**ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ФАХОВИЙ КОЛЕДЖ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»
НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ «ЛЬВІВСЬКА
ПОЛІТЕХНІКА»**

ТЕХНІЧНИЙ ЗВІТ

з переддипломної практики

Студента групи *КН-320*
Цьося Іллі Юрійовича
Прізвище, ім’я по батькові
База практики: Лабораторія “web-програмування” 303н
Керівник практики Бугиль Б. А.
Ім’я, прізвище
Керівник дипломної роботи Бугиль Б. А.
Ім’я, прізвище

Дані про залік:

Оцінка _____

Дата захисту: «___» _____ 2023 р.

Членів комісії:

_____

_____

Львів 2023 р.

АНОТАЦІЯ

Обсяг даної курсової роботи 29 сторінок. Робота містить 8 посилань.

У першому розділі було розглянуто збір а опис теоретичних відомостей з автоматизації виконання юніт-тестів та статистики покриття коду за допомогою GitHub Actions

У другому розділі було проведено налаштування середовища виконання роботи роботи та встановлення програм для автоматизації виконання юніт-тестів та статистики покриття коду за допомогою GitHub Actions

Ключові слова: GITHUB, ТЕСТ, АВТОМАТИЗАЦІЯ, PYTHON, КОД,

ЗМІСТ

ВСТУП

Автоматизація виконання юніт-тестів та отримання статистики покриття коду є важливим етапом розробки програмного забезпечення. GitHub Action надає можливість автоматизувати ці задачі та забезпечити високу якість вашого програмного забезпечення.

GitHub Action дозволяє вам встановлювати та налаштовувати сценарії автоматизації, які будуть виконуватися при певних подіях, таких як коміти до гілки репозиторію. Ви можете налаштувати сценарій для запуску юніт-тестів та збору статистики покриття коду, щоб переконатися, що ваш код працює належним чином та відповідає вимогам якості.

Для автоматизації запуску юніт-тестів можна використовувати різні інструменти, такі як PHPUnit для PHP, Jest для JavaScript, Pytest для Python та інші. Ви можете встановити та налаштувати відповідний інструмент у своєму сценарії автоматизації та визначити, які саме тести мають бути запущені.

Для отримання статистики покриття коду можна використовувати інструменти, такі як Codecov, Coveralls та інші. Ви можете встановити та налаштувати відповідний інструмент у своєму сценарії автоматизації та визначити, які саме дані щодо покриття коду мають бути зібрані.

Отримання статистики покриття коду дозволяє вам зрозуміти, наскільки добре ваши тести покривають код та які частини коду потребують додаткових тестів. Це допомагає вам забезпечити високу якість вашого програмного забезпечення та зменшити кількість помилок, які можуть з'явитися в процесі розробки

РОЗДІЛ 1

ЗБІР ТА ОПИС ТЕОРЕТИЧНИХ ВІДОМОСТЕЙ З РОБОТИ З АВТОМАТИЗАЦІЄЮ ВИКОНАННЯ ЮНІТ-ТЕСТІВ

1.1 Автоматизація та її застосування

Автоматизація - це процес заміни ручної роботи на автоматичний механізм, який виконує задачі з використанням програмного забезпечення, роботів або інших пристроїв, що забезпечують автоматичне виконання операцій.

Автоматизація може бути застосована у різних сферах, включаючи виробництво, логістику, банківський сектор, медицину та багато інших. При цьому автоматизація може забезпечувати підвищення продуктивності, зниження витрат, покращення якості продукції або послуг, зменшення кількості помилок та інші переваги.

Автоматизація дозволяє знизити витрати на працю, зменшити кількість помилок, підвищити продуктивність та покращити якість продукту. Наприклад, у виробництві автоматизація може включати в себе використання роботів для монтажу деталей або використання програмних засобів для контролю якості продукту. В сфері бізнесу автоматизація може включати в себе використання програмного забезпечення для автоматичного створення звітів або виконання рутинних операцій, таких як відправка електронної пошти.

Автоматизація стала надзвичайно важливою в епоху цифрових технологій, коли все більше бізнес-процесів стають електронними та вимагають автоматизації. Це дозволяє компаніям зосередитися на більш складних та високооплачуваних завданнях, замість того, щоб витрачати час та ресурси на повторювані рутинні завдання.

Автоматизація може бути виконана різними способами, включаючи використання механічних пристроїв, роботів, програмного забезпечення та інших технологій. У деяких випадках автоматизація може бути повністю автономною, тобто виконувати завдання без жодної участі людини, а в інших -

вона може вимагати участі людини для контролю за процесом та прийняття рішень. У загальному, автоматизація дозволяє підвищити ефективність та точність роботи, знизити витрати та покращити якість продукту або послуги.

Залежно від сфери застосування та масштабу, автоматизація може бути виконана різними методами та технологіями.

У виробничій сфері, наприклад, можуть бути застосовані різні типи автоматизації. Один з них - це автоматизація виробничих ліній з використанням роботів, які здійснюють різні види операцій на лінії виробництва. Це дозволяє знизити витрати на працю, зменшити час виробництва та збільшити продуктивність.

Інший метод - це використання автоматизованих систем керування виробництвом, що дає можливість контролювати різні аспекти виробництва, включаючи якість продукції, витрати матеріалів, продуктивність працівників та інші параметри.

У сфері логістики автоматизація може включати використання автоматизованих складів, які забезпечують автоматичне зберігання та перевезення товарів, використання систем автоматичного розподілу товарів та інше.

У банківській сфері автоматизація може включати використання програмного забезпечення для обробки фінансових операцій, автоматизовані банкомати, системи розпізнавання голосу для ідентифікації клієнтів та інші технології.

Окрім цього, автоматизація може бути застосована у багатьох інших сферах, таких як медицина, освіта, транспорт, розваги та інші.

Загалом, автоматизація є важливою складовою в усіх сферах діяльності, оскільки дозволяє забезпечити ефективність та точність роботи, знизити витрати та покращити які

1.2 Тестування коду, тести та їх класифікація

Тестування коду - це процес визначення якості програмного забезпечення (ПЗ), під час якого розробники створюють набір тестів, які призначені для перевірки функціональності, стійкості та надійності ПЗ. Ці тести включають в себе запуск програмного коду з різними вхідними даними, щоб перевірити, чи поводиться програма правильно в різних умовах.

Тестування коду є важливою складовою в процесі розробки ПЗ, оскільки допомагає знизити кількість помилок та забезпечити високу якість програмного продукту. Відсутність тестів може призвести до помилок, які можуть бути складні для виявлення та виправлення, особливо у великих проектах зі складним кодом.

Тест (англ. test) - це процес або процедура, яка використовується для оцінки якості чогось, такого як програмне забезпечення, продукти, послуги, знання тощо. Тестування може здійснюватися з різних позицій і з метою перевірки різних характеристик, таких як відповідність вимогам, правильність роботи, швидкість, надійність, безпека тощо.

У контексті програмування, тест - це програмний код, написаний з метою перевірки правильності роботи іншого програмного коду. Цей тестовий код запускається з різними вхідними даними, щоб перевірити, чи поводиться програма правильно в різних ситуаціях. Тести можуть бути написані розробниками програмного забезпечення або іншими спеціалістами, які відповідають за якість ПЗ.

Тести можуть бути ручними або автоматизованими. Ручні тести виконуються вручну спеціалістами, що може бути часом затратно та неефективно, особливо в великих проектах. Автоматизовані тести створюються з використанням спеціальних програм та бібліотек, що дозволяє прискорити процес тестування та забезпечити більш високу точність результатів.

Для ефективного тестування важливо створювати якісні тести, які покривають якомога більше можливих сценаріїв взаємодії користувачів з програмою. Крім того, слід регулярно проводити тести на різних етапах розробки ПЗ, щоб вчасно виявляти та виправляти помилки та несумісності.

У контексті програмування, тести можуть мати різні форми і виконуватися на різних етапах розробки програмного забезпечення. Основна мета тестування - перевірити, чи відповідає програма вимогам та специфікаціям, чи працює правильно в різних ситуаціях і чи забезпечує вимоги до якості.

Для забезпечення якості програмного забезпечення зазвичай використовуються три типи тестів: модульні тести, інтеграційні тести і функціональні тести.

Модульні тести - це тести, які перевіряють правильність роботи окремих модулів або функцій програми. Модулі можуть бути тестовані індивідуально, щоб забезпечити, що вони працюють правильно перед їх інтеграцією в систему.

Інтеграційні тести - це тести, які перевіряють правильність роботи всієї системи як одного цілого. Ці тести перевіряють взаємодію різних модулів та компонентів системи, щоб переконатися, що вони працюють разом правильно.

Функціональні тести - це тести, які перевіряють функціональність програми, тобто те, як програма повинна працювати відповідно до вимог і специфікацій. Ці тести перевіряють різні входи та виводи програми, щоб переконатися, що вони повністю відповідають вимогам.

Для виконання тестів можуть використовуватися різні інструменти, такі як фреймворки тестування, автоматизовані тести, інструменти профілювання, тести навантаження та інші. Крім того, важливим етапом тестування є аналіз та виявлення помилок, що дозволяє покращити

1.3 Юніт-тести та їх використання

Юніт-тести (англ. unit tests) - це тести, які перевіряють коректність роботи окремих функцій або методів програмного коду. Це найнижчий рівень

тестування в програмуванні, де кожна функція тестується окремо, ізольовано від інших частин програми.

Юніт-тести дозволяють забезпечити, що кожна окрема частина програми працює коректно і не впливає на роботу інших частин. Вони зазвичай виконуються автоматично за допомогою спеціальних фреймворків тестування, таких як JUnit для Java або NUnit для .NET.

Під час написання юніт-тестів для функції або методу програміст визначає очікуваний результат роботи функції або методу для певних вхідних даних. Потім він створює тестові дані, передає їх у функцію або метод, і перевіряє, чи отриманий результат відповідає очікуваному результату.

Написання юніт-тестів дозволяє програмістам виявляти та виправляти помилки в коді на ранніх етапах розробки, забезпечуючи більш високу якість програмного забезпечення та знижуючи ризик виникнення критичних помилок під час виконання програми. Крім того, юніт-тести сприяють підтримці іншими розробниками і використовувачами коду, а також дозволяють зберігати якість програми на стабільному рівні під час змін у коді або вводу нового функціоналу.

Юніт-тестування є невід'ємною частиною розробки програмного забезпечення, і для його виконання використовуються різні утиліти. Ось декілька найпопулярніших утиліт для юніт-тестування:

JUnit: Це найпопулярніший фреймворк для юніт-тестування Java-програм. Він має простий синтаксис, добре документований і підтримує різні функції для тестування, такі як перед- і післятестові дії, перевірка винятків і т.д.

NUnit: Це фреймворк для юніт-тестування .NET-програм. Він підтримує багато функцій, таких як тестування з параметрами, можливість запуску тестів у певному порядку, підтримка кількох фреймворків вводу-виводу і т.д.

pytest: Це фреймворк для тестування Python, який дозволяє легко писати тести для Python-коду. Він має багато функцій, таких як підтримка параметрів, підтримка фікстур (зберігання даних між тестами), інтеграція з іншими фреймворками тестування і т.д.

Mocha: Це фреймворк для тестування JavaScript, який можна використовувати як на серверній, так і на клієнтській стороні. Він підтримує різні функції, такі як асинхронні тести, тести з параметрами, включення і виключення тестів і т.д.

Ці фреймворки для юніт-тестування дозволяють програмістам легко писати тести, забезпечують структуру тестів і забезпечують різноманітні функції для тестування. Крім цього, інструменти для збирання тестів і генерації звітів про тестування допомагають забезпечити повну автоматизацію процесу тестування, що збільшує швидкість розробки і знижує кількість помилок.

Юніт-тести використовуються для тестування окремих функцій або методів програми, які виконують обмежені завдання і мають чітко визначені вхідні та вихідні параметри.

Зазвичай розробники створюють окремий набір тестів для кожного модуля або компоненту програми, який потім виконують автоматично під час процесу розробки, використовуючи спеціальні утиліти для тестування.

При виконанні тестів, програма запускається з різними вхідними параметрами, щоб перевірити, чи повертає функція очікуваний результат. Якщо результат відповідає очікуваному, тест вважається успішним. Якщо результат не відповідає очікуваному, тест вважається неуспішним, і програміст повинен виправити проблему та перевірити тест ще раз.

Користуючись юніт-тестами, розробники можуть переконатися в тому, що окремі частини програми працюють правильно, і необхідні функції тестуються на ранніх етапах розробки. Це допомагає знизити кількість помилок, які можуть з'явитися пізніше, під час інтеграції окремих компонентів в одну програму.

Ще одна важлива перевага використання юніт-тестів полягає в тому, що вони дозволяють розробникам змінювати код програми, не боячись, що це призведе до збоїв у функціонуванні інших частин програми. Коли розробник вносить зміни в код, він може виконати юніт-тести, щоб переконатися в тому, що нічого не зламалось в інших частинах програми.

Також юніт-тести допомагають покращити документацію до програмного продукту, оскільки вони змушують розробників чітко визначати очікуваний результат для кожної функції або методу. Це дозволяє створювати документацію, яка чітко визначає, що робить кожна функція, які вхідні дані потрібні для її виконання та який очікуваний результат.

Окрім того, юніт-тести можуть допомогти зекономити час на тестуванні, оскільки вони автоматизовані та можуть бути запущені автоматично під час процесу збірки програми. Це дозволяє розробникам швидше виявляти та виправляти проблеми, що допомагає збільшити ефективність розробки та знизити загальні витрати на проект.

Отже, використання юніт-тестів є важливою складовою розробки програмного продукту, оскільки допомагає забезпечити правильну роботу програми та знижує кількість помилок, що можуть з'явитися в процесі її розробки.

1.4 Покриття коду тестами та їх призначення

Розкриття коду тестами, або Test Driven Development (TDD), є методологією розробки програмного забезпечення, яка базується на написанні тестів перед написанням фактичного коду. Основна ідея полягає в тому, що розробник спочатку пише тест-кейси, які визначають очікувані результати функції, а потім реалізує ці функції, щоб задовольнити вимоги тестів.

Процес TDD можна розбити на три етапи: "червоний", "зелений" та "рефакторинг".

На етапі "червоного" тесту розробник пише тест, який спочатку не пройде, оскільки функціонал ще не реалізовано. Наприклад, якщо потрібно реалізувати функцію, яка додає два числа, розробник може написати тест, який передбачає, що якщо додати 2 та 3, то результат буде 5. Цей тест відповідає на питання "що має робити програма?".

Після написання тесту розробник запускає його, і очікується, що тест не пройде ("червоний" результат). Це свідчить про те, що функціонал ще не реалізовано, і потрібно переходити до наступного етапу.

На етапі "зеленого" тесту розробник реалізує функціонал, необхідний для проходження тесту. В нашому прикладі, розробник може реалізувати функцію, яка додає два числа. Після написання коду розробник запускає тест, і очікується, що тест пройде ("зелений" результат). Це свідчить про те, що функціонал було успішно реалізовано.

На етапі "рефакторингу" розробник розглядає код і перевіряє, чи можна його покращити. Наприклад, розробник може виявити, що певні частини коду можна спростити або оптимізувати. При цьому потрібно переконатися, що всі тести продовжують працювати як очікувалось.

Таким чином, розкриття коду тестами дозволяє розробникам писати більш стійкий код з меншою кількістю помилок, зменшує час, необхідний для виявлення та виправлення помилок, та забезпечує більшу стабільність продукту.

Розкриття коду тестами є частиною процесу тестування програмного забезпечення, який дозволяє перевірити, чи відповідає код певним вимогам та специфікаціям. Цей підхід полягає у написанні автоматизованих тестів, які виконують певний код та перевіряють, що його результати відповідають очікуваним.

Процес розкриття коду тестами може бути виконаний на різних етапах розробки програмного забезпечення. Наприклад, на початковому етапі розробки можна написати тести для перевірки коректності введення даних, або тестів, які перевіряють, що певні алгоритми працюють правильно. На етапі тестування можуть бути написані додаткові тести для перевірки, що попередні пройшли успішно та не були порушені змінами, які були внесені до коду.

Є кілька типів тестів, які можна використовувати при розкритті коду тестами. Один з найпоширеніших типів - це юніт-тести, які перевіряють, чи працює окрема функція коду, при цьому ізолюючи її від інших функцій та компонентів. Інші типи тестів включають інтеграційні тести, які перевіряють

взаємодію між різними компонентами програмного забезпечення, та прийняття (acceptance) тести, які перевіряють, що продукт повністю задовольняє вимоги користувачів.

Незалежно від типу тестів, розкриття коду тестами дозволяє забезпечити якість та стабільність продукту, зменшити час та затрати на виявлення та виправлення помилок, та забезпечити більшу відповідність коду вимогам та специфікаціям. Використання автоматизованих тестів також може допомогти покращити розуміння коду та його функцій, та зробити процес розробки більш ефективним та продуктивним. Крім того, розкриття коду тестами допомагає виявляти проблеми та помилки ще до того, як вони стануть критичними, тим самим зменшуючи ризики для користувачів та забезпечуючи вищу якість продукту.

Загалом, розкриття коду тестами є важливим етапом в процесі тестування програмного забезпечення, який допомагає забезпечити вищу якість продукту та підвищити ефективність процесу розробки. Розкриття коду тестами (англ. code coverage) - це процес визначення того, яка частина програмного коду була виконана під час запуску тестів. Зазвичай цей процес проводиться за допомогою спеціальних програмних інструментів, які відслідковують виклики функцій, логічні гілки, оператори та інші елементи коду, що виконуються під час тестування.

Розкриття коду тестами дозволяє визначити, наскільки повні та ефективні тести були написані. Недостатнє покриття коду може вказувати на те, що певні фрагменти коду не були викликані під час тестування, що може призвести до того, що певні проблеми та помилки не будуть виявлені. Крім того, розкриття коду тестами може допомогти виявити неявні залежності та взаємодії між частинами коду, що дозволить покращити архітектуру програми та зменшити ризики для користувачів.

Інструменти розкриття коду тестами можуть бути використані для визначення таких метрик, як загальне покриття коду, покриття окремих функцій, класів або пакетів, покриття логічних гілок та виразів, а також для

порівняння покриття коду між різними тестами та версіями програмного забезпечення.

1.5 Призначення GitHub Actions

GitHub Action - це сервіс, що дозволяє автоматизувати процеси від GitHub репозиторію. Один з найбільш корисних варіантів автоматизації - це автоматичне виконання юніт-тестів під час відправки змін в репозиторій.

Звичайно! GitHub Action має безліч можливостей, які дозволяють автоматизувати не тільки виконання юніт-тестів, але й інші процеси, такі як автоматична збірка, розгортання та багато інших.

Окрім цього, GitHub Action має декілька корисних функцій, таких як:

Візуалізація результатів тестування: GitHub Action дозволяє вам відслідковувати результати вашого тестування на графіках та гістограмах. Це допомагає швидко виявляти помилки в коді та вдосконалювати ваші тести.

Кастомні сценарії: GitHub Action дозволяє вам налаштувати сценарії автоматизації, які відповідають вашим потребам. Наприклад, ви можете налаштувати GitHub Action, щоб автоматично запускати тести при кожному коміті в гілку репозиторію або при пул-реквесті.

Налаштування середовища: GitHub Action дозволяє вам налаштувати середовище, у якому виконуються ваші тести. Наприклад, ви можете вибрати ОС, встановити необхідні залежності та налаштувати змінні середовища.

GitHub Action дозволяє вам налаштовувати сценарії автоматизації, які виконуються при певних подіях, таких як коміти до гілки репозиторію, пул-реквести, створення відгалуження та інші. Кожен сценарій складається з одного або декількох кроків, які виконуються послідовно.

GitHub Action має безліч готових шаблонів, які допомагають вам почати автоматизувати свої задачі з мінімальними зусиллями. Ви можете вибрати шаблон, який найкраще підходить вашому проекту, та налаштувати його відповідно до своїх потреб.

GitHub Action інтегрується з іншими сервісами, такими як Docker, Amazon Web Services, Google Cloud та інші, що дозволяє вам використовувати їх у своїх сценаріях автоматизації.

Загалом, GitHub Action є потужним інструментом для автоматизації вашого процесу розробки та забезпечення високої якості вашого програмного забезпечення. Він дозволяє вам економити час та зусилля, які ви можете витратити на більш складні задачі.

Інтеграція з іншими сервісами: GitHub Action дозволяє інтегрувати ваші тести з іншими сервісами, такими як Slack, Email та інші. Це допомагає швидко отримувати повідомлення про помилки та виявляти їх.

Загалом, автоматизація виконання юніт-тестів за допомогою GitHub Action дозволяє вам швидко виявляти помилки в вашому коді та забезпечувати високу якість вашого продукту.

РОЗДІЛ 2

НАЛАШТУВАННЯ СЕРЕДОВИЩА ВИКОНАННЯ РОБОТИ ТА ВСТАНОВЛЕННЯ ПРОГРАМИ GITHUB ACTION ТА ЙОГО КОНФІГУРАЦІЯ

2.1 Налаштування локального проекту

Visual Studio Code (VS Code) - це безкоштовний та відкритий редактор коду, розроблений компанією Microsoft. Він є дуже популярним редактором, який використовують розробники з різних мов програмування.

VS Code забезпечує зручне та налаштоване середовище для написання коду, налагодження та контролю версій.

Для того, щоб автоматизувати виконання юніт-тестів за допомогою GitHub Action, спочатку потрібно створити файл у вашому репозиторії з назвою .github/workflows/test.yml. У цьому файлі ви можете вказати конфігурацію вашого тестування, включаючи інструкції для запуску тестів.

Наприклад, якщо ви використовуєте мову програмування Python, ваш файл конфігурації може виглядати наступним чином:

```yaml
steps:
- uses: actions/checkout@v2

- name: Set up Python 3.8
  uses: actions/setup-python@v2
  with:
    python-version: 3.8

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

- name: Run tests
  run: |
    pytest --cov=./
```

У цьому прикладі ми вказали, що ми хочемо запустити юніт-тести під час кожного push в гілку репозиторію, використовуючи Python 3.x. Ми також вказали, що ми хочемо встановити залежності з requirements.txt і запустити юніт-тести, використовуючи python -m unittest discover -s tests.

Після того, як ви створили файл конфігурації, ви можете перевірити його на помилки та зберегти його у вашому репозиторії. Кожного разу, коли ви відправляєте зміни до свого репозиторію, GitHub Action буде автоматично запускати ваші тести на серверах GitHub і повертати результати вам.

Таким чином, автоматизація виконання юніт-тестів за допомогою GitHub Action дозволяє вам відслідковувати помилки тестування на ранніх етапів розробки та забезпечує, що ваш код завжди працює належним чином. GitHub Action - це сервіс, який дозволяє автоматизувати ваші розробницькі процеси, включаючи тестування коду.

Коли ви налаштуєте GitHub Action для виконання юніт-тестів, він буде запускатися автоматично кожен раз, коли ви здійснюєте зміни в коді та відправляєте їх на GitHub. Якщо під час виконання тестів виявляється помилка, ви отримуєте повідомлення про це, що дозволяє вам виправити проблему до того, як вона потрапить до основної гілки вашого проекту.

Автоматизація виконання юніт-тестів також зменшує кількість ручної роботи, яку потрібно виконувати розробникам. Замість того, щоб запускати тести вручну, вони можуть сконцентруватися на написанні якісного коду, знаючи, що юніт-тести запустяться автоматично та перевірять правильність роботи коду.

Отже, використання GitHub Action для автоматизації виконання юніт-тестів є корисним інструментом для забезпечення якості вашого коду та покращення продуктивності вашої розробки.

Отже, для створення віртуального середовища знадобиться встановлений пакетний менеджер pip, а також модуль virtualenv. Якщо користуватись ОС Windows, то потрібно встановити ще й програму virtualenvwrapper-win. Нижче наведено приклади Windows:

1. Відкрийте командний рядок та введіть наступну команду для встановлення модуля virtualenv: pip install virtualenv
2. Встановіть virtualenvwrapper-win за допомогою команди: pip install virtualenvwrapper-win
3. Створіть нове віртуальне середовище командою: mkvirtualenv env_name, де env_name - назва віртуального середовища.
4. Активуйте віртуальне середовище командою: workon env_name.

Після активації віртуального середовища, всі пакети, які ви встановлюєте за допомогою pip, будуть доступні лише в цьому середовищі. Це дозволяє уникнути конфліктів між різними версіями пакетів, які використовуються в різних проектах. Щоб вийти з віртуального середовища, достатньо ввести команду deactivate.

2.2 Створення юніт-тестів

Юніт-тести — це тести, які перевіряють роботу окремих модулів програмного забезпечення. Зазвичай, юніт-тести створюються для перевірки правильності роботи окремих функцій, методів та класів.

Основна ідея юніт-тестів полягає у створенні спеціального коду, який автоматично запускається та перевіряє поведінку тестованого модуля. При створенні юніт-тестів, зазвичай, використовуються спеціальні бібліотеки для тестування, такі як unittest, pytest, nose, mocha та інші.

Нижче наведено приклад створення юніт-тесту за допомогою бібліотеки unittest для Python:

```python
import unittest

def square(x):
    return x ** 2

class TestSquareFunction(unittest.TestCase):

    def test_square_with_positive_number(self):
        self.assertEqual(square(2), 4)

    def test_square_with_negative_number(self):
        self.assertEqual(square(-2), 4)

    def test_square_with_zero(self):
        self.assertEqual(square(0), 0)

if __name__ == '__main__':
    unittest.main()
```

У цьому прикладі ми створюємо функцію square, яка повертає квадрат числа x. Далі ми створюємо клас TestSquareFunction, який наслідується від класу unittest.TestCase. У цьому класі ми описуємо тести, які перевіряють правильність роботи функції square для різних вхідних даних.

Кожен тест в цьому класі описується як окремий метод з префіксом test_. У тілі методу ми викликаємо функцію square з різними вхідними даними та перевіряємо результат за допомогою метода assertEqual.

Нарешті, ми викликаємо функцію unittest.main(), яка запускає всі тести. Якщо всі тести успішно пройдуть, то у консоль виведеться повідомлення про успішне виконання тестів, в іншому випадку будуть виведені повідомлення про помилки.

У цьому прикладі ми створюємо функцію square, яка повертає квадрат числа x. Далі ми створюємо клас TestSquareFunction, який наслідується від класу unittest.TestCase. У цьому класі ми описуємо тести, які перевіряють правильність роботи функції square для різних вхідних даних.

Кожен тест в цьому класі описується як окремий метод з префіксом test_. У тілі методу ми викликаємо функцію square з різними вхідними даними та перевіряємо результат за допомогою метода assertEqual.

Нарешті, ми викликаємо функцію unittest.main(), яка запускає всі тести. Якщо всі тести успішно пройдуть, то у консоль виведеться повідомлення про успішне виконання тестів, в іншому випадку будуть виведені повідомлення про помилки.

Це лише простий приклад створення юніт-тесту, але він демонструє основні концепції тестування. Зазвичай, при створенні юніт-тестів потрібно враховувати багато нюансів та деталей, таких як генерація випадкових вхідних даних, обробка виключень, мокування (mocking) залежностей та інше.

В будь-якому випадку, створення юніт-тестів є важливою складовою розробки програмного забезпечення, оскільки вони дозволяють перевіряти коректність роботи окремих модулів та зменшують кількість помилок у роботі програми в цілому.

Загальний формат написання юніт-тестів на Python дуже схожий на той, який я описав вище для мови Java. Ось приклад тестування функції, яка повертає середнє арифметичне двох чисел:

```python
import unittest

def average(x, y):
    return (x + y) / 2

class TestAverage(unittest.TestCase):

    def test_average(self):
        self.assertEqual(average(2, 4), 3)
        self.assertEqual(average(0, 0), 0)
        self.assertEqual(average(-2, 2), 0)

if __name__ == '__main__':
    unittest.main()
```

У цьому прикладі ми спочатку імпортуємо модуль unittest, який надає функціональність для написання юніт-тестів на Python. Далі ми оголошуємо функцію average, яка приймає два аргументи та повертає їх середнє арифметичне.

Потім ми оголошуємо клас TestAverage, який успадковується від unittest.TestCase. У цьому класі ми оголошуємо тест test_average, який перевіряє правильність роботи функції average на трьох тестових наборах.

Тестування виконується за допомогою методу self.assertEqual, який перевіряє, чи дорівнює результат виконання функції очікуваному значенню. Якщо це так, то тест пройдений успішно.

Нарешті, ми запускаємо тест за допомогою методу unittest.main(). Він автоматично знайде всі тести, оголошені у класі TestAverage, та виконає їх. Якщо будь-який тест не пройдений успішно, то виведеться відповідне повідомлення про помилку.

2.3 Автоматизація юніт тестів за допомогою GitHub Actions

Для автоматизації запуску юніт-тестів на пайтон за допомогою GitHub Actions потрібно створити файл .github/workflows/tests.yml, який міститиме

наступний код:

```yaml
name: Run Tests

on: [push]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Set up Python
      uses: actions/setup-python@v2
      with:
        python-version: 3.8

    - name: Install dependencies
      run: pip install -r requirements.txt

    - name: Run unit tests
      run: python -m unittest discover tests/
```

Цей файл містить наступне:

- Назва воркфлоу: Run Tests
- Подія, при якій запускається воркфлоу: push (тобто при кожному коміті в гітхабі)
- Налаштування воркфлоу: він буде виконуватись на останній версії Ubuntu
- Кроки воркфлоу:
    - Клонування репозиторію за допомогою actions/checkout@v2
    - Налаштування Python 3.8 за допомогою actions/setup-python@v2
    - Встановлення залежностей з requirements.txt за допомогою pip install
    - Запуск юніт-тестів за допомогою python -m unittest discover tests/

Цей файл можна модифікувати, щоб відповідати конкретним потребам вашого проекту, наприклад, змінити версію Python або шлях до каталогу з тестами.

Після додавання цього файлу в репозиторій GitHub, GitHub Actions автоматично запустить воркфлоу кожного разу, коли буде здійснений коміт в гітхаб. Результати тестів можна переглянути в журналі воркфлоу веб-інтерфейсу GitHub, де можна побачити, які тести пройшли успішно, а які - ні.

Конфігурація юніт-тестів за допомогою GitHub Actions зазвичай включає в себе кілька кроків, що дозволяють виконати тести та зібрати інформацію про їх виконання.

Ось приклад конфігурації для Python проекту, який використовує pytest:

```yaml
name: Run Tests
on: [push]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Set up Python
      uses: actions/setup-python@v2
      with:
        python-version: 3.8

    - name: Install dependencies
      run: pip install -r requirements.txt

    - name: Run tests
      run: pytest
      env:
        DB_HOST: localhost
        DB_PORT: 5432
        DB_NAME: test_db
        DB_USER: test_user
        DB_PASSWORD: test_password
```

У цій конфігурації ми використовуємо actions/checkout@v2, щоб склонувати репозиторій, actions/setup-python@v2, щоб налаштувати середовище Python, та pip install, щоб встановити залежності проекту. Далі ми викликаємо pytest, використовуючи команду run.

Також ми передаємо середовищні змінні за допомогою блоку env. У цьому прикладі ми передаємо параметри підключення до тестової бази даних.

Результати виконання тестів можна переглянути в журналі воркфлоу, який буде виконуватися веб-інтерфейсом GitHub. Крім того, за допомогою різних інструментів можна налаштувати повідомлення у Slack, електронну пошту або інші інтеграції залежно від ваших потреб.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ ПРАКТИКИ

1. Мова програмування високого рівня - це мова програмування, яка надає високий рівень абстракції та дозволяє розробнику програмувати на вищому рівні абстракції, що приховує деталі роботи з апаратним забезпеченням та пам'яттю. Основна особливість мов високого рівня полягає в тому, що вони дозволяють розробникам працювати з даними та операціями на більш абстрактному рівні, що зменшує кількість помилок та спрощує розробку програмного забезпечення.

Деякі основні поняття та особливості мов програмування високого рівня:

- Синтаксис: Мови високого рівня мають більш простий та зрозумілий синтаксис порівняно з мовами низького рівня. Це зменшує кількість помилок, що можуть виникати при програмуванні.
- Абстракція: Мови високого рівня надають високий рівень абстракції, що дозволяє розробникам зосередитися на більш важливих завданнях, а не на деталях роботи з апаратним забезпеченням.
- Портативність: Мови високого рівня можуть бути портативними, тобто програми, написані на таких мовах, можуть запускатися на різних платформах без необхідності переписування коду.
- Об'єктно-орієнтований підхід: Більшість мов високого рівня підтримують об'єктно-орієнтований підхід, що дозволяє розробникам створювати програмне забезпечення за допомогою об'єктів та класів, що спрощує розробку та збільшує повторне використання коду.
- Динамічна типіз

2. Тестування програмного забезпечення - це процес перевірки програмного забезпечення на належність вимогам та виявлення помилок та дефектів. Цей процес може включати ручне або автоматизоване виконання тестів, які можуть бути функціональними, не функціональними, регресійними тощо. Тестування програмного забезпечення допомагає підвищити якість продукту, зменшити ризик виявлення помилок в етапі експлуатації та

забезпечити задоволення користувачів. GitHub Action можна використовувати для автоматизації процесу тестування програмного забезпечення. Для цього необхідно створити відповідний конфігураційний файл у форматі YAML, який містить інструкції для виконання тестів та збору статистики покриття коду.

3. Програмування в Інтернеті включає в себе створення програмного забезпечення, яке працює в Інтернеті та взаємодіє з користувачами через веб-браузер або інші інтернет-технології. Це можуть бути різноманітні веб-додатки, веб-сайти, сервіси та інші інтернет-рішення.

Програмування в Інтернеті зазвичай використовується для розробки веб-додатків, що дозволяють користувачам взаємодіяти з різними сервісами та зберігати свої дані в Інтернеті. Це може бути, наприклад, онлайн-магазин, соціальна мережа, хмарне сховище даних тощо.

Одним з найпопулярніших інструментів для програмування в Інтернеті є мова програмування JavaScript. JavaScript є мовою, яка підтримується веб-браузерами та дозволяє створювати інтерактивні веб-сторінки та додатки. Також для програмування в Інтернеті можуть використовуватися інші мови програмування, такі як Python, Ruby, PHP та інші.

4. Ознайомився з правилами техніки безпеки та охорони праці на підприємстві. Пройшов інструктаж по техніці безпеки та охорони праці.

5. Ознайомився із структурою підприємства. Взаємодіяв з його окремими підрозділами.

6. Ознайомився з описом посадових інструкцій та роботи підприємства під час проходження практики.

7. Під час проходження практики використовувався персональний комп’ютер та ноутбук

8. Під час проходження практики використовувався GitHub Actions. Це сервіс автоматизації розробки, що надається GitHub, що дозволяє створювати та налаштовувати автоматичні робочі процеси (workflows) для забезпечення автоматичної інтеграції, тестування та розгортання програмного забезпечення.

GitHub Actions дозволяє підключати додаткові сервіси та інструменти, такі як Docker, AWS, Azure, Slack тощо, для полегшення роботи з проектами.

ВИСНОВОК

Автоматизація юніт-тестів за допомогою GitHub Actions дозволяє значно зменшити витрати часу та зусиль, що потрібні для виконання тестів вручну. Завдяки автоматизованій системі тестування можна швидко виявляти та виправляти помилки в коді, що зменшує ризик виникнення проблем у продакшні.

Конфігурація GitHub Actions дозволяє налаштовувати виконання тестів залежно від потреб проекту. Можна налаштувати виконання тестів при кожному коміті, або використовувати регулярний запуск, щоб переконатися, що код проекту залишається працездатним протягом усього періоду розробки.

GitHub Actions дозволяє використовувати різні інструменти для тестування різноманітних проектів. Наприклад, для тестування веб-додатків можна використовувати Selenium, а для тестування мобільних додатків - Appium.

Таким чином, автоматизація юніт-тестів за допомогою GitHub Actions є потужним інструментом для автоматизації тестування проектів будь-якої складності. Вона дозволяє підвищити якість продукту та зменшити час, необхідний для виявлення та виправлення помилок.

ВИКОРИСТАНІ ДЖЕРЕЛА

1. GitHub Actions Documentation - Тип доступу:

   https://docs.github.com/en/actions

2. Learn GitHub Actions - GitHub Docs - Тип доступу:

   https://docs.github.com/en/actions/learn-github-actions

3. Quickstart for GitHub Actions - Тип доступу:

   https://docs.github.com/en/actions/quickstart

4. Офіційна документація GitHub Actions - Тип доступу:

   https://docs.github.com/en/actions

5. Практичний гайд з автоматизації юніт-тестування за допомогою GitHub Actions - Тип доступу:

   https://dev.to/michaelboganjr/python-unit-testing-with-github-actions-2bfp

6. Посібник з автоматизації тестування на Python за допомогою Pytest та GitHub Actions - Тип доступу:

   https://www.section.io/engineering-education/automated-testing-with-pytest-and-github-actions/

7. Гайд з автоматизації тестування на Node.js за допомогою Jest та GitHub Actions - Тип доступу:

   https://medium.com/swlh/automated-testing-with-jest-and-github-actions-f7cc9d3c3ea3

8. Практичний гайд з автоматизації тестування на React за допомогою Cypress та GitHub Actions - Тип доступу:

   https://www.cypress.io/blog/2020/08/17/testing-react-apps-automation-github-actions/