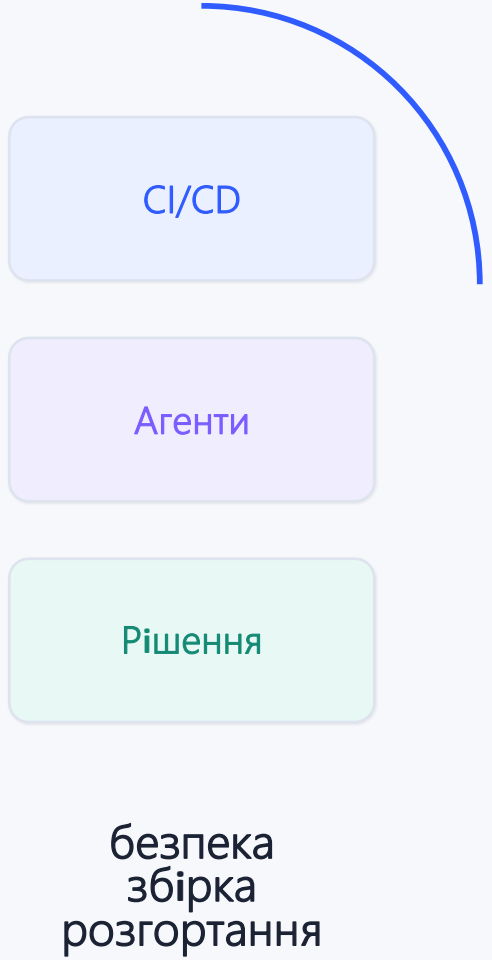
Автоматизована система для безпечної збірки та розгортання вебдодатків з використанням агентних технологій



Малащук Іванна Олександрівна
Освітня програма «Інженерія програмного забезпечення»
Науковий керівник: Кундік В.В.

Мета дослідження

Розробити прототип системи, яка аналізує вебпроєкт, оцінює ризики та приймає рішення щодо можливості подальшого розгортання.

- проаналізувати CI/CD-процеси та загрози безпеці
- обґрунтувати використання агентно-орієнтованого підходу
- розробити архітектуру прототипу системи
- реалізувати статичний і динамічний аналіз
- створити агентну модель оцінювання результатів

Об’єкт:
автоматизація безпечної збірки, тестування та розгортання

Результат:
робочий прототип системи аналізу репозиторіїв

Відомі підходи до розв’язання задачі

Які перевірки зазвичай застосовуються у CI/CD

Статичний аналіз коду

Аналіз залежностей

Пошук секретів

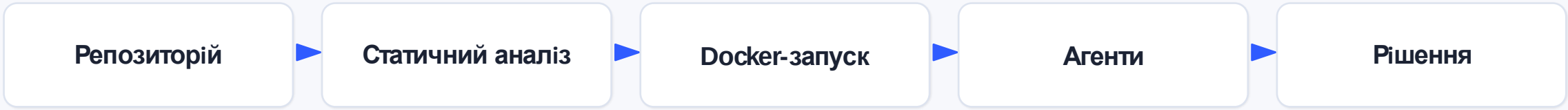
Аналіз конфігурацій

- окремі інструменти добре знаходять конкретні типи проблем
- результати різних перевірок часто не узгоджуються між собою
- система може зафіксувати помилку, але не пояснити її вплив на розгортання
- потрібен механізм комплексної оцінки ризику

Ключова потреба - не лише знайти проблему, а й визначити, що робити з розгортанням далі.

Суть запропонованого методу

Поєднання автоматизованих перевірок і агентного рішення



- система отримує дані репозиторію та аналізує важливі файли проєкту
- перевіряються код, залежності, Dockerfile і конфігураційні файли
- динамічний аналіз виконує встановлення залежностей, збірку та тести в ізольованому середовищі
- агенти узагальнюють результати і визначають рівень ризику

deploy | manual_review | block

Фінальний результат - кероване рішення щодо подальшого розгортання.

Використані технології

основні засоби реалізації прототипу

Node.js та Express

серверна частина, маршрути, контролери, служби аналізу

HTML, CSS, JavaScript

клієнтський вебінтерфейс для запуску перевірки

Git та GitHub

отримання репозиторію і робота з копією проєкту

Docker

ізольований запуск встановлення залежностей, збірки та тестів

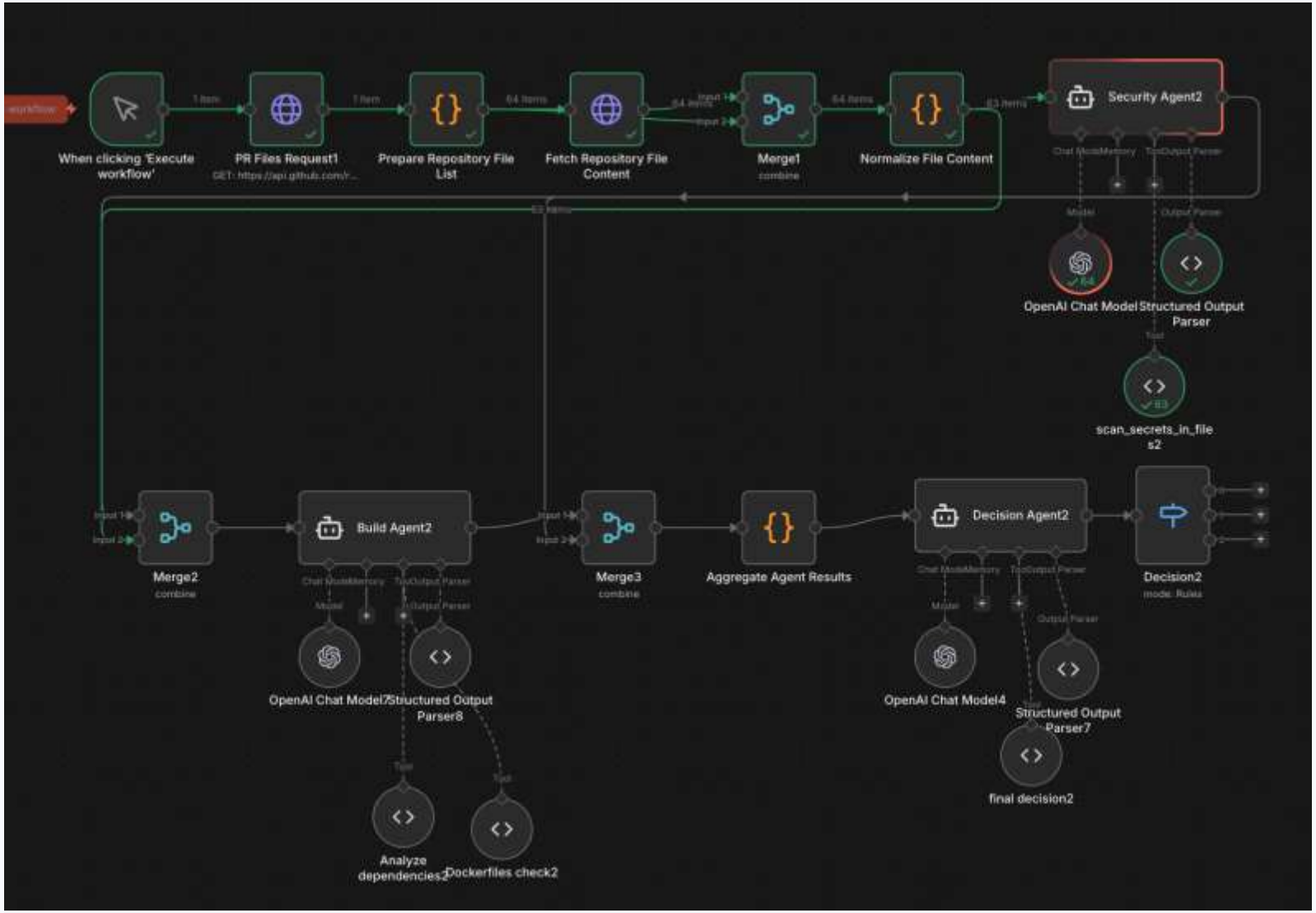
OpenAI 4-mini

LLM

n8n

попереднє моделювання взаємодії агентів

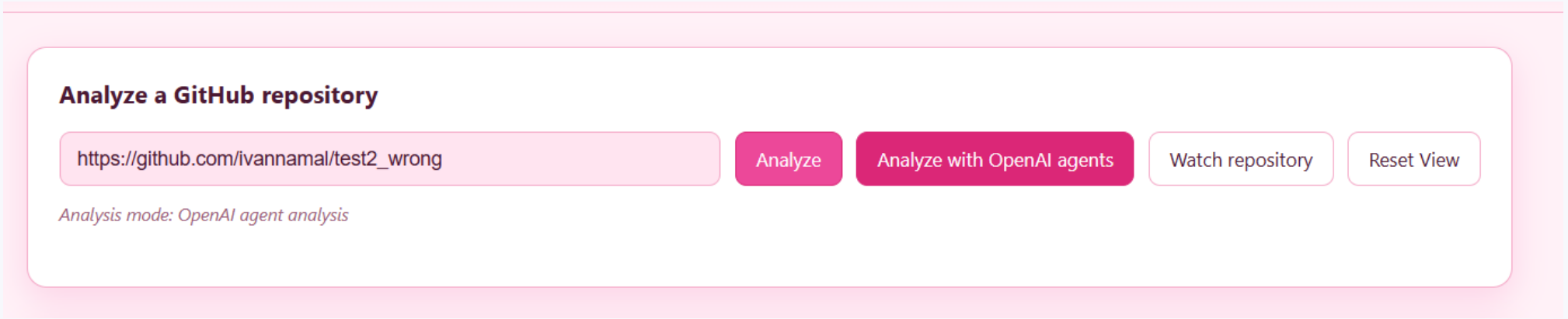
Попереднє моделювання агентного процесу в середовищі n8n



Приклад застосування до реальної задачі

Як система перевіряє репозиторій вебдодатка

Клонування → Вибір файлів → Перевірки → Збірка і тести → Висновок



- користувач передає посилання на репозиторій
- система визначає структуру проєкту та важливі файли
- виконується пошук секретів, небезпечних команд і проблем конфігурації
- проєкт запускається в Docker-контейнері
- за наявності критичної проблеми або невдалих тестів розгортання блокується

Приклад застосування до реальної задачі

(repository-wide)

Tests (npm test) failed inside Docker.

Run "npm test" locally and resolve failing tests. See stdout/stderr tails below for the failing assertions.

Verified AI fix HIGH

The test fails because the environment variable DATABASE_URL is not set, causing getDatabaseConfig() to throw an error. The fix is to set a valid DATABASE_URL environment variable before calling getDatabaseConfig() in the test.js file.

CHANGED FILES

- <code>test.js</code>

VERIFICATION PLAN

- Run 'npm test' locally and verify that all tests pass without errors.
- Ensure that the test output includes 'All tests passed'.
- Confirm that the environment variable DATABASE_URL is correctly used in the test.

RISK NOTES

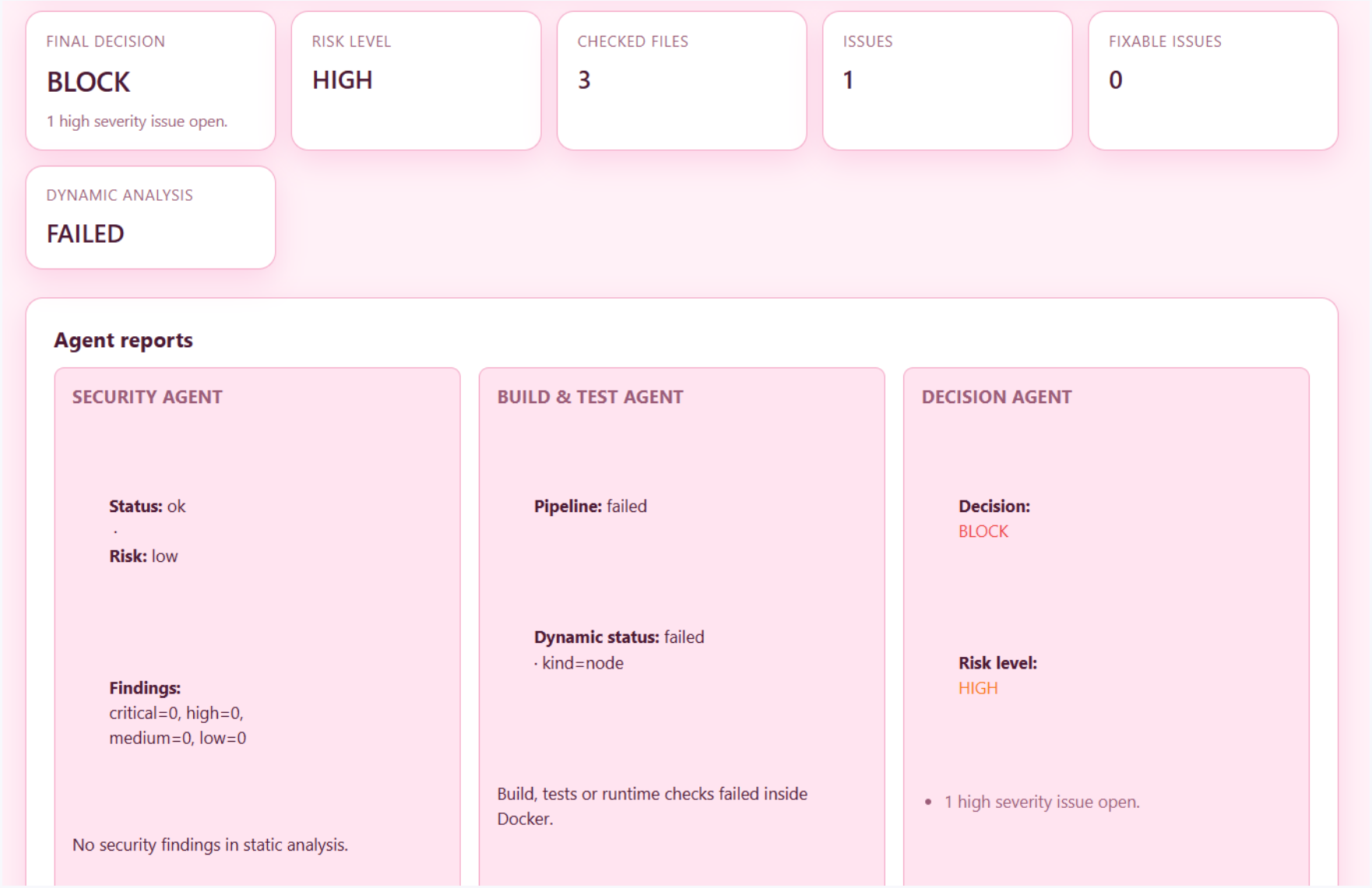
- Setting the DATABASE_URL environment variable in the test file is safe for testing purposes.
- This change does not affect production code or behavior outside the test environment.

▼ Diff preview

```
--- a/test.js
+++ b/test.js
 const assert = require("assert");
 const { getDatabaseConfig, getAppStatus } = require("./index");

 assert.deepStrictEqual(
   getAppStatus(),
   {
     service: "demo-api",
     status: "ready"
   },
   "App status should be ready"
 );

+// Set DATABASE_URL environment variable for testing
+process.env.DATABASE_URL = "postgres://user:password@localhost:5432/mydb";
+
 const config = getDatabaseConfig();

 assert.ok(config.url.startsWith("postgres://"), "DATABASE_URL should be a PostgreSQL URL");

 console.log("All tests passed");
```

Apply verified fix

Приклад застосування до реальної задачі



Приклад застосування до реальної задачі

OpenAI agent analysis

model: gpt-4.1-mini

SECURITY ANALYSIS AGENT

Status: ok

.

Risk:
LOW

No security findings in static analysis.

BUILD & TEST ANALYSIS AGENT

Status: failed

Build, tests or runtime checks failed inside Docker.

- Tests (npm test) failed inside Docker due to missing DATABASE_URL environment variable.

DECISION AGENT

Decision:
BLOCK

Risk level:
HIGH

Reason: High severity test failure inside Docker due to missing environment variable.

Required actions:

- Fix the missing DATABASE_URL environment variable for tests.
- Ensure tests pass inside Docker before deployment.

Інші приклади застосування

де може використовуватися система

історія нещодавніх перевірок

приклад застосованого виправлення

- перевірка запиту на злиття перед додаванням коду в основну гілку
- пошук випадково доданих секретів
- виявлення небезпечних команд у конфігураціях
- аналіз Dockerfile та package.json
- перевірка помилок, що проявляються лише під час запуску
- формування пропозицій щодо виправлення проблем

Порівняння з відомими підходами

традиційний підхід і запропонована система

Традиційний CI/CD

- виконує наперед задані команди
- повертає переважно технічний статус
- потребує ручної інтерпретації помилки
- окремі інструменти часто працюють ізольовано

Запропонована система

- поєднує кілька типів перевірок
- аналізує контекст помилки
- оцінює рівень ризику
- формує рішення deploy, manual_review або block

У попередньому тестуванні мовних моделей найдоцільнішою для подальшого використання була OpenAI 4-mini: вона поєднала високу точність із нижчою вартістю багаторазових перевірок.

Наукова новизна

Запропоновано підхід до побудови агентної CI/CD-системи, у якій результати статичного та динамічного аналізу узгоджуються програмними агентами для прийняття рішення щодо безпечності подальшого розгортання вебдодатка.

відокремлені агенти

обмін результатами аналізу

комплексна оцінка ризику

рішення перед розгортанням

- система не обмежується одним видом перевірки
- враховується рівень ризику та тип проблеми
- результати перевірок впливають на подальший перебіг розгортання

Практична значимість

Розроблений прототип може бути використаний як основа для системи, що автоматично перевіряє вебдодатки перед розгортанням, зменшує ризик помилок і прискорює прийняття рішення щодо безпечності змін.

швидше виявлення проблем

менша ймовірність людської помилки

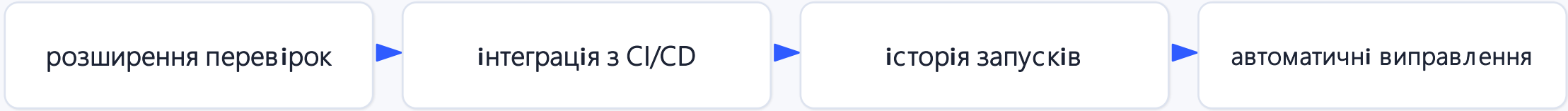
перевірка збірки й тестів у Docker

блокування критичних ризиків

рекомендації щодо виправлення

Система корисна для попередньої перевірки репозиторіїв і подальшої інтеграції з CI/CD-процесами.

Перспективи подальших досліджень



- підтримка більшої кількості мов програмування та типів проєктів
- глибший аналіз залежностей і відомих вразливостей
- інтеграція з GitHub Actions або GitLab CI
- покращення аналізу журналів виконання
- збереження історії перевірок і результатів
- розвиток механізму автоматичного формування та застосування виправлень

Дякую за увагу!