Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет *інфокомунікацій*
(повна назва)

Кафедра *інформаційно-мережної інженерії*
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження шляхів оптимізації системи автоматизації перевірки виконання практичних завдань в локальному віртуальному оточенні користувача
(тема)

Виконав: Верещака Олексій Андрійович
здобувач 2025 року навчання,
групи ІМІм-23-2

Верещака О.А.
(прізвище, ініціали)

Спеціальність 172 Електронні комунікації та радіотехніка
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Інформаційно-мережна інженерія
(повна назва освітньої програми)

Керівник Костромицький Андрій Іванович
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Безрук В.М.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет *інфокомунікацій*

Кафедра *інформаційно-мережної інженерії*

Рівень вищої освіти другий (магістерський)

Спеціальність 172 Електронні комунікації та радіотехніка
(код і повна назва)

Тип програми *освітньо-професійна*

Освітня програма *інформаційно-мережна інженерія*
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«____»_____20 ____ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Верещакі Олексію Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи. *Дослідження шляхів оптимізації системи автоматизації перевірки виконання практичних завдань в локальному віртуальному оточенні користувача* затверджена наказом університету від 28 жовтня 2024 р. № 1148Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 20 січня 2025 р.
3. Вихідні дані до роботи *Доступ до LMS Moodle для інтеграції та тестування системи. Документація на API LMS Moodle, включаючи стандарти REST API та LTI 1.3. Технічні вимоги до системи автоматизації перевірки. Наявність локального віртуального середовища для тестування практичних завдань. Дані щодо методологій Infrastructure as Code (IaC). Зворотний зв'язок від викладачів і студентів для оцінки зручності та продуктивності системи.*
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Вступ
1. Огляд системи керуванням навчання
2. Загальна архітектура додатку
3. Шляхи оптимізації архітектури
4. Практична реалізація поліпшень
5. Дослідження ефективності застосування системи
Висновки

4. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) *Архітектура системи автоматизації перевірки (схема взаємодії компонентів), шляхи покращення, практична реалізація, результати впровадження, висновки тощо.*

5. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Основна частина	*доц. Костромицький А.І.*		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів	Примітка
1	*Ознайомлення із завданням. Уточнення ТЗ.*	*21.10.24*	
2	*Підбір літератури за темою роботи.*	*23.10-29.10.24*	
3	*Виконання розділу 1*	*30.10-04.11.24*	
4	*Виконання розділу 2*	*05.11-09.11.24*	
5	*Виконання розділу 3*	*10.11-14.11.24*	
6	*Виконання розділу 4*	*15.11-19.11.24*	
7	*Виконання розділу 5*	*20.11-31.12.24*	
8	*Оформлення презентаційного матеріалу*	*10.01.24-13.01.25*	

Дата видачі завдання 21 *жовтня* 2024 р.

Здобувач _____
(підпис)

Керівник роботи _____ *Костромицький А.І.*
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 64 с., 18 рис., 43 джерел.

Об'єкт дослідження - система автоматизації перевірки виконання практичних завдань у локальному віртуальному середовищі користувача.

Мета дослідження - розробка рекомендацій щодо оптимізації існуючих систем автоматизації перевірки практичних завдань у локальному віртуальному середовищі користувача.

Аналіз існуючих систем автоматизації та визначення їх сильних і слабких сторін; розробка критеріїв оцінки ефективності систем, включаючи показники якості, продуктивності та зручності використання; пропозиція шляхів вдосконалення системи, зокрема інфраструктури, алгоритмів перевірки та інтеграції з LMS.

PYTHON, MOODLE, LMS, LTI, IAC, СИСТЕМА АВТОМАТИЧНОЇ ПЕРЕВІРКИ

ABSTRACT

Explanatory note: 64 pages, 18 figures, 43 references.

Object of the work – a system for automating the verification of practical tasks in a user's local virtual environment.

Research Objective - development of recommendations for optimizing existing systems for automating the verification of practical tasks in a user's local virtual environment.

Analysis of existing automation systems and identification of their strengths and weaknesses; development of criteria for evaluating the effectiveness of systems, including quality, performance, and usability indicators; proposal of ways to improve the system, particularly its infrastructure, verification algorithms, and integration with LMS.

PYTHON, MOODLE, LMS, LTI, IAC, AUTOMATED VERIFICATION SYSTEM

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API (Application Programming Interface) – прикладний програмний інтерфейс;
AWS (Amazon Web Services) – провайдер хмарних рішень;
CICD (Continuous Integration and Continuous Delivery/Deployment) – неперервна інтеграція та неперервна доставка/розгортання;
HTML (Hyper-Text Markup Language) – мова розмітки гіпертексту;
HTTP (HyperText Transfer Protocol) – протокол передачі даних;
HTTPS (HyperText Transfer Protocol Secure) – захищений протокол передачі даних;
IaC (Infrastructure as Code) – інфраструктура як код;
JSON (JavaScript Object Notation) – текстовий формат обміну даними;
JWT (JSON Web Token) – відкритий стандарт для створення токенів;
JWK (JSON Web Key) – ключ, що використовується для перевірки JWT токену; LMS (Learning Management System) – система керування навчанням;
LTI (Learning Tools Interoperability) – стандарт взаємодії між LMS і зовнішніми інструментами;
OIDC (OpenID Connect) – протокол аутентифікації користувача сторонніми сервісами;
ORM (Object-Relation Mapping) – технологія, що дозволяє пов’язати сутності бази даних з сутностями коду;
REST API – архітектурний стиль для взаємодії клієнта та сервера;
WSGI (Web Server Gateway Interface) – стандарт взаємодії між Python- застосунком на сервері та самим веб-сервером, наприклад, Nginx.

ВСТУП

Сучасний освітній процес все більше орієнтований на використання цифрових технологій, які сприяють індивідуалізації навчання, підвищенню його ефективності та доступності. Системи управління навчанням (Learning Management Systems, LMS) є важливими інструментами цифрової трансформації освіти, забезпечуючи платформу для організації навчального процесу, взаємодії викладачів і студентів, а також для управління контентом. Однак традиційні LMS мають суттєву прогалину: вони не забезпечують ефективної автоматизації оцінювання практичних навичок, що залишається одним із найбільших викликів для цифрового навчання.

Основна проблема полягає в тому, що існуючі системи LMS переважно орієнтовані на управління контентом, проведення тестів та адміністрування курсів. Вони не мають функціоналу, який дозволяє об'єктивно оцінювати виконання практичних завдань, що є ключовим компонентом навчання у багатьох галузях. Це ускладнює процес оцінювання практичних навичок студентів, особливо в умовах дистанційного або змішаного навчання.

Метою цієї роботи є розробка рекомендацій щодо оптимізації існуючих систем автоматизації перевірки практичних завдань у локальному віртуальному середовищі користувача. Досягнення цієї мети передбачає вирішення таких завдань:

1) Аналіз існуючих рішень для автоматизації перевірки практичних завдань, визначення їх сильних і слабких сторін;

2) Розробка критеріїв оцінки ефективності таких систем, включаючи показники продуктивності, зручності використання та інтеграції з іншими платформами;

3) Впровадження нових підходів до автоматизації, які дозволять вдосконалити процес перевірки практичних завдань.

Дослідження має як теоретичне, так і практичне значення. Теоретично робота зосереджена на аналізі архітектури систем автоматизації, їх інтеграції з LMS і методології Infrastructure as Code (IaC). Практичний аспект включає впровадження запропонованих удосконалень, зокрема використання Terraform і Ansible для автоматизації інфраструктури та покращення перевірки завдань.

Значущість цього дослідження полягає в тому, що воно сприяє підвищенню якості освітнього процесу, забезпечуючи швидке, точне та об'єктивне оцінювання практичних навичок. Для освітніх установ це означає зниження адміністративного навантаження та підвищення ефективності навчального процесу. Для розробників створення вдосконалених систем відкриває можливості для розширення функціоналу LMS, інтеграції інноваційних рішень і впровадження сучасних методологій управління інфраструктурою.

У роботі також розглядається потенціал для подальшого вдосконалення систем автоматизації перевірки завдань, що може бути основою для майбутніх досліджень у цій галузі.

1 ОГЛЯД СИСТЕМИ КЕРУВАННЯ НАВЧАННЯМ

1.1 Сутність поняття LMS

Система управління навчанням (Learning Management System, LMS) — це програмна платформа, що спрощує організацію, проведення та контроль навчального процесу. Вона забезпечує інструменти для створення курсів, управління навчальним контентом, відстеження прогресу учнів і взаємодії між учасниками навчання. LMS є ключовим елементом у сучасному цифровому освітньому середовищі та використовується як в академічному, так і корпоративному навчанні.

Система управління навчанням (Learning Management System, LMS) – це програмна платформа, що забезпечує організацію, проведення та контроль навчального процесу. LMS є ключовим елементом сучасного цифрового освітнього середовища, дозволяючи створювати навчальні курси, управляти контентом, відстежувати прогрес студентів та забезпечувати комунікацію між учасниками навчання.

Основні аспекти LMS, які мають значення для цієї роботи, полягають у їхньому функціоналі, інтеграційних можливостях і здатності до адаптації для автоматизації оцінювання практичних навичок.

Ключові функції LMS:

1) Управління контентом: створення, зберігання та організація освітніх матеріалів у зручному цифровому форматі;

2) Оцінювання: проведення тестів, інтерактивних завдань та генерація звітів про успішність студентів;

3) Інтерактивність: підтримка обговорень, форуми та інші інструменти для взаємодії студентів із викладачами;

4) Інтеграція: можливість підключення сторонніх додатків та платформ, таких як API, хмарні сховища або сервіси перевірки завдань.

Типи LMS:

1) Хмарні LMS: забезпечують легкий доступ до ресурсів через Інтернет, дозволяючи студентам працювати з будь-якого пристрою;

2) Системи з відкритим кодом: як-от Moodle, дозволяють розробникам додавати нові модулі, зокрема для автоматизації перевірки практичних завдань;

3) Корпоративні LMS: адаптовані для професійного навчання, часто включають аналітичні інструменти для оцінки ефективності навчання.

Переваги LMS:

1) Автоматизація процесів: економія часу на адміністративні завдання та оцінювання завдяки автоматичному аналізу виконаних завдань;

2) Доступність: можливість навчання з будь-якого місця у зручний час;

3) Адаптивність: системи можуть бути налаштовані під специфічні вимоги освітніх установ або корпоративних клієнтів;

4) Інтеграція з інструментами перевірки: наприклад, використання API для впровадження автоматизованої оцінки практичних завдань.

Таким чином, LMS є потужним інструментом для організації навчального процесу, однак для вирішення завдань автоматизації перевірки практичних завдань необхідна адаптація або інтеграція із зовнішніми рішеннями, що є предметом подальшого дослідження в цій роботі.

1.2 Основні функції LMS, переваги та недоліки використання

LMS забезпечують централізоване управління навчальним процесом, об’єднуючи всі аспекти освіти в єдиній цифровій платформі. Це включає управління навчальними матеріалами, автоматизацію оцінювання, аналітику та адаптивність, які дозволяють викладачам персоналізувати навчання під конкретні потреби студентів.

Ключовим досягненням сучасних LMS є інтеграція інструментів для онлайн-комунікації, таких як форуми, відеоконференції та чати. Вони не тільки полегшують обмін інформацією, але й сприяють створенню спільнот, де студенти та викладачі можуть взаємодіяти у реальному часі. LMS також надають можливість інтеграції з іншими технологіями, такими як автоматизовані перевірки завдань, що робить їх універсальним рішенням для різних навчальних середовищ.

Популярність LMS значною мірою зумовлена їх здатністю забезпечувати доступність навчальних матеріалів у будь-який час і з будь-якого пристрою. Це особливо актуально для дистанційного навчання, яке стало нормою у постпандемічному світі. Крім того, інструменти аналітики допомагають викладачам слідкувати за прогресом студентів, що підвищує якість викладання та рівень успішності учнів.

Проте використання LMS не позбавлене викликів. Серед них – складність інтеграції практичних завдань, особливо в технічних дисциплінах, де необхідно оцінювати навички роботи з конкретними програмами чи обладнанням. Впровадження таких систем також вимагає значних ресурсів для навчання користувачів та підтримки інфраструктури. Брак соціальної взаємодії є ще одним обмеженням, оскільки студентам важко розвивати так звані «м’які навички», які зазвичай формуються у традиційному навчальному середовищі.

Сучасні LMS активно впроваджують адаптивні алгоритми, що дозволяє значно поліпшити ефективність навчання. Однак, для забезпечення повноцінного інтерактивного навчання необхідно враховувати потребу в інтеграції практичних завдань і розвивати відповідні модулі, які б могли оцінювати не лише теоретичні знання, а й навички застосування цих знань на практиці.

1.3 Історія розвитку LMS

Історія розвитку LMS починається з простих систем управління навчальними матеріалами, які з’явилися на початку 90-х років. Вони мали обмежену функціональність, орієнтовану переважно на викладання лекцій і тестування. Перші комерційні LMS забезпечували базову можливість розміщення контенту, а їх користувачами були переважно великі навчальні заклади.

З розвитком інтернету з’явилися перші веборієнтовані LMS, які дозволяли студентам отримувати доступ до матеріалів у будь-який час. Це стало значним проривом, адже навчальні ресурси стали доступними поза межами аудиторії. Платформи почали підтримувати відео та мультимедійний контент, а також інструменти комунікації, що суттєво розширило їх функціонал.

Новітні технології значно змінили підхід до навчання. Зокрема, розвиток генеративного штучного інтелекту відкрив можливості для створення адаптивного навчання, яке підлаштовується під потреби кожного студента. Такі технології використовуються для аналізу прогресу, визначення прогалин у знаннях та формування індивідуальних траєкторій навчання.

Сучасні LMS також включають інтеграцію з хмарними платформами, що дозволяє зберігати значні обсяги даних і забезпечувати стабільну роботу систем. Вони пропонують інструменти для підвищення академічної доброчесності, такі як перевірка на плагіат, що забезпечує високий рівень якості освіти. Разом з тим, інтеграція мобільних додатків зробила навчання ще більш доступним, дозволяючи студентам навчатися в будь-якому місці та в будь-який час.

У майбутньому розвиток LMS буде зосереджений на впровадженні нових технологій, таких як віртуальна та доповнена реальність, що дозволить створювати симуляції реальних умов для практичного навчання. Ці системи продовжують трансформувати освіту, роблячи її більш доступною, персоналізованою та інтерактивною.

2 ЗАГАЛЬНА АРХІТЕКТУРА ДОДАТКУ

2.1 Типова функціональність та архітектура додатку

Системи з клієнт-серверною архітектурою є основою багатьох сучасних додатків, оскільки вони забезпечують гнучкість, масштабованість і централізоване управління. У контексті проєкту, ця архітектура дозволяє створити додаток, який може обробляти значну кількість запитів від клієнтів, забезпечувати доступ до навчальних ресурсів і підтримувати інтерактивність між студентами та викладачами.

Основними компонентами клієнт-серверної архітектури є сервер, який обробляє запити, зберігає дані та виконує основні обчислення, і клієнт, який забезпечує інтерфейс користувача та відправляє запити до сервера. Такий підхід дозволяє розподілити функціональність між різними рівнями, що підвищує ефективність роботи системи.

У випадку проєкту було обрано платформу Moodle як базу для розробки додатку. Moodle — це відкрита система управління навчанням, яка забезпечує широкий спектр функцій для організації навчального процесу. Вибір Moodle ґрунтувався на наступних перевагах:

1) Гнучкість і адаптивність. Moodle підтримує розширення за допомогою модулів і плагінів, що дозволяє адаптувати систему до специфічних потреб проєкту. Це особливо важливо для інтеграції функціоналу автоматизації перевірки практичних завдань;

2) Підтримка стандартизованих протоколів: Moodle використовує відкриті стандарти, такі як LTI (Learning Tools Interoperability), що спрощує інтеграцію з іншими системами та платформами. Це забезпечує можливість обміну даними та створення комплексних рішень;

3) Масштабованість: Moodle дозволяє обслуговувати велику кількість користувачів без суттєвих втрат у продуктивності. Це забезпечує стабільну роботу навіть у разі значного навантаження на систему.

4) Підтримка аналітики навчального процесу: Moodle має вбудовані інструменти для збору та аналізу даних про успішність студентів, що сприяє персоналізації навчання та підвищенню його ефективності.

5) Спільнота та підтримка: Moodle має велику активну спільноту розробників і користувачів, які постійно працюють над вдосконаленням системи, створенням нових плагінів і наданням технічної підтримки.

У цьому проєкті клієнт-серверна архітектура Moodle використовується для забезпечення функціональності взаємодії користувача з навчальними матеріалами, автоматизації перевірки практичних завдань і управління базами даних користувачів. Серверна частина забезпечує централізоване управління всіма ресурсами, тоді як клієнтська частина надає інтуїтивно зрозумілий інтерфейс для роботи зі студентами та викладачами.

Для реалізації системи було вирішено використати готову платформу Moodle як базу. Moodle — це одна з найпопулярніших систем управління навчанням (LMS), яка надає широкі можливості для адаптації та розширення.

Основні переваги вибору Moodle:

1) Економія часу: використання готової LMS знижує час, необхідний на розробку базової функціональності;

2) Можливість інтеграції: архітектура Moodle дозволяє легко додавати нові модулі та підтримку інших LMS у майбутньому;

3) Гнучкість та масштабованість: Moodle підтримує налаштування під потреби користувачів та забезпечує зручний доступ до навчальних матеріалів;

4) Широкий функціонал: система пропонує інструменти для управління курсами, тестування, звітності, а також підтримує інтеграцію з зовнішніми сервісами;

Таким чином, обрана архітектура ґрунтується на готовій платформі, що поєднує функціональність клієнт-серверної моделі з перевагами відкритого вихідного коду Moodle. Це забезпечує стабільність, ефективність та гнучкість у розробці проєкту.

2.2. Вибір технологій для реалізації проекту

Архітектура програм Angular базується на деяких базових концепціях. Основне з чого складається проект на Angular – це компоненти, які, своєю чергою, організовані у NgModules. NgModules збирає зв'язані компоненти, та інший схожий по функціоналу також; Angular програми складаються з набору NgModules. Завжди, як мінімум, повинен існувати хоча б один кореневий модуль, що дозволяє завантажувати інші функціональні модулі.

Компоненти визначають уявлення, які є наборами елементів екрана, які Angular може вибирати та змінювати відповідно до логіки та даних програми.

Компоненти звертаються до сервісів, які надають конкретні функції, які пов'язані безпосередньо з уявленнями. Сервіси можуть бути введені в компоненти залежно від того, що робить код модульним, багатовикористовуваним та ефективним.

Модулі, компоненти та послуги – це класи, де використовуються декоратори. Ці декоратори позначають тип класу та надають метадані, які повідомляють Angular, щоб той міг їх використати.

Метадані для сервісного класу надають інформацію, необхідну Angular, щоб зробити її доступною компонентам за допомогою ін'єкції залежностей (DI).

Компоненти програми є графічними елементами, розташованими ієрархічно. Angular надає модуль Router, щоб допомогти визначити шляхи навігації між уявленнями.

Архітектура програми на Angular складається з таких основних елементів: модулі, компоненти, шаблони, директиви, сервіси, маршрутизація тощо, на рис. 2.1 зображено діаграму взаємозв'язку між концепціями [33].

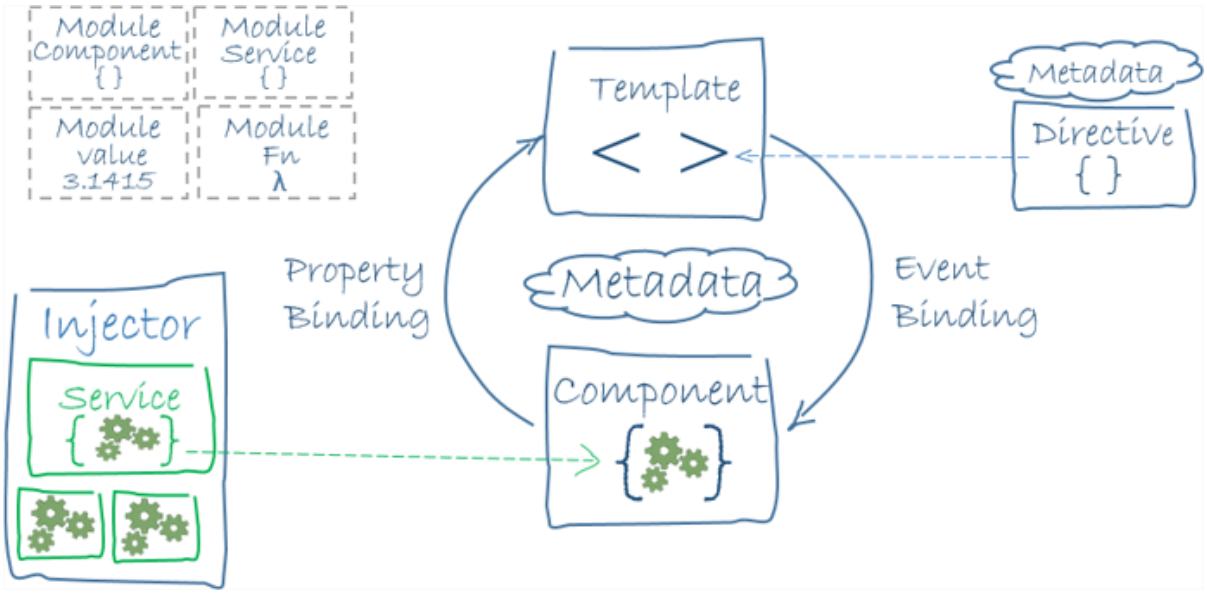

Рисунок 2.1 – Архітектура Angular: взаємодія компонентів, шаблонів і метаданих

Модулі Angular NgModules відрізняються від модулів JavaScript (ES6) та доповнюють їх.), щоб утворити функціональні одиниці. Кореневий модуль, який зазвичай називається AppModule, забезпечує механізм запуску програми, як правило, складається з функціональних модулів. службу маршрутизатора у своїй програмі, потрібно скористатися маршрутизатором NgModule. Організація коду в окремі функціональні модулі допомагає керувати розробкою складних програм та розробляти їх для повторного використання. Крім того, ця техніка дозволяє скористатися перевагами відкладеного завантаження, тобто. запуску.

Компоненти. Кожна програма містить щонайменше один компонент, який називається кореневим компонентом, який з'єднує ієрархію компонентів з об'єктною моделлю документа сторінки (DOM) Кожен компонент визначає клас, що містить дані програми та логіку. пов'язаний з ним шаблон HTML, який визначає графічний елемент, що відображається у браузері Декоратор @Component() визначає клас безпосередньо під ним як компонент і надає шаблон і пов'язані метадані, специфічні компоненту.

Шаблони, директиви та прив'язка даних Шаблон поєднує HTML з розміткою Angular, яка може змінювати елементи HTML перед відображенням. даних:

1) Прив'язка подій дозволяє вашій програмі реагувати на введення користувача в цільове середовище, оновлюючи ці програми;

2) прив'язка властивостей дозволяє інтерполювати значення, обчислені з даних програми HTML.

Перед відображенням представлення Angular оцінює директиви та вирішує синтаксис прив'язки у шаблоні, щоб змінити елементи HTML та DOM відповідно до даних та логіки вашої програми. також відображаються в даних програми.

Сервіси та ін'єкція залежностей Створюються класи сервісів для даних або логіки, які не пов'язані з певним уявленням та є загальними між компонентами. Певний клас сервісу визначається безпосередньо декоратором @Injectable. клас. Ін'єкція залежностей дозволяє підтримувати класи компонентів упорядкованими та ефективними. сервера, які не валідують дані, введені користувачем, вони делегують наступні завдання сервісам.

Angular Router NgModule надає функцію, яка дозволяє визначати шлях навігації між різними станами програми та переглядати ієрархію маршрутизації у браузері маршрутизатор замість сторінки відображає шлях перегляду, подібний до URL, коли користувач виконує операцію, наприклад, натискає посилання, яка завантажує нову сторінку, маршрутизатор перехопить поведінка браузера і покаже чи приховає ієрархію перегляду [34].

Основою для проєкту є розроблена система автоматизації перевірки практичних завдань у локальному віртуальному оточенні, яка інтегрована з LMS Moodle. Постановка завдання проєктування полягає у вдосконаленні існуючого рішення з урахуванням сучасних технологічних викликів і потреб користувачів.

Розроблена система автоматизації перевірки практичних завдань має кілька ключових переваг:

1) Централізоване управління навчальними ресурсами: Система використовує Moodle як базову платформу, що забезпечує простоту організації навчального процесу, доступ до навчальних матеріалів і зручність управління курсами;

2) Інтеграція з локальним середовищем: Система дозволяє студентам виконувати завдання у власному середовищі, після чого результати автоматично передаються на сервер для перевірки. Це забезпечує гнучкість навчання та врахування індивідуальних потреб студентів;

3) Автоматизація перевірки: Система виконує перевірку завдань без участі викладача, що суттєво економить час та підвищує об’єктивність оцінювання;

4) Звітність та аналітика: Система генерує звіти про успішність студентів, що дозволяє викладачам отримувати детальну інформацію про прогрес учнів.

Однак, система має й виклики, які потребують уваги:

1) Обмеження інтеграції: Хоча Moodle підтримує широкий спектр функцій, існують труднощі в інтеграції з іншими платформами та сервісами через відсутність стандартизованих інструментів обміну даними;

2) Масштабованість: Поточна реалізація має обмежену пропускну здатність, що ускладнює обслуговування великої кількості користувачів одночасно;

3) Відсутність підтримки сучасних технологій: Система не використовує можливості штучного інтелекту та машинного навчання для адаптації завдань і автоматизації перевірки;

4) Обмежена автоматизація інфраструктури: Процеси розгортання та управління середовищем виконуються вручну, що підвищує ризик помилок і знижує ефективність розробки.

Для вирішення виявлених проблем та вдосконалення системи були сформульовані такі завдання:

1) Розширення інтеграційних можливостей. Забезпечення підтримки стандартів обміну даними, таких як LTI 1.3, для полегшення взаємодії з іншими LMS та сторонніми сервісами. Розробка API для інтеграції зі сторонніми додатками;

2) Масштабованість та продуктивність. Оптимізація бази даних для зниження навантаження на сервер. Впровадження балансувальника навантаження для забезпечення стабільної роботи системи при значному збільшенні кількості користувачів;

3) Автоматизація інфраструктури. Використання Terraform або Ansible для автоматизованого розгортання інфраструктури. Налаштування CI/CD процесів для зниження ризику помилок під час впровадження нових функцій;

Таким чином, проєктування оновленої системи передбачає вдосконалення поточної архітектури та функціоналу, що дозволить підвищити її ефективність, масштабованість і гнучкість. У наступних розділах розглядається реалізація зазначених завдань із використанням сучасних технологій та інструментів.

3 ШЛЯХИ ОПТИМІЗАЦІЇ АРХІТЕКТУРИ

3.1 Аспекти покращення архітектури

Розширення підтримки та інтеграції з іншими платформами. Для підвищення універсальності додатку можна впровадити інтеграцію з різними сервісами, такими як платіжні шлюзи, системи аналітики або сторонні LMS. Це дозволить розширити функціонал та забезпечити кращу інтеграцію із зовнішніми інструментами.

Пропоновані покращення:

1) Використання API для створення універсального модуля інтеграції; Використання стандартних протоколів, таких як OAuth 2.0, для забезпечення безпеки підключення до сторонніх сервісів;

2) Забезпечення гнучкості модулів для легкого додавання нових інтеграцій;

3) Покращення якості та безпеки коду. Впровадження автоматизованого аналізу коду для виявлення потенційних помилок, недоліків стилю та уразливостей. Це підвищить якість програмного забезпечення та зменшить ризики, пов’язані з безпекою;

4) Впровадження інструментів статичного аналізу коду, таких як SonarQube або Flake8;

5) Використання інструментів перевірки на вразливості, наприклад, Fortify або Veracode;

6) Налаштування CI/CD процесів із перевіркою коду на кожному етапі.

7) Оптимізація запитів до бази даних. Використання технологій кешування.

Розширення функціональності існуючої архітектури є важливим кроком у вдосконаленні системи автоматизації перевірки практичних завдань. Основна мета цього процесу полягає в адаптації системи до сучасних викликів і забезпеченні її інтеграції з інноваційними технологіями. У цьому розділі розглядаються аспекти покращення архітектури, конкретні приклади інтеграції з API, а також представлення результатів впроваджених змін.

Одним із головних напрямів покращення є інтеграція системи з сучасними API. Наприклад, використання LTI (Learning Tools Interoperability) 1.3 дозволяє забезпечити безпечну взаємодію між LMS Moodle та іншими освітніми платформами. Це відкриває нові можливості для викладачів і студентів, дозволяючи обмінюватися даними між платформами без необхідності дублювання інформації.

Для забезпечення ефективної інтеграції запропоновано впровадження REST API, які дозволяють виконувати такі операції:

1) Обмін інформацією про студентів та їхній прогрес;

2) Додавання та оновлення завдань із зовнішніх платформ;

3) Отримання детальних звітів про виконані завдання;

4) Наприклад, інтеграція із сервісом Turnitin може автоматизувати перевірку текстових завдань на унікальність. Це зменшує час викладача на перевірку та покращує якість оцінювання.

Масштабованість і продуктивність. Для забезпечення роботи системи під час значного збільшення кількості користувачів запропоновано такі покращення:

1) Використання індексів для пришвидшення пошуку даних.

2) Нормалізація таблиць для усунення дублювання даних.

3) Балансувальник навантаження.

4) Використання хмарних сервісів, таких як AWS Load Balancer.

Автоматизація процесів розгортання та управління інфраструктурою є важливим аспектом вдосконалення. Використання Terraform дозволяє описувати та створювати інфраструктуру як код. Це забезпечує:

1) Швидке розгортання нових середовищ.

2) Мінімізацію ризику людських помилок.

3) Уніфікацію конфігурації для різних середовищ (розробка, тестування, продакшн).

3.2 Практична реалізація покращень

Лінтери – інструменти статичного аналізу, які перевіряють відповідність коду специфікаціям чи відповідним стандартам. Далі розглянемо деякі з них.

Лінтери бувають різні, що реалізують базові потреби, наприклад flake8, який перевіряє лише стиль написання коду, складніші, які крім стилістики перевіряють семантику, зокрема pylint, і ті, що впроваджують додаткові перевірки, такі як type checking, але для python це новий функціонал , і зараз він знаходиться осторонь [28-30].

Flake8 не має великої кількості перевірок, і не здатний перевіряти семантику, але підтримує плагіни і може ними розширювати свій функціонал, враховуючи специфіку окремого проекту.

Крім лінтерів, існує інший вид інструментів, які забезпечують статичний аналіз даних - аналізатори вразливості безпеки. Ці інструменти призначені для виявлення потенційних уразливостей, які можуть бути використані зловмисниками для атаки на систему. неправильним керуванням доступом, уразливістю типу буферного переповнення та іншими. виявляють можливості для виконання SQL-інекцій, введення шелл-команд, атак на основні XSS, CSRF та інші.

Такі можливості у SonarQube, Checkmarx. Деякі аналізатори можуть перевіряти потік, яким поширюються дані в додатку, що виявляють шляхи атак і точки введення, які можуть бути використані хакерами, такий функціонал має зокрема Fortify, також Veracode.

Аналізатори можуть застосовувати набір правил для перевірки дотримання кодом стандартів безпеки.

Деякі інструменти можуть інтегруватися з базами даних відомих загроз і використовувати їх для виявлення вразливостей.

Аналізатори якості коду у статичному аналізі є інструментами, спрямованими на визначення та оцінку якості коду без його фактичного виконання, вони стежать за відповідністю коду певним стандартам, виявляють потенційні проблеми та рекомендують покращення з погляду читабельності, ефективності та обслуговуваності.

Аналізатори використовують певні стандарти коду та метрики для оцінки якості програми. Це включає перевірку належного форматування коду, дотримання стилів та певних критеріїв якості.

Інструменти виявляють забруднення або погані практики в програмному коді, які можуть призвести до невиправних проблем у майбутньому.

Деякі аналізатори обчислюють цикломатичну складність коду, визначальну кількість незалежних шляхів коду.

Аналізатори можуть виявляти потенційні проблеми та недоліки в коді, такі як невикористовувані змінні, недоступний код або інші конструкції, які можуть викликати проблеми.

Flake8 – інструмент побудований над PyFlakes їх використанням PyCodeStyle (pep8) та цикломатичної перевірки складності (використовується для пошуку складного коду).

Flake8 – може бути розширений різними модулями, існує велика кількість плагінів під будь-які потреби. Flake8 має потужний механізм налаштування, використовуючи один із файлів конфігурації «flake8», «setup.cfg», «tox.ini». Плагіни та попередження можуть бути відключені у кожному файлі чи рядку.

Код наведено на рисунку 3.2, проаналізовано за допомогою цього інструменту Flake8, результати наведено на рисунку 3.3.

```
def calculate_sum(a, b):  1 usage
    result = a + b
print(f"Результат додавання: {result}") def main():
    num1 = 10
num2 = 20 calculate_sum(num1, num2) num3=5
num4=5
if  name     == "     main     ": main()
```

Рисунок 3.2 - Тестовий код для аналізу за допомогою Flake8

```
test.py:5:1: E302 expected 2 blank lines, found 1
test.py:11:5: F841 local variable 'num3' is assigned to but never used
test.py:11:9: E225 missing whitespace around operator
test.py:13:5: F841 local variable 'num4' is assigned to but never used
test.py:13:11: E225 missing whitespace around operator
test.py:15:1: E305 expected 2 blank lines after class or function definition, found 1
test.py:16:11: W292 no newline at end of file
```

Рисунок 3.3 – Результат аналізу тестового коду за допомогою Flake8

Як було зазначено раніше, Flake8 це інструмент, що включає функції декількох інших, тому був створений код для показового тестування. На відміну від попереднього інструменту, Flake8 має кольорове виділення інформації про помилки.

У цьому коді функція calculate_sum приймає два аргументи, додає їх та виводить результат. Функція main визначає дві змінні num1 та num2 та викликає функцію calculate_sum з цими аргументами. Далі створено, але так і не використано дві змінні num3 та num4.

Так було допущено декілька помилок, у вигляді зайвих порожніх рядків, та пробілів.

Bandit – інструмент, призначений для пошуку поширених проблем безпеки у коді Python. Для цього він аналізує кожен файл, будує з нього AST та запускає відповідні плагіни для вузлів AST.

Він призначений для виявлення дефектів безпеки:

1) Захардкожені паролі;
2) Неправильна серіалізація/десеріалізація pickle;
3) Ін'єкції оболонки;
4) Sql-ін'єкції.

Код наведено на рисунку 3.4, та проаналізовано за допомогою інструменту Bandit, результати наведено на рисунку 3.5

```
import os
def insecure_function(user_input):  1 usage
    os.system("rm -rf /") # Небезпечна операція def safe_function(user_input):
print (f"Введене значення: {user_input}") if    name    == "    main    ":
    user_input = input("Введіть дані: ") insecure_function(user_input) safe_function(user_input)
```

Рисунок 3.4 - Тестовий код для аналізу за допомогою Bandit

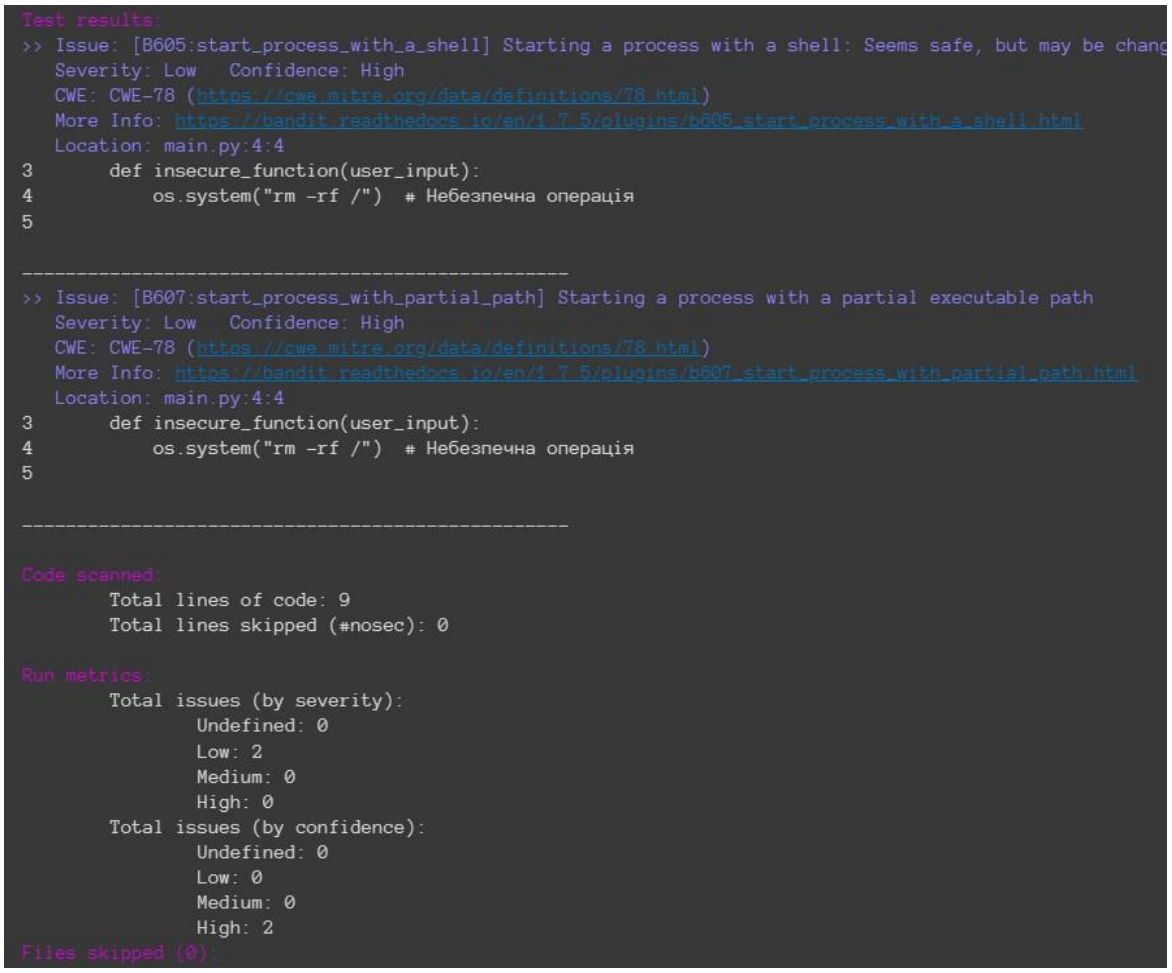

Рисунок 3.5 – Результат аналізу тестового коду за допомогою Bandit

Кожна з представлених бібліотек досить популярна і використовується розробниками для поліпшення свого коду, але слід зазначити, що в різних випадках слід використовувати відповідні інструменти.

3.3. Перспективні можливості вдосконалення системи доставки коду.

Швидкість та якість створення продукту – ключові елементи успіху у розробці програмного забезпечення, що робить їх основними чинниками конкуренції. У зв'язку з цим застарілі моделі розробки, такі як імперативні, структурні та модульні, замінені на новішу модель CI/CD.

CI/CD – це процес безперервної інтеграції та безперервної доставки продукту.

Концепція CI/CD відноситься до гнучких методологій розробки ПЗ і орієнтована на увагу до бізнес-вимог і безпеки, а також забезпечення високої якості кінцевого продукту. Цей підхід включає:

1) Автоматизацію процесів складання, пакування та тестування програмного забезпечення;

2) Автоматичне розгортання програми у різних оточеннях (наприклад, на тестових, проміжних та продуктових серверах);

3) Мінімізацію помилок та вразливостей у програмному продукті шляхом постійної перевірки та коригування коду.

Наведемо приклад: для того, щоб виконати розрахунки в EXCEL, зручніше скористатися формулою та отримати результат за секунду, ніж вираховувати кожен рядок вручну на калькуляторі.

Основними принципами CI/CD є поділ відповідальності за різні етапи процесу, мінімізація ризиків кожному етапі життєвого циклу продукту. Скорочення часу зворотний зв'язок з допомогою автоматизації. Створення єдиного робочого простору для розробників, який можна порівняти з виробничою, тестовою та середовищем для розробки.

Переваги CI/CD:

1) Підвищення швидкості розробки: розробники можуть швидше тестувати та вводити зміни до продукту;

2) Якість тестування: можливість виявлення помилок на ранній стадії розробки.

3) Вибір оптимального варіанта: розробники можуть протестувати різні варіанти коду швидко та впровадити найбільш оптимальний;

Під безперервною інтеграцією ми розуміємо розгортання коду в тестових системах, а під безперервною доставкою – доставку додатка до продукту (максимально спрощуючи: написав – відправив до репозиторію, все саме зібралося, автоматично протестувалося, встановилося).

CI/CD складається з git (інструмент для автоматизації рутинних завдань, що виникають у процесі розробки програмного забезпечення).

Автоматизовані інструменти розгортання: GitLab (інструмент для автоматизації процесів тестування, розгортання та моніторингу проектів).

Для того, щоб працювала CI/CD, потрібно налаштувати гід, розгорнути тестове оточення з необхідним переліком: фронт, тест, залежно від того, які рівні у замовника, та кінцевий сервер, на якому все має з'явитися після схвалення.

До розробки CI/CD підключені:

1) DevOps – одноосібний організатор усієї концепції;
2) Тімлід керує всіма процесами, частіше взаємодіє із замовником, ставить завдання девопсу, збирає концепцію додатку докупи;
3) Менеджер проекту веде облік витрат.

Інструменти для CI/CD:

1) GitLab – платформа для управління репозиторіями, документування результатів тестування та розробки, аналізу та покращення функціональності проекту, виявлення та виправлення помилок;

2) Docker – CD-система для контейнеризації проекту;

3) Travis-CI – сервер, який підключається до віртуальних GitHub-репозиторій з мінімальними налаштуваннями та не потребує окремої установки;

4) Jenkins – сумісний з різними плагінами для адаптації до різних проектів та завдань;

5) PHP Censor – CI-сервер для автоматизації збирання PHP-проектів. Може працювати з репозиторіями GitLab, Mercurial та ін, з тестовими бібліотеками Atoum, PHP Spec, Behat.

Суть безперервної інтеграції у тому, що це учасники процесу розробки регулярно публікують зміни, внесені у код, у центральному репозиторії. Початок впровадження безперервної інтеграції полягає у надсиланні змін до системи контролю вихідного коду.

Що стосується безперервного розгортання, то це вже процес випуску програмного забезпечення, в ході якого проводиться автоматичне тестування для перевірки правильності та стабільності змін у коді та подальшого автоматичного розгортання програмного забезпечення в робочому середовищі. Варто зазначити, що безперервна доставка є складовою процесу розгортання. Розгортання, по суті, є доставкою програмного забезпечення плюс його тестування перед офіційним випуском для користувачів.

4 ПРАКТИЧНА РЕАЛІЗАЦІЯ ПОЛІПШЕНЬ

4.1 Принципи роботи Terraform

Terraform — це потужний інструмент для автоматизації управління інфраструктурою на основі принципу "інфраструктура як код" (Infrastructure as Code, IaC). Він дозволяє описувати інфраструктуру в декларативних конфігураційних файлах, що забезпечує повторюваність, надійність і гнучкість процесу управління.

Декларативний підхід. Замість програмування послідовності дій, Terraform дозволяє описувати бажаний стан інфраструктури. Система сама визначає, які зміни потрібні для досягнення заданого результату.

Планування змін (Plan). Перед застосуванням будь-яких змін Terraform створює план, що демонструє, як нові конфігурації вплинуть на існуючу інфраструктуру. Це дозволяє уникнути небажаних наслідків.

Збереження стану (State). Terraform зберігає поточний стан інфраструктури у спеціальному файлі (state). Це дозволяє точно відстежувати зміни та автоматично синхронізувати нові конфігурації з поточним станом.

Підтримка модулів. Для спрощення роботи з великими проєктами використовуються модулі, які дозволяють повторно використовувати стандартні конфігурації.

Підтримка багатьох провайдерів. Terraform підтримує понад 100 провайдерів, включаючи AWS, Azure, Google Cloud та інші. Це дозволяє працювати з мультихмарними середовищами та інтегрувати різні сервіси.

Щодо переваг у використання Terraform у системі можна виділити автоматизація інфраструктури, тобто використання Terraform дозволяє повністю автоматизувати процеси розгортання та управління інфраструктурою.

Наприклад, у системі можна швидко налаштувати середовище для перевірки завдань.

Масштабованість. Завдяки підтримці масштабування ресурсів, система легко адаптується до збільшення кількості користувачів чи складності завдань.

Прозорість та контроль змін. Функція планування дозволяє аналізувати вплив запланованих змін ще до їх впровадження, що підвищує стабільність роботи системи.

Підтримка CI/CD процесів. Terraform інтегрується з інструментами безперервної інтеграції та доставки (CI/CD), такими як GitLab CI/CD. Це забезпечує автоматичне розгортання нових версій інфраструктури та підвищує ефективність розробки.

Зниження людського фактору. Завдяки автоматизації рутинних процесів мінімізується ризик помилок, пов'язаних із ручним управлінням інфраструктурою.

Практичне застосування у проєкті. У поточній системі Terraform був використаний для автоматизації створення та управління ключовими компонентами, такими як:

1) Віртуальні машини для виконання завдань;

2) Мережеві ресурси для забезпечення захищеного доступу;

3) Бази даних для зберігання результатів перевірки.

Одним з найпоширеніших сценаріїв використання Terraform є створення та управління віртуальними машинами у хмарних оточеннях. Припустимо, у вас є проект, який вимагає кілька віртуальних машин для хостингу програми. За допомогою Terraform це виконується так:

1) Створіть конфігураційний файл, наприклад main.tf, у якому визначте параметри віртуальних машин, такі як типи, кількість, образи та інші параметри

2) Ініціалізація. У командному рядку виконайте команду terraform init у директорії, де знаходиться ваш файл конфігурації. Це завантажить необхідні плагіни та налаштує робоче середовище;

3) Планування та застосування. Після успішної ініціалізації виконайте terraform plan для перегляду плану змін. Після перевірки, що все коректно, виконайте terraform apply для створення віртуальних машин;

4) Управління інфраструктурою. Можна використовувати Terraform для зміни кількості віртуальних машин, їх типів та інших параметрів. ПЗ виявить різницю між поточним станом та бажаним станом (визначеним у конфігураційних файлах) та застосує необхідні зміни;

5) Налаштування мережної інфраструктури. Terraform також дозволяє легко налаштовувати мережеві аспекти вашої інфраструктури. Припустимо, ви хочете створити віртуальну приватну мережу (VPC) і налаштувати в ній підмережі та правила фаєрволу;

6) Планування та застосування змін. Після визначення конфігурації виконайте terraform plan і terraform apply так само, як і для віртуальних машин. Terraform створить і налаштує VPC, підмережі та правила фаєрволу відповідно до ваших вказівок;

7) Управління. Подібно до віртуальних машин, ви можете вільно змінювати параметри, такі як підмережі та правила фаєрволу, шляхом редагування конфігураційних файлів та впровадження необхідних змін;

Як бачимо, Terraform став невід'ємною частиною сучасного програмування хмар. Його простота у використанні, гнучкість та підтримка різних хмарних провайдерів роблять його кращим вибором для багатьох компаній та розробників.

4.2. Впровадження інфраструктури як коду за допомогою Terraform в проект

Щоб створювати ресурси без ризику щось забути і робити це швидко при повторенні — вигадали Infrastructure as Code. Це підхід до управління інфраструктурою через набори конфігураційних файлів (коду), у яких декларативно (тобто специфікації фінального результату) описується бажаний стан цієї інфраструктури. Такий підхід вирішує багато завдань та проблем, наприклад:

1) Конфігурація є джерелом правди у тому, як усе має бути налаштовано;

2) Вона часто є і документацією;

3) Повторне застосування цієї конфігурації (наприклад, вам потрібно кілька оточень) гарантовано призводять до того ж результату;

4) Усувається людський фактор (коли забули щось додати чи налаштувати);

5) Terraform – це один із інструментів реалізації Infrastructure as Code;

Terraform (утиліта командного рядка) працює з конфігураційними файлами (кодом) та інтерпретує їх усередині cloud або on-premise платформи – створює, змінює чи видаляє ресурси.

Terraform підтримує велику кількість платформ: від основних хмарних провайдерів (AWS, Azure, GCP) до скромніших платформ, наприклад Hetzner або 1&1. Крім того, він підтримує роботу з таким програмним забезпеченням, як Docker, Kubernetes, Chef, що розширює функціонал і дозволяє використовувати їх у зв'язці. Тому Terraform такий популярний – це справжній швейцарський ніж у світі управління інфраструктурою.

Код Terraform використовує свій власний синтаксис, але виглядає дуже доброзичливо. Наприклад, на рисунку 4.1 можна побачити мінімальний блок коду, який створює EC2 instance в AWS.

```
resource "aws_instance" "web_server" {
  ami = "ami-a1b2c3d4"
  instance_type = "t3.micro"
}
```

Рисунок 4.1- мінімальний блок коду для створення EC2

Terraform підтримує безліч так званих провайдерів, які дозволяють інтерпретувати код для тієї чи іншої платформи хостингу.

Щоб застосувати конфігурацію, використовується CLI-додаток, який запускає код на обробку та застосування провайдером по відношенню до тієї чи іншої платформи.

При першому використанні Terraform відбувається ініціалізація проекту, каталогу, в якому знаходяться конфігураційні файли (ваш код). Для цього використовується команда terraform init. Ця команда дозволяє завантажити необхідні версії провайдерів та інших залежностей, які використовуються у конфігурації інфраструктури.

Далі під час першого застосування команди terraform apply Terraform створить описані ресурси на хостинг-платформі та створить спеціальний файл – state file – у каталозі проекту.

Саме цей файл буде використовуватися Terraform для порівняння поточного стану вашої інфраструктури з новою версією вашої конфігурації щоразу при виконанні terraform apply. У такий спосіб він зрозуміє: які ресурси змінити, видалити чи додати.

У синтаксису Terraform є безліч інструментів для організації коду та розширення можливостей роботи з ним, наприклад:

Modules – дозволяють створювати логічні блоки (абстракції) з різних наборів ресурсів та знову використовувати такі блоки надалі.

Variables – дозволяють параметризувати код, роблячи його універсальнішим.

Expressions — дозволяє обчислювати або звертатися до різних типів даних у коді. Наприклад, взяти елемент зі списку або вибрати якесь значення на підставі умови.

Functions — вбудовані функції, які ще більше розширюють можливості роботи з даними всередині конфігурації. Наприклад, перетворення рядків, математичні обчислення, робота зі списками тощо.

Це та багато іншого робить Terraform потужним інструментом для управління інфраструктурою: як усередині однієї хостинг-платформи, так і між кількома одночасно.

Система автоматизації перевірки виконання практичних завдань в локальному віртуальному оточенні є важливим інструментом для оцінювання навичок учнів у віддаленому навчанні. Основною метою даного дослідження є вдосконалення архітектури цієї системи, що включає в себе аналіз існуючих рішень та пропозицію нових підходів до автоматизації.

Традиційні системи управління навчанням (LMS) часто не забезпечують достатньої точності та ефективності в оцінюванні практичних навичок. Це пов'язано з тим, що вони зазвичай призначені для управління контентом та проведення тестувань, але не мають механізмів для оцінювання практичних завдань, що є критично важливим у багатьох галузях.

Для дослідження було обрано методику, яка включає:

1) Аналіз існуючих систем: Оцінка поточних архітектур рішень для автоматизації перевірки;

2) Розробка прототипу: Створення вдосконаленого прототипу системи, що включає локальне віртуальне середовище для виконання практичних завдань;

3) Інтеграція з LMS: Розробка механізмів для інтеграції з існуючими LMS через протоколи, що дозволяє забезпечити безперебійну взаємодію між компонентами системи.

Архітектура системи. Система складається з кількох ключових компонентів:

1) Серверний додаток: Відповідає за обробку запитів на перевірку завдань та управління даними;

2) Автоматизований перевірник: Окремий сервіс, який виконує роль перевірника завдань;

3) Клієнтське програмне забезпечення: Локальне середовище учня, яке використовує віртуальну машину для виконання завдань;

4) Скрипти синхронізації: Забезпечують обмін даними між локальним середовищем і сервером перевірки.

Починаємо написання коду із створення головного файлу для створення інфраструктури проекту – main.tf. Він представлений рисунку 4.2

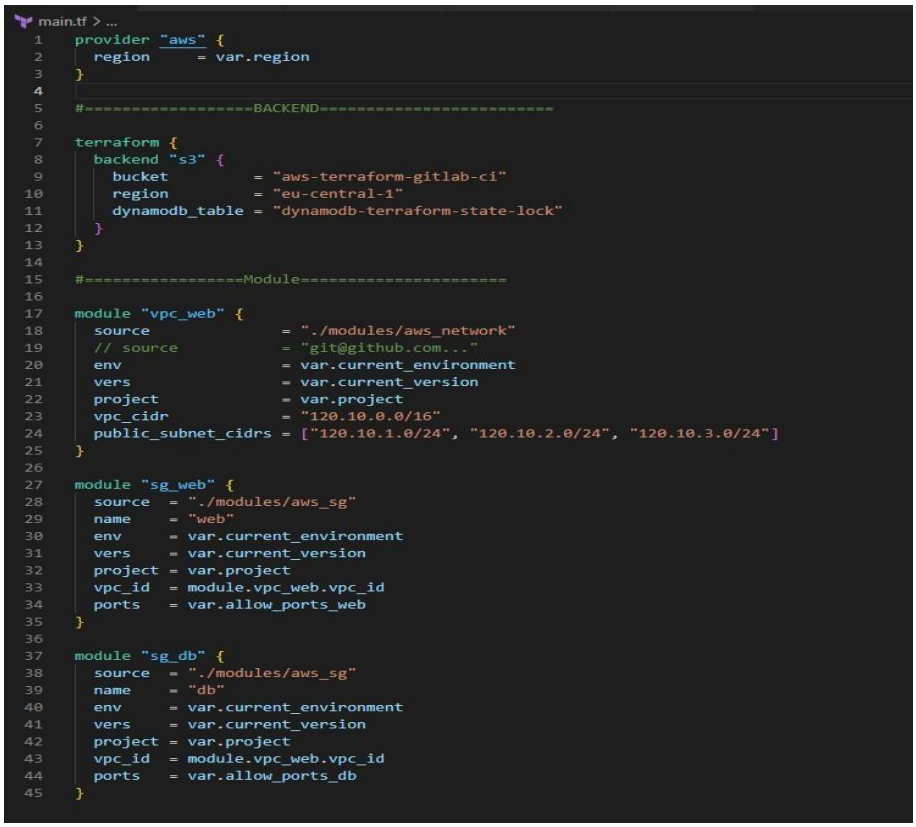

Рисунок 4.2 – Код файлу main.tf

У цьому файлі ми визначаємо провайдера AWS, налаштовуємо backend для збереження стану VPC та груп безпеки (Security Group).

Використання модулів дозволяє нам повторно використовувати налаштування та забезпечує шаблонизацію інфраструктури для різних середовищ та проектів.

Модуль aws_network, код якого наведено на рисунку 4.3, виконує налаштування мережі та публічних підмереж (subnets), необхідних для коректної роботи інфраструктури.

Код цього модуля починається з визначення даних про доступні зони доступності AWS (Availability Zones) Ці дані використовуються для розміщення публічних підмереж у різних зонах доступності, щоб забезпечити високу доступність та відмовостійкість.

Далі слідує визначення ресурсів. Першим ресурсом є aws_vpc, який визначає віртуальну приватну хмарну мережу (VPC) із зазначеним діапазоном IP-адрес.

Наступний ресурс – aws_internet_gateway, який створює шлюз до Інтернету для забезпечення зовнішнього доступу до публічних підмереж VPC.

Після цього слід визначення ресурсу aws_route_table, який встановлює таблицю маршрутизації для громадських підмереж.

Ресурс aws_route_table_association пов'язує кожну публічну мережу з відповідною таблицею маршрутизації.

```
data "aws_availability_zones" "availability"{}

resource "aws_vpc" "main" {
    cidr_block = var.vpc_cidr

    instance_tenancy        = "default"
    enable_dns_hostnames    = true
    tags                    = {Name = "VPC-${var.project}-${var.env}-V${var.vers}"}
}

resource "aws_internet_gateway" "main" {
  vpc_id    = aws_vpc.main.id
  tags      = {Name = "GateWay-${var.project}-${var.env}-V${var.vers}"}

}

resource "aws_route_table" "public_subnets" {
  vpc_id = aws_vpc.main.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.main.id
  }
  tags = {Name = "Route-${var.project}-${var.env}-V${var.vers}"}
}

resource "aws_route_table_association" "public_routes" {
  count          = length(aws_subnet.public_subnets[*].id)
  route_table_id = aws_route_table.public_subnets.id
  subnet_id      = element(aws_subnet.public_subnets[*].id, count.index)
}
#=========================Subnets=======================================

resource "aws_subnet" "public_subnets" {
  count = length(var.public_subnet_cidrs)
  vpc_id = aws_vpc.main.id
  availability_zone = data.aws_availability_zones.availability.names[count.index]
  cidr_block = element(var.public_subnet_cidrs, count.index)
  map_public_ip_on_launch = true
  tags = {Name = "Subnet-public-${var.project}-${var.env}-V${var.vers}-${count.index + 1}"}
}
```

Рисунок 4.3 – Код модулю aws_network

Останній ресурс – aws_subnet визначає публічні підмережі, які розташовані в різних зонах доступності. Кожна публічна мережа має свій IP-діапазон і зв'язується з таблицею маршрутизації та VPC.

Цей модуль допомагає створити та налаштувати основну мережну інфраструктуру, що включає VPC, публічні підмережі та необхідні налаштування маршрутизації для забезпечення зовнішнього доступу до ресурсів у цій мережі.

Наступний модуль aws_sg визначає ресурс aws_security_group, який виконує налаштування групи безпеки для контролю доступу до ресурсів у мережі. Код цього модуля представлено рисунку 4.4.

```
resource "aws_security_group" "main" {
  name        = "Security group-${var.name}-${var.project}-${var.env}-V${var.vers}"
  description = "SecurityGroup for ${var.project}"
  vpc_id      = var.vpc_id
  dynamic "ingress" {
    for_each = var.ports
    content {
          from_port        = ingress.value
          to_port          = ingress.value
          protocol         = "tcp"
          cidr_blocks      = ["0.0.0.0/0"]
    }
  }

  egress {
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
    cidr_blocks      = ["0.0.0.0/0"]
  }

  tags = {
    Name = "SG-${var.name}-${var.project}-${var.env}-V${var.vers}"
  }
}
```

Рисунок 4.4 – Код модулю aws_sg

Основні атрибути ресурсу включають назву групи безпеки (name), опис (description) і ідентифікатор VPC (vpc_id), до якого вона відноситься. Також використовується директива "dynamic" визначення вхідних правил доступу. Кожне правило визначається за допомогою блоку "content", де вказується порт (from_port, to_port), протокол (protocol) та дозволені IP-адреси (cidr_blocks). У цьому випадку вхідні правила дозволяють доступ з будь-яких IP-адрес ("0.0.0.0/0") на вказані порти.

Також визначається блок "egress", що встановлює правила для вихідного трафіку групи безпеки. У цьому випадку всім вихідним з'єднанням дозволяється надсилати будь-який трафік ("0.0.0.0/0").

Назва та теги групи безпеки також визначаються на основі вхідних змінних, що містять інформацію про назву (name), проект (project), середовище (env) та версію (vers).

Модуль допомагає створити та налаштувати групу безпеки, яка визначає правила доступу до ресурсів мережі, що забезпечує контроль трафіку та безпеку системи.

Після створення VPC та Security Group починаємо описувати, як виглядатиме наша база даних RDS у файлі database.tf. Код цього файлу наведено рисунку 4.5.

```
resource "aws_db_instance" "db" {
  allocated_storage       = 10

  db_subnet_group_name    = aws_db_subnet_group.main.name

  identifier              = "db-${var.project}-${var.current_environment}"
  engine                  = "mysql"
  engine_version          = "5.7"
  instance_class          = "db.t3.micro"
  parameter_group_name    = "default.mysql5.7"
  skip_final_snapshot     = true
  #publicly_accessible     = true

  db_name                 = var.namedb
  username                = var.usernamedb
  password                = var.passworddb

  vpc_security_group_ids  = [module.sg_db.sg_id]
  availability_zone       = data.aws_availability_zones.availability.names[1]
  #lifecycle {
  #  prevent_destroy = true
  #}
}

resource "aws_db_subnet_group" "main" {
  name            = "${var.project}-${var.current_environment}-v${var.current_version}"
  subnet_ids      = flatten([module.vpc_web.public_subnet_ids])
  tags            = merge(var.common_tag, {Name = "DB-SubnetGroup-${var.project}-${var.current_environment}-v${var.current_version}"})

}
```

Рисунок 4.5 – Код файлу database.tf

Цей код визначає два ресурси: aws_db_instance та aws_db_subnet_group, які виконують налаштування бази даних та групи підмереж бази даних у середовищі AWS.

Ресурс aws_db_instance визначає параметри та налаштування для створення екземпляра бази даних. Він містить такі атрибути, як обсяг виділеного сховища (allocated_storage), група підмереж бази даних (db_subnet_group_name), ідентифікатор (identifier), тип бази даних (engine), версія бази даних (engine_version), клас екземпляра бази даних (instance_class), група параметрів (parameter_group_name), наявність publicly_accessible) і т.д.

Вказуються назва бази даних (db_name), ім'я користувача (username) та пароль (password), а також ідентифікатори групи безпеки VPC (vpc_security_group_ids) та доступні зони доступності (availability_zone). Цей ресурс допомагає створити та налаштувати екземпляр бази даних MySQL.

Ресурс aws_db_subnet_group визначає групу підмереж бази даних, яка використовується для налаштування доступу до бази даних. Цей ресурс допомагає створити групу підмереж бази даних для використання в ресурсах бази даних.

Код дозволяє створити та налаштувати екземпляр бази даних та групу підмереж бази даних, що забезпечує інфраструктуру для зберігання та доступу до даних у середовищі AWS.

Для Terroform, щоб отримати доступ до AWS необхідно попередньо створити обліковий запис відвідавши веб-сайт aws.amazon.com. Далі налаштовуємо політику Identity and Access Management (IAM), створити користувача і надати необхідні дозволи йому, зокрема AdministratorAccess, для доступу до ресурсів AWS.

Після цього створюємо та зберігаємо необхідні ідентифікатори доступу (Access Key ID) та секретні ключі (Secret Access Key) для автентифікації нашого коду Terraform.

Для зберігання стану та блокування ресурсів Terraform також необхідно створити S3 bucket та таблицю DynamoDB.

Попередньо створивши обліковий запис на сайті gitlab.com, створюємо репозиторій, в якому зберігатимемо наші файли. Далі створюємо потрібні нам гілки, вони будуть вказувати на наше середовище, тобто для кожної обраної гілки (dev, prod, test, main), створюватиметься своя інфраструктура для майбутнього управління та розвитку коду.

Наступним кроком перед написанням конфігураційного файлу .gitlab-ci.yml потрібно встановити в GitLab, у налаштуваннях CI/CD, наші змінні, як показано на рисунку 4.6 Вони будуть використовуватися в конвеєрі, щоб передати GitLab Runner ідентифікатори доступу (Access Key ID) та секретні ключі (Secret Access Key), а також пароль, ім'я користувача та назву docker image, який буде встановлений на EC2.

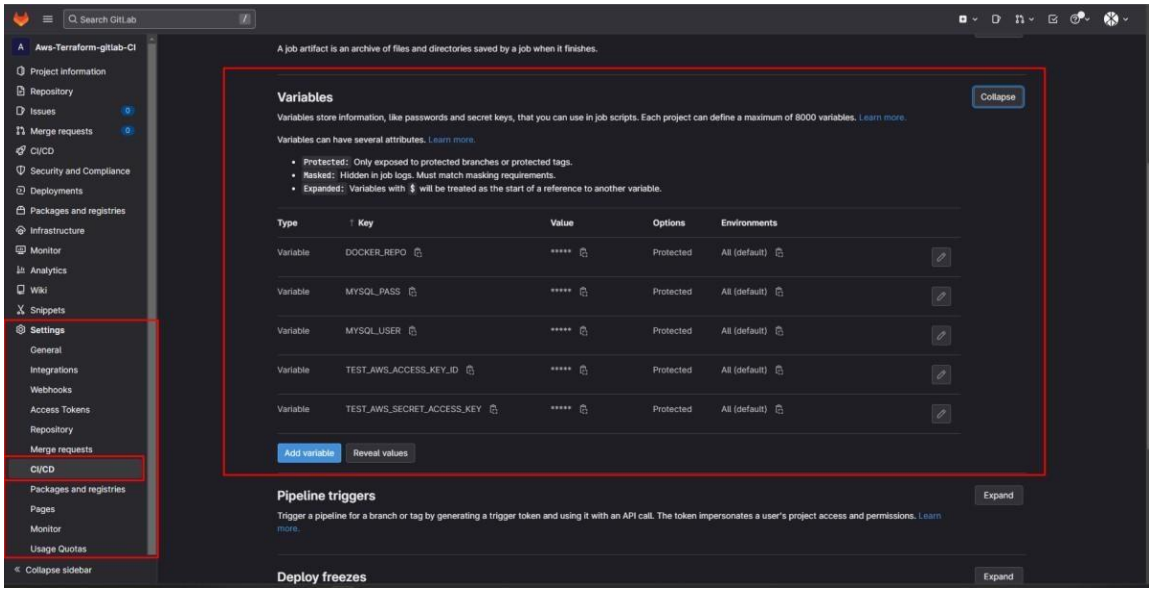

Рисунок 4.6– Налаштування змінних у GitLab CI/CD

Після цього переходимо до створення GitLab Runner. Він може бути встановлений будь-де, на будь-якій операційній системі та хмарному середовищі. Для зручності, GitLab Runner, була встановлена локально на віртуальну машину Ubuntu.

У налаштуваннях CI/CD знаходимо поле Runners та встановлюємо GitLab Runner на Ubuntu за інструкцією. На рисунку 4.7 показані команди для встановлення Runner на Linux.

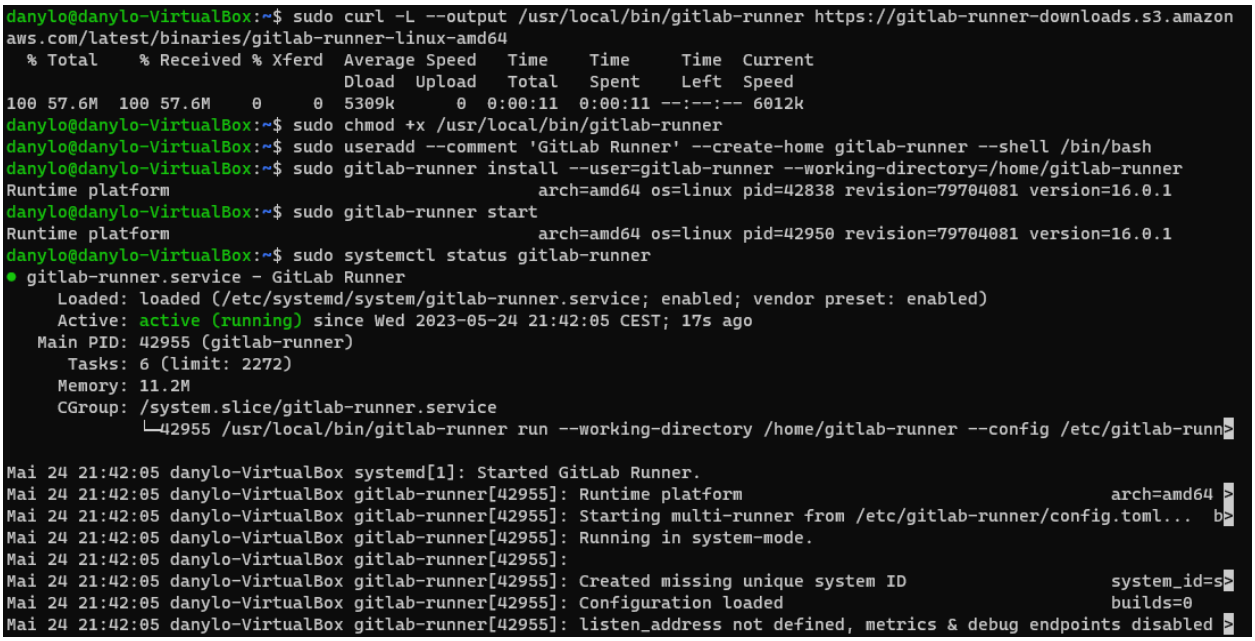

Рисунок 4.7 – Команди для встановлення GitLab Runner

Далі потрібно зареєструвати Runner, щоб з'єднати його з GitLab. Для цього потрібно використовувати токен, вказаний у налаштуваннях CI/CD Runners.

На рисунку 4.8 показано приклад команди, з допомогою якої можна зробити. Після її запуску дані вводяться в інтерактивному режимі, залишаємо все за замовчуванням, крім description for the runner, tags та executor.

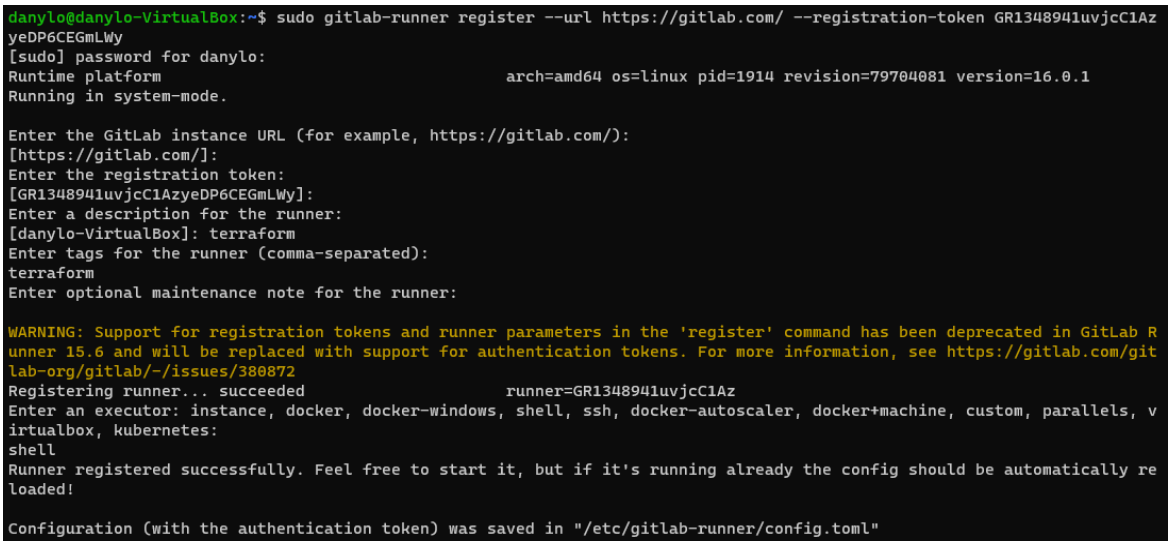

Рисунок 4.8 – Приклад реєстрації GitLab Runner

Важливо, щоб сервер, на якому встановлений GitLab Runner, мав доступ до інтернету, а також щоб на ньому було встановлено Terraform. Вдосконалена архітектура системи позитивно вплинула на якість і швидкість доставки програмного коду до серверного додатка. Використання локального оточення дозволило значно знизити експлуатаційні витрати на підтримку роботи системи4. Прототип був успішно апробований у навчальному процесі, що підтвердило його ефективність у реальних умовах.

Дослідження показало, що автоматизація перевірки виконання практичних завдань у локальному віртуальному оточенні є перспективним напрямком для покращення якості освіти в умовах дистанційного навчання. Подальші дослідження можуть зосередитися на розширенні функціоналу системи та її інтеграції з новими технологіями навчання.

Цей підхід дозволяє не лише підвищити ефективність оцінювання, але й забезпечити гнучкість у використанні ресурсів, що є особливо важливим в умовах обмежених бюджетів на освіту.

Для управління інфраструктурою було впроваджено CI/CD пайплайн, який автоматизує основні процеси роботи з Terraform. Це рішення дозволило значно підвищити ефективність та зменшити кількість ручних операцій, пов'язаних із розгортанням і змінами в інфраструктурі.

Пайплайн реалізовано в системі GitLab CI/CD. Основними етапами його роботи є:

1) Перевірка коду: Перевірка валідності конфігурацій за допомогою команди terraform validate. Цей етап забезпечує впевненість у правильності синтаксису та логіки написаного коду ще до внесення змін до інфраструктури;

2) Планування змін: Генерація плану змін через terraform plan, що дозволяє команді ознайомитися з тим, які саме ресурси будуть створені, змінені або видалені. Цей етап надає прозорість процесу та можливість схвалити зміни перед застосуванням;

3) Застосування змін: Автоматичне виконання команди terraform apply, яка безпосередньо розгортає зміни в інфраструктурі. Цей етап повністю виключає необхідність ручного виконання команд;

4) Управління знищенням ресурсів: У пайплайні також передбачено етап terraform destroy, який виконується вручну для знищення інфраструктури, якщо це потрібно.

Код пайплайна структурований у файлі .gitlab-ci.yml, який розміщено у ДОДАТКУ Б, де представлено детальний вміст цього файлу, включаючи змінні середовища та залежності між етапами.

Інтеграція пайплайна з Terraform дозволила досягти значних покращень в управлінні інфраструктурою. Автоматизація процесу зменшує середній час розгортання, забезпечуючи прозорість на кожному етапі. Кожен етап пайплайна документується, а результати зберігаються у вигляді логів та артефактів, доступних для аналізу, що підвищує зручність роботи команди. Крім того, перевірки на кожному етапі знизили ризик виникнення помилок у процесі роботи з інфраструктурою, забезпечуючи стабільність і надійність. Таким чином, впровадження CI/CD пайплайна стало важливим кроком до підвищення ефективності та якості управління інфраструктурою.

5 ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ЗАСТОСУВАННЯ СИСТЕМИ

5.1 Ефективність застосування Terraform

Terraform – це інструмент інфраструктурного управління, створений компанією HashiCorp. Він став популярним серед інженерів та розробників завдяки здатності автоматизувати створення, налаштування та управління інфраструктурою у хмарних сховищах та централізованих дата-центрах.

До оптимізації код Terraform мав велику кількість дублювань, що ускладнювало його підтримку та повторне використання. Відсутність модульного підходу значно впливала на гнучкість у роботі з конфігураціями, а всі операції виконувалися вручну, що призводило до тривалого часу розгортання та ризиків людських помилок. Завдяки впровадженню модулів вдалося винести спільні компоненти, такі як VPC, EC2 та Security Groups, у повторно використовувані блоки, що зменшило обсяг коду приблизно на 25%. Структуризація та параметризація дозволили швидко масштабувати інфраструктуру, додаючи нові ресурси шляхом змін значень змінних. Також перевірка коду через terraform validate допомогла знизити кількість помилок на етапі конфігурації.

Після оптимізації процес роботи з Terraform виглядає наступним чином. На першому етапі виконується перевірка валідності коду, що дозволяє впевнитись у відсутності синтаксичних помилок. Далі генерується план змін за допомогою terraform plan, який показує, які ресурси будуть створені, змінені чи видалені (рис 5.1). Після затвердження плану команда terraform apply (рис. 5.2) вносить зміни до інфраструктури. Такий підхід значно підвищив ефективність і зручність управління інфраструктурою.

```
Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
  + create
Terraform will perform the following actions:
  # aws_instance.this[0] will be created
  + resource "aws_instance" "this" {
      + ami                                  = "ami-046d18c147c36bef1"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
      + iam_instance_profile                 = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_lifecycle                   = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t2.micro"
      + ipv6_address_count                   = (known after apply)
      + ipv6_addresses                       = (known after apply)
      + key_name                             = (known after apply)
      + monitoring                           = false
      + outpost_arn                          = (known after apply)
      + password_data                        = (known after apply)
      + placement_group                      = (known after apply)
      + placement_partition_number           = (known after apply)
      + primary_network_interface_id         = (known after apply)
      + private_dns                          = (known after apply)
      + private_ip                           = (known after apply)
      + public_dns                           = (known after apply)
      + public_ip                            = (known after apply)
      + secondary_private_ips                = (known after apply)
      + security_groups                      = (known after apply)
```

Рисунок 5.1 – Результат виконання команди terraform plan для EC2

```
module.vpc.random_integer.random: Creating...
module.vpc.random_integer.random: Creation complete after 0s [id=65]
module.vpc.aws_vpc.kp_vpc: Creating...
module.vpc.aws_vpc.kp_vpc: Still creating... [10s elapsed]
module.vpc.aws_vpc.kp_vpc: Creation complete after 12s [id=vpc-0870f7a15ace372ab]
module.vpc.aws_subnet.kp_pb_sn[1]: Creating...
module.vpc.aws_subnet.kp_pb_sn[0]: Creating...
module.vpc.aws_security_group.kp_pb_sg: Creating...
module.vpc.aws_security_group.kp_pb_sg: Creation complete after 3s [id=sg-0da9dcc31a7b58639]
module.ec2.aws_launch_template.asg_webserver: Creating...
module.ec2.aws_launch_template.asg_webserver: Creation complete after 1s [id=lt-058a134f972b88e45]
module.vpc.aws_subnet.kp_pb_sn[0]: Still creating... [10s elapsed]
module.vpc.aws_subnet.kp_pb_sn[1]: Still creating... [10s elapsed]
module.vpc.aws_subnet.kp_pb_sn[0]: Creation complete after 11s [id=subnet-07af5417da4009281]
module.vpc.aws_subnet.kp_pb_sn[1]: Creation complete after 11s [id=subnet-09e63c9ba13cf05d7]
module.ec2.aws_autoscaling_group.asg_webserver: Creating...
module.ec2.aws_autoscaling_group.asg_webserver: Still creating... [10s elapsed]
module.ec2.aws_autoscaling_group.asg_webserver: Creation complete after 17s [id=asg_webserver]

Apply complete! Resources: 7 added, 0 changed, 0 destroyed.
```

Рисунок 5.2 - Результат виконання команди terraform apply

Автоматизація роботи з Terraform була реалізована через інтеграцію репозиторію GitLab та зберігання стану інфраструктури в S3-бакеті на AWS. Це рішення забезпечує централізований підхід до управління інфраструктурою, дозволяючи кільком інженерам працювати з одними і тими ж конфігураціями. Завдяки GitLab CI/CD, будь-які зміни у коді, наприклад, у файлі main.tf, автоматично запускають пайплайн. Інженери можуть вносити правки до конфігурацій, які після коміту перевіряються, генерується план змін, а потім відбувається їх застосування після ручного підтвердження.

Коли код редагується, GitLab має запустити конвеєр і автоматично оновити зміни. Щоб емулювати це, ми змінили код, щоб мати бажану ємність трьох екземплярів EC2 замість двох (рис 5.3).

main.tf

```
# --- ec2/main.tf

resource "aws_launch_template" "asg_webserver" {
  name_prefix            = "webserver"
  image_id               = "ami-026b57f3c383c2eec"
  instance_type          = var.webserver_type
  vpc_security_group_ids = [var.kp_pb_sg]
  key_name               = var.key

  tags = {
    Name = "webserver"
  }
}

resource "aws_autoscaling_group" "asg_webserver" {
  name                = "asg_webserver"
  vpc_zone_identifier = tolist(var.kp_pb_sn)
  min_size            = 3
  max_size            = 4
  desired_capacity    = 3

  launch_template {
    id      = aws_launch_template.asg_webserver.id
    version = "$Latest"
  }
}

```

Рисунок 5.3 – Зміна кількості екземплярів EC2

Зафіксували зміни, і ми бачимо, що знову запустився конвеєр (рис 5.4).

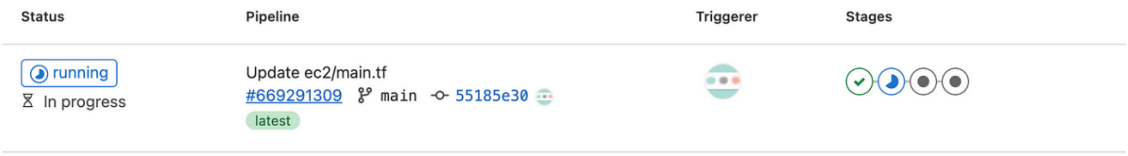

Рисунок 5.4 – Автоматичний запуск конвеєра після змін в коді

Далі ми можемо побачити в консолі управління AWS, що у нас створюється третя нова EC2 машина (рис. 5.5).

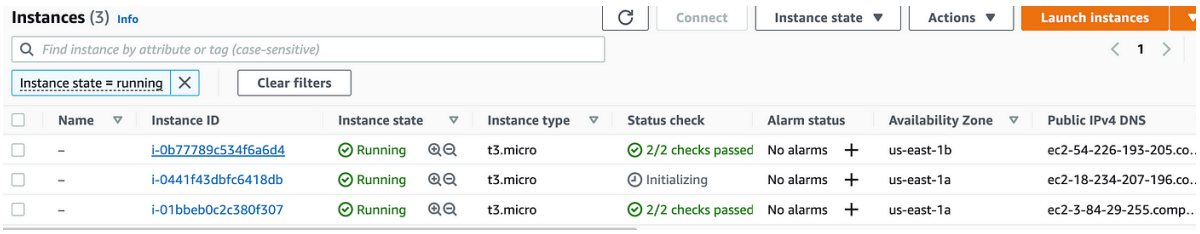

Рисунок 5.5 – Створення нової EC2 машини

Інтеграція GitLab CI/CD з Terraform та використання S3-бакета для збереження стану значно спростили управління інфраструктурою. Автоматизація всіх етапів, від перевірки валідності до застосування змін, забезпечила прозорість і стабільність процесів, а також дозволила кільком інженерам працювати над одним проектом без конфліктів. Впровадження таких практик дозволяє не лише оптимізувати робочі процеси, але й забезпечити готовність системи до масштабування та швидкого внесення змін. Це демонструє ефективність підходу Infrastructure as Code у сучасному управлінні інфраструктуро.

ВИСНОВКИ

Виконана кваліфікаційна робота підтвердила актуальність автоматизації інфраструктури за допомогою сучасних технологій. Основна мета, що полягала у вдосконаленні існуючої системи управління інфраструктурою через впровадження Terraform та CI/CD, була досягнута. У ході роботи виконано низку практичних задач, які забезпечили значне покращення продуктивності та зручності використання системи.

По-перше, реалізація модульного підходу в Terraform дозволила оптимізувати код, зробивши його більш структурованим і придатним для повторного використання. Це значно спростило процес управління інфраструктурою та забезпечило стабільність при її масштабуванні.

По-друге, налаштовано CI/CD пайплайн у GitLab для автоматизації процесів перевірки, планування та застосування змін. Завдяки цьому вдалося значно скоротити час на виконання рутинних операцій і зменшити ризик помилок, пов'язаних із людським фактором.

По-третє, проведена інтеграція з AWS S3 для зберігання стану інфраструктури. Це дозволило забезпечити синхронну роботу кількох інженерів у єдиному середовищі, що сприяло підвищенню ефективності командної роботи.

Крім того, розглянуто перспективи подальших покращень, таких як впровадження кешування за допомогою Redis та механізмів балансування навантаження для підвищення продуктивності системи. Однак реалізація цих рішень потребує додаткових ресурсів, що є основою для подальших досліджень та розвитку.

Іншим перспективним напрямком є перехід на повністю хмарну інфраструктуру, що дозволить забезпечити глобальну доступність системи, мінімізувати витрати на підтримку локального середовища та підвищити її продуктивність. Додатково варто розглянути можливість створення мобільної версії системи, яка значно розширить аудиторію користувачів, особливо серед студентів, які активно використовують мобільні пристрої для навчання.

У підсумку, результати роботи підтверджують доцільність впровадження сучасних технологій автоматизації в освітній процес. Запропоновані рішення не лише покращили існуючу систему, але й створили основу для її подальшого розвитку, забезпечуючи високу ефективність, зручність і стабільність роботи. Розроблений підхід може бути використаний як модель для інших проєктів, спрямованих на автоматизацію та вдосконалення освітніх процесів у сучасному цифровому світі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Документація до LMS Moodle [Електронний ресурс]. Режим доступу: https://docs.moodle.org/403/en/Main_page.
2. Специфікація LTI [Електронний ресурс]. Режим доступу: https://www.imsglobal.org/spec/lti/v1p3/impl/.
3. Communication protocol [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/Communication_protocol.
4. Serverless and IaC // International Journal For Science Technology And Engineering [Електронний ресурс]. Режим доступу: https://www.ijraset.com/bestjournal/serverless-and-iac.
5. Continuous Integration and Continuous Deployment (CI/CD) for Web Applications on Cloud Infrastructures [Електронний ресурс]. Режим доступу: https://www.atlassian.com/microservices/microservicesarchitecture/microservi ces-vs-monolith.
6. Федорченко О. М., Костромицький А. І. Дослідження ефективності системи автоматизації перевірки виконання практичних завдань у локальному віртуальному оточенні користувача. // 28-й Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті», Харків, 2024.
7. Документація Terraform [Електронний ресурс]. Режим доступу: https://registry.terraform.io/.
8. AWS Documentation [Електронний ресурс]. Режим доступу: https://aws.amazon.com/documentation/.
9. HashiCorp. Terraform: Infrastructure as Code [Електронний ресурс]. Режим доступу: https://www.terraform.io/.
10. Jenkins Documentation [Електронний ресурс]. Режим

доступу: https://www.jenkins.io/doc/.

11. Kubernetes Documentation [Електронний ресурс]. Режим доступу: https://kubernetes.io/docs/.
12. Docker Documentation [Електронний ресурс]. Режим доступу: https://docs.docker.com/.
13. Puppet Documentation [Електронний ресурс]. Режим доступу: https://puppet.com/docs/.
14. Ansible Documentation [Електронний ресурс]. Режим доступу: https://docs.ansible.com/.
15. Python Official Documentation [Електронний ресурс]. Режим доступу: https://docs.python.org/.
16. GitLab Documentation [Електронний ресурс]. Режим доступу: https://docs.gitlab.com/.
17. CI/CD Best Practices [Електронний ресурс]. Режим доступу: https://www.atlassian.com/continuous-delivery/principles.
18. DevOps Practices in Modern IT Environments [Електронний ресурс]. Режим доступу: https://www.redhat.com/en/topics/devops.
19. HashiCorp. Infrastructure as Code Patterns [Електронний ресурс]. Режим доступу: https://learn.hashicorp.com/tutorials/terraform.
20. Martin Fowler. Continuous Delivery [Електронний ресурс]. Режим доступу: https://martinfowler.com/bliki/ContinuousDelivery.html.
21. Atlassian. Microservices vs. Monolithic Architecture [Електронний ресурс]. Режим доступу: https://www.atlassian.com/microservices.
22. VMware Tanzu. Kubernetes Operations Guide [Електронний ресурс]. Режим доступу: https://tanzu.vmware.com/content/operations.
23. HashiCorp Whitepaper: Infrastructure Automation [Електронний

ресурс]. Режим доступу: https://www.hashicorp.com/resources.

24. OpenStack Documentation [Електронний ресурс]. Режим доступу: https://docs.openstack.org/.

25. Postman API Documentation [Електронний ресурс]. Режим доступу: https://learning.postman.com/docs/.

26. Moodle Integration with Third-party Systems [Електронний ресурс]. Режим доступу: https://docs.moodle.org/dev/Integration.

27. Infrastructure as Code: Best Practices [Електронний ресурс]. Режим доступу: https://www.thoughtworks.com/insights/articles.

28. Nginx Documentation [Електронний ресурс]. Режим доступу: https://nginx.org/en/docs/.

29. Google Cloud Documentation [Електронний ресурс]. Режим доступу: https://cloud.google.com/docs/.

30. Helm Kubernetes Package Manager Documentation [Електронний ресурс]. Режим доступу: https://helm.sh/docs/.

31. Федорченко О. М. Впровадження автоматизації перевірки завдань у локальному середовищі. // Праці міжнародної конференції, 2023.

32. Шкарупа О. В., Ларіонова І. О. Інтеграція хмарних сервісів у освітній процес. // Вісник Київського університету, 2022.

33. Ашурков А. І., Сидоренко Б. В. Використання Docker та Kubernetes у дистанційному навчанні. // Журнал технологій освіти, 2021.

34. Microsoft Azure Documentation [Електронний ресурс]. Режим доступу: https://docs.microsoft.com/en-us/azure/.

35. ElasticSearch Documentation [Електронний ресурс]. Режим доступу: https://www.elastic.co/guide/index.html.

36. OpenID Connect Specification [Електронний ресурс]. Режим

доступу: https://openid.net/specs/openid-connect-core-1_0.html.

37. Apache Kafka Documentation [Електронний ресурс]. Режим доступу: https://kafka.apache.org/documentation/.

38. GitHub Actions Documentation [Електронний ресурс]. Режим доступу: https://docs.github.com/en/actions.

39. Cloud Native Computing Foundation. Kubernetes Resources [Електронний ресурс]. Режим доступу: https://www.cncf.io/.

40. Jenkins Pipeline Syntax [Електронний ресурс]. Режим доступу: https://www.jenkins.io/doc/pipeline/.

41. Дослідження шляхів автоматизації перевірки виконання практичних завдань в локальному віртуальному оточенні користувача. Розробка API [Електронний ресурс]. Режим доступу: https://openarchive.nure.ua/

42. Дослідження шляхів автоматизації перевірки виконання практичних завдань в локальному віртуальному оточенні користувача. Розробка архітектури системи [Електронний ресурс]. Режим доступу: https://openarchive.nure.ua/.

43. Дослідження ефективності системи автоматизації перевірки виконання практичних завдань в локальному віртуальному оточенні користувача [Електронний ресурс]. Режим доступу: https://openarchive.nure.ua/