МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп’ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

Матіяш Юрій

Програмний модуль опрацювання зображень з використанням DevOps засобів/ Software module for image processing using DevOps tools

спеціальність: 123 – Комп’ютерна інженерія
освітньо–професійна програма – Комп’ютерна інженерія

Кваліфікаційна робота

Виконав студент групи КІ–41
Ю. Р. Матіяш
_____

Науковий керівник:
к.т.н., О.Й. Піцун
_____

Кваліфікаційну роботу
допущено до захисту:
"_____"__ 20_____р.

_____Л. О. Дубчак

ТЕРНОПІЛЬ - 2024

АНОТАЦІЯ

Кваліфікаційна робота на тему «Програмний модуль опрацювання зображень з використанням DevOps засобів» зі спеціальності 123 «Комп'ютерна інженерія» освітнього ступеня «бакалавр» містить 49 сторінок пояснюючої записки, 16 рисунків, 1 таблиця, 2 додатки. Обсяг графічного матеріалу 2 аркуші формату А3.

Метою кваліфікаційної роботи є розробка пайплайну та програмного модулю для опрацювання зображень з використанням DevOps засобів.

Методи дослідження включають методи хмарних обчислень, DevOps підходів до інтеграції та доставки програмного коду.

DevOps — це методологія, яка інтегрує процеси розробки програмного забезпечення (Dev) та інформаційних операцій (Ops), і вона стала ключовою для успішного впровадження технологічних рішень у багатьох сферах. Завдяки принципам автоматизації, неперервної інтеграції та швидкої ітерації, DevOps сприяє значному підвищенню якості та швидкості розробки, що є критично важливим у сферах, де потрібна висока точність обробки даних.

Головна мета даного дослідження полягає в розробці DevOps пайплайну для програми обробки зображень на Python, що включає повну автоматизацію від розробки до впровадження. Такий підхід має на меті забезпечення ефективного та надійного середовища для обробки зображень, з особливим акцентом на використання середовища Ubuntu.

Ключові слова: DEVOPS, CI/CD, ОПРАЦЮВАННЯ ЗОБРАЖЕНЬ.

ANNOTATION

The qualification work on the topic "Software module for image processing using DevOps tools" from specialty 123 "Computer engineering" of the bachelor's degree contains 49 pages of explanatory note, 16 figures, 1 tables, 2 appendices. The amount of graphic material is 2 sheets of A3 format.

The goal of the qualification work is to develop a pipeline and software module for image processing using DevOps tools.

Research methods include methods of cloud computing, DevOps approaches to the integration and delivery of software code.

DevOps is a methodology that integrates software development (Dev) and information operations (Ops) processes, and it has become key to the successful implementation of technology solutions in many areas. Thanks to the principles of automation, continuous integration and rapid iteration, DevOps contributes to a significant increase in the quality and speed of development, which is critical in areas where high accuracy of data processing is required.

The main goal of this research is to develop a DevOps pipeline for a Python image processing application that includes full automation from development to implementation. This approach aims to provide an efficient and reliable image processing environment, with a special emphasis on using the Ubuntu environment.

Keywords: DEVOPS, CI/CD, IMAGE PROCESSING.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ПК – Персональний комп’ютер

HSV – Hue Saturation Value

BGR – Blue Green Red

ПЗ – Програмне забезпечення

CI/CD – Continuous Integration/Continuous Deployment

ВСТУП

DevOps — це методологія, яка інтегрує процеси розробки програмного забезпечення (Dev) та інформаційних операцій (Ops), ставши ключовою для успішного впровадження технологічних рішень у багатьох сферах. Завдяки принципам автоматизації, неперервної інтеграції та швидкої ітерації, DevOps сприяє значному підвищенню якості та швидкості розробки, що є критично важливим у сферах, де потрібна висока точність обробки даних.

Основні принципи DevOps включають культуру співпраці між командами розробки та експлуатації, що забезпечує безперервний обмін інформацією і швидке вирішення проблем. Автоматизація знижує ймовірність людських помилок і підвищує ефективність роботи, наприклад, через автоматизоване тестування, яке дозволяє швидко виявляти дефекти.

Неперервна інтеграція та неперервне розгортання (CI/CD) є ключовими аспектами DevOps. Постійна інтеграція коду в спільне сховище, що проходить автоматизовані тести, після чого нові функції автоматично розгортаються на виробничих серверах знижуючи час від написання коду до його впровадження.

DevOps також включає принципи моніторингу та зворотного зв'язку, що дозволяє вчасно виявляти та виправляти проблеми, забезпечуючи стабільну роботу системи. Зворотний зв'язок від користувачів допомагає покращувати продукти та процеси, відповідаючи на потреби клієнтів.

Впровадження DevOps вимагає розвитку відкритості до змін і постійного вдосконалення. Такий підхід дозволяє організаціям швидше адаптуватися до змін, підвищувати конкурентоспроможність і задовольняти потреби клієнтів.

Об’єкт дослідження – підходи до безперервно доставки і інтеграції програмного коду на веб-серверах.

Предмет дослідження – підхід до розгортання проекту з опрацювання зображення на основі технології DevOps.

Головна мета даного дослідження полягає в розробці DevOps пайплайну для програми обробки зображень на Python, що включає повну автоматизацію від розробки до впровадження. Такий підхід має на меті забезпечення ефективного та надійного середовища для обробки зображень, з особливим акцентом на використання середовища Ubuntu.

Для досягнення мети необхідно виконати такі задачі:

- проаналізувати підходи до використання пайплайнів при розробці пз;
- проаналізувати засоби опрацювання зображень;
- розробити алгоритм обробки цифрових зображень;
- розробити життєвий цикл розгортання проекту;
- розробити алгоритм опрацювання зображень в хмарному середовищі;
- здійснити порівняльний аналіз розроблено підходу з аналогами.

У першому розділі проведено порівняльний аналіз існуючих підходів до розробки CI/CD пайплайнів, що дозволило виділити основні блоки конфігураційного файлу.

У другому розділі наведено алгоритм опрацювання зображень в хмарному середовищі та життєвий цикл розгортання проекту.

У третьому розділі наведено результати опрацювання зображень на хмарному сервісі з використання CI/CD пайплайнів.

1 АНАЛІЗ ІНСТРУМЕНТІВ РОЗРОБКИ DEVOPS

1.1 Використання пайплайнів при розробці ПЗ

В сучасному світі розробка програмного забезпечення стає складнішою та вимагає від розробників постійного покращення їхніх методів та підходів. Одним із ключових інструментів, що забезпечують ефективну та якісну розробку програм використовуються пайплайни. Пайплайн — це послідовність процесів, які обробляють вхідні дані та передають їх наступному процесу для подальшої обробки. У кваліфікайній роботі буде розглянуто, що таке пайплайни, як вони використовуються в розробці програмного забезпечення та як вони сприяють покращенню процесу розробки [1].

Багато організацій активно впроваджують практики та інструментарій DevOps для подолання внутрішньоорганізаційної ізоляції, підвищення якості та ефективності доставки програмного забезпечення. Однак, науковою спільнотою було недостатньо досліджено вплив конкретних практик DevOps на продуктивність організацій. Встановлено, що існує позитивне співвідношення між комплексним впровадженням цих практик та незалежною оцінкою рівня зрілості організацій. В особливості, практики, що включають створення мінімальних робочих середовищ (пісочниць), тестування засноване на розробці, та неперервну інтеграцію в головну вітку коду, показують найслабшу кореляцію в аналізованих даних. Загалом, вплив на процес доставки програмного забезпечення та продуктивність організацій переважно оцінюється як позитивний. Тим не менш, існують відгуки про потенційний негативний вплив DevOps, зокрема, зниження передбачуваності доставки продуктів розробки [2].

Пайплайн у розробці програмного забезпечення являє собою послідовність автоматизованих кроків або операцій, що виконуються над вихідним кодом або даними з метою забезпечення якості, ефективності та надійності програмного продукту. Вони включають в себе такі етапи, як збірка (build), тестування (testing), відлагодження (debugging), деплоймент (deployment) та багато інших [3].

Їхнє використання у процесі розробки програмного забезпечення надає значні переваги, зокрема через автоматизацію різних рутинних процедур, що не тільки підвищує швидкість розробки, але й мінімізує ризики виникнення помилок. Завдяки пайплайнам забезпечується ефективний контроль версій, кожен етап розробки асоціюється з певною версією коду, що сприяє кращій організації і збереженню історії змін. Також ці системи полегшують комплексну перевірку якості коду, таку як, статичний аналіз, модульне тестування та аналіз покриття коду тестами, що значно підвищує надійність розроблюваних продуктів. Швидкість розгортання програмних рішень зростає, що дозволяє оперативно реагувати на зміни вимог замовників і ринкових умов. Також, вони відрізняються високою масштабованістю, що робить їх придатними для застосування як в малих, так і в масштабних складних проектах, що гарантує їхню ефективність і адаптивність.

Типовий пайплайн в розробці програмного забезпечення може містити наступні компоненти:

1) Етап збірки(Building) програмного забезпечення виконує різні процеси, такі як компіляція програмного коду, створення виконуваних файлів та інших важливих компонентів необхідних для функціонування програми. Під час виконання основна функція це перетворення написаного коду в формат, який може бути виконаний на комп'ютері, та підготовку всіх супутніх ресурсів, які потрібні для його запуску і коректної роботи.

2) Відлагодження (Debugging) якщо під час тестування виявляються помилки, цей етап допомагає їх виправити та забезпечити стабільну роботу програми.

3) Деплоймент (Deployment), на цьому етапі програмне забезпечення розгортається на цільовій платформі або сервері та готується до використання.

4) Моніторинг (Monitoring) в контексті пайплайнів здійснюється з метою постійного відстеження роботи програми в реальному часі, що дозволяє оперативно виявляти та реагувати на нові проблеми. Цей процес важливий для

забезпечення стабільності та надійності програмного продукту, а також дозволяє оптимізувати роботу відповідно до потреб користувачів і технічних умовам.

Компанії розробники програмного забезпечення перебувають під сильним тиском необхідності надавати високоякісне програмне забезпечення вчасно через низку причин, включаючи швидкий розвиток технологій, зміну споживчих уподобань. Щоб вирішити цю проблему, індустрія програмного забезпечення постійно шукає відповіді на питання, як пришвидшити час обробки. Серед компаній у Сполучених Штатах сьогодні розробка та експлуатація (DevOps) набуває все більшої популярності завдяки перевагам, які він пропонує. Однак, коли компанії намагаються впровадити процедури DevOps, вони стикаються з низкою проблем. Мета цього аналізу – визначити, які аспекти процесу DevOps можна покращити за допомогою подальших досліджень Основний підхід дослідження зосереджений на всебічному аналізі літератури для визначення ключових елементів, що позитивно впливають на реалізацію. Сьогодні розробники програмного забезпечення можуть продовжувати вдосконалювати свій досвід, підготуватись до наступного кроку DevOps за рахунок зростання, доставки та забезпечення якості. Лідери будь-якої команди, повинні бути готові до того, що їм доведеться зіткнутися з величезними перешкодами. З часом DevOps став популярним інструментом для подолання цих викликів. Покращена співпраця та автоматизація покращила його цикл , дозволяючи розробникам швидше розробляти, тестувати та розгортати програми. DevOps залишатиметься популярним у Сполучених Штатах, де постійні оновлення революціонізують доставку програмного забезпечення практично безмежній споживчій базі. Багато спостерігачів галузі вважали, що 2018 рік – це рік, коли DevOps нарешті повністю розкриє весь свій потенціал і стане галузевою нормою [4].

Наразі багато компаній покладаються на інформаційні технології як основу своїх бізнес-процесів. Було здійснено кілька закупівель інфраструктури та створено кілька додатків. Однак деякі компанії все ще роблять ці дві речі повільно або вручну, що робить їхню технологічну стратегію неефективною, робить

компанії недостатньо гнучкими до змін у конкуренції на їхньому ринку. Мета полягає в дослідженні, як хмарна архітектура (CNA) та культура DevOps можуть прискорити ці два процеси за рахунок автоматизації. Використаний метод аналізу це використання моделі DevOps в рамках життєвого циклу розробки програмного забезпечення (SDLC). Пропонується впровадження інфраструктури як коду (IaC), систем контролю версій (VCS) та безперервної інтеграції/безперервного розгортання (CI/CD). Результати демонструють, що використання CNA та культури DevOps може скоротити час, необхідний для забезпечення інфраструктури та розгортання додатків, зменшити кількість людських помилок, покращити узгодженість документації та підвищити загальну надійність системи в середовищі компанії. Результати можуть розширити базу знань про синергію між CNA та DevOps. Вони також можуть бути корисними для інших компаній, які хочуть одночасно впроваджувати CNA та DevOps у своїх організаціях [5].

Використання конвеєрів при розробці програмного забезпечення є ключем до забезпечення якості, ефективності та надійності програмних продуктів. За допомогою конвеєра можна автоматизувати багато рутинних операцій, швидко виявляти та виправляти помилки та прискорювати процес розгортання програм. Розробники, які вміло використовують пайплайни, отримують можливість створювати високоякісне програмне забезпечення [6].

1.2 Програмні засоби розробки пайплайнів

DevOps – це практика, що об'єднує розробників та операторів для спільної роботи над розробкою та експлуатацією програмного забезпечення. Одним з ключових аспектів є автоматизація процесів, що дозволяє пришвидшити розробку та забезпечити ефективнішу доставку продукту.

Автоматизація процесів відіграє ключову роль у парадигмі DevOps, сприяючи на прискорення розробки та ефективнішій доставці продукту.

Проблеми ,що пов'язані із забезпеченням функціональної коректності коду, є значними і вимагають комплексного підходу. Ефективне впровадження, розробка та розгортання програмного забезпечення здійснюються за допомогою пайплайнів GitActions CI/CD. Основа автоматизації процесів DevOps полягає у застосуванні базових знань Git, що дозволяє фіксувати та перевіряти код з високою ефективністю. В процесі освоєння цих технологій відбувається також знайомство з методами розгортання коду в різноманітних середовищах та розширення функціоналу конвеєрів CI/CD [7].

У цьому аналізі розглянуто різні інструменти, що використовуються в DevOps pipelines, їхні функціональні можливості та те, як вони взаємодіють один з одним.

Автоматизація збірки (Continuous Integration) є важливим аспектом сучасної розробки програмного забезпечення, включаючи автоматизоване об'єднання коду з репозиторію, його перевірку на наявність помилок і створення виконуваних файлів або артефактів. Виконання цього процес допомагає підтримувати високу якість коду та зменшення часу на виявлення та виправлення помилок.

Автоматизація тестування (Continuous Testing) використовується для автоматичного тестування коду на різних етапах конвеєру. Включаючи модульне тестування, інтеграційне тестування та тестування безпеки, забезпечуючи систематичне і вчасне виявлення вразливостей та дефектів.

Автоматизація розгортання включає Continuous Deployment і Continuous Delivery (CD), де Continuous Deployment автоматично розгортає код на робочому сервері, а Continuous Delivery розгортає код на тестовому з можливістю подальшого ручного розгортання. Підвищуючи ефективність розгортання програмних рішень, зменшує ризики затримки та забезпечує надійність використання продукту кінцевими користувачами.

Автоматизація зворотного зв'язку (Continuous Feedback) включає моніторинг робочого середовища, збір важливих метриків та повідомлення про помилки. Дозволяючи командам швидко реагувати на проблеми та оптимізувати процеси, що підтримує стабільну та ефективну роботу програмних продуктів.

Автоматизація операцій (Continuous Operations) включає автоматичне управління інфраструктурою, масштабування систем, резервне копіювання та інші критичні операції, що забезпечують надійність і доступність сервісів. Цей процес є досить важливим для підтримки високої доступності та ефективності інфраструктурних ресурсів, необхідних для сучасного програмного забезпечення [8].

Проаналізуємо основні інструменти DevOps та середовищ розробки конвеєрів (рисунок 1.1).

Рисунок 1.1 – Інструменти DevOps

Для версійного керування використовують Git. Це система контролю з відкритим вихідним кодом, призначена для ефективної роботи з проектами будь-якого масштабу. Основні переваги: простоту вивчення, компактність і висока швидкість обробки даних. Однією з головних особливостей Git є можливість легко створювати локальних гілки, можливість стадійного додавання змін та підтримка різноманітних робочих процесів [9].

Для автоматизації процесів збірки, тестування та розгортання програмного забезпечення використовується Jenkins, що полегшує процес розробки програмного забезпечення, забезпечуючи постійну інтеграцію та доставку.

Система працює на серверах, що підтримують сервлет-контейнери, наприклад Apache Tomcat, і підтримує інструменти керування версіями. Jenkins дозволяє автоматизувати проекти, що використовують Apache Ant, Apache Maven та sbt, виконувати довільні скрипти оболонки та команди Windows batch [10].

Для розробки, доставки та запуску додатків використовується Docker. Цевідкрита платформа, яка дозволяє відокремлювати додатки від інфраструктури для швидкої доставки програмного забезпечення, що спрощує життєвий цикл розробки, забезпечуючи розробникам стандартизоване середовище та полегшуючи розгортання додатків у виробництво. Його функціонал підтримує полегшені контейнери, які містять все необхідне для запуску додатку, що забезпечує високу портативність та ефективність використання ресурсів [11].

Як система відкритого коду для автоматизації розгортання, масштабування та управління контейнеризованими додатками використовують Kubernetes ,що об'єднує контейнери в логічні одиниці для зручного управління та виявлення. Цей інструмент базується на багаторічному досвіді Google та кращих практиках спільноти [12].

Серед сучасних інструментів управління інфраструктурою як кодом особливе місце займає Terraform, який дозволяє безпечно створювати, змінювати та версіонувати ресурси хмарної та локальної інфраструктури. Завдяки Terraform, можна описувати ресурси в конфігураційних файлах, які легко версіонувати, повторно використовувати та ділитися ними [13].

На сьогоднішній день один з найбільш використовуваних інструментів автоматизації відкритого коду є Ansible. Він дозволяє автоматизувати управління конфігурацією системи, безперервне розгортання програмного забезпечення та виконання оновлень без зупинки сервісу [14].

Графічний інструмент Grafana, відомий своєю широкою функціональністю та високою популярністю, використовується для візуалізації даних, моніторингу та отримання сповіщень. Він підтримує різноманітні джерела даних, включаючи: Prometheus, InfluxDB та ElasticSearch. Користувачі мають можливість створювати гнучкі та ефективні для аналізу метрик та логів інфраструктури та додатків [15].

Одним з головних хмарних сервісів у світі є Amazon Web Services (AWS). Він надає обчислювальні потужності, бази даних, місце зберігання та інструменти для розробників та машинного навчання. AWS допомагає підприємствам масштабуватися та рости, спрощуючи управління інфраструктурою, розгортання коду, автоматизацію процесів випуску програмного забезпечення та моніторинг продуктивності додатків [16].

Для створення та керування віртуальними машинами в уніфікованому та простому у використанні середовищі забезпечує Vagrant. Цей інструмент спрощує налаштування розробницького середовища, дозволяючи швидко створювати ізольовані та портативні робочі середовища, які легко відтворювати на різних платформах [17].

Одним з інструмент для розробки пайплайнів є високорівнева мова програмування Python, яка підтримує кілька парадигм програмування, включаючи процедурну, об'єктно-орієнтовану та функціональну. Вона відома своєю читабельністю та спрощенням процесу програмування [18].

Необхідним інструментом для робочих процесів та CI/CD для багатьох розробників стає GitHub Actions. Вона дозволяє автоматизувати завдання, такі як тестування, збірка та розгортання програмного забезпечення, безпосередньо з репозиторію на GitHub [19].

Зручний інтерфейс забезпечує командний рядок та інтерпретатор командного рядка в UNIX-подібних операційних системах Bash. Він дозволяє виконувати різноманітні завдання, включаючи запуск програм, керування файлами та каталогами, а також автоматизацію процесів за допомогою скриптів [20].

Технологія для автоматизації обробки зображень є Levity. Вона використовує алгоритми машинного навчання для класифікації, розпізнавання об'єктів та вилучення важливих даних з візуальних матеріалів. Це значно зменшує потребу в ручній обробці та аналізі зображень, забезпечуючи швидку і точну обробку великих наборів даних [21].

Відкрите програмне забезпечення InfluxDB, створене для управління даними часових рядів, спеціалізується на ефективному зберіганні та обробці цих даних. Використовується в таких ключових сферах, як телеметрія, енергетика та інтернет речей. Однією з його відмінних особливостей є наявність SQL-подібної мови запитів і вбудованих функцій для агрегації даних, що робить його\ ефективним інструментом для аналізу великих обсягів інформації. Часто InfluxDB інтегрується з іншими інструментами для створення повноцінних систем моніторингу та аналізу даних [22].

Технологія для створення розширених пошукових рішень з швидким індексуванням і запитами є ElasticSearch яка заснована на бібліотеці Lucene, вона дозволяє обробляти великі обсяги неструктурованих даних у реальному часі. ElasticSearch підтримує складні функції фільтрації та агрегації, полегшуючи інтеграцію в різноманітні застосунки для глибшого аналізу даних. Часто використовується у складі стеку ELK для логування та візуалізації даних [23].

Розширення стандартного сервісу Rekognition, Amazon Web Services (AWS) Rekognition Custom Labels, дозволяє користувачам створювати, тренувати та налаштовувати власні моделі машинного зору для розпізнавання специфічних об'єктів чи сцен. Це забезпечує гнучкість у підготовці моделей для ідентифікації об'єктів, які можуть бути неправильно розпізнані або взагалі не розпізнані загальними алгоритмами машинного зору [24].

Платформа Roboflow надає широкий спектр функціональних можливостей для допомоги розробникам та підприємствам у створенні та впровадженні власних моделей комп'ютерного зору. Вона пропонує можливість тренування власних моделей на хостингових GPU, використання фундаментальних моделей або вибір з 50 000 попередньо навчених моделей з Roboflow Universe. Це дозволяє користувачам адаптувати фундаментальні моделі до своїх конкретних потреб у даних, підвищуючи ефективність та знижуючи затримку [25].

Одна з головних технологій для роботи з візуальним ШІ є Hasty. Вона істотно спрощує весь цикл машинного навчання. Вона охоплює всі етапи від анотації даних до навчання моделей та їх впровадження, автоматизуючи до 90%

ручної роботи. Hasty також надає можливість побудови моделей без написання коду та підтримує керовані MLOps [26].

Ці інструменти спільно допомагають забезпечити ефективний процес розробки, тестування та розгортання програмного забезпечення. Вибір конкретних інструментів залежить від потреб проекту та вимог до автоматизації.

1.3 Засоби редагування зображень

Засоби редагування зображень можна класифікувати на кілька основних категорій, таких як: растрові редактори, векторні редактори та спеціалізоване програмне забезпечення для автоматизованої обробки. Прикладом растрових редакторів є Adobe Photoshop та GIMP. Вони забезпечують широкий спектр інструментів для маніпулювання пікселями, що дозволяє детально коригувати кольори, текстури та контрасти зображень. Векторні редактори, наприклад Adobe Illustrator та CorelDRAW, використовуються для створення та редагування векторної графіки [27].

Інтеграція штучного інтелекту в програми для редагування зображень збільшила їхні можливості. Його використовується для автоматизації рутинних задач, таких як ретушування шкіри, корекція кольору та компонування зображень.

Технології машинного навчання використовуються для точнішої та ефективнішої обробки, аналізуючи великі обсяги зображень для вдосконалення алгоритмів обробки [28].

Редагування растрових зображень зображень – це процес зміни пікселів зображення для досягнення потрібного візуального ефекту або корекції. Завдання цих інструментів обробляти зображення, що складаються з великої кількості пікселів, кожен з яких містить інформацію про колір і яскравість. Так як кожен піксель може бути індивідуально відредагований, растрові редактори

забезпечують високу точність обробки зображень, що є особливо важливим у галузях, де потрібна деталізована візуалізація, таких як фотографії.

Основними інструментами растрових редакторів є кисті, ластики, засоби для редагування, вибір кольору та фільтри. Вони дають змогу виправляти недоліки, змінювати колірний баланс і додавати різні ефекти без зміни структури зображення. Наприклад, можна застосовувати фільтри для досягнення ефекту старіння або імітації різних типів освітлення.

Сучасні растрові редактори використовують технології штучного інтелекту та машинного навчання для автоматизації багатьох процесів, таких як видалення фону, розпізнавання обличчя та оптимізація кольору, що зменшує кількість часу та зусиль, необхідних для редагування зображень [29].

Векторне редагування зображень ґрунтується на створенні та маніпулюванні графічними елементами, такими як точки, лінії, криві та полігони, що визначаються математичними формулами. Векторні зображення відрізняються від растрових ,тим що вони не втрачають якості при масштабуванні або трансформації, тому що вони базуються на геометричних визначеннях ,а не на пікселях. Однією з головних переваг векторного редагування є гнучкість у зміні розмірів та форм без втрати якості, що робить його ідеальним для створення логотипів, технічних схем, ілюстрацій та іншої графіки, яка потребує частих змін розмірів або детального редагування.

Векторні редактори дозволяють користувачам з точністю впливати на форму та розташування елементів. Крім того, ці програми надають розширені можливості для роботи з колірними палітрами, застосування градієнтів і використання шарів, що дозволяє ефективно організовувати складні дизайни.

Крім технічних можливостей, векторні редактори підтримують роботу з текстом, що робить їх незамінними у створенні графічних елементів, де необхідне поєднання тексту та графіки. Текст у векторних редакторах можна легко адаптувати та інтегрувати з графічними елементами, що забезпечує необмежені можливості для дизайну [30].

1.4 Постановка завдань кваліфікаційної роботи

На основі проведених досліджених методів та алгоритмів було виділено наступні завдання які необхідно виконати для успішного проектування DevOps пайплайну:

- проаналізувати підходи до використання пайплайнів при розробці пз;
- проналізувати засоби опрацювання зображень;
- розробити алгоритм обробки цифрових зображень;
- розробити життєвий цикл розгортання проекту;
- розробити алгоритм опрацювання зображень в хмарному середовищі;
- здійснити порівняльний аналіз розроблено підходу з аналогами.

2 АЛГОРИТМ РОЗРОБКИ ПАЙПЛАЙНІВ

2.1 Алгоритм обробки цифрових зображень

Цей розділ детально описує архітектуру програмного забезпечення, розробленого для обробки цифрових зображень за допомогою використання множини графічних фільтрів. Основна мета програми полягає у наданні користувачам інструментів для модифікації цифрових зображень. Програма була реалізована на мові Python, і для обробки зображень використовується бібліотека OpenCV. Ця бібліотека обрана завдяки своїй продуктивності та широким можливостям у сфері обробки графіки, що робить її зручною для реалізації вимог проекту. В цьому підпункті демонструється послідовність кроків, що виконуються в програмі, та забезпечує візуальне розуміння процесу обробки зображень. Схема починається з ініціалізації програми та проходження через ряд взаємопов'язаних етапів: введення даних користувачем, валідація введеного шляху, завантаження та перевірка зображення, застосування вибраних фільтрів, та завершується збереженням обробленого зображення. Основні блоки та зв'язки між ними викладені нижче, що допомагає зрозуміти логічну послідовність дій та рішень у програмі.

1) Старт програми відбувається при запуску скрипта. Виконання коду починається з визначення вихідної точки, що у багатьох програмах на Python визначається за допомогою конструкції if __name__ == "__main__" ,що забезпечує, що блок коду всередині цієї перевірки запуститься тільки якщо програму запущено як основний скрипт, а не імпортовано як модуль. Такий підхід дозволяє контролювати результати програми при різних сценаріях використання.

2) Під час виконання наступного кроку ,програма просить користувача ввести шлях до зображення. Взаємодія з програмою відбувається через консоль. Використовується функція input(), що полегшує збір інформації, введеної користувачем. Процес отримання необхідної інформації та подальшого завантаження зображення спрощується завдяки використанню цієї

функції.Спочатку програма просить користувача повідомити місцезнаходження файлу зображення. Отримавши цю інформацію, програма перетворює її на текстовий рядок, що буде використано під час завантаження. Під час виконання цієї дії завантаження зображення ще не відбувається , замість цього, програма зосереджується на зборі необхідних даних, що будуть використані в подальших кроках.

3) Наступний крок розробленої програми це перевірка шляху, введеного користувачем. Ця перевірка слугує гарантом, що всі майбутні операції обробки зображень виконуватимуться на самому файлі, уникаючи будь-яких випадкових помилок.

Функція була реалізована за допомогою os.path.exists(), що перевіряє наявність файлу за заданим шляхом. При відсутності файлу, генерується повідомлення про помилку і надається інша спроба щоб ввести коректний файл. Виконання цієї функції відіграє важливу роль в управлінні потоком виконання програми, підвищуючи надійність та забезпечуючи безперервність роботи. Застосування цієї методики у програмуванні сприяє оптимізації часу та ресурсів, а також забезпечує користувачам чітке розуміння причин помилок. Під час запуску програми користувач бачитиме всі опції обробки, що можуть застосуватися для обробки зображень. Завдяки використанню цієї функції підвищується зручність інтерфейсу. Вона реалізована в коді через змінну показати_доступні_фільтри(), що виводить у консоль перелік всіх фільтрів. Кожен з фільтр має свій номер та опис, що робить інтерфейс зручнішим та більш інформативним. Використовуючи такий підхід користувачам буде легше орієнтуватися серед доступних опцій та вибирати ті фільтри, які вони бажають застосувати .

4) Наступним кроком розробленої програми буде вибір фільтрів користувачем. Взаємодіючи з програмою користувач вказує, як саме хоче редагувати зображення. Це впливає на всі подальші кроки обробки зображення та визначає кінцевий результат. Користувачі вводять номери фільтрів, які вони

хочуть застосувати, або прямо вказують назви фільтрів, залежно від уподобань або потреб. Використовується функція input() для збору вхідних даних:

```
    вибрані_фільтри_ввід = input("Введіть номери фільтрів (через
кому) або назви фільтрів, розділені комами: ")
```

Також, функціонал програми дозволяє вибрати декілька фільтрів ,що збільшує її гнучкість під час використання для обробки зображень. Програма виконує перевірку та адаптацію введених даних до наявних опцій обробки. Цей процес включає в себе перетворення числових ідентифікаторів у відповідні назви фільтрів, а також перевірку коректності введення текстових назв.

Програма спочатку аналізує введені дані для виявлення та виправлення можливих помилок чи непорозумінь, які могли виникнути під час введення. Використовуючи розроблену логіку обробки введення, програма переконується, що кожен вибір відповідає допустимому фільтру.

5) Наступним кроком програми є те, що користувач вказує шлях для збереження обробленого зображення. Визначає місце на сервері, куди буде збережено фінальний результат обробки зображення.

Функціонал реалізовано за допомогою того ж input() ,що використовується для збору вхідних даних від користувача. Пропонується ввести шлях, де буде зберігатись зображення. Це може бути як абсолютний шлях до папки ,так і назва файлу, якщо вони бажають замінити оригінал або створити новий файл.

```
    шлях_збереження = input("Введіть шлях для збереження
обробленого зображення (або залиште порожнім для заміни оригіналу):
")
```

Після цього крок виконується завантаження зображення за вказаним шляхом. Це є важливим для всіх наступних операцій з обробки, адже без успішного завантаження файлу подальша робота програми стає неможливою. Завантаження зображення перевіряє валідність введеного шляху і фактичну наявність файлу.

На цьому етапі програма використовує функцію cv2.imread(), що є частиною бібліотеки OpenCV, призначеної для роботи з зображеннями. Функція спробує відкрити зображення у вказаному місці і, у разі успіху, поверне об'єкт зображення, який можна використовувати в подальших обробках. Якщо спочатку програма спробує завантажити зображення, а потім перевірить результат, і якщо змінна image має значення None, це вказуватиме про те, що зображення не було знайдено за цим шляхом, або файл не відповідає підтримуваним форматам. В такому випадку програма виведе повідомлення про помилку та припинить роботу, що уникне спроб обробки неіснуючих даних. Після спроби завантаження зображення, програма аналізує, чи об'єкт зображення був створений правильно.

6) Цей етап програми призначений для завершення процесу обробки шляхом зберігання модифікованого зображення в локації, визначеній користувачем. Фінальна стадія в модифікації зображення, де всі застосовані зміни фіксуються, підготовляючи зображення для подальшого використання чи архівації.

Програма використовує функцію cv2.imwrite(), що вважається надійним інструментом у бібліотеці OpenCV для запису зображень . Залежно від введення користувача про шлях збереження, ця функція забезпечує збереження зображення на вказаному шляху або оновлення оригінального файлу:

Цей крок вважається кульмінаційним для всього процесу обробки зображення і означає остаточне завершення виконання програми. Відображає успішне виконання усіх передбачених дій, від ініціації через обробку до фінального збереження зображення, підсумовуючи результати виконання.

```
    print(f"Оброблене зображення замінило оригінал: 
{шлях_до_зображення}")
```

Реалізація кінцевої функції, що оброблене зображення успішно збережено на сервері, замінивши оригінальний файл або як новий файл за іншим шляхом. Це інформує користувача про завершення основної роботи програми і її готовність до завершення сесії або до нового циклу обробки зображень.

2.2 Життєвий цикл розгортання проекту

Структура пропонованого алгоритму життєвого циклу розгортання проекту подана в додатку Б"

Процес розгортання DevOps відіграє важливу роль у життєвому циклі розробки проекту, інтегруючи розробку та операційні команди з метою підвищення швидкості, ефективності та якості доставки програмного продукту. Основою цього процесу є автоматизація, що охоплює такі етапи як: реалізацію, тестування, розгортання та моніторингу у продуктивному середовищі. Інтегрований підхід не лише сприяє постійному вдосконаленню та виправленню помилок, але й встановлює важливий зв’язок між розробниками та операційними процесами, що є важливим фактором для ефективної реалізації проектів у динамічному середовищі.

1) На початковій стадії роботи ініціюється процес розробки, налаштовуючи робоче середовище. Включаючи оновлення вихідного коду до останньої стабільної версії, що знаходиться в основній гілці репозиторію. Цей крок виключає проблеми різних версій та спрощує подальшу інтеграцію робочого процесу розробників, що включає в себе створення стабільної робочої основи, що базується на останній.

2) Після завершення модифікацій у програмному коді, ініціюється процес фіксації змін у версійному контролі, використовуючи систему Git. Передбачається детальне документування внесених змін, включаючи додавання нових функцій або корекцію помилок. Commit (Коміт) створює запис у репозиторії, що служить для відстеження послідовності змін і забезпечення прозорості розвитку проекту.

3) Етап заповнення конфігураційних файлів для git actions вимагає детального визначення робочого процесу в автоматизованому середовищі CI/CD. Файли формату .yml чи .yaml включають інструкції, які задають послідовність операцій, викликаних при кожному внесенні змін у репозиторій. Такі інструкції

визначають умови запуску процесів, середовища для виконання та команди, що мають бути виконані, і охоплюють весь життєвий цикл розробки від тестування до розгортання. Оптимально структуровані git actions сприяють забезпеченню ефективного та безперебійного потоку робочих процесів, підвищуючи якість та швидкість розробки.

4) Наступним кроком програми це доставка локальних файлів на віддалений репозиторій git. Виконання команди push полягає у синхронізації змін із центральним репозиторієм, розташованим на віддаленому сервері. Забезпечуючи архівацію змін у віддаленому сховищі і запускаючи автоматизовані скрипти, визначені у файлі конфігурації .yml. Активація цих скриптів ініціює процес неперервної інтеграції (CI), та доставки коду і його розгортання на віддаленому сервер (CD) який здійснює перевірки та автоматичні тести, що гарантує якість кінцевого продукту.

Таким чином, процеси збереження змін та відправлення є стартовими точками для всього циклу неперервної доставки, включаючи інтеграцію, тестування, компіляцію та розгортання продукту, що є важливим для забезпечення стабільності та надійності програмного забезпечення.

5) Виконуючи push(пуш) на локальний репозиторій активовується пайплайн CI/CD. Активація пайплайну — це процес, який автоматично запускається після завантаження змін у віддалений репозиторій. Визначені в конфігурації .yml файлі автоматизовані завдання починають виконуватися, включаючи збірку коду, його тестування, перевірку якості, доставку і розгортання коду на хмарному середовищі.

Під час цього етапу система сканує код на предмет помилок компіляції, виконує автоматизовані тести для ідентифікації проблем, які могли бути внесені під час внесення змін. За рахунок цього підвищується загальна надійність програмного забезпечення, забезпечуються швидке виявлення та виправлення помилок та знижує ризик введення дефектів у виробниче середовище. Також пайплайн може включати статичний аналіз коду, що допомагає підтримувати високі стандарти кодування та перевіряти код на відповідність вимогам якості.

Успішне завершення цих завдань є важливим кроком перед тим, як зміни будуть дозволені до злиття з основною гілкою коду, що підтверджує готовність коду до додаткових етапів доставки та розгортання.

CI/CD пайплайн ініціюється після збереження змін у репозиторій, виконуючи автоматизовані процедури, які забезпечують інтеграцію та розгортання коду. Процес починається зі скачування останньої версії проекту, перевірки залежностей та компіляції коду. Наступна дія це виконання ряду автоматизованих тестів, зокрема модульних та інтеграційних, для забезпечення стабільної роботи. Статичний аналіз знаходить неоптимальні частини коду та можливі вразливості. Успіх цих процедур веде до створення детальних звітів про стан коду. У випадку, коли пайплайн не може бути завершено через помилки у коді, це приводить до зупинки всього процесу інтеграції. Коли виконання тестів видає помилку про некоректність ,то система повідомляє про це і вказує на ,що потрібно звернути уваги під час наступного запуску конвеєра. Система повідомляє про помилки, і пропонує рекомендації для їхнього виправлення. Проект залишається на цій стадії, поки всі невирішені проблеми не будуть виправлені та код знову не пройде через конвеєр без помилок. Запобігаючи перенесенню недостовірного або помилкового коду в основну гілку.

6) Після успішного завершення CI пайплайну та затвердження коду наступним етапом є розгортання. Цей процес включає переміщення оновленого або нового програмного продукту з тестового або робочого середовища до основного, щоб забезпечити доступ користувачів до змін. Розгортання може бути автоматизованим або потребувати ручного втручання, залежно від налаштувань та потреб інфраструктури. Важливими компонентами розгортання є перевірка готовності середовища та забезпечення безперебійної роботи під час впровадження змін. Під час цієї стадії здійснюється автоматичне створення резервних копій діючих версій системи, що надає можливість швидко відновити попередні налаштування у разі необхідності. Також часто процес розгортання включає виконання додаткових перевірок. Дозволяючи переконатись що новий реліз не порушує стабільність існуючої інфраструктури. Завершення процесу

розгортання означає те , що програмний продукт успішно пройшов усі етапи від початкової концепції до впровадження у виробниче середовище , готовий до використання користувачами.

7) Після завершення розгортання програмного продукту у виробничому середовищі полягає у моніторингу та підтримці системи. Даний процес включає в себе постійне спостереження за працездатністю додатку, аналіз логів для виявлення можливих помилок та забезпечення відповідності системи заданим показникам продуктивності.

Важливість моніторингу полягає у можливості оперативно реагувати на виникаючі проблеми, які можуть вплинути на доступність та ефективність системи. Допомагаючи вчасно виявляти та виправляти помилки, а також проводити оптимізацію роботи системи для покращення її загальної продуктивності.

Додатково на цьому етапі здійснюється оновлення системи для впровадження покращень або виправлення помилок, виявлених під час експлуатації. Дозволяючи підтримувати програмний продукт у актуальному стані та забезпечує його стабільність та безпеку на довгострокову перспективу.

Таким чином, моніторинг та підтримка є важливою складовою для забезпечення надійної роботи програмного забезпечення, здатного ефективно виконувати свої функції в умовах постійно змінюваної інформаційного середовища. Вони сприяють підвищенню рівня користувацького задоволення та оптимізації ресурсів системи.

2.3 Алгоритм опрацювання зображень в хмарному середовищі

Структура пропонованого алгоритму опрацювання зображень в хмарному середовищі подана в додатку В"

1) Початок. Процес розпочинається з точки входу, що сигналізує про ініціалізацію алгоритму. На цьому етапі всі необхідні ресурси готуються до подальшого використання, а саме середовище налаштовується для обробки вхідних даних.

2) Введення шляху до зображення. Користувач вводить шлях до файлу з зображенням, який потрібно обробити. Це може бути абсолютний або відносний шлях у файловій системі. Введення даних є обов'язковим для забезпечення правильної роботи алгоритму на наступних етапах.

3) Перевірка наявності зображення. Алгоритм перевіряє, чи існує зображення за вказаним шляхом. Якщо зображення існує, алгоритм переходить до наступного кроку, а саме введення вибору фільтрів. Якщо зображення не існує, генерується повідомлення про помилку.

4) Повідомлення про помилку. У випадку відсутності зображення за вказаним шляхом генерується повідомлення про помилку. Повідомлення інформує користувача про те, що зображення не знайдено, і пропонує повернутися до блоку введення шляху до зображення для повторної спроби.

5) Введення вибору фільтрів. Під час цього кроку користувач може вибрати один або декілька фільтрів для редагування фото, використовуючи потенціал програми. Програма перевіряє, чи були використані існуючі фільтри, оскільки реалізовано як цифровий ввід, так і повні назви фільтрів. Це забезпечує гнучкість і зручність для користувача.

6) Фільтри 1, 2, 3. Під час виконання цього кроку програма переформатовує з текстового повідомлення в чисельне значення для розуміння, який фільтр було вибрано. Це перетворення необхідне для подальшого застосування фільтрів у процесі обробки зображення.

7) Застосування фільтрів. Після того як програма визначила, які фільтри були вибрані, вона починає застосовувати їх до зображення в кеші, використовуючи різні функції та алгоритми обробки зображень. Цей процес включає виконання математичних операцій і перетворень для досягнення бажаного ефекту на зображенні.

8) Введення шляху для збереження зображення. Програма запитує у користувача шлях для збереження обробленого зображення або заміни оригіналу, якщо не буде вказано інший шлях. Це дозволяє користувачеві зберегти результати обробки у зручному для нього місці.

9) Збереження зображення на сервері. Після введення шляху для збереження зображення фотографія зберігається у вибраній директорії на сервері. Це дозволяє зберегти результати обробки у хмарному середовищі для подальшого доступу.

10) Збереження зображення на ПК. Якщо користувач захоче зберегти фото на локальному комп’ютері його потрібно буде використовувати сторонню програму для викачування фото, тому що в хмарному середовищі, такому як Ubuntu, може бути обмежена можливість безпосереднього перегляду і завантаження зображення на локальний комп'ютер. Для цього використовується програма FileZilla, яка дозволяє завантажувати файли з хмарного середовища на ПК. Після налаштування FileZilla можна прив'язати до хмарного середовища, що дозволяє користувачеві завантажувати різні файли і папки.

11) Кінець. Завершення процесу алгоритму обробки у хмарному середовищі сигналізує про успішне завершення всіх етапів обробки зображення. Усі ресурси, що використовувалися в процесі, звільняються, а користувач отримує готовий результат обробки.

3 ПРОГРАМНА РЕЛІЗАЦІЯ

3.1 Результат опрацювання зображень

В цьому розділі розглянуто ефективність розробленої програми. Для ілюстрації буде наведено кілька прикладів роботи програми для обробки зображень, що продемонструють її можливості та функціональність.

Для перевірки використано фотографія початкової сторінки wunu.edu.ua (рисунок 3.1).

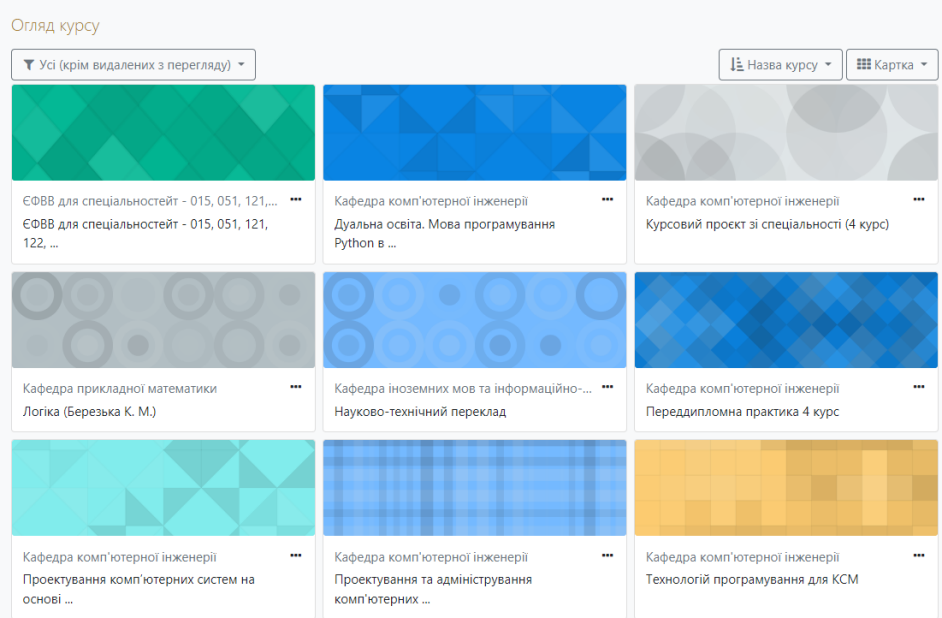

Рисунок 3.1 – оригінал фото

Тестування програми на працездатність розглянемо з фільтру розмиття ця функція реалізована в коді:

```
for selected_filter in selected_filters:
    if selected_filter == 'розмиття':
        image = cv2.blur(image, (10, 10))
```

Реалізація методу використовує функцію cv2.blur() бібліотеки OpenCV, що виконує приглушення високочастотних деталей зображення. Вона може бути корисною для зниження рівня шуму або як попередній крок для подальшої

обробки, такої як контурне виявлення. Розмиття застосовується до зображення за допомогою простого лінійного фільтра, де кожен піксель у вихідному зображенні набуває значення, що є середнім значенням пікселів у його оточенні, розмір якого визначається параметром ядра (10x10 пікселів). Використання цього методу ефективно розмиває зображення, зменшуючи деталізацію та чіткість, що може покращити загальне сприйняття зображення при обробці (рисунок 3.2).

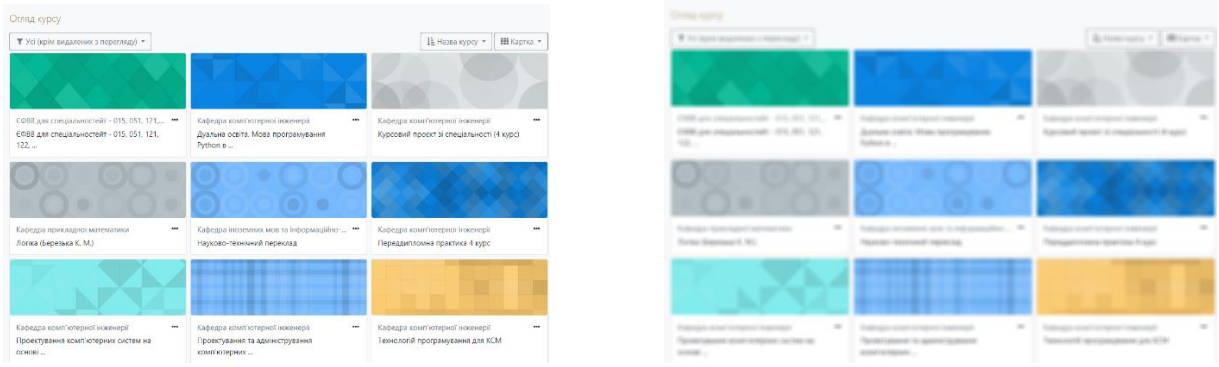

Рисунок 3.2 – оброблене фото

Тестування іншого фільтру під назвою “контурне виявлення ”, реалізований за допомогою методу cv2.Canny() від бібліотеки OpenCV (рисунок3.3).

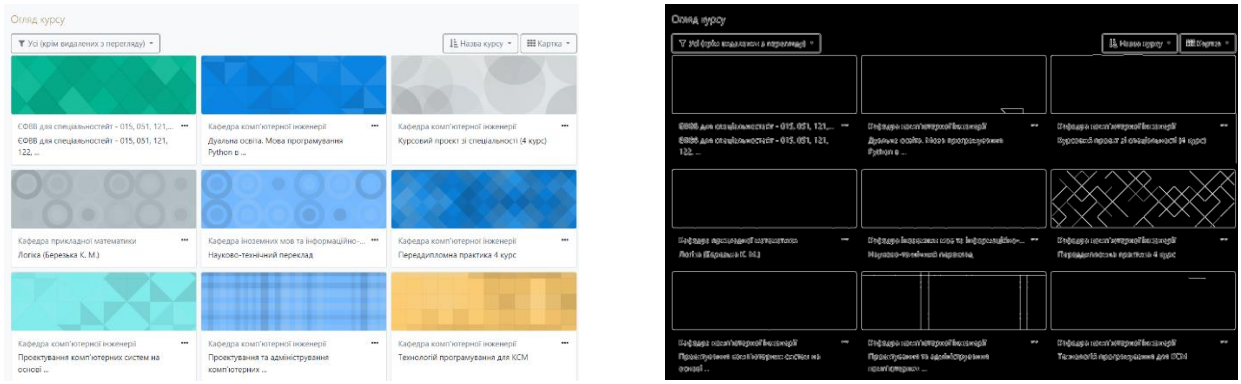

Рисунок 3.3 – використання фільтру контурного виявлення

Код функції

```
elif selected_filter == 'контурне виявлення':
image = cv2.Canny(image, 100, 200)
image = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)
```

Перевірка на правильність файлу, якщо користувач вибирає файл якого не існує програма повідомить про це (рисунок 3.4).

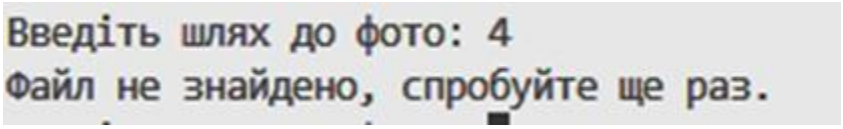

Рисунок 3.4 – перевірка файлу

Дана функція реалізована за допомогою модуля **os** з Python, який використовується для взаємодії з файловою системою. Виконується перевірка , щоб визначити, чи існує файл за вказаним шляхом перед тим, як спробувати його завантажити для обробки .

Підвищення яскравості реалізована за допомогою функції increase_brightness, в коді це реалізовано так:

```
 def increase_brightness(image):
        if image.ndim == 2:  # перевірка, що зображення не
одноканальне
            return image
        hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
        hsv_image [:, :, 2] = np.clip(hsv_image [:, :, 2] * 1.5,
0, 255)
        return cv2.cvtColor(hsv_image, cv2.COLOR_HSV2BGR)
```

Функція для підвищення яскравості зображення на 50% реалізовано шляхом зміни компоненту яскравості в кольорову просторі HSV (Hue, Saturation, Value). Спершу програма перевіряє, чи зображення не є одноканальним, оскільки одноканальні зображення (відтінки сірого) не містять кольорової інформації, тому такі зображення повертаються без змін. Після цього зображення перетворюється з стандартного BGR колірного простору (який використовується в OpenCV) до HSV. HSV компонент Value, що відповідає за яскравість зображення, дозволяючи безпосередню маніпуляцію цим параметром для зміни яскравості. Значення компонента V множиться на 1.5, ефективно підвищуючи

яскравість на 50%, і обмежується максимумом 255 за допомогою функції np.clip, щоб уникнути переповнення кольору. Завершальним кроком є конвертація зображення назад у BGR для подальшої обробки або відображення, завершуючи процес підвищення яскравості. Використання цього методу є ефективним та зберігає якість зображення, забезпечуючи контрольоване підвищення яскравості без втрати деталей у темних або яскравих областях (рисунок 3.5) .

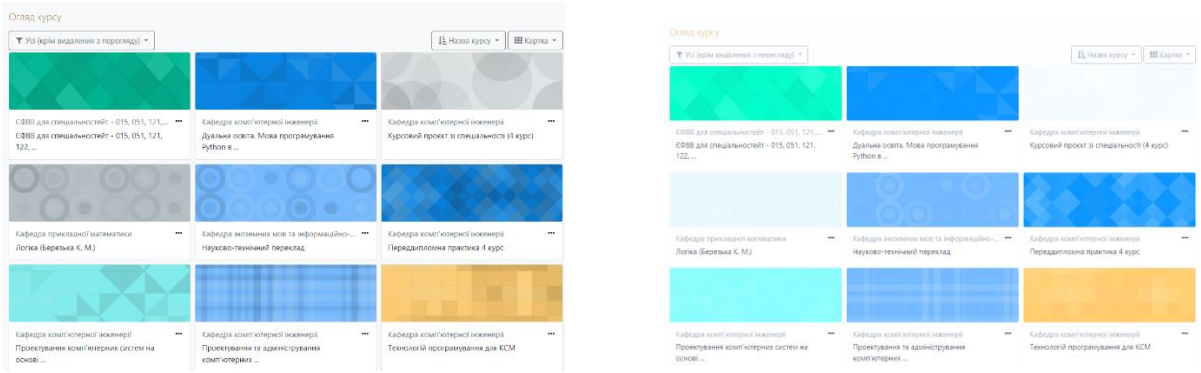

Рисунок 3.5 – функція збільшення яскравості

Однією з важливих особливостей розробленої програми є можливість одночасного застосування декількох фільтрів до одного зображення. Ця функція дозволяє експериментувати з візуальними стилями та обробками без повторного завантаження зображення. Програма реалізована таким чином, що може приймати від користувача список фільтрів, які потім застосовуються по порядку до вибраного зображення, що робить її гнучкою та ефективною у використанні (рисунок 3.6).

```
Введіть шлях до фото: 1.png
Доступні фільтри:
1. розмиття
2. підвищення насиченості
3. контурне виявлення
4. відтінки сірого
5. негатив
6. збільшення яскравості
7. зменшення яскравості
8. агат
9. темно-синій
10. рожевий
Введіть номери фільтрів (через кому) або назви фільтрів, розділені комами:розмиття,6
Введіть шлях для збереження обробленого зображення (або залиште порожнім для заміни оригіналу): 3.png
```

Рис 3.6 – Варіація вибору фільтрів

Результат використання двох фільтрів такі як: розмиття і збільшення яскравості (рисунок3.7).

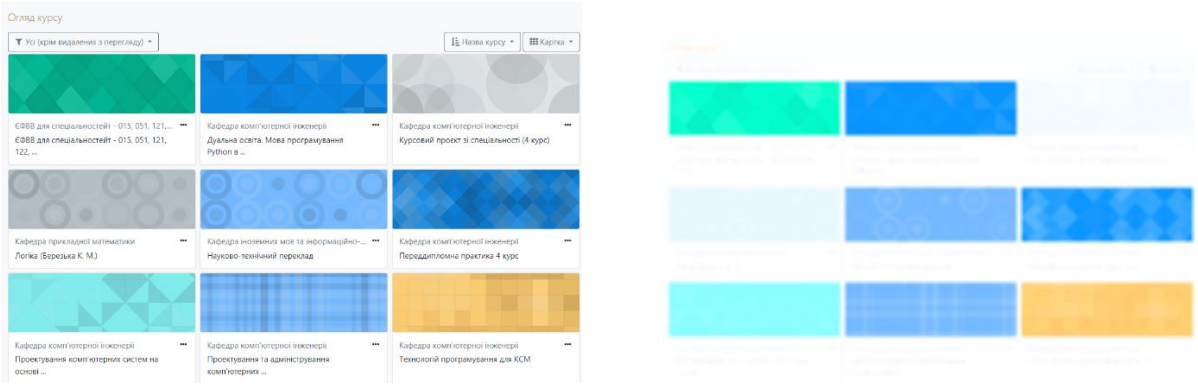

Рис 3.7 – Накладання фільтрів

Якщо спочатку застосувати фільтр розмиття, а потім збільшення яскравості, то буде досягнуте послідовне зменшення видимих деталей та посилена інтенсивність кольорів зображення. Розмиття здійснюється за допомогою середнього фільтра, який замінює кожен піксель середнім значенням пікселів у визначеному радіусі, що ефективно знижує деталізацію та гостроту зображення, одночасно усуваючи шум і вирівнюючи текстурні різниці. Підвищення яскравості реалізується через множення значень у каналі яскравості HSV-простору.

3.2 Результат роботи CICD pipeline

Робота з CI/CD (Continuous Integration/Continuous Deployment) пайплайнами є важливою у сучасних практиках розробки програмного забезпечення. Автоматизуючи тестування і випуск програм, підвищуючи їх ефективність та знижуючи ризик помилок, прискорюючи доставку продукту. Постійна інтеграція змін у код допомагає виявляти проблеми на ранніх стадіях і підтримувати високу якість продукту. Автоматизоване розгортання доставляє оновлення до робочих

середовищ, забезпечуючи безперервне покращення функціональності та безпеки. Зазначені вище процеси забезпечують стабільність і безпеку програмного забезпечення, а також здатність адаптуватися до швидко змінних вимог ринку. Для управління завданнями автоматизації використовується конфігураційний файл форматі YAML, який забезпечує зручний і читабельний спосіб для визначення етапів конвеєру, дозволяючи легко налаштовувати та розширювати автоматизацію процесів (рисунок 3.8).

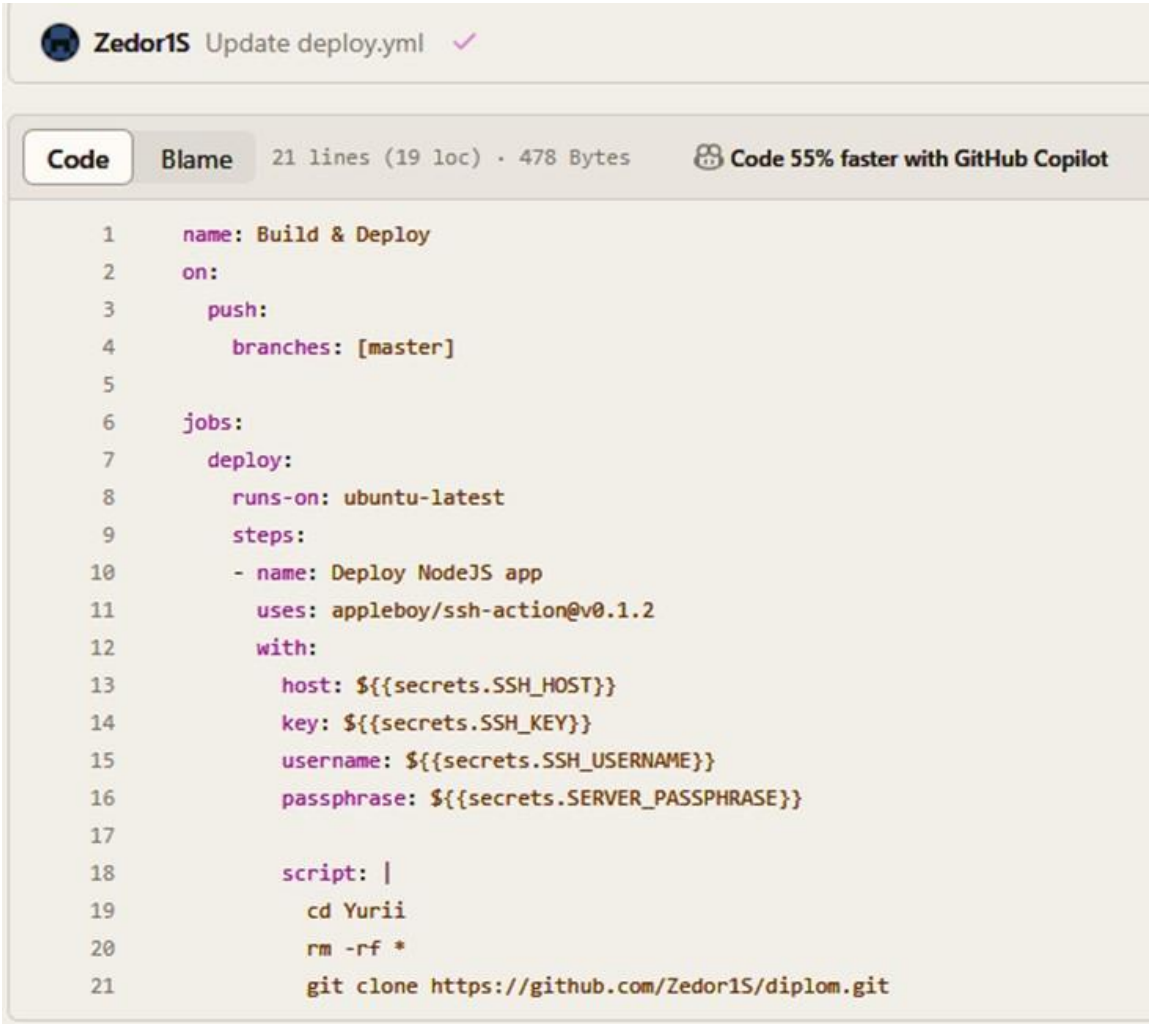

Рис 3.8 – YML file

У файлі міститься визначення робочого процесу під назвою "Build & Deploy", що активується на подію push у гілку master репозиторію. Забезпечуючи,

що кожен commit, здійснений до головної гілки, ініціює зазначені дії автоматизації. Коли будь-які зміни відбуваються з репозиторієм ,на вітці master, то спрацьовує тригер який починає виконувати YML file .

У конфігураційному файлі визначено завдання під назвою deploy, яке ініціюється на віртуальній машині з найновішою версією Ubuntu.

У рамках заданої роботи процес включає декілька кроків, серед яких ключовим є крок під назвою "Deploy NodeJS app", який використовує дію appleboy/ssh-action@v0.1.2 для здійснення SSH-з'єднання з віддаленим сервером. Ця дія дозволяє безпечно здійснювати дистанційну інтеракцію з сервером, використовуючи такі параметри: ім'я хоста, збережене в секретах GitHub для забезпечення конфіденційності та безпеки, приватний ключ SSH для аутентифікації, який також є засекреченим, ім'я користувача для доступу до сервера та пароль, який використовується за потреби (рисунок 3.9).

Рисунок 3.9 – створення секретів

Використання цих параметрів у конфігураційному файлі підвищує рівень безпеки процесу розгортання, забезпечуючи, що доступ до сервера мають лише особи з високим рівнем доступу ,що володіють відповідними криптографічними ключами та інформацією, що знижує ризики несанкціонованого доступу та забезпечує стійкість деплоймент процесу.

Для запуску пайплайну ,потрібно виконати завантаження програми на сервер, використовуючи push ,але для початку повинен бути commit за правилами Git (рисунок 3.10).

```
yuram@DESKTOP-8B1R1MJ MINGW64 ~/OneDrive/Рабочий стол/update (master)
$ git commit -m "update"
[master 37ccfcf] update
 3 files changed, 33 insertions(+), 94 deletions(-)
 delete mode 100644 3.png
```

Рисунок 3.10 – використання commit

У системах контролю версій, таких як Git, commit є основною одиницею для зберігання змін, внесених у файл або групу файлів репозиторія. Він створює точний знімок всіх файлів репозиторія на момент здійснення збереження, що дозволяє детально відстежувати історію змін. Включаючи метадані, такі як унікальний ідентифікатор (зазвичай SHA-1 хеш), час і дату створення, а також дані про автора. Він також має описове повідомлення, яке дає контекст або опис змін, допомагаючи іншим розробникам або автору зрозуміти внесені зміни у майбутньому (рисунок3.11).

```
yuram@DESKTOP-8B1R1MJ MINGW64 ~/OneDrive/Рабочий стол/update (master)
$ git push -u origin master
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 12 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 115.43 KiB | 28.86 MiB/s, done.
Total 4 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Zedor1S/diplom.git
   8906019..37ccfcf  master -> master
branch 'master' set up to track 'origin/master'.
```

Рисунок 3.11 – push на сервер

Перекидання файлів з локального репозиторію на віртуальний ,виконується за допомогою команди push. Після того як файли уже знаходяться на репозиторії

git автоматично спрацьовує перший тригер і виконується конфігураційний файл (рисунок 3.12).

Рисунок 3.12 – успішне виконання CI/CD

Після успішного завершення дій, виконується вхід на сервер та введення команди для ініціації скрипта, який автоматично запускає програму для обробки зображень і відображає функціонал розробленої програми (рисунок 3.13).

```
root@ubuntu-s-1vcpu-1gb-intel-nyc1-01:~# bash script.sh
Введіть шлях до фото: 1.png
Доступні фільтри:
1. розмиття
2. підвищення насиченості
3. контурне виявлення
4. відтінки сірого
5. негатив
6. збільшення яскравості
7. зменшення яскравості
8. агат
9. темно-синій
10. рожевий
Введіть номери фільтрів (через кому) або назви фільтрів, розділені комами:1,7,4
Введіть шлях для збереження обробленого зображення (або залиште порожнім для зам
іни оригіналу): 4.png
Оброблене зображення збережено за шляхом: 4.png
root@ubuntu-s-1vcpu-1gb-intel-nyc1-01:~#
```

Рисунок 3.13 – виконання скрипта

Для того щоб видобути відредаговане фото, потрібно скористатись FileZila, тому що на пряму з Ubuntu її неможливо, через відсутність такої функції, перегляд також недоступний (рисунок 3.14).

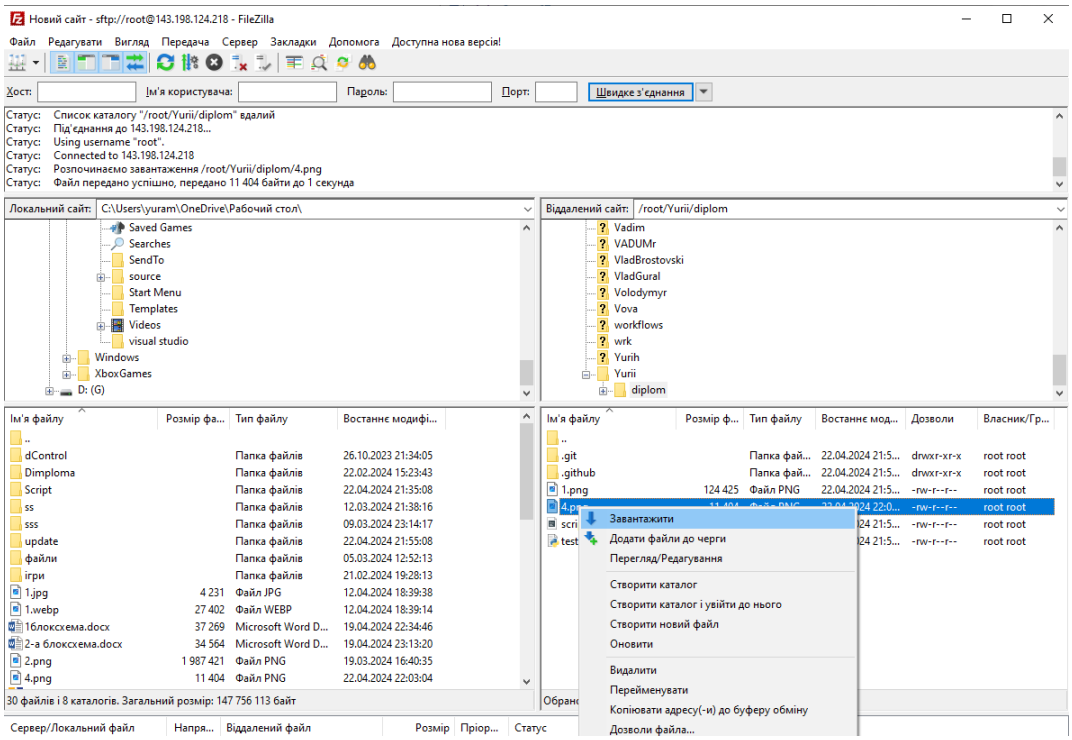

Рисунок 3.14 – програма для видобування фото з сервера Ubuntu

Після успішного завантаження фотографії можливий перегляд її на персональному комп'ютері (рисунок 3.15).

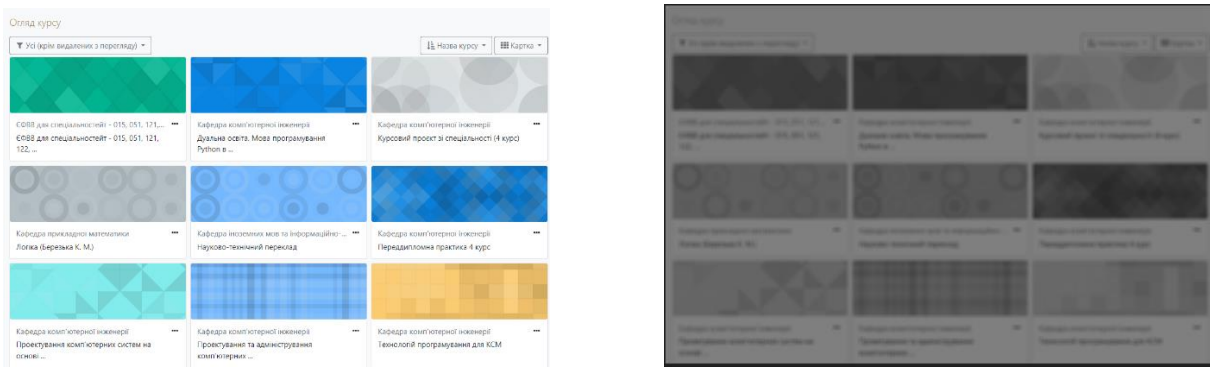

Рисунок 3.15 – Результат виконання фільтрів

Результат роботи CI/CD пайплайну, що був описаний раніше, являє собою ефективний механізм для автоматизації процесів розробки та деплойменту програмного забезпечення. Створений пайплайн включає в себе використання дій SSH для віддаленого виконання команд на сервері, забезпечуючи надійність та безперервність розгортання. Використання останньої версії операційної системи та її інструментів значно знижує ризики, пов'язані з використанням застарілих технологій, та підвищує загальний рівень безпеки в процесі розгортання.

Забезпечуючи актуальність та стійкість системи до нових загроз і вразливостей. Зосередження на роботі з головною гілкою репозиторію в рамках CI/CD пайплайну гарантує, що всі зміни проходять ретельну перевірку на кожному етапі розробки та тестування, що забезпечує їхню готовність до впровадження у виробниче середовище. Описаний CI/CD пайплайн сприяє не тільки швидкій і надійній доставці продукту, але й значно підвищує продуктивність команди розробників, задовольняючи вимоги швидкозмінного ринку програмного забезпечення завдяки автоматизації та інтеграції процесів. Ця методологія дозволяє мінімізувати час від виявлення дефектів до їх виправлення, що є критично важливим для підтримання високого рівня якості кінцевого продукту. Завдяки CI/CD пайплайну, команди можуть безперервно впроваджувати покращення, тестувати нові функції в реальному часі і швидко реагувати на зворотний зв'язок користувачів, що додатково підвищує задоволеність клієнтів і конкурентоспроможність продукту на ринку.

3.3 Порівняння з аналогами–інструментам DevOps

В цьому підпункті детально порівняється з іншими інструментами DevOps котрі також виконують різні маніпуляції з фотографіями. Нижче представлено таблицю порівняння, яка ілюструє основні характеристики розробленої програми для обробки зображень у хмарному середовищі з такими конкурентами, як AWS Rekognition Custom Labels, Roboflow, Hasty, та Levity.

Таблиця 3.1 – Порівняння розробленої програми з іншими DevOps інструментами

Програмна система	image filtration	image tagging	change brightness	object detection	image regression	intensity inversion
my programm	+	–	+	+	–	+

AWS Rekognition Custom Labels	–	+	–	–	–	–
Roboflow	+	+	–	+	–	–
Hasty	–	+	–	+	–	–
Levity	–	+	–	+	–	–

Для порівняння було використано інші інструменти DevOps ,що спеціалізуються на розпізнаванні фото ,обробці і різні види маніпуляцій з об’єктом. Було порівняно по таким параметрам як: накладання фільтрів ,тегування фото, зміна яскравості ,виявлення об’єктів ,регресія фото і інверсійна інтенсивність. Розроблена програма показала досить хороші результати порівнюючи з конкурентами .

Відзначено наступні особливості:

1) Фільтрація зображень. Розроблена програма підтримує функцію фільтрації зображень, забезпечуючи високу точність і швидкість обробки у хмарному середовищі. Конкурентні платформи також пропонують схожі функції, проте можуть мати різні рівні налаштувань і точності.

2) Зміна яскравості. Розроблений проект дозволяє швидко і якіснозмінювати яскравість зображень, що забезпечує гнучкість обробки. Хоча ця можливість є у більшості конкурентів, їхні рішення можуть відрізнятися за швидкістю та якістю результуючих зображень.

3) Виявлення об'єктів. Розроблена програма використовує передові алгоритми для виявлення об'єктів, оптимізовані для хмарної платформи. Забезпечуючи високу продуктивність, що може бути порівнянна з лідером у цій області, AWS Rekognition.

4) Інверсія інтенсивності. Програма включає унікальну можливість інверсії інтенсивності, що забезпечує користувачам додаткову гнучкість у візуалізації

зображень. Це може відіграти ключову роль у певних сценаріях використання, які не підтримуються всіма конкурентами.

Недоліки проекту. Відзначено, що розроблена програма не підтримує маркування зображень та регресію зображень, що може бути важливим для певних користувачів, зокрема у сфері машинного навчання та штучного інтелекту. Впливаючи на конкурентоспроможність проекту в порівнянні з платформами, що надають ці можливості.

ВИСНОВКИ

1) На основі аналітичного підходу проведено порівняльний аналіз сучасних підходів до розгортання програмного забезпечення на хмарних сервісах, що дозволило визначити основні конфігураційні блоки для покладеної задачі.

2) На основі алгоритмічного підходу проведено порівняльний аналіз засобів опрацювання графічних зображень, що дозволило виділити алгоритми для обробки цифрових зображень для даного дослідження.

3) На основі алгоритмічного підходу розроблено алгоритм для роботи програми, що опрацьовує зображення, дозволяючи уніфікувати процес розгортання проекту в сервісі digitalocean.

4) На основі алгоритмічного підходу розроблено життєвий цикл розгортання проекту на хмарному сервісі з використанням підходу безпервної доставки та інтеграції коду.

5) На основі порівняльного аналізу за визначиними критеріями виділено переваги та недоліки розробленого підходу до розгортання програмного забезпечення для опрацювання зображень.

6) Розроблена програма для обробки зображень у хмарному середовищі вирізняється серед конкурентів високою точністю та швидкістю виконання функцій, таких як фільтрація зображень, зміна яскравості, виявлення об'єктів та інверсія інтенсивності. Роблячи її кращою для користувачів, що потребують ефективної обробки зображень. Однак, через відсутність функцій маркування та регресії зображень, її використання може бути обмеженим у галузях, де потрібен більш детальний аналіз. Вказуючи на необхідність подальшого розвитку функціоналу для повного задоволення потреб різних категорій користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Mikael Krief, Learning DevOps: A comprehensive guide to accelerating DevOps culture adoption with Terraform, Azure DevOps, Kubernetes, and Jenkins , Packt Publishing, 2022.
2. T. Offerman, R. Blinde, C. J. Stettina and J. Visser, "A Study of Adoption and Effects of DevOps Practices," 2022 IEEE 28th International Conference on Engineering, Technology and Innovation (ICE/ITMC) & 31st International Association For Management of Technology (IAMOT) Joint Conference, Nancy, France, 2022, pp. 1-9,
3. Humble, J., & Farley, D. (2010). Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. AddisonWesley.
4. T. Pandiyavathi and B. Sivakumar, "Implementing Various Systems with DevOps to Make Successful Decisions based on Intelligent Learning Strategy," 2024 International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics (IITCEE), Bangalore, India, 2024, pp. 1-6,
5. Koren, F. Rinker, K. Meixner, M. Kröger and M. Zeng, "Implementing DevOps Practices in CPPS Using Microservices and GitOps," 2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA), Sinaia, Romania, 2023, pp. 1-4
6. Gaurav Agarwal, Modern DevOps Practices: Implement and secure DevOps in the public cloud with cutting-edge tools, tips, tricks, and techniques , Packt Publishing, 2021.
7. Christopher Cowell; Nicholas Lotz; Chris Timberlake, Automating DevOps with GitLab CI/CD Pipelines: Build efficient CI/CD pipelines to verify, secure, and deploy your code using real-life examples , Packt Publishing, 2023.
8. H. da Gião, R. Pereira and J. Cunha, "CI/CD Meets Block-Based Languages," 2023 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Washington, DC, USA, 2023, pp. 232-234,Git. (n.d.). Git. Retrieved from https://git-scm.com/ (Git SCM).
9. Jesse Liberty; Jon Galloway, Git for Programmers: Master Git for effective implementation of version control for your programming projects , Packt Publishing, 2021.
10. Docker Docs. (n.d.). Docker overview. Retrieved March 23, 2024, from https://docs.docker.com/get-started/overview/
11. Hightower, K., Burns, B., & Beda, J. (2019). Kubernetes: Up and Running (2nd ed.). O'Reilly Media.

12. HashiCorp. (n.d.). What is Terraform?. Terraform by HashiCorp. Retrieved from https://www.terraform.io/intro

13. Ansible Community Documentation. (n.d.). Introduction to Ansible. Retrieved from https://docs.ansible.com/ansible/latest/getting_started/introduction.html

14. Grafana Labs. (n.d.). Grafana. Retrieved from https://grafana.com/

15. R. C. Pushpaleela, S. Sankar, K. Viswanathan and S. A. Kumar, "Application Modernization Strategies for AWS Cloud," 2022 1st International Conference on Computational Science and Technology (ICCST), CHENNAI, India, 2022, pp. 108-110,

16. HashiCorp. (n.d.). Vagrant by HashiCorp. Retrieved from https://www.vagrantup.com/

17. HashiCorp. (2024). Vagrant by HashiCorp. Вилучено 11 травня 2024, з https://www.vagrantup.com

18. Muhammad Asif, Python for Geeks: Build production-ready applications using advanced Python concepts and industry best practices , Packt Publishing, 2021.

19. GitHub. (n.d.). GitHub Actions Documentation. Retrieved from https://docs.github.com/en/actions.

20. Bash - GNU Project. (n.d.). Bash - GNU Project - Free Software Foundation. Retrieved from Bash - GNU Project - Free Software Foundation

21. Sorvisto, D. (2023). *MLOps Lifecycle Toolkit*. Apress. https://doi.org/10.1007/978-1-4842-9642-4

22. Filip, Ion-Dorinel, Cristian-Mihai Iliescu, and Florin Pop. "Assertive, Selective, Scalable IoT-Based Warning System." Sensors 22, no. 3 (January 28, 2022): 1015. http://dx.doi.org/10.3390/s22031015.

23. Vidhya, R., & Vadivu, G. (2016). Research Document Search using Elastic Search. Indian Journal of Science and Technology, 9(37). https://doi.org/10.17485/ijst/2016/v9i37/102108

24. Evans, J. C. (2020). *AWS : Amazon Web Services: A Complete Guide for Beginners and Advanced Users for Amazon Web Services*

25. D. Deepa, A. Sivasangari, R. Roonwal and R. Nayan, "Pothole Detection using Roboflow Convolutional Neural Networks," *2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS)*, Madurai, India, 2023, pp. 560-564,

26. G. Symeonidis, E. Nerantzis, A. Kazakis and G. A. Papakostas, "MLOps - Definitions, Tools and Challenges," *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA, 2022, pp. 0453-0460

27. S. Singla, A. Eldawy, T. Diao, A. Mukhopadhyay and E. Scudiero, "Experimental Study of Big Raster and Vector Database Systems," 2021 IEEE 37th

International Conference on Data Engineering (ICDE), Chania, Greece, 2021, pp. 2243-2248

28. H. W. H. Premachandra, A. Jayakody and H. Kawanaka, "Converting high resolution multi-lingual printed document images in to editable text using image processing and artificial intelligence," 2022 2nd International Conference on Image Processing and Robotics (ICIPRob), Colombo, Sri Lanka, 2022, pp. 1-7

29. Bath, S. Shekhar, J. Döllner and M. Trapp, "COLiER: Collaborative Editing of Raster Images," 2021 International Conference on Cyberworlds (CW), Caen, France, 2021, pp. 33-40

30. Fried, O., Jacobs, J., Finkelstein, A., & Agrawala, M. (2020). Editing self-image. Communications of the ACM, 63(3), 70-79. https://doi.org/10.1145/3326601

31. Методичні вказівки до виконання практичних робіт з дисципліни «Техніко-економічне обґрунтування розробки комп'ютерних систем"/ Н.Я. Савка, І.Р. Паздрій / Під ред. О.М. Березького. Тернопіль: ТНЕУ, 2019. 40 с.

32. Методичні вказівки до оформлення курсових проектів, звітів про проходження практики, випускних кваліфікаційних робіт для студентів спеціальності «Комп'ютерна інженерія» / І.В. Гураль, Л.О. Дубчак / Під ред. О.М. Березького. Тернопіль: ТНЕУ, 2019. 33 с.