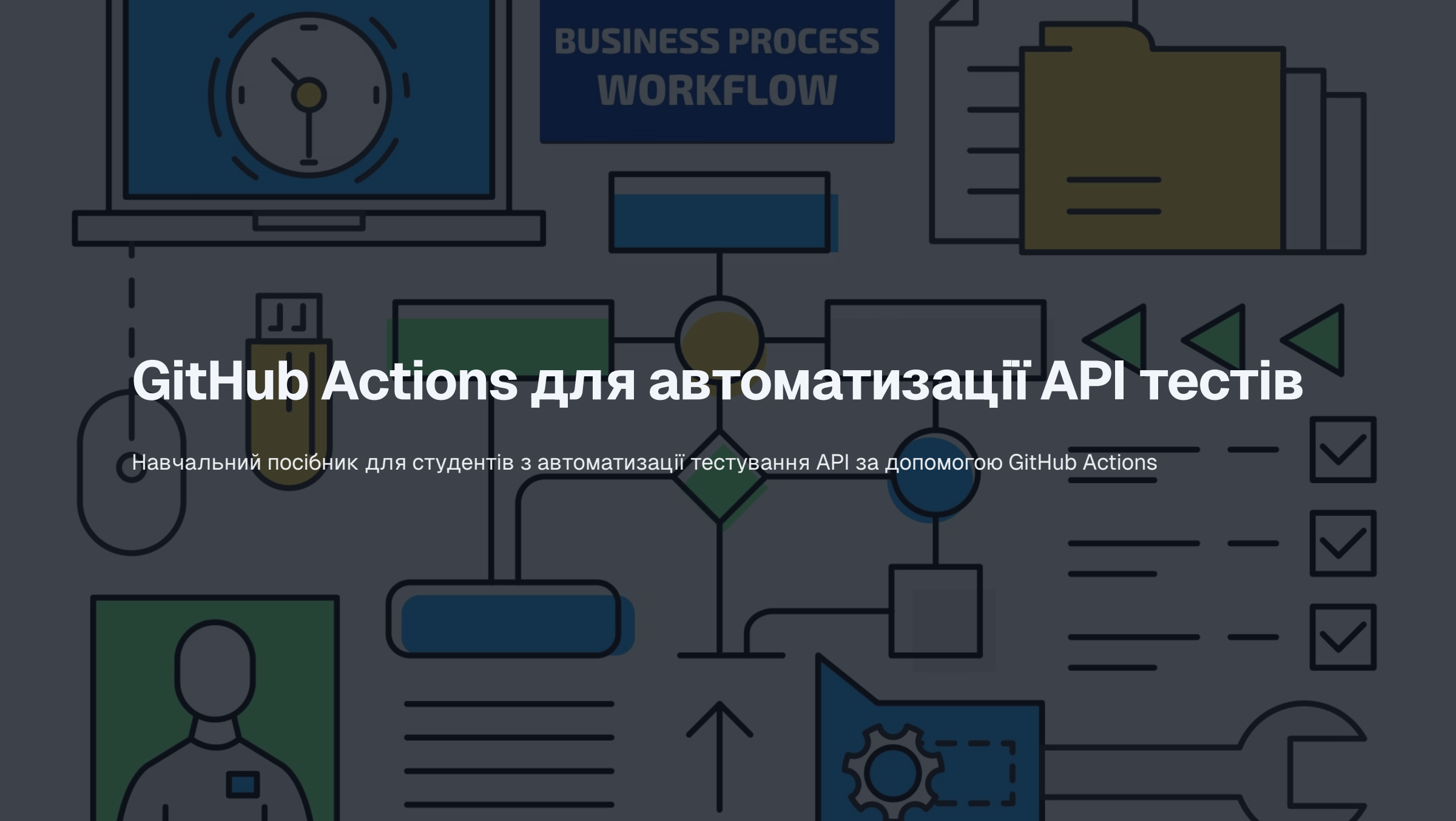

GitHub Actions для автоматизації API тестів

Навчальний посібник для студентів з автоматизації тестування API за допомогою GitHub Actions

Вступ до автоматизації тестування

Що таке CI/CD

CI/CD (Continuous Integration/Continuous Delivery) — це практика автоматичного тестування та розгортання коду. Кожна зміна в коді автоматично перевіряється тестами, що дозволяє швидко виявляти помилки та підтримувати високу якість продукту.

Навіщо автоматизувати API тести

Автоматизація API тестів економить час розробників та QA інженерів, забезпечує стабільність додатку та дозволяє виявляти проблеми на ранніх етапах розробки. Замість ручного запуску тестів після кожної зміни, система робить це автоматично.

Роль GitHub Actions

GitHub Actions — це вбудований інструмент автоматизації від GitHub, який дозволяє налаштувати CI/CD безпосередньо в репозиторії. Він запускає ваші тести автоматично при певних подіях, таких як push коду або створення pull request.

Що таке GitHub Actions

GitHub Actions — це потужна платформа автоматизації, яка дозволяє створювати робочі процеси (workflows) для вашого проєкту. Розуміння основних концепцій допоможе вам ефективно використовувати цей інструмент.

01

Workflow (Робочий процес)

Це автоматизований процес, який складається з одного або кількох jobs. Workflow визначає, коли і як запускати ваші тести. Наприклад, workflow може запускатися при кожному push коду в репозиторій.

02

Jobs (Завдання)

Кожен workflow містить одне або більше завдань. Job — це набір кроків, які виконуються на одному runner (віртуальній машині). Jobs можуть виконуватися паралельно або послідовно.

03

Steps (Кроки)

Це окремі команди або дії, які виконуються в межах job. Наприклад, встановлення залежностей, запуск тестів або публікація результатів. Кожен крок виконується послідовно.

Де зберігаються воркфлови

Всі workflow-файли зберігаються в папці `.github/workflows` вашого репозиторію. Це YAML-файли з розширенням `.yml` або `.yaml`.

Коли вони запускаються

Workflows можуть запускатися автоматично при різних подіях: push, pull request, створення issue, за розкладом (cron) або вручну через інтерфейс GitHub.

Передумови для налаштування

Перед тим, як почати налаштовувати GitHub Actions для автоматизації API тестів, переконайтеся, що у вас є всі необхідні компоненти. Ці передумови є фундаментом для успішної інтеграції CI/CD у ваш проєкт.

1

GitHub репозиторій

Ваш код має бути розміщений у GitHub репозиторії. Це може бути публічний або приватний репозиторій. GitHub Actions доступний для всіх типів репозиторіїв.

2

Node.js проєкт

Ваш проєкт має бути налаштований як Node.js додаток з відповідною структурою файлів. Переконайтеся, що локально проєкт працює коректно.

3

API тести

У вас мають бути написані API тести, які можна запустити командою `npm run api_tests`. Тести повинні успішно виконуватися локально перед налаштуванням CI.

4

Файл package.json

У файлі `package.json` має бути визначений скрипт `api_tests` у секції scripts. Наприклад: `"api_tests": "jest tests/api"` або інша команда для запуску ваших тестів.

Workflow для запуску тестів при push

Найпростіший спосіб автоматизувати тестування — налаштувати запуск тестів при кожному push коду в певну гілку. Це дозволяє миттєво перевіряти, чи не зламали ваші зміни існуючу функціональність. Розглянемо детально, як створити такий workflow.

Тести запускаються автоматично при кожному push у гілку `my_branch`. Це означає, що щоразу, коли ви відправляєте зміни в цю гілку командою `git push`, GitHub Actions автоматично запустить ваші API тести.

Приклад YAML конфігурації

```yaml
name: CI API TESTS
on:
  push:
    branches: ['my_branch']
jobs:
  tests:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
      - name: Install dependencies
        run: npm ci
      - name: Run api tests
        run: npm run api_tests
```

actions/checkout@v3

Цей крок завантажує ваш код з репозиторію на віртуальну машину GitHub. Без цього кроку система не матиме доступу до вашого коду.

actions/setup-node@v3

Встановлює Node.js на віртуальній машині. Це необхідно для запуску npm команд та виконання JavaScript коду.

npm ci

Встановлює всі залежності проєкту з файлу `package-lock.json`. Команда `npm ci` швидша та надійніша для CI, ніж `npm install`.

npm run api_tests

Запускає ваші API тести. Якщо тести провалюються, весь workflow завершується з помилкою, і ви отримаєте сповіщення.

Workflow для запуску тестів при pull request

Окрім автоматичного запуску тестів при push, дуже корисно налаштувати перевірку коду при створенні pull request. Це дозволяє переконатися, що нові зміни не порушують функціональність до того, як вони будуть об'єднані з основною гілкою.

Тести запускаються автоматично при відкритті pull request у гілку `my_branch`. Це означає, що коли хтось створює PR, GitHub Actions автоматично перевірить, чи проходять всі тести з новими змінами.

Приклад YAML конфігурації для PR

```
name: CI API Tests
on:
  pull_request:
    types: ['opened']
    branches: ['my_branch']
jobs:
  tests:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
      - name: Install dependecies
        run: npm ci
      - name: Run api test
        run: npm run api_tests
```

Різниця між push і pull_request

- **push** — запускається при кожному відправленні коду в гілку
- **pull_request** — запускається при створенні, оновленні або інших діях з PR
- Push перевіряє зміни після їх додавання
- Pull request перевіряє зміни перед об'єднанням

Типи подій pull_request

- `opened` — PR тільки створено
- `synchronize` — додано нові коміти
- `reopened` — PR відкрито знову
- Можна комбінувати: ['opened', 'synchronize']

Два воркфлови в одному репозиторії

GitHub Actions дозволяє мати кілька workflow-файлів в одному репозиторії. Це дуже зручно, коли вам потрібно налаштувати різні процеси автоматизації для різних сценаріїв. Наприклад, один workflow для перевірки коду при push, а інший — для більш детальної перевірки при створенні pull request.

Окремі файли для різних цілей

Створіть файл `push-tests.yml` для швидких перевірок при push та `pr-tests.yml` для детальніших перевірок при PR. Кожен файл працює незалежно.

Паралельне виконання

Різні workflows можуть запускатися одночасно. Наприклад, при створенні PR можуть одночасно виконуватися тести, лінтинг коду та перевірка безпеки.

Зручність підтримки

Розділення workflows на окремі файли робить їх простішими для читання та підтримки. Кожен файл відповідає за конкретну задачу та легко модифікується.

> **Порада:** Використовуйте описові назви для workflow-файлів, наприклад `api-tests-on-push.yml` та `api-tests-on-pr.yml`. Це допоможе швидко зрозуміти призначення кожного файлу.

Зміна існуючого воркфлову

Не завжди потрібно створювати новий workflow з нуля. Часто простіше та ефективніше змінити існуючий workflow, адаптувавши його під нові потреби. Це особливо корисно, коли ви хочете додати нові умови запуску або змінити кроки виконання.

Зміна умов запуску (on:)

Ви можете змінити події, які запускають workflow. Наприклад, додати нові гілки до списку: `branches: ['main', 'develop', 'my_branch']` або змінити тип події з `push` на `pull_request`.

Модифікація кроків (steps:)

Ви можете додавати нові кроки, видаляти непотрібні або змінювати порядок виконання. Наприклад, додати крок для генерації звітів про тестування або змінити версію Node.js у `actions/setup-node`.

Додавання змінних середовища

Можна додати секцію `env:` для визначення змінних середовища, які будуть доступні в усіх кроках. Це корисно для налаштування API ключів або URL тестового середовища.

Приклад розширеного workflow

```
name: Enhanced API Tests
on:
  push:
    branches: ['main', 'develop']
  pull_request:
    types: ['opened', 'synchronize']
env:
  API_URL: https://test-api.example.com
jobs:
  tests:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
        with:
          node-version: '18'
      - run: npm ci
      - run: npm run api_tests
      - run: npm run generate-report
```

Що змінилося

- Додано кілька гілок для push
- Додано pull_request події
- Встановлено змінну середовища
- Вказано версію Node.js
- Додано крок генерації звіту

Типові помилки

При налаштуванні GitHub Actions новачки часто стикаються з типовими помилками. Розуміння цих помилок допоможе вам швидко їх виправити та налаштувати робочий CI/CD процес. Розглянемо найпоширеніші проблеми та способи їх вирішення.

Неправильна назва гілки

Одна з найчастіших помилок — вказати неіснуючу назву гілки в конфігурації. Переконайтеся, що назва гілки в `branches: ['my_branch']` точно відповідає назві гілки в репозиторії. Назви гілок чутливі до регістру!

Відсутній скрипт api_tests

Workflow намагається виконати команду `npm run api_tests`, але якщо такого скрипта немає в `package.json`, виконання завершиться помилкою. Перевірте секцію `scripts` у вашому `package.json` файлі.

Помилки в YAML відступах

YAML дуже чутливий до відступів. Використовуйте тільки пробіли (не табуляцію) та дотримуйтесь однакової кількості пробілів для одного рівня вкладеності. Зазвичай використовують 2 пробіли на рівень.

Забули зробити push у репозиторій

Після створення або зміни workflow-файлу не забудьте зробити commit та push змін у репозиторій. GitHub Actions не побачить ваш workflow, поки він не буде в репозиторії на GitHub.

> **Корисна порада:** Використовуйте онлайн валідатори YAML для перевірки синтаксису вашого workflow-файлу перед commit. Це допоможе виявити помилки форматування на ранньому етапі.

Практичне завдання

Тепер, коли ви ознайомилися з теорією, настав час застосувати знання на практиці. Це завдання допоможе вам закріпити матеріал та створити власний робочий CI/CD процес для автоматизації API тестів. Виконуйте кроки послідовно та уважно.

Створити workflow файл

У вашому репозиторії створіть папку `.github/workflows` (якщо її ще немає). Всередині створіть файл `api-tests.yml`. Скопіюйте один з прикладів YAML конфігурації з попередніх розділів.

Налаштувати запуск на push

Змініть назву гілки в конфігурації на вашу робочу гілку. Переконайтеся, що в секції `on:` вказано `push:` та правильна назва гілки. Збережіть файл.

Зробити commit та push

Додайте створений файл до git командою `git add .github/workflows/api-tests.yml`, зробіть commit `git commit -m "Add API tests workflow"` та відправте зміни `git push`.

Перевірити виконання

Перейдіть на GitHub у розділ **Actions** вашого репозиторію. Ви повинні побачити запущений workflow. Клікніть на нього, щоб переглянути деталі виконання кожного кроку та переконатися, що тести пройшли успішно.

Критерії успіху

- Workflow файл створено у правильній папці
- Конфігурація не містить синтаксичних помилок
- Workflow запускається автоматично при push
- Всі тести проходять успішно
- Результати видимі в розділі Actions

Додаткові завдання

- Створіть окремий workflow для pull requests
- Додайте крок для генерації звіту про тести
- Налаштуйте сповіщення при провалі тестів
- Експериментуйте з різними тригерами

Висновок

Тепер ви розумієте, як CI/CD допомагає підтримувати високу якість коду та чому автоматизація тестування є критично важливою для сучасної розробки програмного забезпечення.

Чому CI важливий для QA

Continuous Integration дозволяє QA інженерам автоматично перевіряти кожну зміну в коді, економлячи час на ручному тестуванні. Це забезпечує швидкий зворотний зв'язок розробникам та підвищує загальну якість продукту. Автоматизація рутинних перевірок дає змогу зосередитися на більш складних тестових сценаріях.

Як це допомагає знаходити баги раніше

Завдяки автоматичному запуску тестів при кожній зміні коду, проблеми виявляються одразу після їх появи. Це значно спрощує процес виправлення, оскільки розробник ще пам'ятає контекст своїх змін. Раннє виявлення багів економить час та ресурси команди.

Наступні кроки у вашому навчанні

Додати репорти

Налаштуйте генерацію детальних звітів про виконання тестів з використанням інструментів як Allure або Mochawesome.

Зберігати артефакти

Навчіться зберігати результати тестів, скріншоти та логи як артефакти для подальшого аналізу.

Налаштувати алерти

Інтегруйте сповіщення в Slack, email або інші канали для миттєвого інформування про провали тестів.

Пам'ятайте: **Автоматизація тестування — це інвестиція в якість вашого продукту.** Час, витрачений на налаштування CI/CD, окупається багаторазово завдяки швидшому виявленню проблем та підвищенню впевненості в стабільності коду.