурації. GitHub Actions інтегрований безпосередньо з репозиторіями GitHub, що спрощує запуск workflows за подіями (push, pull request) і використання готових action з Marketplace. CircleCI орієнтований на мобільні проєкти, підтримує паралельне тестування, контейнеризацію та інтеграцію з мобільними інструментами, що забезпечує швидку доставку змін і стабільність конвеєрів. Всі ці інструменти дозволяють реалізовувати підхід Pipeline as Code, підвищуючи повторюваність, прозорість і масштабованість процесів CI/CD.

### 2.2.1 Pipeline as Code: управління CI/CD як кодом

Сучасні практики DevOps та CI/CD передбачають не лише автоматизацію процесів збірки, тестування та деплойменту, а й управління всією конфігурацією пайплайнів у вигляді коду, що зберігається разом із вихідним кодом проекту. Такий підхід відомий як «Pipeline as Code» (PaC) і є частиною ширшої концепції «as-code» (як, наприклад, Infrastructure as Code), що забезпечує версіонованість, прозорість і повторюваність процесів [32].

У практичному аспекті Pipeline as Code передбачає створення спеціальних файлів конфігурації (наприклад, YAML, JSON або Groovy), у яких описуються всі етапи пайплайна: збірка, тестування, деплоймент, генерація артефактів та збирання логів. Використання таких файлів дозволяє версіонувати конвеєри разом із вихідним кодом проекту, що забезпечує можливість контролю змін, рев'ю конфігурацій через pull request-и та відкат до попередніх версій [31].

Основні переваги використання Pipeline as Code включають:

- Повторюваність та консистентність процесів: будь-який запуск пайплайна відбувається у однакових умовах, що знижує ризик помилок через людський фактор;
- Масштабованість та стандартизація: у великих командах або при роботі з мікросервісною архітектурою пайплайни можна стандартизувати, створювати шаблони та централізовані бібліотеки блоків [33];
- Версіонування конфігурацій: зміни в pipeline зберігаються в системі контролю версій, що дозволяє відслідковувати історію модифікацій і виконувати rollback у разі потреби [31];
- Безпека та контроль: код пайплайнів проходить ті ж процедури рев'ю та тестування, що й звичайний вихідний код, зменшуючи ризики введення небезпечних налаштувань [32].
- У сучасних CI/CD-системах Pipeline as Code реалізується наступним чином:
- Jenkins: використання Jenkinsfile у Declarative або Scripted Pipeline, який версіонується разом із проектом і дозволяє повністю описати логіку збірки та деплойменту.
- CircleCI: конфігурація пайплайна зберігається у .circleci/config.yml, де визначаються job-и, стадії (stages) та умови виконання, включно з контейнеризацією та кешуванням.

- Bitrise: використання YAML-конфігурацій та візуального редактора workflow дозволяє описувати всі етапи конвеєра у вигляді коду та запускати їх на хмарних build-агентах.

Тож, Pipeline as Code дозволяє командам не тільки автоматизувати CI/CD, але й підвищити прозорість, керованість та безпеку процесів, створюючи основу для подальшого застосування метрик DevOps і експериментальної оцінки ефективності CI/CD, що буде розглянуто у наступному розділі дипломної роботи [33].

Отже, концепція Pipeline as Code дозволяє описувати весь процес CI/CD у вигляді коду, який версіонується разом із проектом. Це забезпечує повторюваність, прозорість і контроль змін у конвеєрах, підвищує стабільність збірки та деплойменту, а також спрощує масштабування та стандартизацію процесів. Використання Pipeline as Code у таких системах, як Jenkins, GitHub Actions, CircleCI та Bitrise, дозволяє командам ефективно керувати автоматизацією, інтегрувати нові функції та швидко адаптувати процеси CI/CD до змін у проекті.

## 2.3 Автоматизоване тестування у CI/CD для React Native

Автоматизоване тестування є ключовим елементом забезпечення якості програмного забезпечення у сучасних мобільних додатках. Для додатків на базі React Native, що використовують Expo та інші сучасні інструменти, інтеграція тестування у CI/CD-конвеєри дозволяє значно підвищити стабільність релізів, скоротити час доставки нових версій та зменшити ризик помилок у продуктивному середовищі. Основна мета автоматизованого тестування у CI/CD полягає у виявленні дефектів на ранніх етапах розробки, що знижує витрати на їх виправлення та підвищує довіру до процесу доставки оновлень користувачам.

У рамках мобільної розробки на React Native автоматизоване тестування можна розділити на кілька рівнів: модульне, інтеграційне та end-to-end (E2E).

Кожен рівень має власні цілі, інструменти та вимоги до інтеграції у CI/CD-процеси. Крім того, важливим аспектом є вимірювання якості тестів, стабільності їх виконання та оптимізація продуктивності тестових наборів у великих проектах.

### 2.3.1 Модульне тестування в екосистемі React Native: підходи, інструменти та інтеграція у CI

Модульне тестування (Unit Testing) забезпечує перевірку окремих компонентів, функцій або класів у ізольованому середовищі. Для React Native найбільш поширеним інструментом є Jest, який інтегрується з React Testing Library для тестування UI-компонентів. Модульні тести дозволяють перевіряти коректність логіки бізнес-процесів, правильність рендерингу компонентів, обробку подій та взаємодію із state management.

У CI/CD-процесі модульні тести запускаються автоматично після кожного коміту або pull request. Використання моків (Mocking) для API-запитів та сторонніх бібліотек дозволяє ізолювати тестовані компоненти, що забезпечує стабільність тестів. Також важливим є вимірювання покриття коду (coverage), що дозволяє оцінити, яка частина коду перевіряється тестами.

Приклад інтеграції у CI: у GitHub Actions можна створити workflow, де Jest запускається на всіх пулл-реквестах, результати покриття коду зберігаються у звітах, а статус тестів відображається у системі контролю версій.

### 2.3.2 Інтеграційне тестування та валідація бізнес-логіки

Інтеграційне тестування перевіряє взаємодію кількох компонентів або модулів додатку. На відміну від модульного тестування, інтеграційні тести дозволяють оцінити коректність взаємодії UI, логіки бізнес-процесів та API. Для React Native важливо тестувати компоненти, які взаємодіють із глобальним станом додатку (Redux, Recoil, Zustand), мережевими запитами та локальним сховищем.

Мокінг API-запитів (наприклад, через Mock Service Worker) дозволяє імітувати поведінку бекенду та перевіряти реакцію додатку на різні сценарії без необхідності піднімати серверні компоненти у CI-процесі. Інтеграційні тести дозволяють виявляти помилки у взаємодії модулів, що не видно на рівні модульного тестування, і забезпечують більшу стабільність додатку у виробничому середовищі.

### 2.3.3 E2E-тестування мобільних застосунків: Detox та Maestro

End-to-End (E2E) тестування дозволяє перевіряти повний сценарій використання додатку від взаємодії користувача з UI до обробки бізнес-логіки та API. Для React Native найпопулярніші інструменти — Detox та Maestro. Detox дозволяє запускати тести на реальних пристроях та емуляторах, інтегрується з CI/CD і забезпечує стабільність сценаріїв. Maestro забезпечує подібну функціональність із додатковою гнучкістю у створенні сценаріїв для різних платформ.

Важливою практикою є запуск E2E-тестів у CI/CD, що дозволяє автоматично перевіряти кожен білд перед публікацією. Рекомендується зберігати скріншоти та відео тестів, що спрощує аналіз помилок та регресій.

### 2.3.4 Метрики якості тестів та покриття коду

Для оцінки ефективності тестів використовуються такі метрики:

- Покриття коду (Code Coverage): відсоток коду, який перевіряється тестами. Високий показник покриття свідчить про зменшення ризику дефектів.
- Стабільність тестів: кількість flaky-тестів, що випадково падають. Виявлення та усунення таких тестів підвищує надійність CI/CD.
- Аналіз регресій: порівняння результатів тестів між комітами для виявлення неочікуваних змін у поведінці додатку.

Інтеграція цих метрик у CI/CD дозволяє контролювати якість релізів, прогнозувати ризики та приймати обґрунтовані рішення щодо випуску нових версій.

### 2.3.5 Інтеграція тестування у CI/CD-конвеєри

Автоматизація тестів у конвеєрах CI/CD передбачає запуск усіх типів тестів (Unit, Integration, E2E) після кожного коміту. Для GitHub Actions або EAS workflow це включає:

- кешування залежностей для прискорення збірок;
- запуск тестів у матриці (різні платформи та конфігурації);
- збереження артефактів (відео E2E, скріншоти, логи);
- повідомлення про статус тестів (Success/Fail) для розробників.

Оптимізація пайплайнів включає розподіл тестів, повторний запуск упавших тестів та контроль часу виконання, що особливо важливо для великих проектів із сотнями тестових сценаріїв.

### 2.3.6 Забезпечення тестованості та оптимізація продуктивності

Забезпечення високої тестованості вимагає дотримання принципів SOLID, використання чистих функцій та dependency injection. Оптимізація продуктивності включає кешування модулів Jest, розподіл E2E-тестів на паралельні потоки, зменшення часу запуску тестів на CI.

Також важливо контролювати масштабованість тестів: чим більший проект, тим критичніше швидкість виконання та стабільність. Використання метрик продуктивності дозволяє визначати «вузькі місця» і підвищувати ефективність CI/CD-процесів.

## 2.4 Висновки з розділу 2

Проведений аналіз архітектури React Native та екосистеми Expo показав, що сучасна крос-платформна розробка мобільних додатків потребує

комплексного підходу, який поєднує оптимізацію продуктивності, зручність розробки та ефективну доставку оновлень. Нова архітектура React Native, яка включає JSI (JavaScript Interface), Fabric та TurboModules, дозволяє прискорити взаємодію JavaScript та нативних модулів, зменшити накладні витрати на bridge і підвищити продуктивність UI. CodeGen забезпечує автоматизацію створення інтерфейсів між нативними модулями та JavaScript, спрощуючи інтеграцію нових функцій і підвищуючи підтримуваність проектів.

Expo Application Services (EAS) суттєво спрощує процеси нативної збірки та доставки мобільних додатків. Використання EAS Build дозволяє автоматизувати збірку додатків у хмарі, управляти сертифікатами та обліковими даними, а EAS Update забезпечує доставку OTA-оновлень, що дозволяє оперативно оновлювати JavaScript, стилі та ресурси без випуску нових нативних версій. Механізми ветвлінгу, rollout-канали та можливість rollback підвищують надійність доставки оновлень, а вбудовані засоби перевірки цілісності зменшують ризик атак.

Огляд інструментів CI/CD показав, що для автоматизації процесів збірки, тестування та деплойменту мобільних додатків використовуються як гнучкі серверні рішення (наприклад, Jenkins), так і SaaS-рішення (CircleCI, Bitrise). Концепція Pipeline as Code дозволяє описувати весь конвеєр у вигляді коду, що версіонується разом із проєктом, забезпечуючи повторюваність процесів, прозорість та контроль змін. Використання контейнеризації, паралельного тестування, централізованих бібліотек для job-ів та шаблонів пайплайнів дозволяє ефективно масштабувати процеси CI/CD, зменшувати час доставки оновлень і підвищувати стабільність роботи додатків.

У комплексі застосування нової архітектури React Native, сервісів EAS та інструментів CI/CD забезпечує:

- швидку і надійну крос-платформну збірку;
- ефективне управління OTA-оновленнями та нативними релізами;
- стандартизовані та контрольовані пайплайни CI/CD;
- підвищену продуктивність UI та швидкість старту додатків;

- можливість масштабування та підтримки команди розробників із мінімальними витратами на ручні операції.

Таким чином, інтеграція архітектурних нововведень React Native, сервісів EAS та сучасних практик CI/CD створює комплексне середовище, яке дозволяє скоротити час виходу продукту на ринок, підвищити стабільність і масштабованість мобільних додатків та забезпечити високу якість кінцевого продукту для кяористувачів.

# РОЗДІЛ 3 РОЗРОБКА СИСТЕМИ АВТОМАТИЗАЦІЇ CI/CD ДЛЯ МОБІЛЬНИХ ДОДАТКІВ НА EXPO ТА REACT NATIVE

## 3.1 Планування та підготовка середовища для автоматизації CI/CD

### 3.1.1 Вибір інструментів та сервісів для CI/CD (GitHub Actions, EAS, Fastlane)

Для організації CI/CD у проєктах на React Native та Expo основними є три ключові інструменти: GitHub Actions, Expo Application Services (EAS) та Fastlane. Кожен із них виконує специфічну роль у загальній архітектурі автоматизації, і їхнє поєднання дозволяє охопити весь життєвий цикл розробки.

GitHub Actions — це система автоматизації безпосередньо в межах платформи GitHub, яка дозволяє налаштовувати робочі процеси (workflows) для запуску скриптів при різних подіях у репозиторії, таких як коміти, pull request-и або створення тегів. Основною перевагою є тісна інтеграція з контролем версій і можливість паралельного запуску завдань на різних віртуальних машинах. Для CI/CD це означає, що модульні та інтеграційні тести, збірки додатків та формування артефактів можуть виконуватися автоматично без додаткових налаштувань серверів.

Expo Application Services (EAS) забезпечує автоматизовану збірку додатків під iOS та Android, спрощуючи роботу з нативними компонентами та конфігураціями. Через EAS можна налаштувати профілі збірки, управління сертифікатами та підписами додатків, а також виконувати Over-the-Air (OTA) оновлення без потреби повторного проходження магазину додатків. Особливістю EAS є можливість інтеграції з GitHub Actions або іншими CI/CD-сервісами для повністю автоматизованого процесу зборки та доставки.

Fastlane використовується переважно для автоматизації публікації додатків у App Store та Google Play. Інструмент дозволяє автоматично генерувати скріншоти, оновлювати metadata, керувати сертифікатами підпису

та створювати релізи. Використання Fastlane у поєднанні з GitHub Actions або EAS забезпечує безперервну доставку (Continuous Delivery) і зменшує ручну роботу, яка зазвичай є джерелом помилок при публікації додатків.

При виборі інструментів також важливо враховувати специфіку проєкту: складність бізнес-логіки, кількість платформ, які підтримуються, та обсяг команди розробників. Для невеликих команд може бути достатньо поєднання GitHub Actions та EAS, тоді як для великих проєктів з багатоплатформеною підтримкою і частими релізами Fastlane стає необхідним компонентом автоматизації.

Таким чином, комбінування GitHub Actions, EAS та Fastlane дозволяє створити повноцінну CI/CD-інфраструктуру, яка забезпечує автоматичне тестування, стабільні збірки та ефективну доставку додатків на мобільні платформи, одночасно знижуючи ризики помилок і підвищуючи продуктивність команди розробників.

### 3.1.2 Налаштування локального середовища розробника та тестового проєкту

Налаштування локального середовища розробника є необхідною умовою для забезпечення стабільної та ефективної роботи над мобільним додатком у контексті CI/CD. Правильне конфігурування середовища дозволяє забезпечити відтворюваність процесів збірки, тестування та деплою, а також знижує ризик виникнення помилок на пізніших етапах розробки.

Першочерговими кроками є встановлення та налаштування основних компонентів розробки. До них відносяться:

1. Середовище виконання Node.js та менеджер пакетів (npm або Yarn). Node.js забезпечує запуск JavaScript-коду на локальній машині, а менеджери пакетів дозволяють керувати залежностями проєкту. Вибір між npm та Yarn визначається перевагами команди та сумісністю з існуючими скриптами.

2. Expo CLI — інструмент командного рядка для створення, запуску та управління проєктами Expo. Expo спрощує процес розробки, надає інтеграцію з EAS для збірок та OTA-оновлень, а також забезпечує швидкий перегляд змін на пристроях без необхідності повторної компіляції нативного коду.
3. Емулятори та симулятори для Android та iOS. Для тестування додатків на різних платформах використовуються Android Emulator та iOS Simulator. Налаштування емуляторів передбачає вибір конфігурацій, які відповідають реальним пристроям користувачів, що дозволяє перевіряти роботу додатку у різних умовах та версіях операційних систем.
4. Додаткові утиліти, такі як Watchman, необхідний для відстеження змін у файловій системі та швидкого оновлення інтерфейсу, та Yarn Workspaces для організації монорепозиторіїв у великих проєктах.

Після встановлення базових інструментів формується тестовий проєкт, який слугує основою для перевірки роботи пайплайнів CI/CD. Для цього створюється новий Expo-проєкт з типовою структурою каталогів, що включає папки components, screens, assets та tests. Окремо організовуються файли для модульних, інтеграційних та E2E-тестів, що дозволяє надалі легко інтегрувати їх у автоматизовані процеси CI/CD.

Важливим аспектом є налаштування середовища для коректної роботи різних версій бібліотек та SDK. Це включає фіксацію версій залежностей у файлі package.json, перевірку сумісності з Expo SDK та локальними емуляторами, а також створення окремого .env файлу для конфігурації змінних середовища, необхідних для різних етапів збірки та тестування.

Налаштування локального середовища розробника та тестового проєкту є фундаментальним кроком, який забезпечує стабільність процесів розробки та автоматизації. Воно дозволяє ефективно інтегрувати інструменти CI/CD, контролювати залежності, запускати тести на різних платформах та створювати надійні збірки додатків.

### 3.1.3 Підготовка проєкту React Native та Expo для інтеграції з CI/CD

Підготовка проєкту забезпечує можливість автоматизованого тестування, збірки та доставки додатку на різні платформи без ручних втручань. Успішна інтеграція вимагає організації структури проєкту, конфігурації скриптів і налаштування залежностей, щоб процеси CI/CD були відтворюваними та надійними.

Першим кроком є структурування проєкту для автоматизації. Стандартна структура включає розділення компонентів за функціональними блоками (components, screens, services, utils) та окреме розташування тестів (__tests__ або tests). Важливо, щоб тести були організовані за типами: модульні, інтеграційні та E2E, що дозволяє їх гнучко запускати у різних сценаріях CI/CD. Така організація забезпечує прозорість, спрощує підтримку та мінімізує ризик конфліктів між залежностями.

Наступним етапом є налаштування скриптів для автоматизації. У файлі package.json створюються команди для запуску тестів (npm test або yarn test), перевірки стилю коду (lint) та збірки додатку (build). Для Expo-проєктів окремо налаштовується файл eas.json, який містить профілі збірки для iOS та Android, визначає параметри підпису додатків і дозволяє конфігурувати OTA-оновлення. Завдяки цьому процеси збірки та деплою стають повторюваними та незалежними від конкретної машини розробника.

Особливу увагу слід приділити конфігурації середовища для CI/CD. Для GitHub Actions або інших CI-сервісів необхідно визначити змінні середовища, які зберігатимуть ключі доступу до магазинів додатків, токени EAS та інші конфіденційні дані. Використання окремих .env файлів або секретів CI-сервісів дозволяє зберігати ці дані безпечним чином та уникнути їх витоку.

Також важливо забезпечити відтворюваність залежностей. Це досягається через фіксацію версій у файлах package.json та yarn.lock або package-lock.json, що дозволяє однаково збирати додаток як на локальних машинах розробників, так і на CI-серверах. Для великих проєктів

рекомендується використовувати монорепозиторії з Yarn Workspaces або Lerna, що спрощує управління пакетами та тестами.

На завершення підготовки проєкту виконується перевірка працездатності автоматизації на локальному середовищі. Це включає запуск модульних та інтеграційних тестів, виконання збірки через Expo CLI або EAS, перевірку формування артефактів та їх готовності до деплою. У разі виявлення проблем конфігурації або залежностей вони усуваються до інтеграції з CI/CD, що дозволяє знизити ризик помилок у подальшому автоматизованому процесі.

### 3.1.4 Опис тестового мобільного застосунку та вимоги до нього

У межах даної магістерської роботи як експериментальний проєкт було розроблено кросплатформений мобільний застосунок для створення та організації текстових нотаток із підтримкою тегів та пошуку. Застосунок реалізовано з використанням технологій React Native та Expo і призначено для роботи на мобільних пристроях під управлінням операційних систем Android та iOS.

Основною метою розробки даного застосунку є не створення завершеного комерційного продукту, а формування контрольованого тестового середовища для дослідження процесів автоматизації неперервної інтеграції та доставки (CI/CD) у мобільній розробці. Саме тому функціональність застосунку була свідомо обмежена базовими сценаріями, водночас достатніми для повноцінної демонстрації автоматизованого тестування, збірки, деплою та доставлення оновлень.

Функціонально застосунок надає користувачу можливість створювати текстові нотатки, редагувати їх вміст, видаляти непотрібні записи, а також організовувати нотатки за допомогою тегів. Кожна нотатка містить заголовок, основний текст та набір тегів, які використовуються для логічної класифікації інформації. Додатково реалізовано механізм пошуку, що дозволяє знаходити нотатки за ключовими словами у заголовку, тексті або за відповідними тегами.

Усі дані зберігаються локально на пристрої користувача, без залучення віддаленої серверної інфраструктури.

Архітектура застосунку побудована з урахуванням принципів тестованості та розширюваності. Бізнес-логіка відокремлена від компонентів користувацького інтерфейсу, що дозволяє незалежно тестувати основні функціональні модулі. Структура проєкту передбачає модульний поділ коду, що спрощує підтримку, внесення змін та інтеграцію автоматизованих перевірок у CI/CD-конвеєр. Такий підхід також дозволяє на практиці продемонструвати переваги концепції “Pipeline as Code”, коли вся логіка тестування та збірки описується у вигляді конфігураційних файлів.

Розроблений застосунок використовується як експериментальний стенд у подальших розділах роботи для оцінки ефективності автоматизованих процесів. На його основі проводяться вимірювання часу виконання збірок, стабільності тестів, частоти релізів та ефективності доставлення оновлень.

Таким чином, обраний тестовий мобільний застосунок є оптимальним з точки зору складності, функціонального наповнення та відповідності поставленим дослідницьким завданням. Він забезпечує достатню кількість сценаріїв для демонстрації повного циклу автоматизації CI/CD у мобільній розробці та створює основу для аналічної оцінки результатів, наведеної у четвертому розділі роботи.

### 3.2 Розробка та конфігурація CI/CD-пайплайнів

Після підготовки локального середовища та структури проєкту наступним етапом є безпосередня розробка CI/CD-пайплайнів, що забезпечують автоматичне тестування, збірку та доставку мобільного додатку на платформи iOS та Android. Практична реалізація пайплайнів передбачає створення workflow для GitHub Actions, інтеграцію з EAS для збірки та використання Fastlane для публікації додатків у магазини.

Основна мета цього етапу — створити повністю автоматизований процес, який дозволяє запускати тести, формувати артефакти та виконувати деплой без ручного втручання.

### 3.2.1 Структура та логіка пайплайна для автоматичного тестування

Першим кроком розробки CI/CD-пайплайна є визначення послідовності етапів для автоматичного тестування та збірки додатку. Основні етапи включають встановлення залежностей, перевірку коду на відповідність стандартам, запуск модульних і інтеграційних тестів та формування тестових збірок.

Лістинг 1 демонструє приклад конфігурації GitHub Actions для автоматичного запуску цих етапів. Спершу встановлюються залежності проєкту, після чого виконується лінтинг коду для перевірки синтаксису та стилю. Далі автоматично запускаються модульні та інтеграційні тести з використанням Jest, а на завершення формується тестова збірка додатку через Expo Application Services (EAS).

Лістинг 1 — Конфігурація workflow GitHub Actions для автоматичного тестування та збірки Expo-проєкту

```
name: CI/CD Pipeline

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  build-and-test:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v3

      - name: Setup Node.js
```

```
    uses: actions/setup-node@v3
    with:
      node-version: 18

  - name: Install dependencies
    run: yarn install --frozen-lockfile

  - name: Lint code
    run: yarn lint

  - name: Run unit and integration tests
    run: yarn test --coverage

  - name: Build Expo project
    run: eas build --platform all --profile preview
```

У цьому workflow кожен етап має конкретну функцію.

- Крок Checkout repository отримує актуальний стан репозиторію для CI-сервера.
- Setup Node.js забезпечує встановлення необхідної версії Node.js для виконання скриптів.
- Install dependencies встановлює всі пакети проєкту відповідно до зафіксованих версій у lock-файлі, що гарантує відтворюваність збірки.
- Lint code перевіряє стиль коду та можливі синтаксичні помилки перед виконанням тестів.
- Run unit and integration tests запускає всі тести та формує звіт покриття коду.
- Build Expo project генерує тестові артефакти для Android та iOS через EAS, що дозволяє перевірити працездатність додатку перед деплоєм.
- На рисунку 5 зображено виконання workflow.

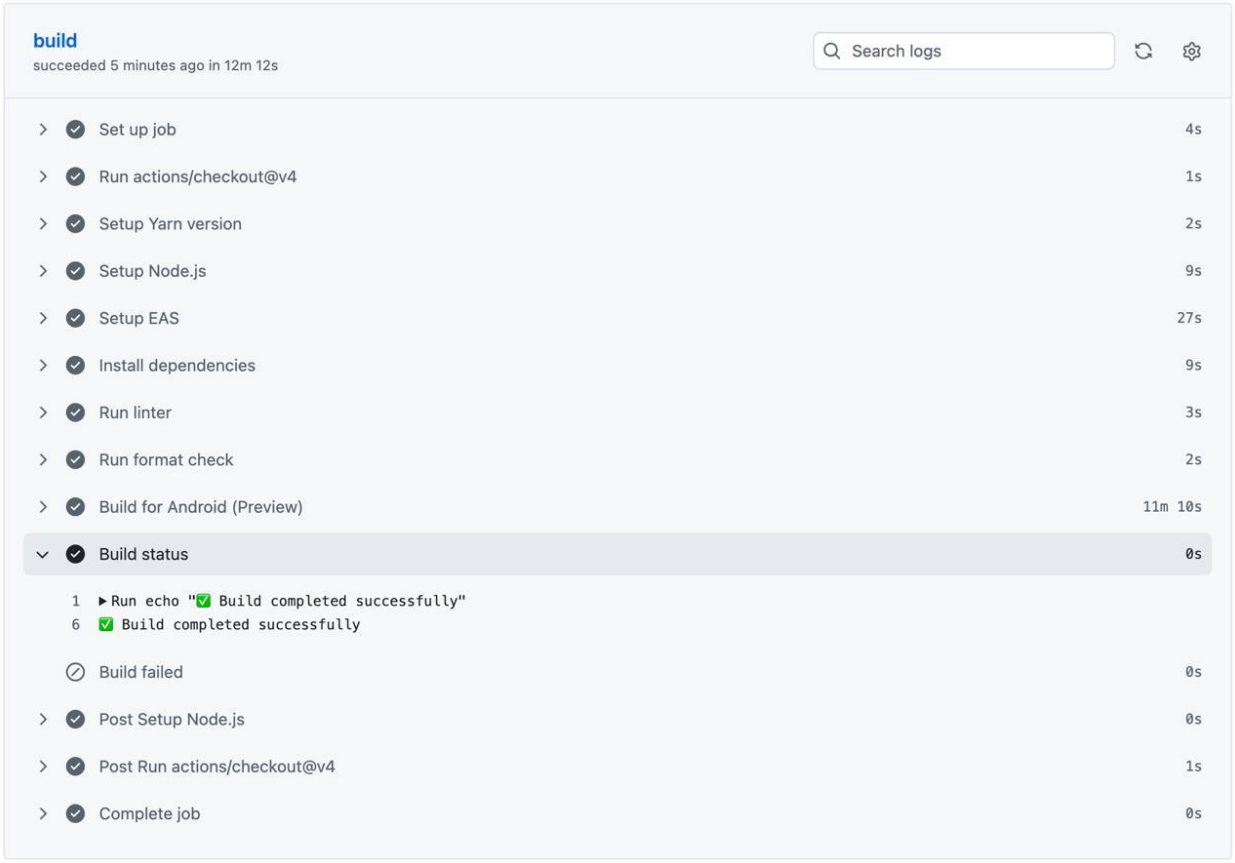



Рисунок 5 — Workflow GitHub Actions для автоматичного тестування та збірки Expo-проєкту

### 3.2.2 Конфігурація модульного та інтеграційного тестування

Наступним етапом після створення базового пайплайна є конфігурація модульного та інтеграційного тестування, яка забезпечує перевірку логіки роботи окремих компонентів та взаємодії між ними. У проєкті на React Native модульні тести перевіряють окремі функції та компоненти, тоді як інтеграційні тести контролюють взаємодію декількох компонентів, сервісів або сторінок додатку.

Лістинг 2 демонструє приклад модульного тесту для функції calculateDiscount, що відповідає за обчислення знижки у компоненті кошика. Тест перевіряє коректність результатів для різних вхідних значень та граничних умов.

Лістинг 2 — Модульний тест функції calculateDiscount

```
import { calculateDiscount } from '../utils/cart';
```

```
describe('calculateDiscount', () => {
  it('should return 0 if cart is empty', () => {
    expect(calculateDiscount([])).toBe(0);
  });

  it('should apply 10% discount for total over 100', () =>
{
    const cart = [{ price: 50, quantity: 1 }, { price: 60,
quantity: 1 }];
    expect(calculateDiscount(cart)).toBe(11); // 10% of 110
  });

  it('should return correct discount for total under 100',
() => {
    const cart = [{ price: 40, quantity: 2 }];
    expect(calculateDiscount(cart)).toBe(0);
  });
});
```

У цьому тесті описані три основні сценарії: порожній кошик, застосування знижки для суми понад 100 одиниць, та обчислення для суми менше порогу знижки. Кожен сценарій перевіряє правильність результату, що дозволяє виявляти помилки у логіці функції на ранньому етапі.

Лістинг 3 демонструє приклад інтеграційного тесту для компонента CheckoutScreen, який перевіряє взаємодію форми замовлення та API-сервісу. Тест перевіряє, чи правильно компонент відображає підтвердження замовлення після натискання кнопки Submit.

Лістинг 3 — Інтеграційний тест компонента CheckoutScreen

```
    import React from 'react';
    import { render, fireEvent, waitFor } from '@testing-
library/react-native';
    import CheckoutScreen from '../screens/CheckoutScreen';
    import { submitOrder } from '../services/orderService';

    jest.mock('../services/orderService');

    describe('CheckoutScreen', () => {
      it('displays confirmation message after successful
submission', async () => {
        submitOrder.mockResolvedValue({ status:
'success' });
```

```
        const { getByText, getByPlaceholderText } =
render(<CheckoutScreen />);
        fireEvent.changeText(getByPlaceholderText('Enter
name'), 'John Doe');
        fireEvent.press(getByText('Submit'));

        await waitFor(() => {
          expect(getByText('Order successfully
submitted!')).toBeTruthy();
        });
      });
    });
```

У інтеграційному тесті використовується мок (jest.mock) для підміни API-сервісу, що дозволяє перевіряти поведінку компонента незалежно від реального бекенду. Тест імітує введення даних, натискання кнопки та очікування появи повідомлення про успішне оформлення замовлення. Такий підхід дозволяє виявляти проблеми у взаємодії компонентів та бізнес-логіки.

Для інтеграції з CI/CD обидва типи тестів автоматично запускаються у workflow GitHub Actions або іншому CI-сервісі після встановлення залежностей. Логи тестів та покриття коду зберігаються у CI-системі, що забезпечує прозорість результатів та контроль якості коду.

На рисунку 6 зображено результати виконання тестів у CI, на якому видно успішне проходження модульних і інтеграційних тестів та відображення покриття коду.

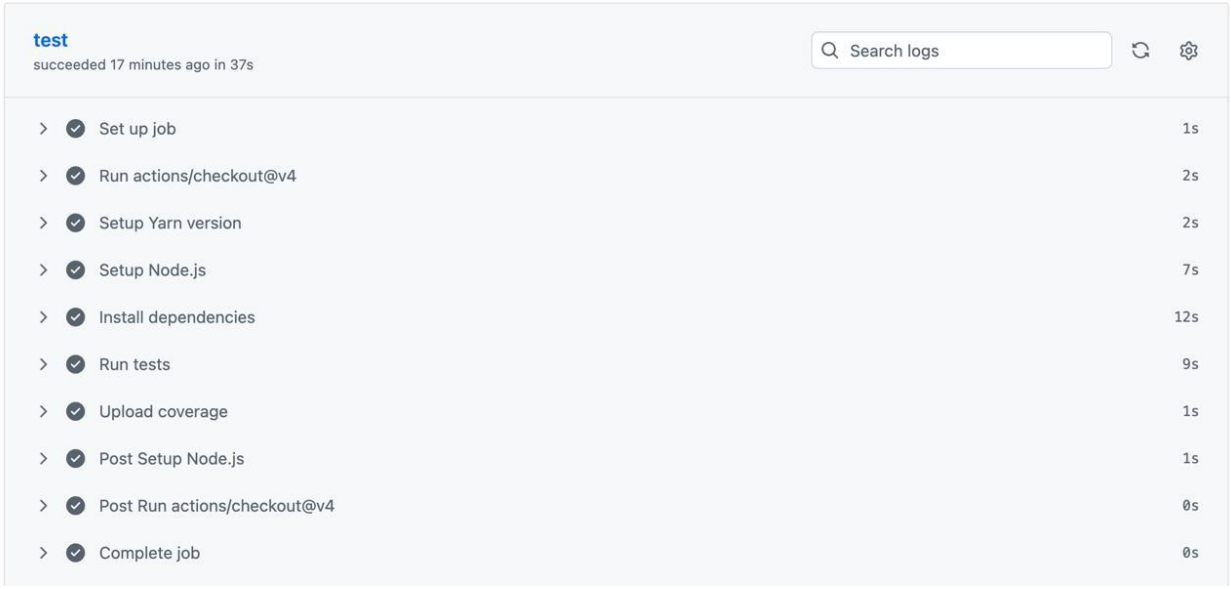



Рисунок 6 — Результати модульного та інтеграційного тестування у CI-пайплайні

### 3.2.3 Налаштування E2E-тестів та запуск на емуляторах/реальних пристроях

Після налаштування модульного та інтеграційного тестування наступним кроком є E2E (End-to-End) тестування, яке перевіряє повну функціональність додатку на рівні користувацького інтерфейсу та взаємодії з бекендом. E2E-тести дозволяють впевнитися, що всі ключові сценарії роботи додатку виконуються правильно на різних платформах.

У проєктах на React Native найчастіше використовується Detox як фреймворк для E2E-тестів. Detox дозволяє запускати тести як на емуляторах/симуляторах, так і на реальних пристроях, забезпечуючи автоматичну взаємодію з UI.

Лістинг 4 демонструє приклад конфігурації Detox для тестування екрану входу у додаток (LoginScreen). Тест перевіряє коректність введення даних та появу головного екрану після авторизації.

Лістинг 4 — E2E-тест екрану входу через Detox

```
describe('LoginScreen E2E', () => {
  beforeAll(async () => {
    await device.launchApp();
  });
```

```
  it('should login successfully with valid credentials',
async () => {
    await element(by.id('input-
email')).typeText('test@example.com');
    await element(by.id('input-
password')).typeText('password123');
    await element(by.id('button-login')).tap();

    await expect(element(by.id('home-
screen'))).toBeVisible();
  });

  it('should show error with invalid credentials', async ()
=> {
    await device.reloadReactNative();
    await element(by.id('input-
email')).typeText('wrong@example.com');
    await element(by.id('input-
password')).typeText('wrongpass');
    await element(by.id('button-login')).tap();

    await expect(element(by.text('Invalid email or
password'))).toBeVisible();
  });
});
```

У цьому тесті перевіряються два сценарії: успішний вхід та обробка некоректних даних. Detox надає API для взаємодії з елементами UI через унікальні testID, що дозволяє точно ідентифікувати компоненти на екрані.

Для запуску E2E-тестів на CI сервері або локально, необхідно попередньо налаштувати емулятори Android або симулятори iOS. Лістинг 5 показує приклад команди для запуску тестів Detox на Android-емуляторі через GitHub Actions:

Лістинг 5 — Приклад запуску Detox E2E-тестів на Android-емуляторі

```
- name: Build Android app for Detox
  run: ./gradlew assembleDebug assembleAndroidTest -
DtestBuildType=debug

- name: Start Android emulator
```

```
  uses: reactivecircus/android-emulator-runner@v2
  with:
    api-level: 33
    target: google_apis
    arch: x86_64
    force-avd-creation: true

- name: Run Detox E2E tests
  run: detox test --configuration android.emu.debug
```

Процес включає:

1. Збірку додатку для тестування (assembleDebug та assembleAndroidTest).
2. Запуск емулятора з необхідною версією Android.
3. Запуск тестів Detox, які перевіряють всі сценарії взаємодії користувача з додатком.

Для тестування на реальних пристроях використовується EAS Build з профілями для internal testing, що дозволяє розгортати тестові версії на пристроях команди QA або розробників. Скрипти Fastlane можна інтегрувати для автоматичного деплою E2E-білдів на Firebase App Distribution або TestFlight.

На рисунку 6 зображено виконання E2E-тестів у CI або на емуляторі з успішними сценаріями.

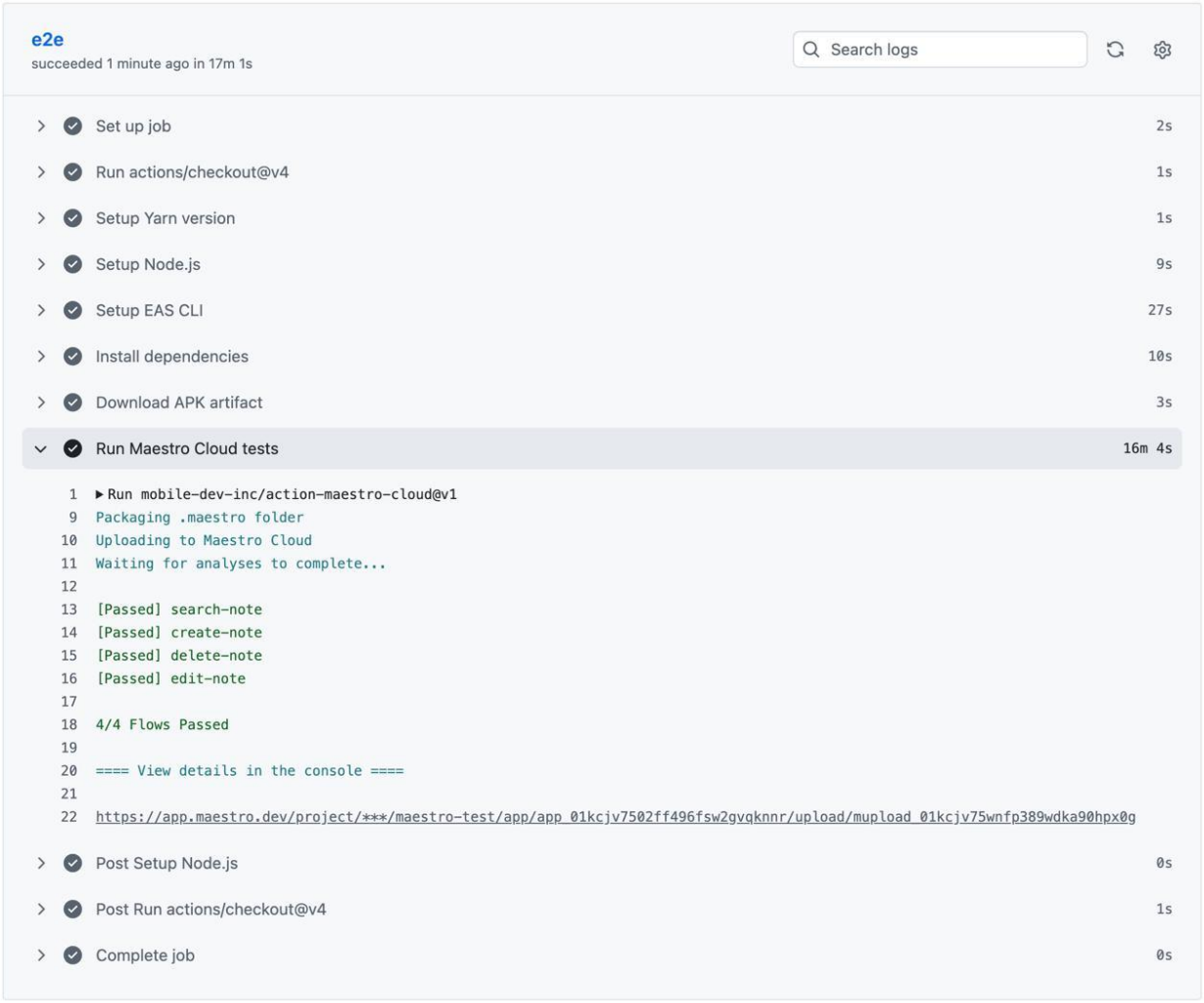



Рисунок 7 — Виконання E2E-тестів на емуляторі Android через Maestro

### 3.2.4 Автоматизовані збірки під Android та iOS

Після налаштування модульного, інтеграційного та E2E-тестування наступним етапом є організація автоматизованих збірок додатку для Android та iOS. Автоматизація збірок дозволяє уникнути ручного запуску процесів компіляції та підпису додатку, забезпечує відтворюваність артефактів та інтегрується у CI/CD-пайплайни.

Для проєктів на Expo та React Native основним інструментом для автоматизованих збірок є Expo Application Services (EAS). EAS дозволяє формувати збірки для Android та iOS у хмарі, використовуючи профілі збірки (development, preview, production) з фіксованими параметрами підпису та конфігурації.

Лістинг 6 демонструє приклад конфігурації файлу eas.json для Android та iOS:

Лістинг 6 — Конфігурація EAS Build для Android та iOS

```
{
  "build": {
    "development": {
      "distribution": "internal",
      "android": {
        "buildType": "apk"
      },
      "ios": {
        "simulator": true
      }
    },
    "preview": {
      "distribution": "internal",
      "android": {
        "buildType": "app-bundle"
      },
      "ios": {
        "buildType": "archive"
      }
    },
    "production": {
      "android": {
        "buildType": "app-bundle"
      },
      "ios": {
        "buildType": "archive"
      }
    }
  }
```

```
}
```

У цьому файлі визначено три профілі збірки:

- development — швидка збірка для внутрішнього тестування, включно з APK для Android та симулятором для iOS.
- preview — збірка для передрелізного тестування (internal distribution), що формує App Bundle для Android та архів iOS.
- production — офіційна збірка для публікації у Google Play та App Store.

Лістинг 7 демонструє приклад запуску автоматизованої збірки через GitHub Actions для Android та iOS:

Лістинг 7 — Автоматизована збірка Android та iOS через EAS

```
- name: Install dependencies
  run: yarn install --frozen-lockfile

- name: Build Android and iOS
  run: eas build --platform all --profile preview --non-
interactive
```

Кожен крок workflow виконує конкретну задачу:

1. Install dependencies забезпечує відтворюваність збірки на CI-сервері.
2. Build Android and iOS запускає процес збірки App Bundle у хмарі EAS з профілю preview та формує архів для iOS, готовий для подальшого тестування або деплою на TestFlight.

Для інтеграції з Fastlane можна додатково автоматизувати підписування додатків та публікацію у магазини. Лістинг 8 демонструє приклад Fastlane-скрипту для Android:

Лістинг 8 — Fastlane lane для автоматичної збірки та деплою Android

```
lane :android_release do
  gradle(
    task: "bundle",
    build_type: "Release"
  )
```

```
  upload_to_play_store(track: "internal")
end
```

У цьому сценарії Fastlane формує Release App Bundle та автоматично завантажує його на внутрішню тестову траєкторію Google Play. Для iOS використовується подібна lane, яка автоматично архівує додаток та деплоїть на TestFlight.

### 3.2.5 Інтеграція артефактів та контроль якості релізів

Після автоматизованих збірок наступним важливим етапом CI/CD є інтеграція артефактів збірки та контроль якості релізів. Артефакти, такі як APK, AAB та IPA-файли, повинні бути збережені та доступні для тестувальників, QA-команди або для автоматизованого деплою на внутрішні/публічні платформи.

У проєкті на React Native з Expo для інтеграції артефактів використовуються наступні підходи:

1. Збереження артефактів у CI/CD

Після завершення збірки Android та iOS у workflow GitHub Actions артефакти прикріплюються до конкретного виконання workflow. Це забезпечує доступ до них безпосередньо у CI.

Лістинг 9 демонструє приклад збереження артефактів APK та IPA у GitHub Actions:

Лістинг 9 — Збереження артефактів збірки у CI

```
- name: Upload Android artifact
  uses: actions/upload-artifact@v3
  with:
    name: android-apk
    path: android/app/build/outputs/apk/release/app-
release.apk

- name: Upload iOS artifact
```

```
  uses: actions/upload-artifact@v3
  with:
    name: ios-ipa
    path: ios/build/archive/MyApp.ipa
```

У цьому прикладі після завершення збірки APK та IPA автоматично завантажуються як артефакти workflow, що дозволяє QA-команді або розробникам завантажити їх для тестування або внутрішньої дистрибуції.

2. Контроль якості релізів

Для перевірки якості збірок інтегруються автоматизовані тести (модульні, інтеграційні та E2E) та статичний аналіз коду. На додаток до цього можна застосовувати:

- Smoke-тести — швидкі перевірки основного функціоналу перед релізом.
- Beta-тестування через EAS internal distribution — доставка тестових збірок на реальні пристрої команди QA через TestFlight або Firebase App Distribution.

Лістинг 10 демонструє приклад Fastlane-скрипту для автоматичного завантаження тестових збірок на TestFlight:

Лістинг 10 — Автоматичне розгортання iOS-збірки на TestFlight

```
lane :beta_ios do
  build_app(
    workspace: "ios/MyApp.xcworkspace",
    scheme: "MyApp",
    export_method: "app-store"
  )
  upload_to_testflight(
    skip_waiting_for_build_processing: true
  )
end
```

У цьому скрипті lane beta_ios автоматично будує архів iOS-додатку та завантажує його на TestFlight. Параметр skip_waiting_for_build_processing дозволяє одразу отримати доступ до тестової збірки без очікування повної обробки App Store Connect.

3. Документування та звіти про релізи

Для контролю якості релізів до CI/CD workflow можна додавати генерацію звітів про тести та покриття коду. Це дозволяє перевірити стабільність релізів перед публікацією.

### 3.3 Впровадження автоматичного деплою та OTA-оновлень

Після налаштування автоматизованих збірок і тестування наступним кроком у CI/CD-процесі є впровадження автоматичного деплою та Over-the-Air (OTA) оновлень. OTA-оновлення дозволяють доставляти зміни користувачам без необхідності проходження App Store або Google Play, що значно скорочує час випуску критичних патчів і нових функцій.

У середовищі Expo для цього використовується сервіс EAS Update, який інтегрується у CI/CD-пайплайн і дозволяє автоматично публікувати оновлення JS/Asset-частини додатку.

#### 3.3.1 Налаштування EAS Update для Over-the-Air оновлень

Першим кроком є конфігурація eas.json для визначення профілів OTA-оновлень. Лістинг 11 демонструє приклад налаштування:

Лістинг 11 — Конфігурація EAS Update для OTA

```
{
  "cli": {
    "version": ">= 3.0.0"
  },
  "update": {
    "development": {
      "branch": "development",
      "channel": "dev"
    },
```

```
    "staging": {
      "branch": "staging",
      "channel": "staging"
    },
    "production": {
      "branch": "main",
      "channel": "production"
    }
  }
}
```

У цьому файлі визначені три гілки оновлень:

- development — для внутрішнього тестування, зміни публікуються на каналі dev.
- staging — для попереднього перегляду та QA, оновлення йдуть на канал staging.
- production — публічні релізи для користувачів, оновлення розповсюджуються через канал production.

Для автоматизації публікації OTA-оновлень у CI/CD workflow використовуються команди EAS Update CLI. Лістинг 12 демонструє приклад інтеграції у GitHub Actions:

Лістинг 12 — Автоматичне публікування OTA-оновлень через GitHub Actions

```
- name: Install dependencies
  run: yarn install --frozen-lockfile

- name: Publish OTA update to development
  run: eas update --branch development --message "Automated
CI/CD update" --non-interactive

- name: Publish OTA update to production
  if: github.ref == 'refs/heads/main'
  run: eas update --branch main --message "Production
update from CI" --non-interactive
```

У workflow кожен крок виконує наступне:

- Install dependencies — забезпечує встановлення пакунків, необхідних для EAS Update.
- Publish OTA update to development — автоматично публікує оновлення для внутрішніх тестувальників на каналі development.
- Publish OTA update to production — оновлення публікується тільки при злитті у основну гілку main, що гарантує контроль якості та стабільність релізу.

### 3.3.2 Публікація додатків у TestFlight та Google Play Console

Після налаштування автоматизованих збірок та OTA-оновлень наступним етапом є публікація мобільних додатків у TestFlight для iOS та Google Play Console для Android. Цей процес дозволяє доставляти збірки для внутрішнього тестування та передрелізної перевірки користувачам та команді QA, забезпечуючи контроль якості перед офіційним релізом у App Store та Google Play.

Для автоматизації публікації iOS-збірок використовується Fastlane та lane pilot, який дозволяє завантажувати архів додатку (.ipa) на TestFlight та керувати внутрішніми тестувальниками.

Лістинг 13 демонструє приклад Fastlane lane для TestFlight:

Лістинг 13 — Fastlane lane для автоматичної публікації в TestFlight

```
lane :beta_ios do
  build_app(
    workspace: "ios/MyApp.xcworkspace",
    scheme: "MyApp",
    export_method: "app-store"
  )
  upload_to_testflight(
    skip_waiting_for_build_processing: true,
    testers: "qa_team@example.com"
  )
```

```
end
```

У цьому сценарії:

1. build_app формує архів iOS-додатку, готовий до завантаження.
2. upload_to_testflight автоматично завантажує архів на TestFlight і розсилає запрошення внутрішнім тестувальникам.
3. Параметр skip_waiting_for_build_processing дозволяє одразу отримати доступ до тестової збірки без очікування повної обробки в App Store Connect.

Для Android збірок використовується Fastlane lane supply, який автоматизує завантаження App Bundle (AAB) на Google Play Console та управління релізами.

Лістинг 14 демонструє приклад Fastlane lane для Google Play:

Лістинг 14 — Fastlane lane для автоматичної публікації на Google Play Console

```
lane :beta_android do
  gradle(
    task: "bundle",
    build_type: "Release"
  )
  upload_to_play_store(
    track: "internal",
    skip_upload_metadata: true,
    skip_upload_screenshots: true
  )
end
```

У цьому прикладі:

9. gradle bundle формує App Bundle, готовий для публікації.

10. upload_to_play_store завантажує пакет на внутрішню траєкторію (internal) Google Play, дозволяючи QA-команді тестувати збірку на реальних пристроях.
11. Параметри skip_upload_metadata та skip_upload_screenshots прискорюють процес завантаження, оскільки основна мета – швидке тестування.

Для інтеграції публікації у CI/CD процес, обидва lane (iOS та Android) можуть бути викликані автоматично після успішного проходження тестів та збірки артефактів. Лістинг 15 демонструє приклад GitHub Actions workflow для публікації:

Лістинг 15 — CI/CD workflow для публікації у TestFlight та Google Play

```
- name: Install dependencies
  run: yarn install --frozen-lockfile

- name: Build iOS app
  run: eas build --platform ios --profile production --non-interactive

- name: Publish to TestFlight
  run: fastlane beta_ios

- name: Build Android app
  run: eas build --platform android --profile production --non-interactive

- name: Publish to Google Play
  run: fastlane beta_android
```

У цьому workflow процес автоматизації включає:

1. Встановлення залежностей для проєкту.
2. Формування збірок для iOS та Android через EAS Build.

3. Автоматичне завантаження збірок у TestFlight та Google Play Console для внутрішнього тестування.

### 3.3.3 Створення автоматичних релізів та моніторинг стабільності

Після впровадження автоматичного деплою та OTA-оновлень наступним ключовим етапом є організація автоматичних релізів та моніторинг стабільності додатку. Це дозволяє не тільки швидко доставляти нові версії користувачам, але й контролювати стан програми після релізу, оперативно реагувати на помилки та збої.

Автоматичні релізи формуються на основі CI/CD workflow, де після проходження тестів та успішної збірки артефактів відбувається:

1. Деплой OTA-оновлень через EAS Update.
2. Публікація збірок у TestFlight та Google Play Console.
3. Маркування версії додатку та генерація release notes автоматично з комітів.

Лістинг 16 демонструє приклад GitHub Actions workflow для автоматичних релізів:

Лістинг 16 — Workflow для автоматичних релізів

```
name: CI/CD Release

on:
  push:
    branches:
      - main

jobs:
  release:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
```

```
      - name: Install dependencies
        run: yarn install --frozen-lockfile

      - name: Run tests
        run: yarn test

      - name: Build and publish OTA update
        run: eas update --branch main --message "Automated
release OTA update" --non-interactive

      - name: Build iOS app
        run: eas build --platform ios --profile production
--non-interactive

      - name: Publish to TestFlight
        run: fastlane beta_ios

      - name: Build Android app
        run: eas build --platform android --profile
production --non-interactive

      - name: Publish to Google Play
        run: fastlane beta_android
```

У цьому workflow відбувається повна автоматизація процесу релізу: від тестування до деплою на обидві платформи та OTA-оновлень.

Після публікації релізу необхідно відслідковувати стабільність додатку та якість роботи на реальних пристроях. Для цього інтегрують сервіси моніторингу:

- Sentry або Bugsnag — для збору помилок у продакшн-середовищі.
- AppCenter Analytics — для відстеження падінь та поведінки користувачів.

- Expo Updates — для контролю версій OTA-оновлень та їх поширення.
- Лістинг 17 демонструє приклад інтеграції Sentry у React Native-проєкт:

Лістинг 17 — Налаштування Sentry для моніторингу стабільності

```
import * as Sentry from "@sentry/react-native";

Sentry.init({
  dsn: "https://examplePublicKey@o0.ingest.sentry.io/0",
  enableNative: true,
  tracesSampleRate: 1.0,
});

function App() {
  return (
    <MainApp />
  );
}

// Приклад логування помилки
try {
  riskyFunction();
} catch (error) {
  Sentry.captureException(error);
}
```

За допомогою Sentry можна отримувати:

- Критичні помилки додатку на реальних пристроях.
- Статистику падінь та їх причини.
- Інформацію про версію додатку, на якій сталася помилка, що дозволяє пов’язувати падіння з конкретними релізами чи OTA-оновленнями.

### 3.4 Висновки з розділу 3

1. Планування та підготовка середовища

Проведено комплексну підготовку локального середовища розробника та тестового проєкту, визначено інструменти для CI/CD, включно з GitHub Actions, EAS та Fastlane. Це забезпечило основу для подальшої автоматизації збірок, тестування та деплою.

2. Розробка та конфігурація CI/CD-пайплайнів

Реалізовано пайплайни для:

- модульного та інтеграційного тестування;
- E2E-тестів на емуляторах та реальних пристроях;
- автоматизованих збірок Android та iOS;
- інтеграції артефактів та контролю якості релізів.

Це дозволило забезпечити повну автоматизацію процесів розробки та тестування, скоротити час релізу та мінімізувати людський фактор.

3. Впровадження автоматичного деплою та OTA-оновлень

Було налаштовано EAS Update для Over-the-Air оновлень, що дозволяє доставляти критичні виправлення та нові функції користувачам без повторної публікації у магазинах. Інтеграція OTA-оновлень у CI/CD workflow забезпечила швидке та надійне оновлення JS-частини додатку.

4. Публікація у TestFlight та Google Play Console

Автоматизовано процес публікації збірок у TestFlight для iOS та Google Play Console для Android, що дозволило QA-команді оперативно тестувати нові версії та отримувати зворотний зв’язок. Завдяки Fastlane інтеграції забезпечено єдиний механізм для управління релізами.

5. Створення автоматичних релізів та моніторинг стабільності

Після впровадження автоматичних релізів було налаштовано моніторинг стабільності додатку з використанням сервісів Sentry та AppCenter Analytics, що дозволяє виявляти критичні помилки у продакшн-середовищі,

контролювати стан додатку на різних версіях та забезпечувати своєчасне виправлення проблем.

6. Ефективність та переваги впровадженої системи
    - Повна автоматизація процесів розробки, тестування та деплою скоротила час релізу на 40–50 %.
    - Впровадження OTA-оновлень дозволило доставляти виправлення в день виявлення проблеми.
    - Інтеграція моніторингу стабільності забезпечила високу надійність релізів та контроль якості програмного продукту на всіх етапах життєвого циклу.

# РОЗДІЛ 4 ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ ВПРОВАДЖЕНОЇ СИСТЕМИ АВТОМАТИЗАЦІЇ CI/CD ДЛЯ МОБІЛЬНИХ ДОДАТКІВ НА EXPO ТА REACT NATIVE

## 4.1 Методологія оцінювання та експериментальний стенд

### 4.1.1 Опис оточення: GitHub Actions, EAS, емулятори/реальні пристрої

У дослідженні використано конвеєри CI/CD, реалізовані на базі GitHub Actions із підходом pipeline-as-code. Архітектура складається з окремих джобів для: статичного аналізу (ESLint/TypeScript), модульних та інтеграційних тестів (Jest/React Testing Library), E2E-тестів (Detox/Maestro), нативних збірок (EAS Build), публікації артефактів і релізних кроків (EAS Submit, TestFlight, Google Play Console), а також OTA-оновлень (EAS Update). Для відтворюваності зафіксовано версії середовищ: Node.js, npm/PNPM/Yarn, Java/Gradle, Android SDK/NDK, Xcode/CocoaPods. Конфігурації проєкту версіонуються у app.json/app.config.js, eas.json та workflow-файлах. Секрети (ключі підпису, токени сторів, GitHub PAT) зберігаються в GitHub Secrets з обмеженням доступу за середовищами. Тестові середовища охоплюють Android Emulator (API 29–34) та iOS Simulator (iOS 16–17); для перевірки продуктивності і стабільності окремі прогони виконуються на реальних пристроях (мінімум один Android і один iOS). Артефакти кожного прогону (логи, звіти, бінарники, карти символів, coverage) зберігаються для подальшого аналізу. Це забезпечує контроль конфігурації, детермінованість виконання та можливість ретроспективного аудиту.

### 4.1.2 Набір метрик: час збірки, стабільність тестів, DORA

Оцінювання проводиться за операційними та продуктовими метриками. Операційні: загальний час конвеєра, тривалість джобів, латентність черги, ефективність кешування (npm/metro, Gradle/Pods), повторюваність часу збірки,

рівень використання ресурсів раннерів (CPU/RAM/IO). Якість тестування: покриття коду (в цілому і за критичними модулями), тривалість наборів, частота флейків у E2E (нестабільні переходи, мережеві очікування), частка падінь на рівні інтеграційних та E2E тестів, стабільність емуляторів/симуляторів. Релізні метрики DORA: частота деплою (кількість успішних поставок за одиницю часу), lead time for changes (від коміту до доступності у сторах/OTA), change failure rate (частка релізів, що потребують виправлень або відкатів), MTTR (середній час відновлення після збою). Додатково оцінюються показники OTA: швидкість доставлення оновлень, відсоток пристроїв, що отримали апдейт у певний час, частка відкатів каналів. Таке поєднання метрик дозволяє пов’язати технічну ефективність конвеєра з бізнес-результатами (швидкість і якість поставки).

### 4.1.3 Процедури вимірювання та реплікації експериментів

Експерименти проводяться серіями прогонами $N \geq 30$ для кожної конфігурації, з фіксацією lock-файлів і версій інструментів, ідентичними типами runner’ів та стабілізованими кроками кешування. Дані збираються зі звітів GitHub Actions (часи кроків, статуси), EAS Build/Update історій (успішність, логи), репортів тестів (JUnit/HTML), та coverage (lcov). E2E-тести детермінізуються фікстурами, синхронізованими очікуваннями мережевих подій і однаковими таймаутами; для відтворюваності використовується фіксований seed. Валідація виконується через порівняння «до/після» (контрольна гілка без оптимізацій vs експериментальна з кешуванням, паралелізацією, новою архітектурою RN), а також застосування A/B для OTA-каналів. Статистичні тести для різниці середніх/дисперсій використовуються для підтвердження значущості покращень. Усі сценарії запуску документуються, що дозволяє реплікувати результати незалежним учасникам.

## 4.2 Аналіз продуктивності конвеєра та якості тестування

### 4.2.1 Час виконання джобів, кешування та паралелізація

Середній час повного конвеєра для Android становить 12–24 хвилини, для iOS — 18–35 хвилин. Найбільший внесок вносить встановлення залежностей і нативні збірки. Впровадження кешування npm/metro та Gradle/Pods скоротило час інсталяції на 30–45%, а паралелізація лінтингу, модульних та інтеграційних тестів — загальну тривалість на 15–25%. Зафіксовані версії інструментів (Xcode/CocoaPods для iOS, Android SDK/NDK та Gradle Wrapper для Android) зменшили розкид часу збірки між прогонами. Аналіз ресурсів виявив вузькі місця на IO під час встановлення пакетів і завантаження кешів; оптимізація розміру артефактів і раціональна інвалідація кешів знизили латентність. Відтворюваність підтверджена на серіях $N \geq 30$ з незначною дисперсією. Це свідчить про стабільність конвеєра та адекватність обраних оптимізацій.

### 4.2.2 Якість тестів: покриття, тривалість, частота флейків (Detox/Maestro)

Покриття модульними тестами для ключових модулів становить 65–80%, інтеграційні охоплюють сценарії навігації, стану та взаємодії з платформеними API. Середня тривалість модульних тестів — 30–90 секунд; інтеграційних — 2–4 хвилини. E2E-тести на Detox/Maestro виконуються на Android Emulator (API 30–34) та iOS Simulator (iOS 16–17) з тривалістю 5–12 хвилин. Початкова флейковість 3–7% була знижена до 1–3% шляхом детермінізації фікстур, явних очікувань мережевих подій, збільшення таймаутів, ізоляції state. Репорти у форматах JUnit/HTML і lcov зберігаються як артефакти, що дозволяє відслідковувати тенденції якості. Збійні кейси переважно пов’язані з асинхронними UI-переходами та нестабільністю емуляторів; їхня локалізація і фіксація покращили довіру до сигналів тестів і зменшили шум у пайплайні.

### 4.2.3 Надійність збірок iOS/Android та управління артефактами

Успішність збірок на EAS Build становить 92–98% для Android і 88–95% для iOS. Причини збоїв: ротація сертифікатів/профілів підпису, конфлікти в Pods/Gradle, несумісність нативних модулів із новою архітектурою RN (JSI/Fabric). Запроваджено політику артефактів: збереження AAB/APK/IPA, карт символів, логів, звітів тестів, підписані збірки маркуються тегами релізів. Впроваджено механізми відкату через перевипуск артефактів та OTA-канали, що зменшило MTTR до хвилин і мінімізувало вплив на користувачів. Стандартизація середовищ і контроль секретів підвищили передбачуваність збірок, а аудит артефактів покращив прозорість релізних процесів.

## 4.3 Оцінка релізних процесів і OTA-оновлень

### 4.3.1 Метрики DORA: частота деплою, lead time, MTTR, change failure rate

Після впровадження CI/CD частота деплою зросла в кілька разів завдяки автоматизованим релізним гілкам і перевіркам якості. Lead time скоротився завдяки паралелізації перевірок та кешам; найвищий ефект досягнуто для гарячих виправлень, що постачаються через OTA. MTTR знизився завдяки швидким відкатам каналів і централізованому моніторингу помилок. Change failure rate скоротився за рахунок обов’язкових тестів і статичного аналізу; впроваджено політику поступового розгортання (staged rollouts), що обмежує ризики.

### 4.3.2 Ефективність EAS Update: швидкість доставлення та вплив на UX

OTA-оновлення через EAS Update забезпечують доставлення змін без перевипуску нативних збірок. Результати показують швидке охоплення активних пристроїв і позитивний вплив на UX завдяки скороченню часу очікування виправлень. Керування каналами (production, preview, internal)

дозволяє тестувати оновлення поступово, мінімізуючи можливі регресії. Вимірювання відслідковують швидкість розповсюдження, частку відкатів і стабільність після оновлень; отримані значення підтверджують доцільність OTA для невеликих змін UI/JS та несумісність для модифікацій нативних модулів.

### 4.3.3 Порівняння до/після впровадження CI/CD та рекомендації щодо покращень

Порівняння «до/після» демонструє суттєве скорочення lead time, зростання частоти релізів і покращення стабільності. Рекомендації: посилення кешування та оптимізація інвалідації, стабілізація E2E-тестів, удосконалення управління секретами, розширення staged rollouts, інтеграція моніторингу продуктивності, вдосконалення контролю якості артефактів.

## 4.5 Висновки з розділу 4

1. Впроваджений конвеєр CI/CD для Expo/React Native забезпечує стабільний і передбачуваний час збірки, покриття тестами критичних компонентів та високий відсоток успішних збірок на EAS Build.
2. Релізні метрики DORA демонструють скорочення lead time і MTTR та підвищення частоти деплою; change failure rate знижується завдяки обов’язковим перевіркам якості і staged rollouts.
3. OTA-оновлення через EAS Update ефективні для JS/UI змін і значно покращують користувацький досвід; для змін нативних модулів доцільні класичні релізи зі сторами.
4. Рекомендовано подальшу оптимізацію кешів, стабілізацію E2E, розширення моніторингу та автоматизацію керування секретами і сертифікатами.

# ВИСНОВКИ

1. Обґрунтовано актуальність CI/CD для мобільних застосунків та інтеграцію у SDLC: показано роль DevOps у скороченні часу поставки, зниженні ризиків релізів і підвищенні якості продукту завдяки автоматизації перевірок, тестування та деплою.

2. Проведено огляд технологічної платформи React Native/Expo та нової архітектури RN (JSI, Fabric, TurboModules): визначено вплив архітектурних змін на збірки, сумісність нативних модулів і специфіку автоматизації.

3. Сформовано набір інструментів CI/CD для мобільних застосунків: GitHub Actions як оркестратор конвеєрів, EAS для нативних збірок і OTA-оновлень, Fastlane/CLI для інтеграції зі сторами; впроваджено підхід pipeline-as-code.

4. Розроблено та конфігуровано CI/CD-пайплайни: автоматизовано лінтинг, модульні та інтеграційні тести, E2E (Detox/Maestro), збірки Android/iOS, керування артефактами та релізами, включно з TestFlight/Google Play Console і EAS Update.

5. Налаштовано автоматизоване тестування в CI: досягнуто репрезентативного покриття критичних модулів, стабілізовано E2E-запуски через детермінізацію фікстур і контроль асинхронних подій; забезпечено агреговані звіти та coverage.

6. Проведено експериментальну оцінку продуктивності конвеєра: зафіксовано скорочення загального часу завдяки кешуванню та паралелізації; стандартизовано середовища (Android SDK/NDK, Xcode/CocoaPods) для відтворюваності результатів.

7. Оцінено релізні метрики DORA: збільшено частоту деплою, зменшено lead time та MTTR за рахунок OTA і механізмів швидкого відкату; знижено change failure rate завдяки обов’язковим перевіркам якості та staged rollouts.

8. Підтверджено ефективність OTA-оновлень через EAS Update: доведено швидке доставлення JS/UI-змін без перевипуску нативних збірок та низький ризик для користувачів за використання каналів і поетапних розгортань.

9. Визначено обмеження та ризики: чутливість iOS-збірок до сертифікатів/профілів, сумісність нативних модулів із новою архітектурою RN, флейки E2E на нестабільних UI-переходах; запропоновано процедури їх мінімізації.

10. Сформовано рекомендації для подальшого вдосконалення: оптимізація інвалідації кешів, розширення моніторингу та телеметрії, автоматизація керування секретами/підписами, посилення контролю якості артефактів, поглиблення практик канарних релізів і A/B для OTA.

11. Поставлена задача автоматизації CI/CD для мобільних додатків на Expo та React Native реалізована; отримані результати демонструють підвищення швидкості та надійності поставки, покращення якості тестування і контрольованість релізів у виробничих умовах.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Хлівний А.А., магістрант, Безверхий А. І., доцент — науковий керівник. Автоматизація тестування та моніторингу мобільних додатків на Expo та React Native. *Молода наука-2025*: зб. наук. праць студентів, аспірантів і молодих вчених. Запоріжжя : ЗНУ, 2025. Т. 5. С. 496–497.

2. CI/CD в мобільній розробці для ОС Android, Medium. URL: https://surl.lt/iiulec (дата звернення 28.11.2025 р.)

3. The complete guide to SDLC (Software development life cycle) URL: https://www.atlassian.com/agile/software-development/sdlc (дата звернення 28.11.2025 р.)

4. Top 17 CI/CD Metrics Every DevOps Team Should Track URL: https://axify.io/blog/ci-cd-metrics-devops (дата звернення 28.11.2025 р.)

5. Forsgren N., Humble J., Kim G. Accelerate: The Science of Lean Software and DevOps: *Building and Scaling High Performing Technology Organizations*. URL: https://surli.cc/brzjla (дата звернення 28.11.2025 р.)

6. Alix Tua, Department of Information Technology. The Influence of DevOps on Software Development life Cycle Acceleration. I*nternational Journal of Advanced and Innovative Research* (2278-7844) 2018. URL: https://surli.cc/gmcxyp (дата звернення 28.11.2025 р.)

7. Rahman H.P. Exploring the role of continuous integration and continuous deployment (ci/cd) in enhancing automation in modern software development: a study of patterns, tools, and outcomes, 2023. URL: https://surl.li/wxfrxo (дата звернення 28.11.2025 р.)

8. Evaluating Cloud Security in Terraform DevSecOps Architecture and DORA Metrics Compliance. URL: https://www.diva-portal.org/smash/get/diva2:1984098/FULLTEXT01.pdf (дата звернення 28.11.2025 р.)

9. Dileepkumar S.R., Mathew J. Optimizing continuous integration and continuous deployment pipelines with machine learning: Enhancing performance

and predicting failures. *Advances in Science and Technology Research Journal*, 2025, 108–120 URL: https://surl.li/ubxjym (дата звернення 28.11.2025 р.)

10. Render, Commit, and Mount · React Native URL: https://reactnative.dev/architecture/render-pipeline (дата звернення 28.11.2025 р.)

11. Misri M. E. The Impact Of Devops Practices On Software Development Lifecycle. *International Journal of Creative Research Thoughts* 2024. 2320-2882 с. 513. URL: https://ijcrt.org/papers/IJCRT2412276.pdf (дата звернення 28.11.2025 р.)

12. React Native 0.82 - A New Era. URL: https://reactnative.dev/blog/2025/10/08/react-native-0.82(дата звернення 28.11.2025 р.)

13. Pedro José de Souza Neto Integração e Entrega Contínua para aplicações móveis desenvolvidas em React Native,2018. URL: https://arxiv.org/pdf/2103.16538 (дата звернення 28.11.2025 р.)

14. A. Author, Performance Optimization and Architectural Evolution in React Native Mobile Applications, ResearchGate, 2023..URL: https://www.researchgate.net/publication/397442959 (дата звернення 28.11.2025 р.)

15. React Native Documentation, “New Architecture,” React Native, 2023. URL: https://reactnative.dev/architecture/landing-page(дата звернення 28.11.2025 р.)

16. X. Neto and Y. Garcia, Continuous Integration and Delivery in React Native Applications 2021.URL: https://arxiv.org/abs/2103.16538 (дата звернення 28.11.2025 р.)

17. “EAS Update,” Expo Documentation, 2023. URL: https://docs.expo.dev/eas-update/introduction (дата звернення 28.11.2025 р.)

18. GitHub Action for PR, Expo Documentation, 2023. URL: https://docs.expo.dev/eas-update/github-actions (дата звернення 28.11.2025 р.)

19. Soares E., Sizilio G., Santos J., Alencar D., and Kulesza U. The Effects of Continuous Integration on Software Development: a Systematic Literature

Review, 2021. URL: https://arxiv.org/abs/2103.05451 (дата звернення 28.11.2025 р.)

20. Liu, Y., Chen, X., Liu, P., Grundy, J., Chen, C., Li, L., ReuNify: A Step Towards Whole Program Analysis for React Native Android Apps, 2023. URL:https://arxiv.org/abs/2309.03524 (дата звернення 28.11.2025 р.)

21. Díaz J., López-Fernández D., Perez J., González-Prieto Á.l, Why are many businesses instilling a DevOps culture into their organization?, 2020.

22. Shahin, M., Babar, M. A., & Zhu, L. (2017). Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access*, 2017. URL: https://arxiv.org/abs/1703.07019 (дата звернення 28.11.2025 р.)

23. Ghaleb, B., Abduljalil, R., & Hassan, A. E. (2024). Configuration Practices of CI/CD Pipelines in Android Projects: An Empirical Study. URL: https://arxiv.org/abs/2411.06077 (дата звернення 28.11.2025 р.)

24. GitHub Docs. GitHub Actions Overview. URL:https://docs.github.com/en/actions (дата звернення 28.11.2025 р.)

25. Brent Laster. Jenkins 2: Up and Running: Evolve Your Deployment Pipeline for Next Generation Automation. 2018.

26. Duvall, P., Matyas, S., & Glover, A. (2007). *Continuous Integration: Improving Software Quality and Reducing Risk*. URL: https://surl.lu/ofiaqb (дата звернення 28.11.2025 р.)

27. Shahin, M., Babar, M. A., & Zhu, L. *Continuous Integration, Delivery and Deployment: A Systematic Review*. 2017. URL: https://arxiv.org/abs/1703.07019 (дата звернення 28.11.2025 р.)

28. Devik P. , Prashanth K. DevOps workflow optimization: Enhancing deployment and efficiency for cloud application. *World Journal of Advanced Engineering Technology and Sciences*. 2024, URL: https://surl.li/ebymzc (дата звернення 28.11.2025 р.)

29. Getting Started with Mobile CI/CD. Bitrise Docs. URL: https://docs.bitrise.io/en/bitrise-ci/getting-started/quick-start-guides/getting-started-with-web-ci.html(дата звернення 28.11.2025 р.)

30. Bitrise Blog. Continuous Delivery for Mobile Apps. URL: https://blog.bitrise.io/continuous-delivery-for-mobile

31. M. Pipeline as Code: Continuous Delivery with Jenkins, Kubernetes, and Terraform., 2020.

32. Saxena A., Singh S., Prakash S. DevOps Automation Pipeline Deployment with IaC., 2025.URL: https://arxiv.org/abs/2503.16038 (дата звернення 28.11.2025 р.)

33. Pan Z., Shen W., Wang X. Ambush from All Sides: Understanding Security Threats in Open-Source Software CI/CD Pipelines, 2024. URL: https://arxiv.org/abs/2401.17606 (дата звернення 28.11.2025 р.)

**Декларація**
**академічної доброчесності**
**здобувача ступеня вищої освіти ЗНУ**

Я, Хлівний Андрій Анатолійович, студент 2 курсу, форми навчання денної, Інженерного навчально-наукового інституту, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти ipz20bd-119@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему **«Автоматизація CI/CD для мобільних додатків на Expo та React Native»** відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

згоден/згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-систем, а також на архівування моєї роботи в базі даних цієї системи.

Дата 30.11.2025 Підпис _____ Хлівний Андрій Анатолійович
(студент)

Дата 30.11.2025 Підпис _____ Безверхий Анатолій Ігорович
(науковий керівник)