UNICHECK

Ім'я користувача:
приховано налаштуваннями конфіденційності

ID перевірки:
1015576974

Дата перевірки:
13.06.2023 09:20:03 EEST

Тип перевірки:
Doc vs Library

Дата звіту:
13.06.2023 09:23:13 EEST

ID користувача:
100011372

Назва документа: Lapin 1-3

Кількість сторінок: 42 Кількість слів: 7671 Кількість символів: 57279 Розмір файлу: 1.45 MB ID файлу: 1015227382

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

0.5%
Схожість

Найбільша схожість: 0.23% з джерелом з Бібліотеки (ID файлу: 1015146119)

Пошук збігів з Інтернетом не проводився

0.5% Джерела з Бібліотеки | 18 ..... Сторінка 44

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування | 9 сторінок

1 ЗБІР І ОПИС ТЕОРЕТИЧНИХ ВІДОМОСТЕЙ З РОБОТИ З GITHUB ACTIONS ДЛЯ ПОБУДОВИ СИСТЕМИ АВТОМАТИЗАЦІЇ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Вступ до GitHub Actions. Огляд історії розвитку технології

GitHub Actions - це інструмент автоматизації розробки, який був запущений в 2018 році та надає можливість автоматизувати процес розробки безпосередньо в GitHub. Історія GitHub Actions може бути розділена на наступні етапи:

Етап 1: Перший анонс GitHub Actions. На конференції GitHub Universe в 2018 році було оголошено про запуск GitHub Actions. На той момент він був доступний лише для тестування та використання в рамках приватної бета-версії.

Етап 2: Публічний бета-реліз. У 2019 році GitHub Actions був запущений в публічній бета-версії. На той момент він був доступний для всіх користувачів GitHub. Для використання Actions потрібно було створити файл конфігурації в кореневій директорії репозиторію.

Етап 3: Введення в експлуатацію GitHub Marketplace. У 2019 році було оголошено про запуск GitHub Marketplace, де користувачі можуть знайти та встановлювати різноманітні додатки для своїх репозиторіїв. Також було додано багато нових функцій, таких як можливість використовувати зовнішні сервіси, налаштування графіку, розширені настройки безпеки та інше.

Етап 4: Випуск версії 2.0. У 2020 році була випущена версія 2.0 GitHub Actions. Вона містила багато нових функцій, таких як можливість розширювати дії за допомогою Docker-контейнерів, збільшення швидкості виконання дій та покращення інтерфейсу користувача.

Етап 5: GitHub Actions для публічних репозиторіїв. У 2020 році GitHub Actions був доступний для використання у публічних репозиторіях. Це дозволило розробникам використовувати Actions для автоматизації своїх проектів та спрощувати процеси розробки.

Етап 6: GitHub Actions для приватних репозиторіїв . У тому ж 2020 році GitHub оголосив, що Actions стає доступним для використання в приватних репозиторіях, що робить його більш доступним для команд розробників.

Етап 7: Підтримка Docker Container Registry. У 2021 році GitHub Actions отримав підтримку Docker Container Registry. Це дозволило розробникам легко зберігати та керувати контейнерами Docker без необхідності використання зовнішніх сервісів.

Етап 8: GitHub Actions з підтримкою CI/CD. У тому ж 2021 році GitHub оголосив про підтримку Continuous Integration/Continuous Deployment (CI/CD) в GitHub Actions. Це надало можливість розробникам автоматизувати процеси розгортання своїх додатків та забезпечити більшу стабільність та ефективність своїх проектів.

Етап 9: Розширення можливостей GitHub Actions. У 2022 році було оголошено про додання більшої кількості можливостей до GitHub Actions. До них належать нові типи подій, покращена робота з секретами, розширена підтримка мов програмування та платформ, а також інші функції, що дозволяють розробникам максимально ефективно використовувати Actions для своїх потреб.

Загалом, GitHub Actions став все більш інтегрованим та потужним інструментом для автоматизації розробки, який надає розробникам можливість прискорити та спростити процеси розробки та розгортання своїх проектів.

1.2 Основні поняття та терміни GitHub Actions

Робочі процеси. Робочі процеси GitHub Actions - це набір інструкцій у форматі YAML, який визначає, що потрібно зробити при зміні коду в репозиторії. Робочі процеси дозволяють автоматизувати різноманітні задачі, такі як тестування, збірка, розгортання та багато інших.

Кожен робочий процес GitHub Actions починається з події, наприклад, зміни коду в репозиторії. Далі слідують кроки, які потрібно виконати відповідно до цієї

події. Робочі процеси можуть мати кілька кроків, кожен з яких може бути скриптом або командою для виконання.

Робочі процеси можна запускати автоматично при зміні коду в репозиторії або за певного графіку, що дозволяє виконувати регулярні процеси, такі як резервне копіювання або поновлення бази даних.

Окрім того, GitHub Actions має вбудований магазин додатків, який містить більше 6000 різноманітних дій та робіт, які можна використовувати в робочих процесах. Наприклад, додаток для автоматичного тестування коду, додаток для розгортання на хмарні сервіси та багато інших.

Крім того, GitHub Actions дозволяє створювати свої власні дії та роботи, що дозволяє розробникам налаштовувати та пристосовувати робочі процеси під свої потреби.

Конфігураційні файли робочих процесів можуть бути створені в кореневому каталозі репозиторію або в папці .github/workflows.

Кожен конфігураційніфайл містить список кроків, які необхідно виконати відповідно до визначеної події. Наприклад, при зміні коду в репозиторії може запускатися робочий процес, який виконує тестування та збірку проекту. Структура файлів для конфігурації GitHub Actions зазвичай виглядає так:

- .github/workflows/ - ця папка містить файли з інструкціями для GitHub Actions;
- my-workflow.yml - це файл з інструкціями для GitHub Actions, які будуть виконуватися автоматично;
- my-script.sh - це файл зі скриптом, який буде виконуватися автоматично в межах інструкції;
- Dockerfile - це файл з інструкціями для створення Docker-контейнера, які будуть виконуватися автоматично.

Файли з інструкціями для GitHub Actions мають формат YAML і містять наступні ключові елементи:

- name - назва вашої інструкції;
- on - події, під час яких ваша інструкція буде виконуватися;

- jobs - список робочих процесів, які повинні бути виконані під час роботи вашої інструкції;
- steps - список кроків, які повинні бути виконані в межах кожного робочого процесу;
- env: список змінних середовища, які використовуються в робочому процес;
- defaults: значення за замовчуванням для кроків та завдань.

Крім того, конфігураційні файли можуть містити коментарі, які починаються з символу #.

Сценарії дій. GitHub Actions дозволяє визначати робочі процеси за допомогою сценаріїв дій. Сценарій дій - це послідовність кроків, які потрібно виконати в рамках робочого процесу. Кожен крок може містити команди або використовувати дії, що надаються в стандартній бібліотеці дій.

Основні типи дій, які можна використовувати в сценарії дій, включають:

- actions/checkout: завантаження репозиторію в робоче середовище;
- actions/setup-<language>: налаштування середовища для різних мов програмування, таких як Node.js, Python, Ruby тощо;
- actions/cache: кешування залежностей та інших файлів для прискорення часу виконання;
- actions/upload-artifact: завантаження артефактів, створених під час виконання робочого процесу, до GitHub;
- actions/notify: надсилання повідомлень на Slack, Discord, електронну пошту тощо.

Крім того, GitHub Actions дозволяє використовувати власні дії, які можуть бути створені власником репозиторію або використані зі сторонніх джерел. Це дозволяє значно розширити можливості робочих процесів та забезпечити більшу автоматизацію.

Огляд різних видів конфігураційних файлів: workflows, actions, runners, environment, secrets. GitHub Actions має кілька видів конфігураційних файлів, що

визначають як саму систему, так і окремі дії відносно неї. Ось огляд різних видів конфігураційних файлів:

- Workflows: це головний файл конфігурації для GitHub Actions, який описує послідовність дій, які потрібно виконати при виконанні певної операції. В цьому файлі визначається, які дії потрібно виконати, коли потрібно запустити цей процес, які параметри передаються в кожен крок тощо;
- Actions: це окремі файли конфігурації, які визначають кожну окрему дію, яку можна використовувати в більш складних процесах. Наприклад, можна створити дію для розгортання додатку на хмарному сервісі, і використовувати її в своєму основному файлі workflow;
- Runners: це конфігураційні файли для налаштування серверів, які використовуються для виконання коду. Ці сервери можна налаштувати з різними характеристиками, такими як оперативна пам'ять та кількість процесорів, щоб забезпечити оптимальну продуктивність під час виконання тестів;
- Environment: це файли конфігурації, які визначають змінні середовища, які використовуються в процесі розгортання додатку. Наприклад, можна налаштувати змінну середовища, яка містить ключ API для зовнішнього сервісу;
- Secrets: це файл містить конфіденційні дані, які можуть бути використані в конфігураційному файлі workflow. Це може бути, наприклад, ключ доступу до бази даних або API ключ до зовнішнього сервісу. GitHub забезпечує безпечне зберігання секретів, тому їх можна зберігати в захищеному репозиторії та використовувати в конфігураційних файлах workflow;
- Dockerfile: це файл, який містить інструкції для створення Docker-контейнерів, які можуть використовуватись для виконання GitHub Actions. Docker-контейнери забезпечують стандартизоване середовище виконання, що дозволяє забезпечити консистентність виконання дій між різними репозиторіями та уникнути проблем залежності.

Розгляд різних варіантів параметрів налаштування, таких як cron-розклад, тригери подій, матриці параметрів. GitHub Actions надає можливість

налаштування параметрів для виконання workflows з використанням різних конфігураційних файлів. Деякі з варіантів параметрів налаштування:

- Cron-розклад: можна налаштувати workflow на виконання через певний часовий інтервал за допомогою cron-розкладу. Наприклад, виконати певні дії щодня о певному часі;
- Тригери подій: GitHub Actions можна запустити на основі подій, таких як push, pull_request, issues, release, и т.д. Тригери подій можна настроїти в конфігураційному файлі workflow;
- Матриці параметрів: можна налаштувати параметри, які використовуються в різних варіантах виконання workflows. Наприклад, параметри для різних версій ОС або браузерів, для різних варіантів конфігурації середовища виконання тощо;
- Секрети: можна налаштувати конфіденційні дані, такі як ключі API, паролі або інші токени, і зберігати їх в секретах. Секрети доступні з конфігураційних файлів через змінні оточення;
- Матриці мов програмування: можна налаштувати підтримувані мови програмування та їх версії для виконання tests, builds, deployment та інших операцій;
- Інші параметри: в конфігураційних файлах workflow можна налаштувати різні параметри, такі як назву workflow, підтримувану операційну систему, налаштування кешування, налаштування оточення та інші.

Параметри налаштування можуть бути поєднані між собою для створення більш складних і налаштовуваних workflows.

1.3 Структура робочого процесу в GitHub Actions

Запуск та виконання робочого процесу. Робочий процес GitHub Actions можна запустити вручну або автоматично, в залежності від вибраної стратегії подій.

У випадку автоматичного запуску, робочий процес може бути пов'язаний з певними подіями в GitHub, такими як Push коду в репозиторій, створення нового Pull Request або створення нового релізу. В такому випадку, робочий процес буде запущений автоматично при виникненні відповідних подій.

Якщо ж запуск робочого процесу не пов'язаний з певними подіями, його можна запустити вручну зі сторінки GitHub Actions у репозиторії.

Після запуску робочого процесу, GitHub створює відповідне середовище, де буде виконуватися робочий процес. Далі, GitHub запускає послідовність кроків, які визначені в конфігураційному файлі робочого процесу. Кожен крок може мати свій статус (успішно виконаний, виконання завершилося з помилкою або було пропущено) та відповідні журнали виконання.

Після завершення робочого процесу, можна переглянути журнали виконання кожного кроку, а також статус всього робочого процесу. Також, можна виконати налаштування для автоматичної обробки виключних ситуацій або відправлення повідомлень з результатами виконання робочого процесу.

GitHub Actions надає широкі можливості для контролю, налагодження та покращення робочих процесів, що дозволяє значно зменшити час, необхідний для розробки та випуску програмного продукту.

Підготовка середовища виконання. GitHub Actions запускає робочі процеси віртуальних машин (virtual machines), які називаються "середовища виконання" (runners). Ці середовища містять усі необхідні залежності та програмне забезпечення для виконання робочого процесу.

Середовища виконання можуть бути налаштовані різними способами залежно від потреб проекту. GitHub надає кілька варіантів середовищ виконання, які можна використовувати для запуску робочих процесів, включаючи:

- Ubuntu: це стандартне середовище виконання для більшості проектів на GitHub. Воно містить усі необхідні залежності та програмне забезпечення для виконання більшості задач, включаючи збірку та тестування;
- Windows: це середовище виконання для проектів, що працюють під управлінням операційної системи Windows;

- MacOS: це середовище виконання для проектів, що працюють під управлінням операційної системи macOS;
- Docker: це середовище виконання, яке використовує Docker-контейнери. Це дозволяє використовувати спеціалізовані контейнери з певними залежностями та програмним забезпеченням для виконання робочих процесів.

Для налаштування середовищ виконання, можна використовувати спеціальний файл конфігурації, який називається "workflow file". У цьому файлі можна вказати, яке середовище виконання слід використовувати для робочого процесу та які залежності та програмне забезпечення потрібно встановити.

Виконання дій в рамках робочого процесу. Для виконання дій в рамках робочого процесу потрібно визначити список кроків, які потрібно виконати. Кроки можуть бути виконані на віртуальному середовищі, яке визначене в конфігураційному файлі робочого процесу.

Кроки виконуються послідовно, тобто наступний крок не буде виконаний, поки попередній не завершиться. Крок може бути виконаний на тому ж віртуальному середовищі, на якому запущений робочий процес, або на іншому середовищі, якщо воно визначене в конфігураційному файлі.

Перевірка результатів виконання. GitHub Actions дозволяє перевіряти результати виконання робочих процесів за допомогою різноманітних засобів, що допомагає забезпечити якість коду та швидко виявляти можливі проблеми.

Один з основних засобів перевірки результатів виконання GitHub Actions - це створення логів виконання. Кожен робочий процес, який виконується в GitHub Actions, має свій журнал виконання, де зберігається детальна інформація про процес виконання, включаючи кроки, які були виконані, результати їх виконання та помилки, які виникли під час виконання кроків. Приклад сторінки з результатами виконання робочого процесу зображено на рис. 1.1.

Рисунок 1.1 – Сторінка з результатами виконання

Крім того, GitHub Actions дозволяє створювати спеціальні тести, які автоматично перевіряють результати виконання робочого процесу на відповідність заданим очікуванням. Ці тести можуть бути включені в робочий процес та запускатися автоматично при кожному виконанні процесу.

Також, GitHub Actions підтримує різноманітні інструменти для аналізу коду, такі як лінтери, тестувальники та інші. Вони дозволяють виявляти можливі проблеми у коді та попереджати про можливі помилки ще до того, як вони будуть відображені під час виконання робочого процесу.

Загалом, перевірка результатів виконання GitHub Actions є потужним засобом для забезпечення якості коду та швидкого виявлення можливих проблем. Вона дозволяє забезпечити безпеку, якість та надійність програмного забезпечення та зменшити час на виявлення та виправлення проблем.

1.4 Приклади використання GitHub Actions для автоматизації розробки ПЗ

Автоматичне тестування. Одним з найбільш поширених використань GitHub Actions є автоматичне тестування ПЗ під час розробки. GitHub Actions

дозволяє автоматизувати процеси тестування, що дозволяє зменшити час, необхідний для перевірки коду і покращити якість ПЗ. Приклад кроків для автоматичного тестування на мові Python з використанням фреймворку pytest показано на рис. 1.2.

Рисунок 1.2 – Кроки автоматичного тестування

Перевірка стилю коду. Для перевірки стилю коду можна використовувати спеціальні інструменти, такі як RuboCop для Ruby, ESLint для JavaScript або Flake8 для Python. Ці інструменти перевіряють код на відповідність визначеному стилю коду, такому як стиль іменування, відступи, розмірність файлів тощо.

Загальний приклад коду для перевірки стилю коду Python за допомогою GitHub Actions зображено на рис. 1.3.

Рисунок 1.3 – Кроки для перевірки коду

Автоматичний реліз ПЗ. Автоматичний реліз програмного забезпечення можна налаштувати за допомогою GitHub Actions. Для цього потрібно

налаштувати робочий процес, який буде запускатися після того, як відбувається push до основної гілки (зазвичай, до гілки main).

Основним завданням автоматичного релізу є те, щоб забезпечити швидкий та безпечний випуск нових версій ПЗ з мінімальними зусиллями з боку розробників.

Для автоматичного релізу можна використовувати, наприклад, засіб створення релізів, який вбудований у GitHub. Цей засіб дозволяє створити новий реліз та надати йому назву, опис, список змін та іншу інформацію. Для цього використовується REST API GitHub.

Нижче наведений приклад конфігураційного файлу для автоматичного релізу з використанням засобу створення релізів продемонстровано на рис. 1.4.

Рисунок 1.4 – Крок автоматичного створення релізу

Інтеграція з іншими сервісами. GitHub Actions може бути інтегрована з багатьма різними сервісами, щоб забезпечити автоматизовані процеси, які допоможуть полегшити розробку, тестування та розгортання програмного забезпечення. Деякі з популярних сервісів, які можна інтегрувати з GitHub Actions, включають такі:

- Amazon Web Services (AWS): GitHub Actions можна використовувати для автоматизованого розгортання програмного забезпечення на AWS. Для цього необхідно налаштувати з'єднання з AWS та створити скрипти, які виконують необхідні дії;

- Microsoft Azure: GitHub Actions може бути інтегрована з Microsoft Azure, щоб забезпечити автоматизоване розгортання та тестування програмного забезпечення на Azure;
- Docker: GitHub Actions може бути використана для автоматизованого створення, тестування та розгортання Docker-контейнерів;
- Slack: GitHub Actions можна інтегрувати з Slack, щоб сповіщати розробників про результати тестів, нові пул-реквести або інші події, які можуть бути важливі для розробки;
- Jira: GitHub Actions можна інтегрувати з Jira, щоб забезпечити автоматизований процес створення тікетів в Jira, коли відбувається певна подія в GitHub, наприклад, коли розробник створює новий пул-реквест або коміт;
- SonarCloud: GitHub Actions може бути використана для автоматизованої перевірки якості коду за допомогою сервісу SonarCloud;
- Firebase: GitHub Actions можна інтегрувати з Firebase, щоб забезпечити автоматизоване розгортання програмного забезпечення на Firebase.

Інтеграція з цими та іншими сервісами дозволяє забезпечити повністю автоматизовану розробку, тестування та розгортання програмного забезпечення.

1.5 Порівняння GitHub Actions з іншими системами автоматизації розробки ПЗ

GitHub Actions - це одна з найбільш популярних систем автоматизації розробки ПЗ, але на ринку існує багато інших подібних систем. Далі наведено деякі з них:

- **Jenkins** - це одна з найпопулярніших систем автоматизації розробки ПЗ, яка використовується для збірки, тестування та розгортання програмного забезпечення. Jenkins дуже гнучкий та має багато плагінів, що дозволяє налаштувати систему під свої потреби. Однак в порівнянні з GitHub Actions, Jenkins має більше налаштувань та складніший інтерфейс;

- **Travis CI** - це хмарна система автоматизації розробки ПЗ, яка надає можливість збірки, тестування та розгортання програмного забезпечення. Travis CI має дуже простий та зрозумілий інтерфейс та швидке налаштування. Однак, Travis CI є платною системою, тоді як GitHub Actions є безкоштовною для відкритих репозиторіїв та має ширші можливості;
- **GitLab CI/CD** - це інтегрована система для збірки, тестування та розгортання програмного забезпечення в GitLab. GitLab CI/CD має простий та зрозумілий інтерфейс, але вона має менше можливостей в порівнянні з GitHub Actions. Однак, GitLab CI/CD є безкоштовною та легко інтегрується з GitLab.

Переваги GitHub Actions порівняно з іншими системами автоматизації розробки ПЗ:

- **Інтеграція з GitHub**. GitHub Actions створений спеціально для роботи з GitHub і має вбудовану підтримку для нього. Це означає, що ви можете легко налаштувати та використовувати його без необхідності встановлювати та налаштовувати окремий продукт;
- **Легка налаштовуваність**. Простий у використанні синтаксис для налаштування робочих процесів. Ви можете визначити дії у вигляді стандартних або користувацьких сценаріїв, використовуючи YAML-файли. Це дозволяє швидко створювати і змінювати автоматичні робочі процеси без зайвих складнощів;
- **Безкоштовна для відкритих проектів**. Безкоштовна для відкритих проектів, що робить її дуже привабливою для проектів з відкритим вихідним кодом;
- **Надійність та швидкість**. Підтримка безлічі конфігураційних параметрів, таких як матриця параметрів, яка дозволяє розподілити виконання дій між декількома середовищами одночасно;
- **Підтримка Docker-контейнерів**. Вбудована підтримку Docker-контейнерів, що дозволяє використовувати контейнери для запуску ваших тестів та додатків в різних середовищах;
- **Розширення за допомогою сторонніх плагінів**. GitHub Actions підтримує сторонні плагіни, які можна використовувати для розширення

функціональності, що дозволяє розробникам знайти більш гнучкий та персоналізований підхід до автоматизації свого процесу розробки.

2 НАЛАШТУВАННЯ СЕРЕДОВИЩА ВИКОНАННЯ РОБОТИ ТА ВСТАНОВЛЕННЯ ПРОГРАМ ТА ЇХ КОНФІГУРАЦІЯ ДЛЯ GITHUB ACTIONS

2.1 Вступ до робочого середовища Visual Studio Code та GitHub

Visual Studio Code - це безкоштовний та популярний текстовий редактор, розроблений компанією Microsoft. Він використовується для написання та редагування різноманітного програмного коду із зручними інструментами для розробників.

Visual Studio Code має багато функцій, які роблять його популярним серед розробників. Він підтримує багато мов програмування, включаючи популярні, такі як JavaScript, Python, Java, C#, PHP і багато інших. Редактор має розширену систему плагінів, що дозволяє налаштовувати його під потреби кожного розробника.

2.2 Визначення GitHub та його роль у розробці ПЗ

GitHub є веб-платформою для керування версіями програмного забезпечення, яка базується на системі керування версіями Git. Ця платформа надає розробникам зручність зберігання своїх проектів у віддалених репозиторіях, що дозволяє легко співпрацювати над кодом. Завдяки GitHub розробники можуть відстежувати зміни, керувати проблемами та здійснювати такі дії, як push, або pull request, що полегшує процес рецензування та злиття змін.

Використання GitHub сприяє колаборації між розробниками та забезпечує ефективний контроль версій. Він дозволяє розробникам спільно працювати над проектами, дозволяючи кожному учаснику вносити свої зміни та відстежувати розвиток коду. Крім того, GitHub є важливим інструментом для командної розробки та відкритих проектів, де кілька розробників можуть працювати над одним проектом, координувати свої зусилля та зберігати історію змін.

GitHub сприяє співпраці між розробниками, полегшує рецензування коду, дозволяє контролювати версії та забезпечує зручну платформу для розробки програмного забезпечення. Він став незамінним інструментом для розробників у сучасному світі програмування.

2.3 Важливість налаштування робочого середовища для ефективної роботи з GitHub

Налаштування робочого середовища, такого як Visual Studio Code, є важливим етапом для досягнення ефективної роботи з GitHub. Правильні налаштування дозволяють зручно використовувати всі потужні інструменти керування версіями та спільної роботи, які надає GitHub.

Налаштування може включати встановлення та налаштування розширень для Visual Studio Code, які сприяють безпроблемному зберіганню коду у репозиторії GitHub. Ці розширення надають зручний доступ до функцій відстеження змін, синхронізації змін між локальним середовищем та GitHub, а також злиття гілок. Додатково, налаштування робочого середовища може включати інструменти для спілкування з іншими розробниками, такі як коментарі та перегляд коду.

Використання належно налаштованого робочого середовища спрощує роботу з GitHub, допомагає зосередитися на розробці коду та покращує продуктивність розробника. Налаштоване середовище забезпечує зручний та ефективний процес роботи з GitHub, дозволяючи розробникам максимально використовувати його потенціал для спільної роботи та керування версіями своїх проектів.

2.4 Можливості, які є у доступі при використанні функцій Visual Studio Code

Редактор коду: Visual Studio Code надає розширений редактор коду з численними корисними можливостями. Синтаксис вашого коду підсвічується, що сприяє кращому розумінню структури програми та виявленню помилок. Крім того, доступним для використання є автодоповнення для швидкого дописування коду, яку включає в себе імена функцій, методів, змінних та бібліотек. Це спрощує процес написання коду та зменшує кількість можливих помилок.

Керування версіями: Visual Studio Code має вбудовану підтримку систем керування версіями, зокрема Git. Ви можете створювати коміти, що дозволяє фіксувати зміни у коді та відстежувати його історію. Також доступна можливість створювати нові гілки, що дозволяє розробляти функціональність незалежно одна від одної та об'єднувати їх за потреби. Злиття (мердж) гілок дозволяє поєднати зміни з різних гілок в одну. Загальнодоступним є і перегляд та розгляд попередніх комітів для аналізу змін.

Використання вбудованих інструментів: Visual Studio Code надає вбудовані інструменти для перегляду, порівняння та вирішення конфліктів. Це охоплює перегляд різниці між версіями файлу, порівняння різних гілок та вирішування конфліктів під час злиття. Ці інструменти допомагають зберегти консистентність коду та забезпечують ефективну роботу з GitHub.

2.5 Конфігурація та налаштування Visual Studio Code для роботи з GitHub

Налаштування редактора коду: У Visual Studio Code можна налаштувати різні аспекти редактора, щоб забезпечити комфортну та зручну роботу. Різноманітні налаштування, такі як шрифти, розміри вікон та вкладок, кольорові схеми та теми оформлення, можуть бути змінені відповідно до вподобання. Крім того, Visual Studio Code дозволяє встановлювати розширення, які додають додаткові функціональні можливості, наприклад, підтримку конкретних мов програмування або інструментів для роботи з GitHub.

Конфігурація інтегрованих терміналів: Visual Studio Code надає можливість взаємодії з Git та GitHub через інтегровані термінали. Термінал може бути налаштований для використання певної версії Git або вказання шляху до виконуваного файлу Git на вашій системі. Це дозволяє зручно використовувати команди Git та GitHub безпосередньо з Visual Studio Code.

Налаштування сповіщень та підказок: Visual Studio Code може надсилати сповіщення та підказки, пов'язані з репозиторіями на GitHub. Наприклад, можливо налаштувати сповіщення про нові коміти або злиття, що стосуються проектів. Також можна налаштувати підказки, які допоможуть знайти та виправити можливі проблеми з комітами або гілками. Це сприяє підтримці в курсі змін у репозиторії та забезпечує ефективну роботу з GitHub.

2.6 Встановлення Visual Studio Code та GitHub

За допомогою VS Code, розробники можуть швидко та ефективно працювати з GitHub Actions. Вони можуть створювати, редагувати, налагоджувати та керувати своїми робочими процесами, а також бачити відповідну інформацію про стан процесів прямо у своєму редакторі. Це спрощує розробку, тестування та розгортання програмного коду з використанням GitHub Actions.

Visual Studio Code можна завантажити з офіційного веб-сайту code.visualstudio.com. У розділі "Завантажити" або "Download" будуть доступні версії Visual Studio Code для різних операційних систем, таких як Windows, macOS та Linux.

Після успішного завершення встановлення, з’явиться повідомлення про його успішність, яке зображене на рис. 2.1.

Рисунок 2.1 – Повідомлення про успішне встановлення

Для роботи з GitHub також потрібно встановити Git. Git є розподіленою системою керування версіями, що використовується для відстеження змін у кодовій базі, спільної роботи над проектами та керування різними версіями проекту. Git дозволяє розробникам зберігати, відновлювати та об'єднувати зміни, а також працювати з різними гілками розробки.

Git є основою для GitHub Actions, які забезпечують автоматизацію різних робочих процесів в проектах, що зберігаються на GitHub. Для роботи з GitHub Actions необхідно мати Git встановлений на комп'ютері. Git дозволяє зберігати проект у репозиторії на GitHub та взаємодіяти з ним через командний рядок або інтегроване середовище розробки, таке як Visual Studio Code.

Git потрібно завантажити з офіційного веб-сайту git-scm.com. Після відкриття веб-сайту Git, потрібно знайти секцію "Завантажити" або "Download". Там будуть показані доступні версії Git для різних операційних систем, таких як Windows, macOS або Linux. Сторінку завантаження зображено на рис. 2.2.

Рисунок 2.2 – Сторінка завантаження Git

Після завантаження інсталяційного файлу Git, потрібно запустити процес інсталяції. Після завершення інсталяції з’явиться повідомлення про успішне встановлення, яке зображене на рис. 2.3.

Рисунок 2.3 – Повідомлення про успішне встановлення Git

Після завершення інсталяції Git повинен бути встановлений на комп'ютері. Для перевірки, потрібно відкрити командний рядок та ввести команду "**git --version**". Якщо виведеться повідомлення з номером версії Git, це означає, що встановлення пройшло успішно. Тепер буде доступна команда "git" у командному

рядку, яку можна використовувати для керування версіями коду та спілкування з віддаленими репозиторіями.

2.7 Налаштування імені користувача та електронної пошти в Git

Для того, щоб мати змогу керувати репозиторієм з свого комп'ютера, потрібно налаштувати файл конфігурації вписавши туди ім'я користувача та електрону пошту.

Файл конфігурації Git називається .gitconfig і зберігає основні налаштування Git. Він може бути розташований на рівні кореневої директорії вашого користувача (глобальний .gitconfig) або в кожному окремому репозиторії (локальний .git/config).

Налаштувати своє ім'я користувача та електрону пошту можна командою **git config**, приклад показано на рисунку 2.4.

Рисунок 2.4 – Приклад команд для налаштування .gitconfig

2.8 Інтеграція Visual Studio Code з GitHub

Переваги використання Visual Studio Code для роботи з GitHub є незаперечними. Перш за все, Visual Studio Code має глибоку інтеграцію з GitHub, що робить його потужним інструментом для взаємодії з репозиторіями на цій платформі.

Завдяки Visual Studio Code можна зручно та ефективно працювати з репозиторіями на GitHub. Він надає можливості легко клонувати репозиторій, створювати нові файли та папки, редагувати існуючі файли та навіть видаляти їх -

все це безпосередньо з інтерфейсу редактора. Такий зручний доступ до GitHub-репозиторію дозволяє швидко вносити зміни та керувати проєктами.

Крім того, Visual Studio Code надає зручний інтерфейс для керування гілками, комітами та іншими функціями GitHub. Також є можливість легко переключатись між гілками, створювати нові коміти та відправляти їх на GitHub, а також використовувати функцію pull-request для спільної роботи над кодом з іншими розробниками. Всі ці можливості роблять процес роботи з GitHub у VS Code простим та зручним.

2.9 Огляд доступних розширень та плагінів для спільної роботи з GitHub у Visual Studio Code.

Огляд доступних розширень та плагінів для спільної роботи з GitHub у Visual Studio Code є важливим кроком у покращенні досвіду роботи з цією платформою. У Visual Studio Code існує багато розширень та плагінів, які полегшують взаємодію з GitHub та роблять вашу роботу більш продуктивною.

Одним з популярних розширень є "GitHub Pull Requests and Issues", яке надає зручний інтерфейс для перегляду, коментування та проблемами, що виникають у проекті. Також можна легко відстежувати та взаємодіяти з pull-request безпосередньо з Visual Studio Code.

Інше корисне розширення - "GitLens". Воно надає розширену інформацію про коміти, гілки та авторів прямо в редакторі коду. З допомогою "GitLens" можна швидко переглядати історію змін, визначати, хто вніс конкретні зміни та з якої гілки вони походять.

"GitHub Actions" - це розширення, яке дозволяє автоматизувати робочий процес з використанням GitHub Actions. Воно дозволяє налаштувати автоматичні дії, які відбуваються при певних подіях, таких як злиття pull-request або створення нової гілки.

Крім цих розширень, також існує "GitHub Repositories", яке дозволяє швидко переглядати та відкривати репозиторії на GitHub безпосередньо з Visual Studio Code.

Ці розширення, а також багато інших доступних в екосистемі Visual Studio Code, надають додаткові функціональні можливості для роботи з GitHub. Вони полегшують перегляд, коментування та керування пул-реквестами, використання статусів злиття, автоматизують певні дії та забезпечують інтеграцію з GitHub API. Обираючи та налаштовуючи ці розширення відповідно до потреб, ви можете значно полегшити роботу з GitHub у Visual Studio Code

2.10 Інструкції щодо встановлення та налаштування інтеграції між Visual Studio Code та GitHub

Перед використанням інтеграції GitHub у Visual Studio Code, необхідно встановити вибрані розширення для роботи з GitHub. Для цього потрібно відкрити розділ "Extensions" у бічній панелі VS Code та знайти розширення, використовуючи поле пошуку. Після знаходження обрати потрібне розширення зі списку і натиснути кнопку "Install" для його встановлення.

Після встановлення розширення необхідно налаштувати облікові дані GitHub у Visual Studio Code. Це дозволить авторизуватись у обліковому записі GitHub та мати доступ до репозиторіїв та функцій. Для цього можна ввести свої облікові дані GitHub у налаштуваннях Visual Studio Code або скористатися іншими методами авторизації, які надаються розширеннями.

Після успішної авторизації можна почати використовувати встановлені розширення та інтегровані функції для роботи з потрібними репозиторіями на GitHub. Завдяки цим розширенням можна зручно клонувати, створювати, редагувати та видаляти файли та папки у вашому репозиторії безпосередньо з Visual Studio Code. Крім того, вони дають змогу керувати гілками, комітами, пул-реквестами та іншими функціями GitHub.

Для підключення до облікового запису GitHub у Visual Studio Code необхідно виконати наступні кроки:

- Запустити Visual Studio Code;
- У верхньому меню потрібно вибрати "View" (Вигляд), а зі списку оберати "Extensions" (Розширення), щоб відкрити панель розширень;
- В полі пошуку потрібно ввести "GitHub" і знайти розширення під назвою "GitHub Pull Requests and Issues";
- Щоб встановити це розширення, яке дозволить працювати з GitHub у Visual Studio Code потрібно натиснути на кнопку "Install" (Встановити). Приклад сторінки розширення зображено на рис. 2.5;

Рисунок 2.5. – Сторінка завантаження розширення

- Після успішної установки з'явиться нова вкладка "Source Control" (Керування версіями) у бічній панелі Visual Studio Code;
- Далі потрібно увійти до GitHub, натиснувши на вкладку "Sign in to GitHub" (Увійти в GitHub), і ввести свої облікові дані;
- Після успішного входу з’явиться можливість бачити свої репозиторії, працювати з гілками, робити коміти та виконувати інші операції з GitHub безпосередньо у Visual Studio Code.

Варто звернути увагу, що цей підхід використовує автентифікацію на основі токенів GitHub, а не SSH.Тому не потрібно налаштовувати SSH-ключі для з'єднання між VS Code та GitHub, проте потрібно надати токен доступу GitHub під час підключення.

Після підключення до облікового запису GitHub у Visual Studio Code буде можливість створювати нові репозиторії на GitHub або клонувати існуючі репозиторії на свій комп'ютер.

2.11 Управління репозиторіями в Visual Studio Code

Для створення нового репозиторію у Visual Studio Code потрібно виконати наступні кроки:

- У верхньому меню Visual Studio Code оберіть "View" (Вид) > "GitHub Repositories" (Репозиторії GitHub);
- У панелі ліворуч оберіть "Clone Repository" (Клонувати репозиторій);
- Введіть URL репозиторію або виберіть репозиторій зі списку доступних;
- Оберіть локальний каталог, де потрібно зберегти репозиторій;
- Клікніть на "Clone" (Клонувати) для створення локальної копії репозиторію.

Для клонування існуючого репозиторію наступні кроки виконуються:

- Відкрийте Visual Studio Code та оберіть "View" (Вид) > "GitHub Repositories" (Репозиторії GitHub);
- У панелі ліворуч оберіть "Clone Repository" (Клонувати репозиторій);
- Введіть URL репозиторію або виберіть його зі списку доступних;
- Виберіть локальний каталог для збереження клонованого репозиторію;
- Клікніть на "Clone" (Клонувати) для клонування репозиторію на ваш комп'ютер.

Для створення нової гілки такі дії виконуються:

- Відкрийте репозиторій в Visual Studio Code;
- У панелі стану знизу оберіть поточну гілку;

- У випадаючому меню оберіть "Create new branch" (Створити нову гілку).
- Введіть назву нової гілки та натисніть "Enter" (Ентер);
- Нова гілка буде створена та ви будете переміщені на неї.

Для створення коміту такі кроки виконуються:

- Внесіть потрібні зміни в файли вашого проекту;
- У бічній панелі змін оберіть файли, які потрібно додати до коміту;
- Введіть опис змін у поле "Message" (Повідомлення) у верхній частині панелі змін;
- Натисніть на кнопку "Commit" (Зафіксувати) для збереження коміту.

Для злиття змін виконуються такі дії:

- Переключіться на головну гілку (наприклад, "main" або "master");
- У Visual Studio Code оберіть "View" (Вид) > "GitHub Repositories" (Репозиторії GitHub);
- У панелі ліворуч оберіть "Pull Requests" (Запити на злиття);
- Оберіть потрібний запит на злиття та натисніть "Merge" (Злити);
- Виберіть опції злиття та підтвердьте злиття змін.

Синхронізація змін між локальним середовищем Visual Studio Code та репозиторієм на GitHub включає такі дії:

- У Visual Studio Code оберіть "View" (Вид) > "GitHub Repositories" (Репозиторії GitHub);
- У панелі ліворуч оберіть "Pull" (Витягнути);
- Виберіть репозиторій, з якого ви хочете оновити зміни;
- Visual Studio Code оновить локальну копію репозиторію, синхронізуючи зміни.

2.12 Основи роботи з GitHub у Visual Studio Code

Ефективне використання Git-команд: Ознайомлення з основними командами Git, такими як git clone, git add, git commit, git push та git pull, рекомендується. Це дозволить контролювати версію коду, створювати коміти зі змінами та

синхронізувати роботу з GitHub. Знання цих команд допоможе забезпечити більш ефективну роботу з репозиторіями.

Оптимальна організація репозиторіїв: Використання гілок для розвитку функціональності та виправлення помилок рекомендується. Це дозволить розділити різні функції або завдання на окремі гілки, спрощуючи спільну роботу з іншими розробниками та управління змінами. Крім того, слід використовувати правильні назви гілок, що відображають їх мету або функціональність, наприклад, "feature/new-feature" або "bugfix/issue-123".

Забезпечення безпеки та конфіденційності: Важливо дотримуватись найкращих практик щодо безпеки та конфіденційності при роботі з GitHub. Рекомендується використовувати HTTPS або SSH для з'єднання з репозиторіями на GitHub та забезпечувати належний захист облікових записів та репозиторіїв. Додатково, слід уникати додавання конфіденційної інформації, такої як паролі чи ключі доступу, до публічних репозиторіїв. Для цього рекомендується використовувати файли .gitignore, щоб виключити таку інформацію з контролю версій.

2.13 Створення репозиторію

Першим і важливим кроком у початку роботи з GitHub Actions є створення репозиторію та організація структури проекту. Цей крок визначає основну інфраструктуру, на якій будуть будуватися і виконуватися дії в межах GitHub Actions. Для цього було необхідно зареєструвати обліковий запис на сайті GitHub, якщо такого ще не було, та створити репозиторій, який буде використовуватися для управління та збереження кодової бази проекту.

При створенні репозиторію варто врахувати кілька аспектів. По-перше, необхідно обрати адекватне і зрозуміле ім'я репозиторію, яке відображатиме його призначення і зміст. Також рекомендується додати короткий, але змістовний опис, що допоможе іншим користувачам зрозуміти, що саме міститься в репозиторії.

Крім того, важливо встановити належні права доступу до репозиторію. Залежно від призначення проекту та команди розробників, можна обрати публічний репозиторій, який буде доступний для всіх користувачів, або приватний, що забезпечить обмежений доступ тільки для обраної команди.

У процесі створення репозиторію, було організовано структуру проекту. Схематичне зображення структури репозиторію наведено на рис. 2.6.

Рисунок 2.6 – Схема структури репозиторію

Це допоможе систематизувати кодову базу та зробити її більш логічною та зрозумілою для розробників. Наступним кроком було створено репозиторій та завантажено необхідні папки та файли. Вигляд нового репозиторію зображено на рис. 2.7.

Рисунок 2.7 – Вигляд нового репозиторію

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ АВТОМАТИЗАЦІЇ РОЗРОБКИ З ВИКОРИСТАННЯМ GITHUB ACTIONS

3.1 Автоматичне тестування програмного забезпечення

Автоматичне тестування програмного забезпечення пропонує численні переваги в процесі розробки програмного забезпечення. Завдяки автоматизації повторюваних завдань значно підвищується ефективність циклу розробки. Це дозволяє швидше перевіряти програмні компоненти і виявляти проблеми на ранніх стадіях, що призводить до створення більш міцного і надійного кінцевого продукту. Крім того, початкові інвестиції в автоматизацію тестування, хоча і потенційно вищі, та в довгостроковій перспективі приносять значну віддачу, оскільки скорочують час і ресурси, необхідні для ручного тестування.

Було розроблено блок-схему для автоматичного тестування програмного коду. Ця блок-схема дозволяє автоматизувати процес тестування, спрощуючи виявлення помилок та підтримку якості проекту. Блок-схема зображена на рис. 3.1;

Рисунок 3.1 – Блок-схема автоматизації тестів

Коли в репозиторій здійснюється дія push у будь-яку гілку, за винятком гілки "main", запускається відповідна послідовність кроків для тестування. Першим кроком є копіювання репозиторію з програмним кодом до середовища виконання тестів. Наступними кроками є завантаження мови програмування Python, встановлення необхідних бібліотек, налаштування віртуального монітору, у разі виконання робочого процесу на системі Ubuntu, або пропуск налаштування, якщо система – Windows, тоді слідує запуск тестів.

У репозиторії в теці workflow було створено файл CI.yml, у якому зберігаються інструкції для автоматичного тестування.

Тригери для цього сценарію визначені під ключем on. Процес налаштовано на запуск на подію push, але з обмеженням на гілки. Ключ branches-ignore вказує на те, що сценарій не повинен запускатися, коли виконується подія push до головної гілки. Іншими словами, сценарій буде запущено, коли відбудеться подія push у будь-якій гілці, крім головної. Код налаштованого тригеру:

```
on:
  push:
    branches-ignore:
      - main
```

Після створення тригерів було створено блок з кроками. В цьому сценарії визначені кроки під ключем jobs, зокрема, в межах завдання test.

```
jobs:
  test:
```

Також налаштовано матрицю запуску, яка дозволяє виконувати процеси на різних операційних системах. Цей фрагмент коду визначає налаштування для матриці запуску, яка дозволяє виконувати процеси на різних операційних системах:

```
runs-on: ${{ matrix.os }}
strategy:
  matrix:
```

```
os: [ubuntu-latest, windows-latest]
```

Блок з крокам визначений ключем *steps*, та виконуються послідовно і включають наступні кроки:

- Копіювання сховища: На цьому кроці використовується дія actions/checkout@v3 для клонування сховища. Приклад коду цього кроку:

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

- Встановлення Python 3.11: На цьому кроці використовується дія actions/setup-python@v4 для встановлення Python 3.11 у середовищі виконання. Код цього кроку:

```
- name: Set up Python 3.11
  uses: actions/setup-python@v4
  with:
    python-version: 3.11
```

- Встановлення залежностей: Цей крок встановлює необхідні бібліотеки за допомогою pip та файлу requirements.txt. Далі показано створений код для даного кроку:

```
- name: Install dependencies
  run: |
    python -m pip install --upgrade pip
    pip install -r requirements.txt
```

- Налаштування Xvfb та змінної display (Ubuntu): На цьому кроці налаштовується віртуальний монітор (Xvfb) і змінна DISPLAY для запуску тестів на віртуальному моніторі у системі Ubuntu. Для цього використовується умовний оператор if, який перевіряє чи це середовище виконання Ubuntu. Далі наведено приклад коду:

```
- name: Set up Xvfb and display variable (Ubuntu)
  if: matrix.os == 'ubuntu-latest'
  run: |
    sudo apt-get updsate
```

sudo apt-get install xvfb

sudo Xvfb :99 -ac &

export DISPLAY=:99

- Налаштування Xvfb і змінної display (Windows): На цьому кроці налаштування Xvfb для Windows пропускається, оскільки воно не є обов'язковим. Для цього використовується умовний оператор if, який перевіряє чи це середовище виконання windows. Код кроку наведено далі:

- name: Set up Xvfb and display variable (Windows)

if: matrix.os == 'windows-latest'

run: echo "Skipping Xvfb setup for Windows"

- Запуск тестів за допомогою pytest: Цей крок запускає тести за допомогою команди pytest. Далі зображено рядки що визначають крок, який запускає тести:

- name: Run tests with pytest

run: pytest

Весь код робочого процесу, який відповідає за тестування програмного забезпечення наведено у додатку А.

Розгалуження використовується у цьому робочому процесі для виконання певних кроків на основі операційної системи виконавця. Робочий процес буде виконуватися в обох цих операційних системах паралельно.

Розгалуження також відбувається на кроках, які налаштовують віртуальний монітор (Xvfb) і змінну display. Ключ if використовується для умовного виконання цих кроків на основі значення matrix.os. Наприклад, крок "Налаштування Xvfb і змінної відображення (Ubuntu)" буде виконано, тільки якщо matrix.os дорівнює ubuntu-latest. Аналогічно, крок "Налаштування Xvfb і змінної відображення (Windows)" буде виконано, тільки якщо matrix.os дорівнює windows-latest. Таке розгалуження гарантує, що для кожної операційної системи буде виконано відповідне налаштування.

Перегляд результатів виконання CI тестів для проєкту є важливим етапом в розробці програмного забезпечення. Після налаштування файлу CI.yml та запуску GitHub Actions, буде надана корисна інформація про тестування проєкту.

Один з найважливіших результатів, які будуть спостерігатися, - це статус виконання кожного кроку робочого процесу CI. Якщо всі кроки виконуються успішно, буде помічено, що весь робочий процес пройшов без помилок. Це буде свідчити про те, що код успішно скомпілювався та виконав усі визначені тести.

У випадку, якщо помилка виникне на будь-якому кроці, буде повідомлено на вкладці Actions у репозиторії. Буде надана інформація про помилку, включаючи стек викликів, який допоможе знайти проблему. Це дозволить виправити помилку та запустити робочий процес знову для перевірки виправлення.

Доступ до відповідного журналу виконання буде наданий GitHub Actions. На сторінці журналу виконання можна переглянути вивід тестів та інші корисні повідомлення. Також можна перевірити, чи пройшли тести успішно, та переглянути будь-які повідомлення, що були виведені під час виконання робочого процесу. Звіт про успішне виконання тестів зображено на рис. 3.2.

Рисунок 3.2 – Журнал виконання CI

3.2 Автоматичне нічне тестування

Головною метою тестів є виявлення помилок чи проблем, які можуть виникнути через невідповідність між зміненим кодом та вже наявним кодом проекту. Вони спрямовані на перевірку основних функціональних інтеграційних аспектів, таких як збірка проекту, запуск тестів, перевірка кодових стандартів та інші аспекти, необхідні для забезпечення стабільності проекту при злитті коду. Тести зазвичай виконуються при кожному злитті коду у спільний репозиторій. Вони мають бути швидкими, щоб не гальмувати процес Continuous Integration.

Нічні тести виконуються у відведений час, зазвичай протягом ночі або поза робочими годинами команди розробників.

Основна мета нічних тестів полягає в глибокому тестуванні всіх компонентів програмного забезпечення з використанням різних сценаріїв та широкого спектру тестових даних. Це дозволяє виявляти складніші проблеми, які можуть залишатися невиявленими під час простих тестів, такі як накопичення пам'яті, проблеми з масштабуванням, проблеми зі зв'язками між компонентами та інші.

Нічні тести зазвичай виконуються щодня або на регулярній основі після певного часу. Їх виконання може займати тривалий час, тому вони запускаються у відведений період, коли розробники не працюють, щоб не перешкоджати їхній продуктивності.

Було створено блок-схему задля ілюстрації принципу роботи процесу. Блок-схему зображено на рис. 3.3.

Рисунок 3.3 – Блок-схема автоматизації нічних тестів

Далі було створено файл nightly-tests.yml у дерикторії workflow. Щоб увімкнути автоматичне нічне тестування, робочий процес було налаштовано на запуск за розкладом за допомогою ключа розкладу. Цей ключ визначає час і частоту запуску робочого процесу. Наприклад, сценарій можна налаштувати на запуск щоночі в певний час, наприклад, о 2:00 ночі. Код, який дає назву робочому процесу, запускає його щоночі о 2:00 наведено далі:

```
name: Nightly Tests

on:
  schedule:
    - cron: "0 23 * * *"
```

Ця частина коду визначає запуск робочого процесу за розкладом. За встановленим графіком, сценарій буде запускатися щодня о 23:00 по UTC. За Київським часом це буде 2:00 ночі. Значення "*" використовується для позначення "кожне" або "будь-яке" значення. Таким чином, "*" у всіх полях (хвилини, години, днів, місяців, днів тижня) означає, що задача буде запускатись відповідно до заданого розкладу для всіх можливих значень цих полів. В даному

випадку, задача буде запускатись щодня о 2:00, незалежно від конкретного дня, місяця або дня тижня. Токож код робочого процесу, який відповідає за запуск тестування програмного забезпечення наведено у додатку Б.

Під час поглибленої нічної перевірки, яка автоматично запускається що ночі о 2:00 за допомогою нічних тестів, було отримано цінні результати, які сприяють підвищенню якості проєкту. Ця перевірка забезпечує ретельну перевірку коду та функціональності, щоб виявити можливі проблеми та помилки, які можуть вплинути на виконання програми.

Один із основних аспектів, який робить результати поглибленої нічної перевірки корисними, - це виявлення потенційних проблем, які можуть виникнути під час роботи програми. Це дозволяє забезпечити стабільність та надійність проєкту, виправивши виявлені проблеми перед тим, як вони можуть спричинити серйозні наслідки.

Журнал виконання для нічної перевірки надає повну інформацію про виконання тестів, включаючи деталі кожного кроку та його результати. Це дає змогу оцінити, як успішно пройшли нічні тести, та аналізувати будь-які повідомлення або помилки, які були зареєстровані під час виконання.

Журнал виконання також надає стек викликів, що допомагає знайти кореневу причину виявлених проблем. Це спрощує процес виправлення помилок, оскільки він надає доступ до детальних відомостей про послідовність викликів та контексту, що допомагає локалізувати та усунути проблеми швидше та ефективніше. Сторінку з журналом виконання нічних тестів зображено на рис. 3.4.

Рисунок 3.4 – Журнал виконання нічних тестів

3.3 Автоматичне пакування та створення релізу

Автоматичне пакування та створення релізу у GitHub Actions - це процес, який дозволяє автоматизувати пакування програмного коду та створення релізу для проекту на GitHub. Використання GitHub Actions дозволяє забезпечити безпечний та ефективний процес пакування та розгортання програмного забезпечення.

Автоматичне пакування включає у себе збирання коду, його компіляцію, встановлення залежностей та будь-які інші необхідні кроки для створення готового до розгортання пакету. Це може включати в себе використання інструментів збирання, таких як PyInstaller для Python-проектів, або будь-яких інших інструментів, які потрібні для пакування коду в виконуваний файл або інший формат.

Створення релізу означає створення мітки або тегу для певної версії програмного забезпечення та публікацію цього релізу на GitHub. Це може включати в себе завантаження збираного пакету або будь-яких інших артефактів, пов'язаних з вашим релізом. Створення релізу дозволяє користувачам легко

отримати доступ до конкретних версій програмного забезпечення та опису змін, пов'язаних з кожною версією.

Для схематичного зображення процесу створення релізу, було створено блок-схему, яка ілюструє загальні кроки при виконанні робочого процесу. Блок-схему зображено на рис. 3.5.

Рисунок 3.5 – Блок-схема автоматизації створення релізу

Для автоматичного пакування та публікації релізу було створено файл release.yml у дерикторії workflow. Тригери було налаштовано таким чином, щоб робочий процес запускався при події push, або при події push з тегом. Код який дає назву робочому процесу, та визначає тригери при яких буде запускати робочий процес наведено далі:

name: Build and Publish Release

```
on:
  push:
    branches:
      - main
    tags:
      - "v*"
```

GitHub Actions дозволяє налаштувати права доступу для робочого процесу під час виконання, забезпечуючи йому відповідні права для взаємодії з вмістом репозиторію. Приклад коду, який надає потрібні права для створення релізу наведено далі:

```
permissions:
  contents: write
```

Конкретно в цьому випадку, вказано contents: write, що означає, що сценарій має право запису до вмісту репозиторію. Це корисно, коли сценарій потребує доступу до файлів для виконання певних операцій, таких як збирання, зміни або створення нових файлів.

Далі було створено код, що описує кроки, які виконуються під час створення релізу в GitHub Actions на операційній системі Windows:

- **runs-on: windows-latest:** Цей параметр визначає, що робоче середовище для виконання цього задання є операційною системою Windows. Використовується остання доступна версія Windows;
- **steps:** Це секція, де визначаються окремі кроки, які виконуються в рамках цього задання;
- **Checkout repository:** Цей крок використовує дію actions/checkout@v3 для скопіювання репозиторію, щоб його можна було використовувати в подальших кроках;
- **Set up Python 3.11:** Цей крок використовує дію actions/setup-python@v2 для налаштування середовища Python версії 3.11. Це дозволяє використовувати саме цю версію Python для виконання наступних кроків;

- **Install dependencies:** У цьому кроці виконується встановлення необхідних залежностей, вказаних у файлі requirements.txt, за допомогою команд pip install --upgrade pip та pip install -r requirements.txt;
- **Build package:** Цей крок виконує створення білда (компіляцію) програмного пакету. Використовується інструмент PyInstaller для створення виконуваного файлу з коду програми csc.py. Деякі додаткові параметри, такі як --noconfirm, --onefile, --windowed та --icon, вказують специфічні опції компіляції. Код кроку зображено:

```
- name: Build package
  run: pyinstaller --noconfirm --onefile --windowed --icon "pictures/icon.ico"
                   --add-data
                   "C:\hostedtoolcache\windows\Python\3.11.3\x64\Lib\site-
                   packages\customtkinter;customtkinter"  "csc.py"
```

- **Release:** Цей крок використовує дію softprops/action-gh-release@v1 для створення релізу. Він виконується лише у випадку, якщо подія push включає тег. У цьому кроці також вказується, які файли (dist/*) повинні бути включені до релізу. Код цього кроку показано далі:

```
- name: Release
  uses: softprops/action-gh-release@v1
  if: startsWith(github.ref, 'refs/tags/')
  with:
    files: dist/*
```

Весь код робочого процесу, який відповідає за пакування та створення релізу програмного забезпечення наведено у додатку В.

Цей робочий процес GitHub Actions було налаштовано таким чином, щоб він автоматично виконувався при кожному коміті до основної гілки або при створенні нового тегу. Таке налаштування значно полегшує процес пакування та релізу програмного забезпечення, оскільки не потрібно вручну виконувати ці кроки кожного разу.

Щоб запустити цей робочий процес, необхідно створити новий тег. Для цього можна було використано команду ***git tag -a v1.0.0 -m "v1.0.0"***, яка створює

новий тег з відповідним ім'ям та повідомленням про версію. Після створення тегу, його було відправлено до віддаленого репозиторію за допомогою команди ***git push origin v1.0.0***.

Результати виконання цього робочого процесу, який здійснює збірку та публікацію релізу, є надзвичайно важливими для автоматизації процесу випуску програмного забезпечення. Після налаштування цього робочого процесу та виконання пушу з новим тегом, ми отримуємо доступ до важливої інформації, пов'язаної з роботою з релізами. Це може включати звіти про збірку, повідомлення про успіх або невдачу, а також статуси публікації релізів.

Одним з головних результатів виконання цього робочого процесу є створення нового релізу в репозиторії на GitHub. Цей реліз містить зібраний код програми, який готовий для публікації або розгортання. Створення релізу дозволяє ефективно керувати версіями програмного забезпечення та забезпечує доступність оновлень для користувачів.

Після успішного виконання цього робочого процесу отримано повідомлення про успішну збірку та публікацію релізу. Цю інформацію можна переглянути в журналі виконання, де будуть відображені всі кроки процесу, а також можливі повідомлення про помилки або попередження. Такий журнал є важливим джерелом інформації для перевірки стану робочого процесу та виявлення можливих проблем. Журнал виконання зображено на рис. 3.6.

Рисунок 3.6 – Журнал виконання релізу

Це свідчить про те, що процес випуску був виконаний без помилок і новий реліз успішно створений. Вигляд сторінки з автоматично створеним релізом зображено на рис. 3.7.

Рисунок 3.7 –

Схожість

Джерела з Бібліотеки 18

1	**Студентська робота**	ID **файлу**: 1015146119	**Навчальний заклад**: National Aviation University	14 Джерело	0.23%
2	**Студентська робота**	ID **файлу**: 1015213413	**Навчальний заклад**: Lviv Polytechnic National University	2 Джерело	0.23%
3	**Студентська робота**	ID **файлу**: 1015078715	**Навчальний заклад**: National University of Life and Environmenta...		0.13%
4	**Студентська робота**	ID **файлу**: 1013069525	**Навчальний заклад**: Taras Shevchenko National University of Kyiv		0.1%