МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

Навчально-науковий інститут кібернетики, інформаційних технологій та інженерії

Кафедра комп'ютерних наук та прикладної математики

"*До захисту допущена*"
Зав. кафедри комп'ютерних наук
та прикладної математики
д.т н., проф. Ю.В. Турбал
«____» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

Автоматизація тестування вебзастосунку за допомогою Selenium та GitLab CI/CD

Виконав: Комонюк Іван Вікторович _____
(прізвище, ім'я, по батькові) (підпис)

група ІПЗ-41

Керівник: доцент Ярощак С. В. _____
(науковий ступінь, вчене звання, посада, прізвище, ініціали) (підпис)

Рівне – 2025

‘

Національний університет водного господарства та природокористування

(повне найменування вищого навчального закладу)

Навчально-науковий інститут кібернетики, інформаційних технологій та інженерії

Кафедра **комп'ютерних наук та прикладної математики**
Рівень вищої освіти **Бакалаврський (перший)**
Галузь знань 12 Інформаційні технології
(шифр і назва)
Спеціальність 121 Інженерія програмного забезпечення
(шифр і назва)

"Затверджую"
Завідувач кафедри
д.т н., проф. Ю.В. Турбал
2025р.

З А В Д А Н Н Я
НА КВАЛФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Комонюку Івану Вікторовичу
(прізвище, ім’я, по батькові)

1. Тема роботи *" Автоматизація тестування вебзастосунок за допомогою Selenium та GitLab CI/CD"*

керівник роботи *Ярощак Сергій Вікторович доцент кафедри комп'ютерних наук та прикладної математики.*

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання, посада)

затверджені наказом по університету від *"18" квітня 2025 року С №-463.*

2. Термін подання роботи студентом *30 травня 2025 року.*

3. Вихідні дані до роботи: *відомості про інструменти та технології, що використовуються для автоматизації тестування вебзастосунку, включаючи Selenium WebDriver, GitLab CI/CD, PyTest, Allure, а також опис функціональних вимог до тестової системи.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) *розробити структуру тестового середовища, реалізувати функціональні автотести, протестувати захищені області сайту.*

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) *Мультимедійна презентація*

‘

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ 1	*доцент Ярощак С.В.*	*12.02.25*	*12.02.25*
Розділ 2	*доцент Ярощак С.В.*	*27.03.25*	*27.03.25*
Розділ 3	*доцент Ярощак С.В.*	*30.04.25*	*30.04.25*
Розділ 4	*доцент Ярощак С.В.*	*21.05.25*	*21.05.25*

7. Дата видачі завдання *03 лютого 2025 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	*Вивчення літератури за обраною тематикою*	*13.02.25 – 20.03.25*	*виконав*
2	*Розгортання тестового середовища та налаштування середовища CI/CD*	*24.02.25 – 16.03.25*	*виконав*
3	*Створення та запуск автоматизованих функціональних тестів (Selenium)*	*04.03.25 – 29.03.25*	*виконав*
4	*Завершення тестів та обробка результатів виконання тестових сценаріїв*	*21.05.25 – 26.05.25*	*виконав*
5	*Загальні висновки до роботи*	*21.03.25 – 14.05.25*	*виконав*
6	*Підготовка звіту кваліфікаційної роботи*	*02.04.25 – 21.05.25*	*виконав*
7	*Підготовка мультимедійної презентації*	*06.06.25 – 09.06.25*	*виконав*
8	*Підготовка до виступу*	*10.06.25 – 13.06.25*	*виконав*

Здобувач: _____ Комонюк І В.
(підпис) (прізвище та ініціали)

Керівник: _____ Ярощак С.В.
(підпис) (прізвище та ініціали)

‘

ЗМІСТ

‘

‘

РЕФЕРАТ

Кваліфікаційна робота: 51 с., 15 рисунків, 13 джерел.

Мета роботи: створення системи автоматизованого тестування вебзастосунку з використанням Selenium WebDriver та GitLab CI/CD з можливістю генерації звітів і повною інтеграцією у процес розробки.

Об'єкт дослідження — вебзастосунок офіційного сайту Національного університету водного господарства та природокористування, доступний за адресою https://exam.nuwm.edu.ua/.

Предмет дослідження — методи автоматизації тестування вебзастосунків із використанням інструментів Selenium, Pytest, GitLab CI/CD

Методи дослідження — в роботі застосовано методи аналізу, синтезу, тестування, CI/CD-інтеграції, побудови пайплайнів, генерації звітів та автоматизації інтерфейсної взаємодії.

Проведено аналіз сучасних технологій автоматизованого тестування. Реалізовано систему, що складається з функціональних, безпекових та навантажувальних тестів. Вбудовано GitLab CI/CD для безперервного тестування після кожного коміту в репозиторій. Тестування проводилося з використанням Selenium у headless-режимі, із збереженням артефактів тестування.

Інтегровано генерацію звітів через Allure, що надало змогу відстежувати результати тестів у зручному графічному форматі. Звіти зберігаються у вигляді артефактів у GitLab, з можливістю подальшого аналізу результатів.

Загальна структура системи дозволяє масштабувати тести, легко додавати нові сценарії, перевіряти захищеність сайту та оцінювати продуктивність.

Ключові слова: SELENIUM, PYTEST, CI/CD, GITLAB, АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ, ВЕБЗАСТОСУНОК, ALLURE, ІНТЕРФЕЙС, БЕЗПЕКА, PERFORMANCE-ТЕСТУВАННЯ.

‘

ВСТУП

У сучасному цифровому світі якість програмного забезпечення набуває надзвичайної важливості для ефективного функціонування інформаційних систем. Стрімкий прогрес цифрових технологій змушує користувачів очікувати від програмного забезпечення стабільності, швидкості та безпомилковості. Це спричинює збільшення потреби у системному тестуванні програмних продуктів. Особливу увагу в цьому контексті заслуговує автоматизація тестування, що суттєво прискорює процес перевірки функціональності, мінімізує ризики людських помилок та гарантує стабільність розробки на всіх етапах життєвого циклу програмного забезпечення.

Автоматизоване тестування є критично важливим для вебзастосунків, які широко використовуються в різних галузях – освіті, бізнесі, медицині, державному управлінні. Вебзастосунки характеризуються постійно оновлюваною структурою, інтерактивним інтерфейсом і широким колом користувачів, що висуває високі вимоги до їх якості та надійності. Відтак виникає необхідність впровадження передових методів автоматизованого тестування веб-інтерфейсів, які можуть ефективно виявляти помилки на етапі розробки або безпосередньо після внесення змін.

Одним з найпопулярніших інструментів для реалізації автоматизованого тестування вебзастосунків є Selenium. Це бібліотека з відкритим кодом, що дає змогу імітувати дії користувача у браузері та перевіряти відповідність поведінки вебсторінки очікуваним результатам. У поєднанні з GitLab CI/CD – платформою безперервної інтеграції та доставки – Selenium забезпечує повну автоматизацію процесу тестування, яке запускається автоматично при кожному оновленні коду. Такий підхід дозволяє підтримувати високий рівень якості програмного забезпечення та виявляти помилки ще до того, як зміни потраплять на продуктивне середовище.

У даній кваліфікаційній роботі досліджується процес автоматизації тестування вебзастосунку за допомогою Selenium та GitLab CI/CD. Для демонстрації застосованих методів використовується офіційний сайт

‘
Національного університету водного господарства та природокористування (НУВГП). Даний сайт має комплексну структуру, містить значну кількість сторінок, динамічні елементи, новинні блоки та навігаційні меню, що робить його вдалим прикладом для автоматизованого функціонального тестування.

Мета роботи

Розробити, реалізувати та налаштувати процес автоматизації тестування вебзастосунку за допомогою Selenium та GitLab CI/CD.

Завдання, які необхідно виконати для досягнення поставленої мети:

- Ознайомитись із сучасними підходами до автоматизованого тестування;
- Вивчити принципи роботи Selenium;
- Вивчити можливості GitLab CI/CD;
- Розробити тести для функціональності сайту НУВГП;
- Інтегрувати тести у GitLab CI/CD pipeline;
- Проаналізувати результати виконання автоматизованих тестів.

Об’єкт дослідження

Вебзастосунок – офіційний сайт Національного університету водного господарства та природокористування.

Предмет дослідження

Процес автоматизації тестування вебзастосунку за допомогою Selenium та GitLab CI/CD.

Методи дослідження

У процесі виконання роботи використовувалися методи аналізу, синтезу, моделювання, практичного програмування та тестування.

Структура роботи

Кваліфікаційна робота містить вступ, п’ять розділів, висновки, список використаних джерел та додатки. У роботі розглянуто теоретичні аспекти автоматизованого тестування, проведено аналіз предметної області, розроблено та реалізовано систему автоматизованого тестування вебзастосунку з використанням Selenium і GitLab CI/CD.

‘

РОЗДІЛ 1
ОГЛЯД ЛІТЕРАТУРИ ТА ТЕХНОЛОГІЙ

1.1 Сучасні підходи до автоматизованого тестування

Автоматизоване тестування – ключовий елемент гарантії якості ПЗ, особливо у сучасному динамічному світі, де швидкість розробки, часті оновлення та потреба у безперервній інтеграції визначають успіх. Суть автоматизації полягає у створенні програмних скриптів, які імітують дії кінцевого користувача, взаємодіючи з елементами інтерфейсу, що дає можливість автоматично перевірити, чи система відповідає очікуванням. Такі тести виконуються багаторазово, без втручання людини, що зменшує ризик людської помилки та покращує загальну ефективність тестування.

Основні переваги автоматизованого тестування:

- Зменшення часу на виконання тестових сценаріїв, порівняно з ручним тестуванням;
- Підвищена точність тестів та зменшення шансів пропустити дефекти;
- Можливість багаторазового використання створених тестів;
- Ефективна реалізація регресійного тестування;
- Інтеграція у безперервний цикл розробки, завдяки CI/CD.

Існують різні типи автоматизованого тестування: юніт-тестування, функціональне, інтеграційне, регресійне, навантажувальне та інші. Вебзастосунки, які часто мають складний інтерфейс з великою кількістю інтерактивних компонентів, зазвичай потребують саме функціонального тестування, що перевіряє відповідність дій користувача бажаним результатам.

1.2 Огляд технології Selenium

Selenium – один з найпопулярніших фреймворків для автоматизованого тестування веб-інтерфейсів. Проєкт відкритий та підтримується активною спільнотою. Його ключова перевага – підтримка різних мов програмування

(Python, Java, C#, JavaScript тощо), що робить Selenium універсальним інструментом у багатьох середовищах розробки.

Основні компоненти Selenium:

- Selenium WebDriver – API для взаємодії з веб-браузерами (Google Chrome, Mozilla Firefox, Safari та інші);
- Selenium IDE – браузерне розширення для запису і відтворення дій користувача (підходить для початкового рівня);
- Selenium Grid – дозволяє запускати тести у різних браузерах і на різних платформах паралельно.

Selenium WebDriver дозволяє писати тести, які управляють браузером на рівні коду, імітуючи натискання кнопок, введення даних у поля, перемикання між вкладками, валідацію повідомлень про помилки тощо. Це забезпечує точну перевірку функціональності вебзастосунків.

Переваги Selenium:

- Підтримка великої кількості браузерів і платформ;
- Сумісність з CI/CD-системами та тестовими фреймворками (наприклад, pytest, unittest);
- Можливість масштабування тестів для великих систем;
- Активна спільнота та доступна документація.

До недоліків Selenium можна віднести складність у тестуванні динамічного контенту (наприклад, AJAX-запитів), потребу у додаткових бібліотеках для створення звітів, а також труднощі у підтримці великої кількості тестів при зміні інтерфейсу.

1.3 Огляд системи GitLab CI/CD

GitLab – потужна DevOps-платформа з відкритим вихідним кодом, яка допомагає командам керувати всім життєвим циклом розробки: від планування до розгортання. Важливою складовою GitLab є GitLab CI/CD – система безперервної інтеграції та доставки.

‘

CI/CD (Continuous Integration / Continuous Delivery або Deployment) – це практика постійної інтеграції змін в основну гілку проєкту з автоматичним запуском перевірок (тестів, збірки, аналізу коду). GitLab CI/CD реалізує цю концепцію за допомогою pipeline, які описуються у файлі .gitlab-ci.yml. Цей файл містить інструкції про те, які дії треба виконати (тести, збірка, деплой тощо) та на якому етапі.

Компоненти GitLab CI/CD:

- Runner – програма, що виконує завдання, описані у pipeline;
- Stages – етапи CI/CD-процесу (наприклад, build, test, deploy);
- Jobs – окремі завдання, що виконуються на певному етапі.

GitLab CI/CD забезпечує:

- Автоматичний запуск тестів при кожному коміті чи злитті гілок;
- Швидке виявлення помилок ще до потрапляння коду у продуктивне середовище;
- Масштабування за допомогою декількох runner'ів;
- Централізоване зберігання логів, артефактів та результатів тестів.

Інтеграція з Selenium дозволяє запускати браузерні тести як частину CI/CD, що є дуже зручно для постійної перевірки якості веб-інтерфейсу після кожного оновлення коду.

1.4 Приклади застосування Selenium та GitLab CI/CD у проєктах

Багато компаній різного масштабу активно впроваджують автоматизоване тестування у свої CI/CD-процеси. Наприклад:

- Spotify використовує Selenium для перевірки змін у своєму вебінтерфейсі після кожного коміту в основну гілку, що забезпечує швидке інформування розробників;
- GitLab Inc. тестує свою платформу GitLab за допомогою GitLab CI/CD та Selenium;

‘

- Стартапи використовують Selenium як доступне та безкоштовне рішення, у поєднанні з GitLab для прискорення випуску продуктів на ринок.

Отже, Selenium у поєднанні з GitLab CI/CD – це ефективне і масштабоване рішення, що дозволяє реалізувати високоякісні автоматизовані перевірки для проєктів різного рівня складності.

У наступному розділі буде детально розглянуто вебзастосунок, що підлягає тестуванню, визначено його функціональні особливості та сформульовано технічні вимоги до впровадження системи автоматизації тестування.

‘

РОЗДІЛ 2
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

2.1 Характеристика об’єкта тестування

Об’єктом тестування в цій кваліфікаційній роботі визначено вебсайт Національного університету водного господарства та природокористування (НУВГП), розміщений за адресою https://exam.nuwm.edu.ua/. Це багатофункціональний веб-ресурс, який виступає як інформаційний, навігаційний та сервісний портал університету. Основне його завдання – надання актуальної інформації про діяльність закладу вищої освіти, забезпечення взаємодії із зацікавленими сторонами та підтримка внутрішніх освітніх процесів.

Функціонал сайту охоплює широкий спектр задач:

- Публікація новин, офіційних оголошень, розпоряджень адміністрації;
- Надання студентам доступу до розкладів занять, результатів сесій, академічних довідок;
- Інтерактивна взаємодія з абітурієнтами: прийом заяв, надання інформації про спеціальності, контакти приймальної комісії;
- Сторінки факультетів, кафедр, адміністративних підрозділів із детальним описом їхньої діяльності;
- Мультимовна підтримка, адаптивний дизайн для комфортного перегляду на мобільних пристроях;
- Реалізація зворотного зв’язку, створення звернень та запитів.

Цей вебзастосунок є важливим елементом цифрової інфраструктури університету, від якого залежить ефективність комунікації із зовнішніми та внутрішніми користувачами. Його стабільна робота, швидкість реакції та функціональна цілісність вимагають систематичного контролю, який найкраще реалізувати через автоматизоване тестування.

2.2 Особливості структури сайту

‘

Архітектура сайту побудована як набір тематичних модулів, організованих у логічні блоки. Основні розділи розташовані у верхньому навігаційному меню та мають вкладену структуру, що полегшує навігацію серед великої кількості контенту.

Ключові елементи структури сайту:

- Головна сторінка – з інтерактивним банером, стрічкою новин, інформаційними блоками та контактами;
- Навігація – реалізована у вигляді багаторівневого меню, яке динамічно завантажується;
- Система пошуку – дає змогу здійснювати пошук на сайті за ключовими словами;
- Сторінки структурних підрозділів – з власним дизайном, наповненням та підрозділами всередині;
- Форми – для контактів, реєстрації, зворотного зв’язку та опитувань;
- Інтегровані ресурси – підключення до зовнішніх систем (електронний журнал, кабінет студента тощо).

Крім того, сайт підтримує адаптивну верстку, що забезпечує його правильне відображення на екранах з різною роздільною здатністю. Динамічні елементи, розроблені за допомогою JavaScript, надають інтерактивності, але ускладнюють процес тестування, оскільки потребують очікування асинхронного завантаження даних.

2.3 Технічні вимоги до системи автоматизованого тестування

На основі аналізу функціональних та структурних особливостей сайту визначено такі технічні вимоги до майбутньої системи автоматизованого тестування:

1. Кросплатформеність та гнучкість – можливість запуску тестів у середовищах Windows та Linux, підтримка популярних браузерів (Chrome, Firefox, Edge).

2. Параметризованість тестів – реалізація тестових сценаріїв з можливістю легкого налаштування для різних користувацьких сценаріїв (гостьовий вхід, авторизований користувач тощо).
3. Масштабованість – структура тестів має дозволяти додавання нових сценаріїв без значних змін існуючих модулів.
4. Журналювання та звітність – система повинна формувати структуровані звіти (HTML, XML або PDF) з результатами виконання, вказуючи помилки, час виконання, скріншоти у випадку збоїв.
5. Інтеграція з CI/CD – автоматичний запуск тестів через GitLab Runner після кожної зміни в репозиторії, можливість виконання в ізольованому середовищі (Docker або віртуальна машина).
6. Сумісність з інструментами моніторингу – потенційна інтеграція з системами моніторингу якості, такими як Allure або SonarQube.

Всі ці технічні вимоги будуть враховані при розробці тестової архітектури на наступних етапах роботи.

2.4 Постановка задачі

На основі проведеного аналізу сформульовано основну задачу розробки: створити систему автоматизованого тестування функціональності вебзастосунку НУВГП, яка дозволить:

- своєчасно виявляти помилки у роботі ключових елементів сайту (меню, форми, інтерактивні блоки);
- забезпечити швидку перевірку працездатності після внесення змін у код або структуру сторінок;
- зменшити залежність від ручного тестування та підвищити якість релізів;
- скоротити час зворотного зв’язку для розробників та адміністраторів сайту;
- генерувати зрозумілі та наочні звіти про результати виконання тестів;

‘

- впровадити систему безперервного тестування з використанням GitLab CI/CD.

Система має бути спроектована з урахуванням можливості подальшого масштабування, мати відкриту архітектуру для розширення тестових сценаріїв та бути зручною в підтримці й оновленні.

У наступному розділі безпосередньо розглядатиметься процес побудови автоматизованої системи тестування: вибір інструментів, проєктування структури тестів, написання коду та підключення до CI/CD-середовища GitLab.

‘

РОЗДІЛ 3
РОЗРОБКА СИСТЕМИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ

3.1 Підготовка середовища розробки

Для втілення системи автоматизованого тестування вебзастосунку сайту НУВГП було обрано мову програмування Python, спільно з бібліотекою Selenium WebDriver. Такий вибір ґрунтується на легкості синтаксису Python, значній підтримці Selenium у фаховій спільноті, а також можливості ефективного розширення системи у майбутньому.

Перед початком роботи необхідно підготувати відповідне середовище розробки, яке містить:

- Python (версія 3.10 або вище) — основна мова програмування для написання тестів;
- pip — менеджер пакетів Python, що дає змогу встановлювати необхідні бібліотеки;
- Selenium WebDriver — бібліотека, яка надає інтерфейси для управління браузерами;
- Драйвери браузерів — ChromeDriver або GeckoDriver (для Firefox), що забезпечують взаємодію з конкретними браузерами;
- IDE (інтегроване середовище розробки) — Visual Studio Code або PyCharm для написання, налагодження та запуску тестів;
- Git — система контролю версій, яка дає змогу інтегрувати тестовий код з GitLab;
- GitLab Runner — віртуальний агент, що виконує сценарії CI/CD для автоматичного запуску тестів після комітів.

Встановлення основних компонентів

Для встановлення бібліотеки Selenium необхідно скористатись pip:

```
pip install selenium
```

Для завантаження драйвера браузера Chrome потрібно:

- перейти на офіційний сайт https://chromedriver.chromium.org;

‘

- обрати версію драйвера відповідно до встановленої версії браузера;
- розпакувати архів та помістити виконуваний файл до директорії, яка входить до змінної середовища PATH, або чітко вказати шлях у коді.

В процесі втілення також доцільно застосовувати віртуальне середовище (virtualenv або venv), що дає змогу ізолювати залежності тестового проєкту від глобальної системи Python.

3.2 Структура проєкту автоматизованого тестування

Для забезпечення зручності масштабування, підтримки та повторного використання коду, проєкт було структуровано за принципами модульності з використанням патерну Page Object Model (POM). Цей патерн дає змогу розмежувати логіку взаємодії зі сторінками сайту від логіки самих тестів.

Структура проєкту має вигляд:

```
auto-tests-nuwm/
├── tests/              # Основні тестові сценарії
│   ├── test_homepage.py
│   ├── test_navigation.py
│   ├── test_forms.py
│   └── ...
├── pages/              # Об’єкти сторінок (Page Object Model)
│   ├── base_page.py
│   ├── homepage.py
│   └── contact_page.py
├── drivers/            # Драйвери браузерів
│   └── chromedriver.exe
├── config/             # Конфігураційні файли
│   └── settings.py
├── .gitlab-ci.yml      # Опис CI/CD пайплайну
└── README.md           # Документація до проєкту
```

‘

Такий підхід дає змогу централізовано змінювати логіку взаємодії зі сторінками без потреби редагувати самі тести, що є важливим з огляду на масштабованість та тривалу підтримку системи.

3.3 Приклади реалізації автоматизованих тестів

Розглянемо приклади реалізації тестових сценаріїв. Один із базових тестів — перевірка наявності ключового слова у заголовку головної сторінки:

Код:

```
from selenium import webdriver
from selenium.webdriver.common.by import By

def test_homepage_title():
    driver = webdriver.Chrome()
    driver.get("https://nuwm.edu.ua")
    assert "НУВГП" in driver.title
    driver.quit()
```

Інший тест — перевірка коректності меню навігації:

Код:

```
from selenium import webdriver
from selenium.webdriver.common.by import By

def test_navigation_menu():
    driver = webdriver.Chrome()
    driver.get("https://nuwm.edu.ua")
    nav = driver.find_element(By.CLASS_NAME, "menu-main")
    links = nav.find_elements(By.TAG_NAME, "a")
    assert len(links) >= 5  # має бути щонайменше 5 пунктів меню
    driver.quit()
```

Більш складні тести можуть включати:

- перевірку заповнення та валідації форм;

‘

- перевірку реакції на некоректні введення;
- тестування елементів AJAX (наприклад, динамічні завантаження);
- виконання авторизації та перевірка доступу до обмежених ресурсів.

Кожен тестовий сценарій має бути простим, ізольованим та таким, що може запускатись незалежно від інших.

3.4 Інтеграція з GitLab CI/CD

Для автоматизації процесу тестування інтегрується система CI/CD, що запускає тести при кожному оновленні коду. GitLab CI/CD забезпечує виконання тестів у заздалегідь визначеному середовищі та надає зворотний зв’язок розробникам.

Приклад .gitlab-ci.yml:

Код:

```
stages:
  - test

test_site:
  stage: test
  image: python:3.10
  before_script:
    - pip install selenium
     wget
https://chromedriver.storage.googleapis.com/114.0.5735.90/chromedriver_linux64.zip
      unzip chromedriver_linux64.zip
      mv chromedriver /usr/bin/chromedriver
      chmod +x /usr/bin/chromedriver
  script:
      python3 tests/test_homepage.py
```

‘

Цей конфігураційний файл описує одну стадію — test, яка виконується у Docker-контейнері на основі образу python:3.10. Перед запуском тестів система:

- встановлює бібліотеку selenium;
- завантажує ChromeDriver;
- виконує основний тестовий скрипт.

Після виконання pipeline користувач отримує сповіщення про успішність або помилки в тестах. У випадку падіння тесту GitLab автоматично зберігає лог-файли, що дає змогу оперативно ідентифікувати проблему.

Інтеграція з GitLab CI/CD дає змогу вбудувати автоматизовані тести в процес розробки та підтримки вебзастосунку, забезпечуючи стабільність системи навіть при частих оновленнях коду.

У наступному розділі будуть представлені результати виконання тестів, візуалізація звітів, інтерпретація отриманих даних та загальний аналіз ефективності впровадженого рішення.

3.5 Практичне втілення автоматизованого тестування сайту NUWM Exam через Selenium та GitLab CI/CD

У цьому підрозділі я детально покажу, як організувати автоматичне тестування вебзастосунку на прикладі сайту https://exam.nuwm.edu.ua/ за допомогою Selenium, Pytest та GitLab CI/CD. Опис містить повний цикл — від налаштування проєкту до отримання результатів тестування в пайплайні.

3.5.1 Створення проєкту та підключення до GitLab

Перший крок — створити новий репозиторій на платформі GitLab, куди будуть завантажуватись усі файли для автоматизованого тестування. Для цього я:

- Зайшов на сайт gitlab.com
- Створив новий порожній проєкт у своїй групі (команда “Створити порожній проєкт”)
- Вказав зручну назву, наприклад, “komoniuk-project”.

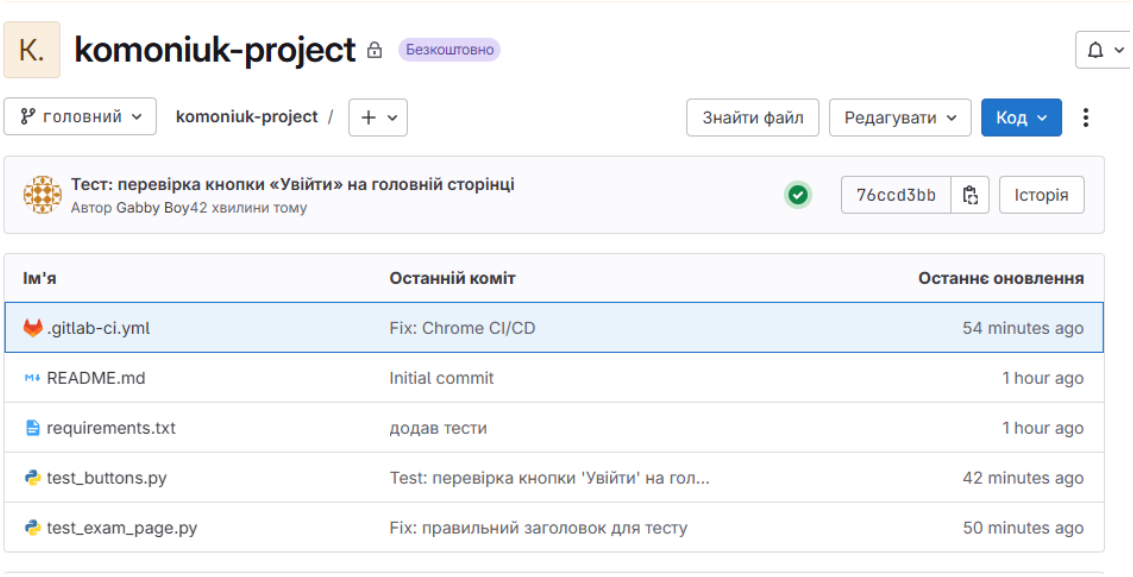

Рис. 3.5.1 Репозиторій GitLab

Після того, як проєкт було створено, GitLab пропонує інструкцію щодо під'єднання репозиторію за допомогою git. Я застосував команду:

git remote add origin https://gitlab.com/komoniuk-group/komoniuk-project.git

Це дозволило встановити зв'язок між локальною директорією та проєктом у GitLab.

3.5.2 Створення структури проєкту для тестування

У головній папці проєкту було створено основні файли:

- requirements.txt – список потрібних Python бібліотек для автоматичного тестування (Selenium та Pytest)
- .gitlab-ci.yml – конфігураційний файл CI/CD, який визначає, як GitLab буде запускати автоматичні тести після внесення змін у проєкт
- test_exam_page.py – файл із тестовим кодом на Python для Selenium/Pytest
- (Для проведення додаткових перевірок було створено ще один файл з тестами — наприклад, test_buttons.py)

‘

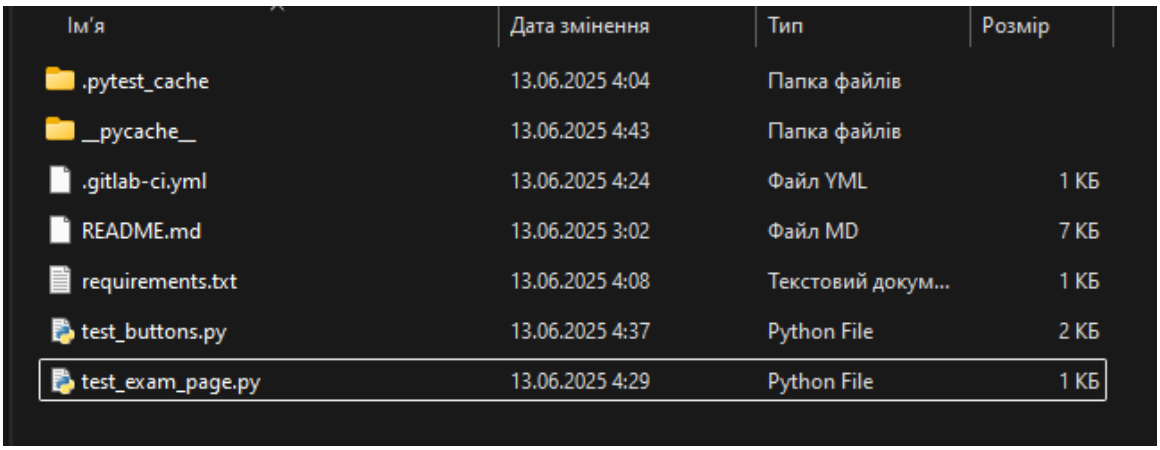

Рис. 3.5.2 Структура файлової системи проєкту з тестами

На цьому зображенні показано файлову структуру локального проєкту, призначеного для автоматизації тестування вебзастосунку. У кореневій директорії розташовані наступні ключові файли:

- .gitlab-ci.yml – конфігураційний файл для CI/CD GitLab, що визначає етапи та середовище для автоматичного запуску тестів;
- README.md – файл з основною інформацією про проєкт;
- requirements.txt – перелік залежностей Python, необхідних для виконання тестів (зокрема, selenium та pytest);
- test_exam_page.py, test_buttons.py – тестові скрипти, що містять автоматизовані сценарії для перевірки функціональності сайту.

Така структура забезпечує зручну організацію файлів для роботи з автотестами та CI/CD.

3.5.3 Вміст файлів проєкту

requirements.txt

selenium

pytest

Пояснення:

- selenium — бібліотека для автоматизованого управління браузером
- pytest — фреймворк для легкого запуску й організації тестів у Python

.gitlab-ci.yml

‘

Цей файл забезпечує автоматичний запуск тестів після кожного push у репозиторій.

Код:

```
image: python:3.11

stages:
  - test

before_script:
    pip install -r requirements.txt
    apt-get update
    apt-get install -y wget unzip chromium-driver chromium
    ln -s /usr/bin/chromedriver /usr/local/bin/chromedriver || true
    ln -s /usr/bin/chromium /usr/local/bin/chrome || true

test:
  stage: test
  script:
      pytest --maxfail=1 --disable-warnings -q
```

Пояснення:

- Використовується docker-образ Python 3.11.
- Встановлюються необхідні бібліотеки, браузер Chromium та його драйвер.
- Запускається команда для автоматичного тестування.

test_exam_page.py

Код:

```
from selenium import webdriver

def test_title():
    options = webdriver.ChromeOptions()
```

‘

```
    options.add_argument('--headless')
    options.add_argument('--no-sandbox')
    options.add_argument('--disable-dev-shm-usage')
    driver = webdriver.Chrome(options=options)
    driver.get("https://exam.nuwm.edu.ua/")
    assert "На головну | НУВГП" == driver.title
    driver.quit()
```

Пояснення:

- Браузер Chrome запускається у безголовому режимі (без графічного інтерфейсу).
- Відкривається головна сторінка сайту.
- Перевіряється відповідність заголовка сторінки очікуваному значенню (“На головну | НУВГП”).
- При успішному проходженні тесту браузер закривається.

test_buttons.py (додатковий тест)

Код:

```
from selenium import webdriver
from selenium.webdriver.common.by import By

def test_login_button():
    options = webdriver.ChromeOptions()
    options.add_argument('--headless')
    options.add_argument('--no-sandbox')
    options.add_argument('--disable-dev-shm-usage')
    driver = webdriver.Chrome(options=options)
    driver.get("https://exam.nuwm.edu.ua/")
    login_button = driver.find_element(By.XPATH, "//a[text()='Увійти']")
    assert login_button.is_displayed()
    driver.quit()
```

Пояснення:

‘

- Відкривається головна сторінка.
- Кнопка "Увійти" шукається за її текстовим вмістом.
- Перевіряється наявність кнопки на сторінці.

3.5.4 Завантаження файлів у репозиторій та запуск тестів

Щоб файли з'явилися в репозиторії та автоматично запустилися тести, я виконав команди в терміналі:

git add .

git commit -m "Додаю автоматизовані тести"

git push -u origin main

Після push-у GitLab автоматично запускає пайплайн (Pipeline), який відображається у вкладці “CI/CD → Pipelines”.

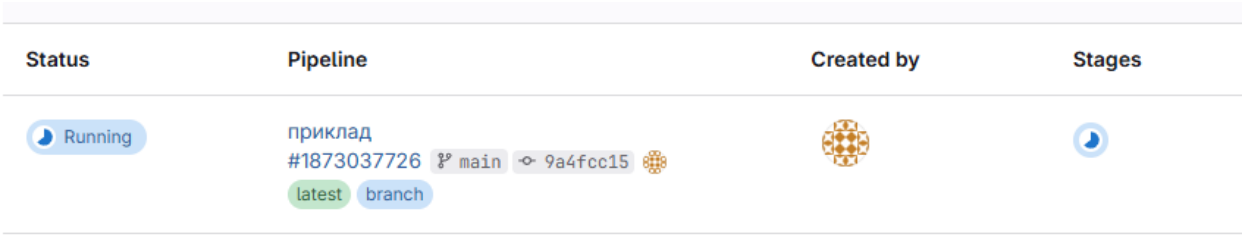

Рис. 3.5.4 Момент запуску пайплайну в GitLab

3.5.5 Автоматизоване функціональне тестування сайту

Після створення проєкту, налаштування середовища та написання потрібних тестових скриптів, була здійснена перевірка функціоналу реального сайту https://exam.nuwm.edu.ua/ за допомогою Selenium та Pytest.

Мета тестування — переконатися, що основна сторінка сайту коректно відкривається, а ключові елементи інтерфейсу (наприклад, кнопки входу) функціонують відповідно до вимог.

Етапи тестування

1. Перевірка відкриття головної сторінки

Було автоматично відкрито сайт у браузері через Selenium. Тест перевірив, що в заголовку сторінки міститься напис "Єдина система тестування НУВГП", що гарантує завантаження правильної сторінки.

‘

Код:

```
from selenium import webdriver


def test_title():
    driver = webdriver.Chrome()
    driver.get("https://exam.nuwm.edu.ua/")
    assert "Єдина система тестування НУВГП" in driver.title
    driver.quit()
```

2. Перевірка наявності та працездатності кнопки "Увійти"

Тест автоматично знаходить на сторінці кнопку "Увійти" та перевіряє, що вона відображається та є доступною для взаємодії.

Код:

```
from selenium import webdriver
from selenium.webdriver.common.by import By


def test_login_button():
    driver = webdriver.Chrome()
    driver.get("https://exam.nuwm.edu.ua/")
    button = driver.find_element(By.XPATH, "//button[contains(text(),'Увійти')]")
    assert button.is_displayed()
    driver.quit()
```

Автоматичне виконання тестів

Всі тести були додані у GitLab-репозиторій проєкту та автоматично виконувалися при кожному новому коміті завдяки інтеграції з GitLab CI/CD.

Це дозволяє своєчасно виявляти помилки або некоректну роботу сайту відразу після змін у коді чи налаштуваннях, а також фіксувати прогрес виконання тестів через інтерфейс пайплайнів.

‘

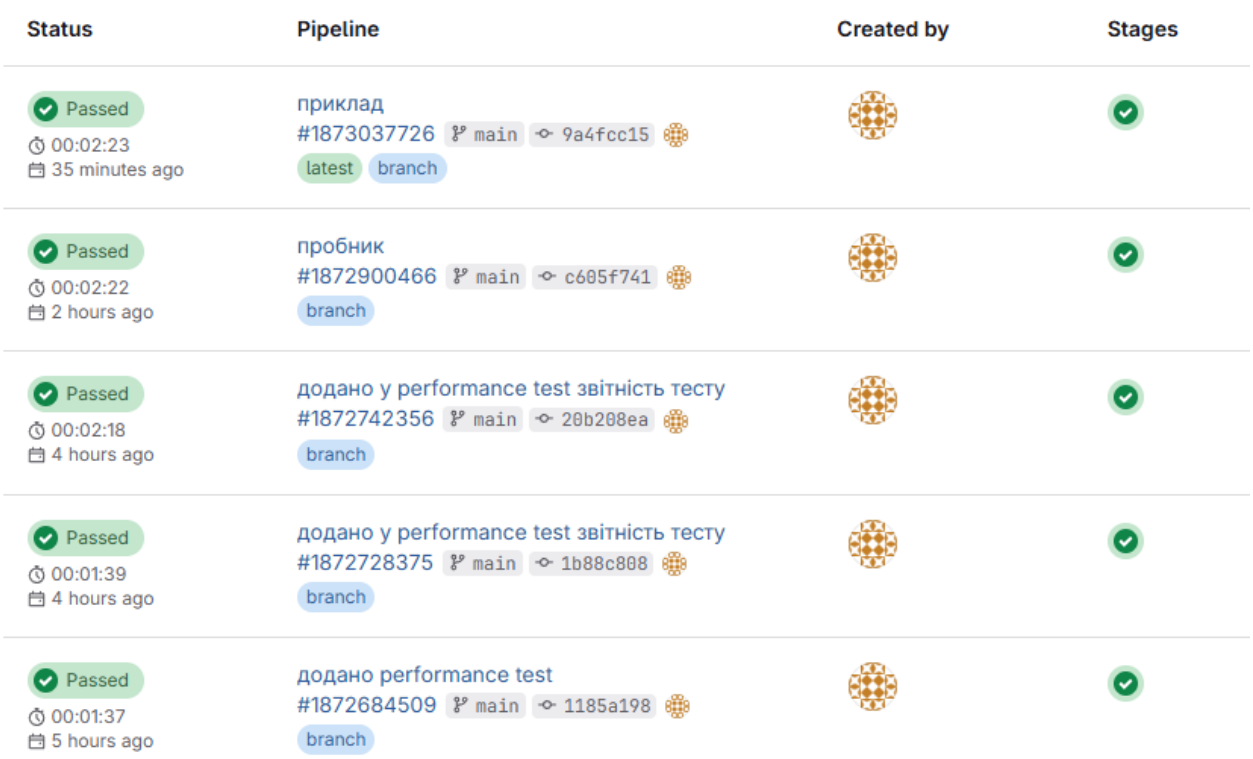

Рис. 1 3.5.5 Успішно виконані пайплайни

На всіх етапах тестування було отримано успішне проходження тестів (Job succeeded).

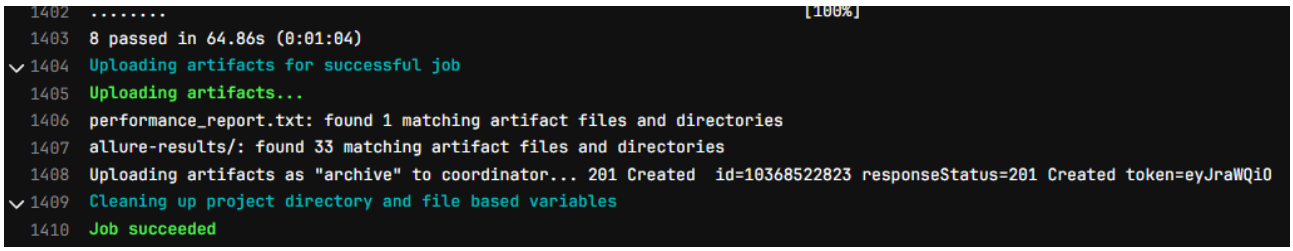

Рис. 2 3.5.5. Успішне завершення всіх етапів тестування в середовищі GitLab CI/CD.

У разі виникнення помилок CI/CD надає детальний лог, що значно спрощує пошук та виправлення проблем.

3.5.6 Аналіз результатів тестування

Щоб переглянути результати тестів, необхідно:

- Зайти в GitLab → розділ "Трубопроводи" (Pipelines)

- Відкрити останній запуск та обрати завдання “test”
- У логах відображаються всі етапи виконання та статус тестів (успішно або з помилкою)

Якщо у коді виникне помилка або тест завершиться невдачею, GitLab негайно відобразить це (наприклад, “Job failed” з описом проблеми).

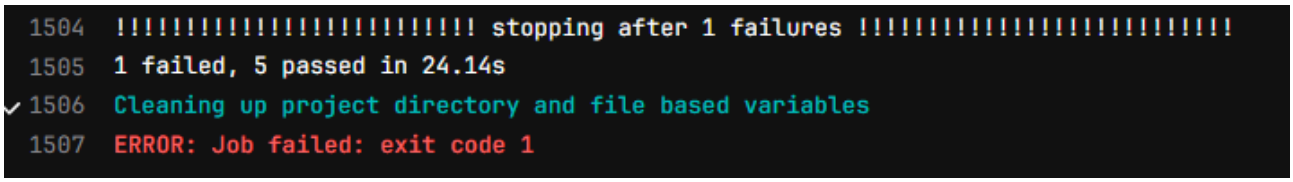

Рис. 3.5.6. Приклад невдалого виконання одного з тестів у CI/CD-процесі GitLab

3.5.7 Роз'яснення важливості автоматизації

Автоматизація тестування за допомогою GitLab CI/CD забезпечує:

- Миттєвий зворотний зв'язок після кожної зміни коду (функціонує чи ні)
- Перевірку сайту в "чистому" оточенні, як його бачить кінцевий користувач
- Спрощення процесу виявлення помилок та поліпшення якості розробки

Підсумок розділу

Такий підхід дозволяє завжди бути впевненим, що зміни у коді не призведуть до збою основних функцій сайту. Всі члени команди мають змогу оперативно отримувати інформацію про помилки та негайно їх виправляти.

‘

РОЗДІЛ 4
МОЖЛИВОСТІ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ САЙТУ (НА ПРИКЛАДІ EXAM.NUWM.EDU.UA)

Автоматизоване тестування вебзастосунку дає змогу не тільки перевіряти виконання окремих сценаріїв, а й забезпечує комплексний контроль якості програмного продукту. На прикладі сайту exam.nuwm.edu.ua розглянемо типові категорії тестів, які можливо реалізувати за допомогою Selenium та Pytest. У цьому розділі структуровано можливості тестування із місцями для практичних прикладів коду та відповідних ілюстрацій.

4.1 Функціональні тести (без авторизації; специфіка тестування входу через Google)

Функціональне тестування слугує для перевірки базових функцій сайту, як-от для пересічного користувача (студента) перед входженням у систему. Для ресурсу exam.nuwm.edu.ua це вкрай важливо, зважаючи на те, що переважна більшість інформації доступна лише тим студентам, котрі вже мають корпоративну пошту та можуть здійснити вхід через Google-акаунт.

Чому тести проходять без авторизації?

Вхід можливий лише для студентів, що володіють корпоративною поштою NUWM.EDU.UA.

Реєстрація на сайті відсутня – всі акаунти генеруються університетом для студентів.

Авторизація відбувається виключно через Google OAuth (натискання кнопки “Увійти з Google”), а не стандартні поля “логін/пароль”.

Технічні обмеження автоматизації тестів із Selenium:

Google блокує автоматичні сценарії у своєму вікні входу задля безпеки облікових записів (Selenium не може обійти цей захист автоматично).

Отримання та підстановка access token вручну – складне, небезпечне та суперечить політиці безпеки.

‘

Відтак, у кваліфікаційній роботі тести реалізовано лише для відкритої частини сайту, що доступна без авторизації.

Що конкретно перевіряють функційні тести?

Наявність та правильність головного меню (“На головну”, “Колонка новин”, “Інститути”, “Перевірка на плагіат”, “Вибіркові дисципліни”).

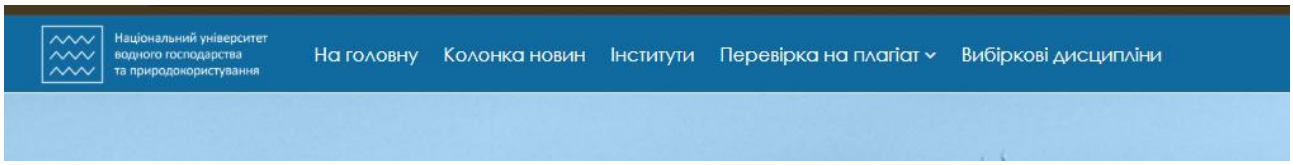

Рис. 1 4.1. Інтерфейс головної сторінки сайту НУВГП

Відображення важливих інформаційних блоків (приміром, перелік інститутів).

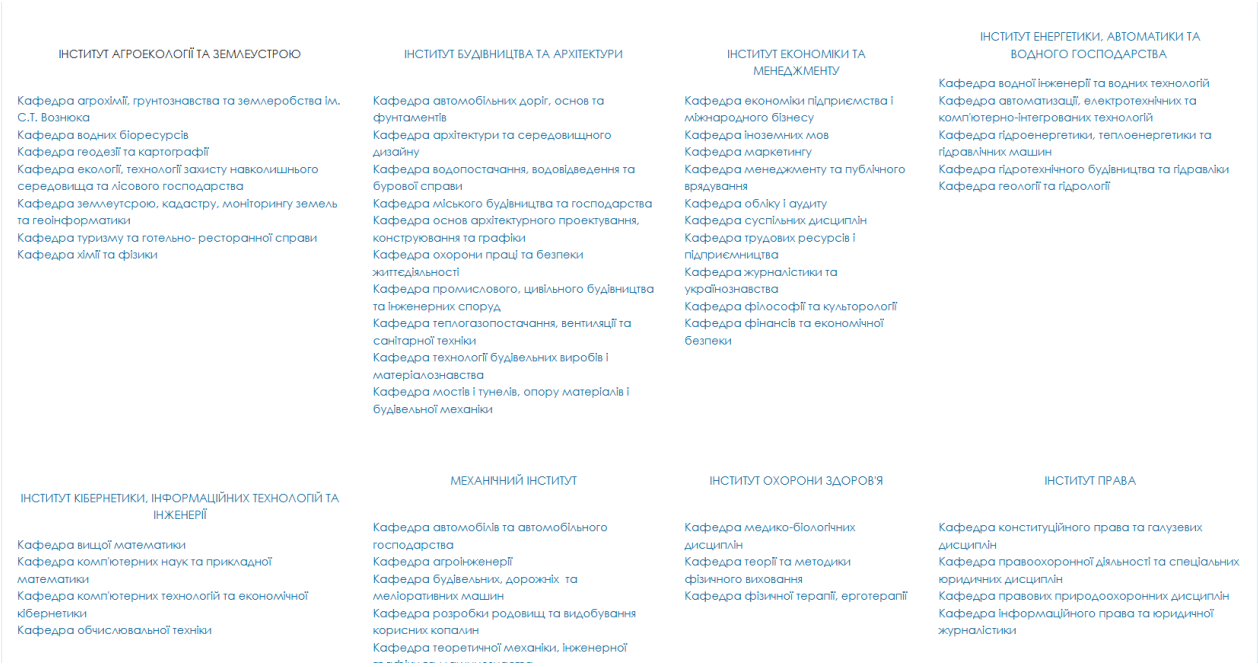

Рис. 2 4.1. Перелік інститутів університету

Доступність основної інформації, що є важливою для студентів: новини, розклад, контактні дані.

Перевірка роботи основних посилань та переходів на сторінках.

Чому це важливо саме для студентів?

Лише студенти з активною корпоративною поштою мають змогу увійти у закриту частину сайту.

‘

Відкрита частина – це перше, з чим зустрічається студент, коли знайомиться з сайтом або ж у випадку проблем зі входом чи втрати паролю.

Відсутність помилок та доступність інформації ще до авторизації – гарантія того, що студент не натрапить на критичні перепони на старті.

Приклад автоматизованого тесту (без входу):

Код:

```
from selenium import webdriver
import time

def test_public_main_menu_items():
    options = webdriver.ChromeOptions()
    options.add_argument('--no-sandbox')
    options.add_argument('--headless')
    options.add_argument('--disable-dev-shm-usage')
    options.add_argument('--disable-gpu')
    driver = webdriver.Chrome(options=options)

    try:
        driver.get("https://exam.nuwm.edu.ua/")
        time.sleep(2)
        menu_texts = ["На головну", "Колонка новин", "Інститути", "Перевірка на плагіат", "Вибіркові дисципліни"]
        for text in menu_texts:
            assert text in driver.page_source
    finally:
        driver.quit()
```

‘

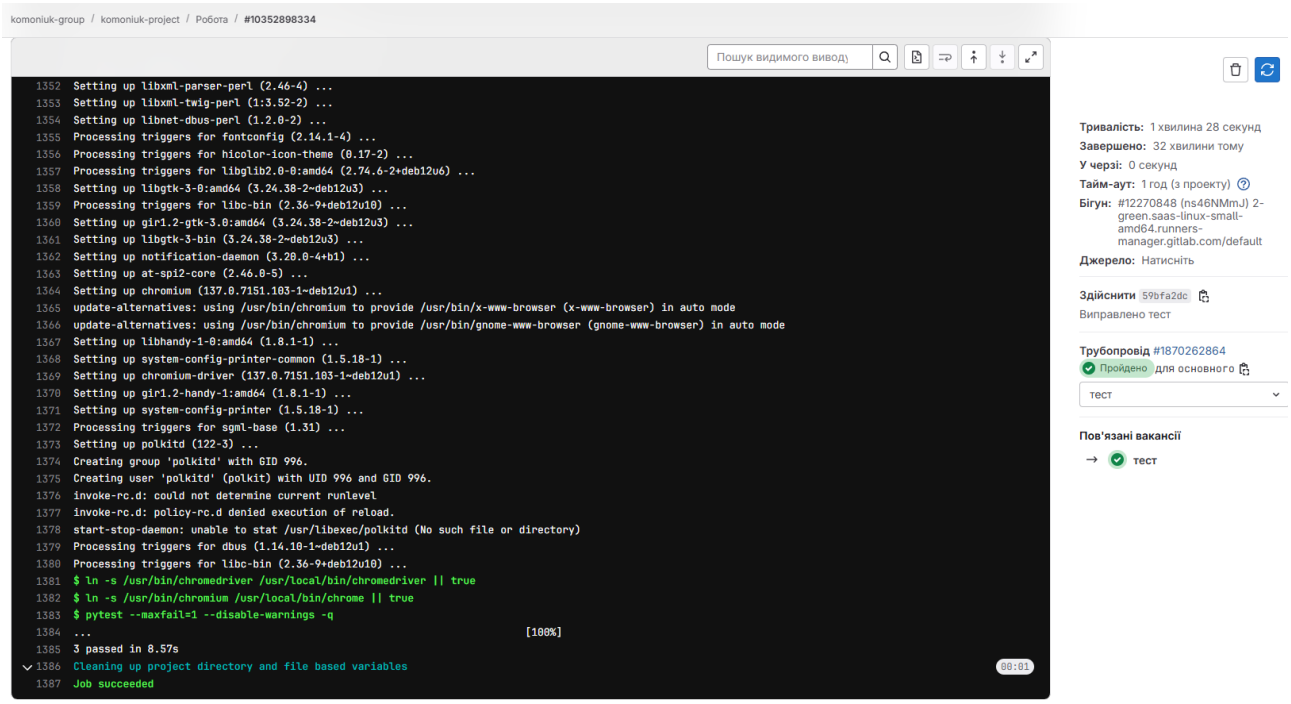

Рис. 3 4.1. Статус виконання пайплайну

4.2 Тестування інтерфейсу користувача

Тестування інтерфейсу користувача (UI testing) — це перевірка зовнішнього вигляду сайту, правильності відображення елементів, відповідності їхнього розташування та змісту очікуванням користувача. Для сучасного вебсайту університету цей тип тестування критично важливий, адже студенту важливо швидко знаходити потрібну інформацію серед багатьох розділів, кнопок і меню.

Що саме перевіряє UI-тестування на exam.nuwm.edu.ua:

- Коректність структури сторінки: чи відображаються всі колонки з інститутами, чи правильно показано заголовки.
- Наявність і читабельність посилань: чи всі інститути мають власні лінки, чи можна за ними перейти.
- Відсутність зламаних або прихованих елементів: всі важливі блоки мають бути помітні та доступні для натискання.
- Адаптивність: елементи не повинні накладатися чи “зникати” при зміні розміру вікна (опціонально для базового тесту).

‘

Чому це важливо для студентів?

Від коректного інтерфейсу напряму залежить досвід взаємодії із сайтом.

Якщо кнопки не відображаються, посилання не працюють або сторінка виглядає “зламаною”, студент може не знайти важливу інформацію, що вплине на його навчальний процес.

Приклад автоматизованого UI-тесту для блоку інститутів:

Код:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
import time

def test_institutes_ui_elements():
    options = webdriver.ChromeOptions()
    options.add_argument('--no-sandbox')
    options.add_argument('--headless')
from selenium import webdriver
from selenium.webdriver.common.by import By
import time
import re

def test_institutes_links():
    options = webdriver.ChromeOptions()
    options.add_argument('--no-sandbox')
    options.add_argument('--headless')
    options.add_argument('--disable-dev-shm-usage')
    options.add_argument('--disable-gpu')
    driver = webdriver.Chrome(options=options)

    try:
        driver.get("https://exam.nuwm.edu.ua/")
```

```
        time.sleep(2)

        institutes_patterns = [
            r"АГРОЕКОЛОГІЇ.*?ЗЕМЛЕУСТРОЮ",
            r"БУДІВНИЦТВА.*?АРХІТЕКТУРИ",
            r"ЕКОНОМІКИ.*?МЕНЕДЖМЕНТУ",
            r"ЕНЕРГЕТИКИ.*?АВТОМАТИКИ.*?ВОДНОГО.*?ГОСПОДАРСТВА",
            r"КІБЕРНЕТИКИ.*?ІНФОРМАЦІЙНИХ.*?ІНЖЕНЕРІЇ",
            r"МЕХАНІЧНИЙ ІНСТИТУТ",
            r"ОХОРОНИ ЗДОРОВ'Я",
            r"ПРАВА"
        ]
        all_links = driver.find_elements(By.TAG_NAME, "a")
        links_text = [link.text.replace('\n', ' ').replace('\xa0', ' ').strip() for link in all_links]

        for pattern in institutes_patterns:
            found = False
            for text in links_text:
                if re.search(pattern, text, re.IGNORECASE):
                    found = True
                    break
            assert found, f"Не знайдено інститут за патерном: {pattern}"

    finally:
        driver.quit()
```

‘

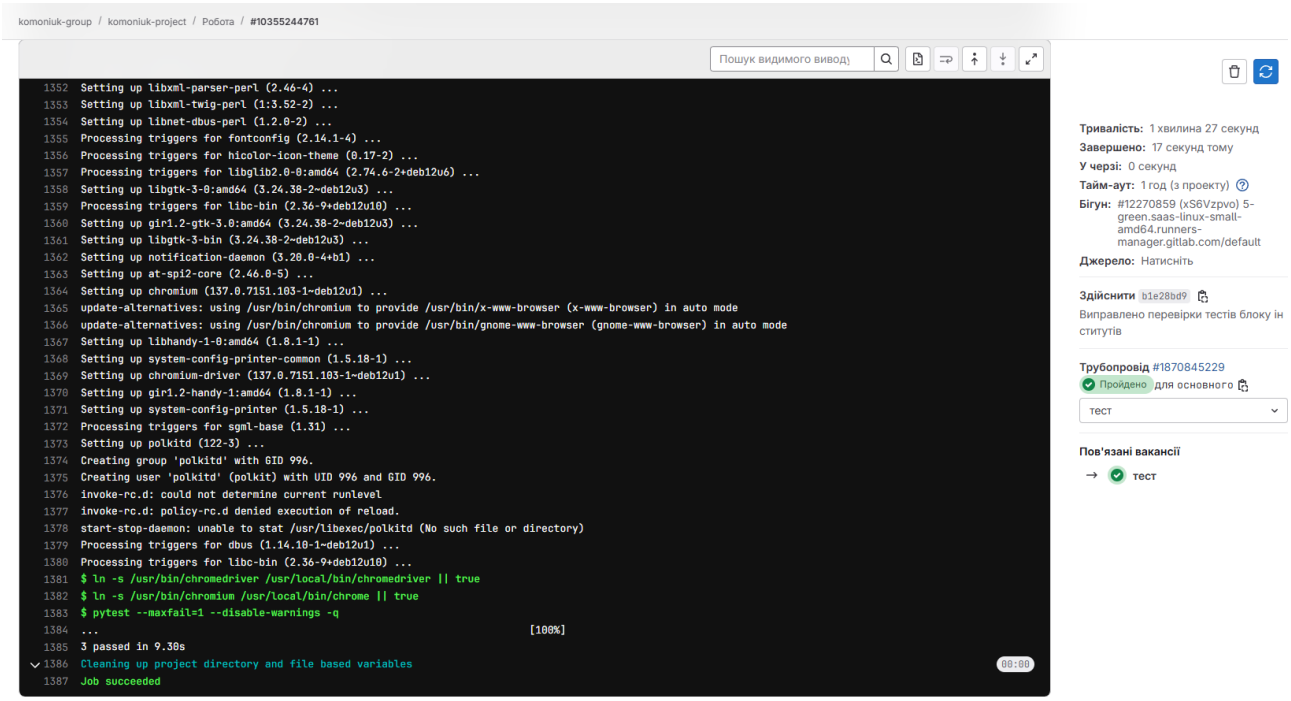

Рис. 4.2. Статус виконання пайплайну

4.3 Тестування безпеки вебзастосунку

Одним із ключових аспектів забезпечення надійності та довіри до сучасного вебзастосунку є перевірка його захищеності від типових векторів атак та помилок реалізації доступу. Для цього у кваліфікаційній роботі було розроблено низку автоматизованих тестів за допомогою Selenium, які імітують некоректну поведінку користувача, а також спроби отримати несанкціонований доступ до захищених частин системи. Такі перевірки дозволяють на ранньому етапі виявляти й усувати потенційні уразливості ще до потрапляння оновлень на продакшн.

4.3.1 Перевірка обмеження доступу до захищених розділів

Перший тест перевіряє, чи не відкривається адміністративний розділ сайту для неавторизованого користувача. Це важливий критерій для будь-якого ресурсу, оскільки відкритий адмін-інтерфейс може стати точкою входу для атак.

Код:

```
from selenium import webdriver
```

‘

```
import time


def test_access_protected_area():
    options = webdriver.ChromeOptions()
    options.add_argument('--no-sandbox')
    options.add_argument('--headless')
    options.add_argument('--disable-dev-shm-usage')
    options.add_argument('--disable-gpu')
    driver = webdriver.Chrome(options=options)
    try:
        driver.get("https://exam.nuwm.edu.ua/admin")
        time.sleep(2)
        assert "Увійти" in driver.page_source or "login" in driver.current_url or "Авторизація" in driver.page_source
    finally:
        driver.quit()
```

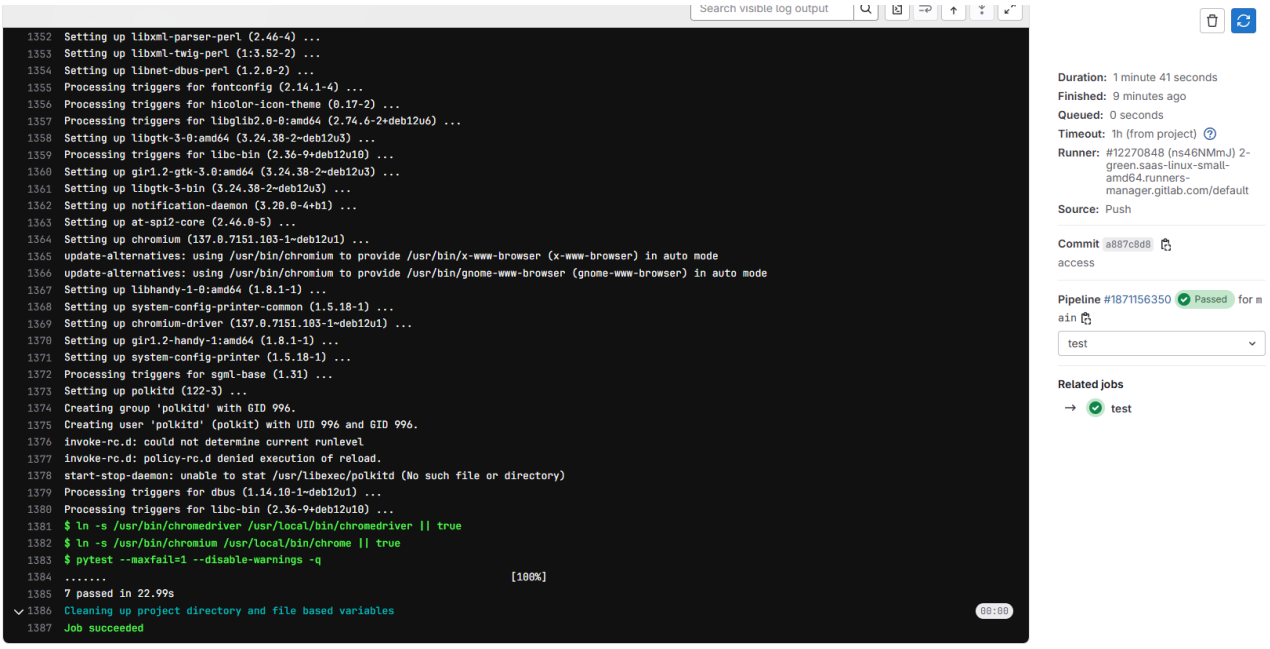

Рис. 4.3.1. Статус виконання пайплайну

Пояснення:

‘

Тест відкриває сторінку адміністративного розділу без попередньої авторизації та перевіряє, що користувач бачить сторінку входу, а не інтерфейс адміністратора. Це гарантує, що прямий доступ до захищених функцій обмежений.

4.3.2 Тестування поля пошуку на XSS-уразливість

Другий тест перевіряє, чи коректно сайт екранує небезпечний скриптовий код, що вводиться у поле пошуку на сторінці форуму (Колонка новин). XSS-атаки можуть призвести до виконання стороннього коду у браузері користувача, тому правильна фільтрація — обов'язкова.

Код:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import uuid

def test_xss_field():
    profile_path = f"/tmp/chrome-user-data-{uuid.uuid4()}"
    options = webdriver.ChromeOptions()
    options.add_argument('--no-sandbox')
    options.add_argument('--headless')
    options.add_argument('--disable-dev-shm-usage')
    options.add_argument('--disable-gpu')
    options.add_argument(f'--user-data-dir={profile_path}')
    driver = webdriver.Chrome(options=options)
    try:
        driver.get("https://exam.nuwm.edu.ua/mod/forum/view.php?id=1")
        search_input = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.NAME, "search"))
```

‘

```
        )
        payload = "<script>alert('xss')</script>"
        search_input.send_keys(payload)
        search_input.submit()
        WebDriverWait(driver, 5).until(EC.staleness_of(search_input))
        page_source = driver.page_source.lower()
        assert "<script>alert('xss')</script>" not in page_source
        assert "&lt;script&gt;alert('xss')&lt;/script&gt;" in page_source or "search" in page_source
    finally:
        driver.quit()
```

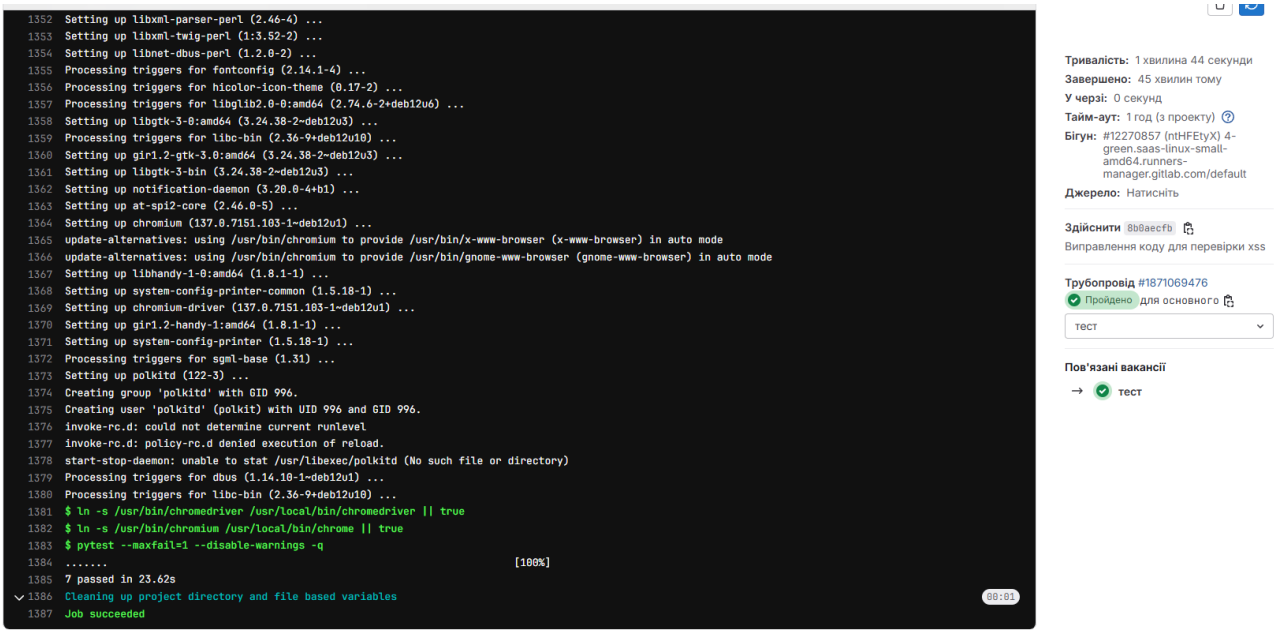

Рис. 4.3.2. Статус виконання пайплайну

Пояснення:

Тест здійснює спробу XSS-ін'єкції, вводячи у поле пошуку спеціальний JavaScript-код. Після відправлення форми перевіряється, що скрипт не виконався (тобто в коді сторінки немає неекранованого <script>alert('xss')</script>), а введення було коректно екрановано (відображається у вигляді <script>alert('xss')</script>). Це підтверджує, що сайт не вразливий до XSS саме у цій формі.

‘

4.3.3 Перевірка сторінки помилки 404

Третій тест імітує перехід на неіснуючу сторінку та перевіряє, чи коректно вебзастосунок повертає повідомлення про помилку 404.

Код:

```
from selenium import webdriver
import time
import uuid
import os

def test_404_page():
    profile_path = f"/tmp/chrome-user-data-{os.getpid()}-{uuid.uuid4()}"
    options = webdriver.ChromeOptions()
    options.add_argument('--headless')
    options.add_argument('--no-sandbox')
    options.add_argument('--disable-dev-shm-usage')
    options.add_argument('--disable-gpu')
    options.add_argument(f'--user-data-dir={profile_path}')
    driver = webdriver.Chrome(options=options)
    try:
        driver.get("https://exam.nuwm.edu.ua/nonexistent-page-123456")
        time.sleep(2)
        assert "404" in driver.page_source or "не знайдено" in driver.page_source.lower(), "Сторінка помилки не відображається коректно!"
    finally:
        driver.quit()
```

‘

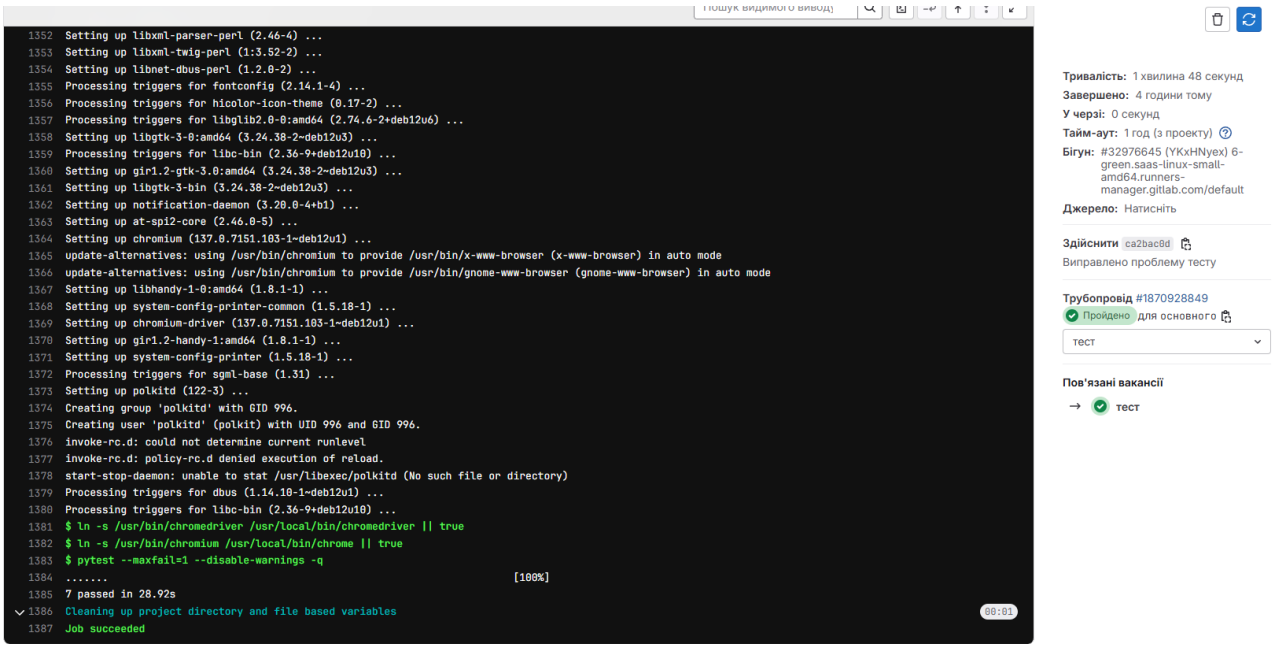

Рис. 4.3.3. Статус виконання пайплайну

Пояснення:

Тест відкриває довільну неіснуючу сторінку сайту і перевіряє, чи відображається на ній повідомлення про помилку (наприклад, "404" або "не знайдено"). Це свідчить про коректне оброблення невалідних запитів і відсутність зайвої інформації для потенційного зловмисника.

4.4 Тестування навантаження (performance)

Тестування навантаження (performance testing) є ключовим етапом оцінки працездатності вебзастосунку в умовах багаторазового або інтенсивного доступу. Незважаючи на те, що Selenium переважно застосовується для автоматизації функціонального тестування, він може бути використаний для базового вимірювання часу завантаження вебсторінки у циклічному режимі, що дає змогу отримати загальне уявлення про продуктивність сайту.

Для прикладу реалізовано простий скрипт, який запускає кілька сесій браузера з headless-режимом і вимірює час завантаження сайту https://exam.nuwm.edu.ua/. За результатами роботи коду обчислюються середнє,

‘
мінімальне та максимальне значення часу відповіді.

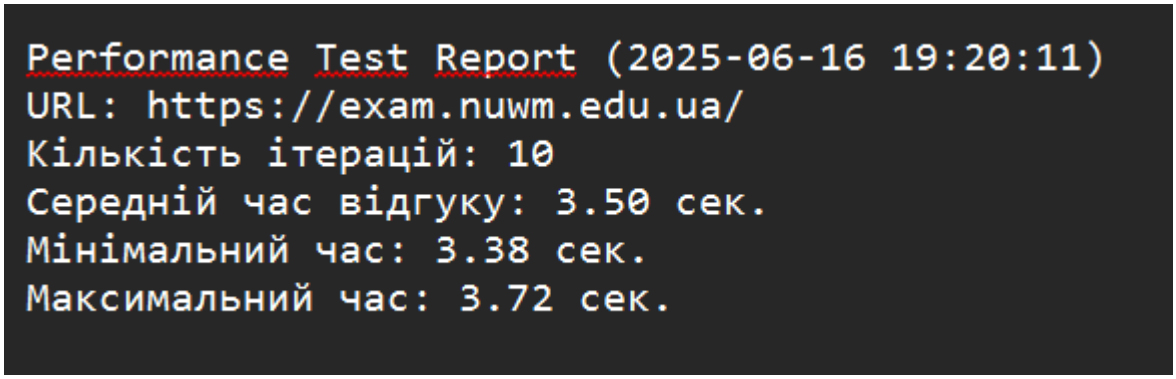

Рис. 1 4.4. Звіт про тестування продуктивності

Код:

```
from selenium import webdriver
import time
import datetime

def performance_test(url, iterations=10):
    options = webdriver.ChromeOptions()
    options.add_argument('--headless')
    options.add_argument('--no-sandbox')
    options.add_argument('--disable-dev-shm-usage')
    options.add_argument('--disable-gpu')

    response_times = []

    for i in range(iterations):
        driver = webdriver.Chrome(options=options)
        start = time.time()
        driver.get(url)
        time.sleep(1)
        end = time.time()
        response_times.append(end - start)
```

‘

```
        driver.quit()


    avg_time = sum(response_times) / len(response_times)
    report = f"Performance Test Report ({datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')})\n"
    report += f"URL: {url}\n"
    report += f"Кількість ітерацій: {iterations}\n"
    report += f"Середній час відгуку: {avg_time:.2f} сек.\n"
    report += f"Мінімальний час: {min(response_times):.2f} сек.\n"
    report += f"Максимальний час: {max(response_times):.2f} сек.\n"

    with open("performance_report.txt", "w", encoding="utf-8") as f:
        f.write(report)

def test_dummy():
    performance_test("https://exam.nuwm.edu.ua/", iterations=10)
    assert True
```

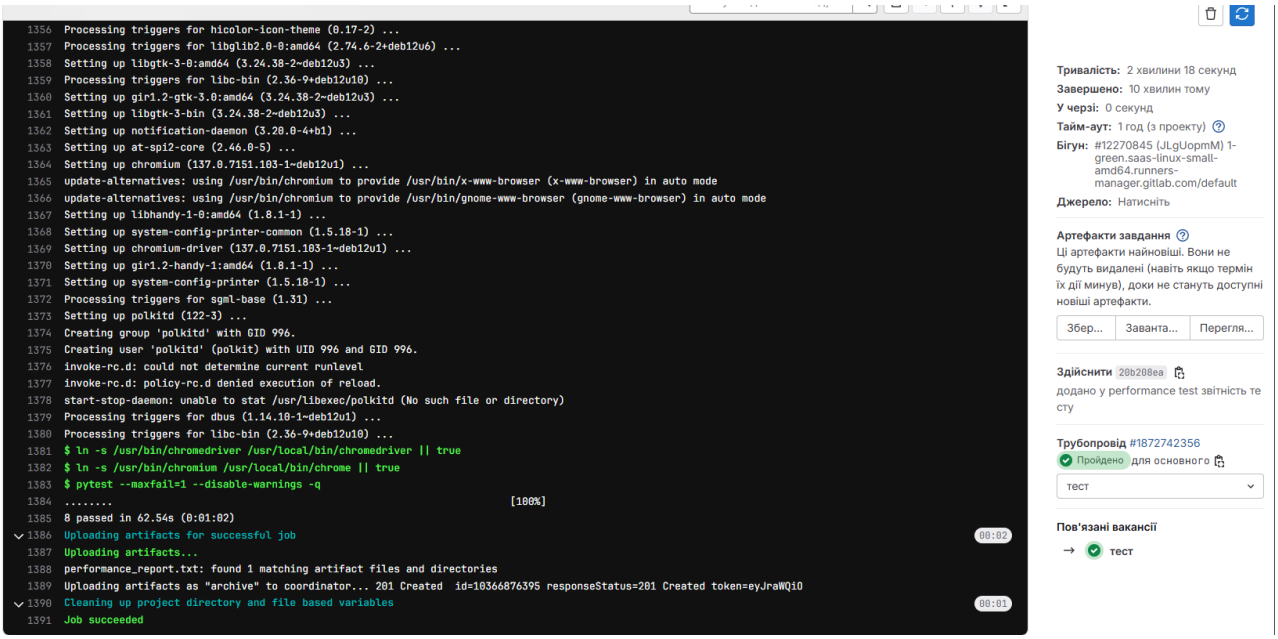

Рис. 2 4.4. Статус виконання пайплайну

Переваги використання Selenium для тестування навантаження:

‘

- **Простота реалізації:** Не потребує складної конфігурації чи зовнішніх інструментів. Достатньо лише Python і Selenium.
- **Реалістичне завантаження:** Вебсторінка завантажується так само, як і у звичайному браузері, що імітує поведінку реального користувача.
- **Інтеграція з CI/CD:** Код можна легко включити до пайплайну GitLab або іншої платформи CI/CD.
- **Мінімальні вимоги до системи:** Headless-режим дозволяє запускати тест навіть на сервері без GUI.

Недоліки:

- **Не підходить для великого навантаження:** Неможливо симулювати одночасний доступ великої кількості користувачів.
- **Обмежена аналітика:** Відсутність деталізованих метрик (розбиття на час відповіді серверу, рендеринг, мережеві запити тощо).
- **Нестабільність результатів:** Через роботу з реальним браузером результати можуть варіюватися в залежності від навантаження на машину.
- **Повільність:** У кожній ітерації відкривається нова сесія браузера, що потребує більше ресурсів і часу, ніж, наприклад, HTTP-запити через requests чи навантаження через JMeter.

4.5 Можливості розширення автоматизованих тестів

У процесі розробки системи автоматизованого тестування важливо передбачити можливість масштабування та інтеграції з додатковими інструментами. Проєкт, створений для тестування сайту НУВГП, було реалізовано з урахуванням розширюваності, що дозволяє безболісно додавати нові тести, інтеграції та сервіси.

Одним із ключових кроків розширення стала інтеграція системи генерації звітів — **Allure**.

Переваги використання Allure

- Візуалізація результатів тестів у зручному HTML-форматі.

‘

- Наявність повного логування кожного тесту (успішного чи проваленого).
- Можливість зберігати звіти як артефакти CI/CD пайплайну.
- Висока сумісність із pytest, що використовується в основному проєкті.
- Автоматичне збереження даних у папку allure-results.

Внесені зміни

1. Файл requirements.txt

У цей файл було додано необхідні бібліотеки для роботи з Allure:

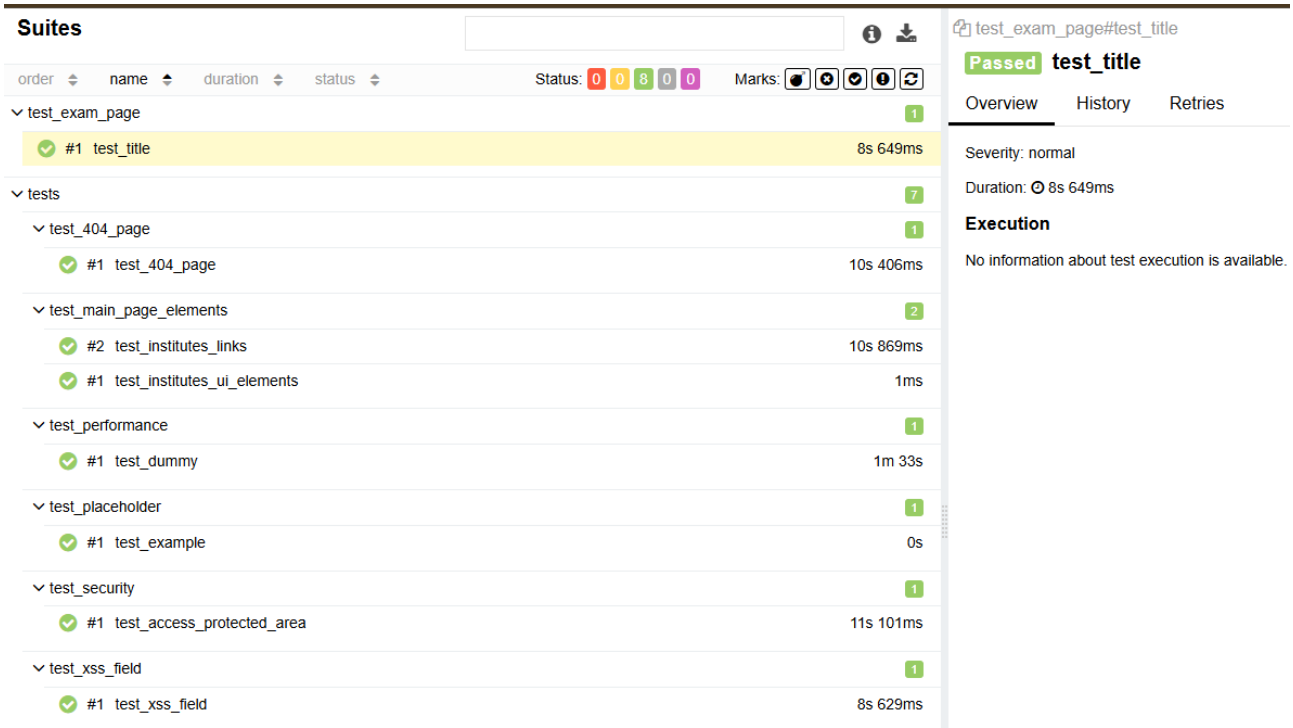

Рис. 4.5. Інтерфейс звіту Allure

Selenium
pytest
allure-pytest

2. Файл .gitlab-ci.yml

Оновлений файл CI/CD для запуску тестів з генерацією звітів і збереженням результатів:

```
image: python:3.11

stages:
  - test
```

‘

```
before_script:
  - pip install -r requirements.txt
  - pip install allure-pytest
  - apt-get update
  - apt-get install -y wget unzip chromium-driver chromium
  - ln -s /usr/bin/chromedriver /usr/local/bin/chromedriver || true
  - ln -s /usr/bin/chromium /usr/local/bin/chrome || true

test:
  stage: test
  script:
    - pytest --maxfail=1 --disable-warnings -q --alluredir=allure-results
  artifacts:
    paths:
      - performance_report.txt
      - allure-results/
    expire_in: 1 week
```

Ці зміни дозволяють не лише зберігати текстовий файл зі статистикою performance-тестів, але й автоматично збирати raw-дані для звіту Allure. Це важливо для подальшого перегляду історії запусків і візуального аналізу помилок.

3. Тестові файли

Не було потрібно змінювати жодного з тестів — команда pytest --alluredir=allure-results збирає дані автоматично. Втручання в код не є обов’язковим, якщо вже встановлено allure-pytest.

4. Запуск звіту Allure (на локальній машині)

Для перегляду звіту на локальному ПК після завершення тестування:

```
allure generate allure-results -o allure-report --clean
allure open allure-report
```

Якщо команда allure не розпізнається, потрібно встановити Allure CLI, додати його до системного шляху і перезапустити термінал.

Загальні переваги розширення

- **Масштабованість** — можливо додати нові тести, не змінюючи основну структуру CI.
- **Візуальна звітність** — Allure дає змогу швидко побачити статус тестів, помилки і тривалість.
- **Мінімальні залежності** — всі дії виконуються через Pytest та GitLab Runner.

‘

- **Автоматизація** — повна інтеграція з CI/CD забезпечує автономність тестування.

Можливі недоліки

- Необхідність ручного відкриття звіту Allure (якщо не використовується сервер або спеціалізований плагін GitLab).
- Потрібна установка Allure CLI на локальну машину.
- Звіти зберігаються лише обмежений час у GitLab (у нашому випадку — 1 тиждень).

У підсумку, впровадження Allure значно підвищило інформативність автоматизованого тестування. Система стала більш зручною для подальшого аналізу, розширення та рефакторингу. Цей крок підвищує рівень професіоналізму тестування й наближає його до промислових практик QA.

‘

ВИСНОВКИ

У даній кваліфікаційній роботі було проведено повноцінне дослідження та практичну реалізацію процесу автоматизованого тестування вебзастосунку з використанням інструментів Selenium та GitLab CI/CD, що дозволило розробити ефективну, гнучку і масштабовану систему контролю якості вебресурсу.

На етапі теоретичного аналізу було опрацьовано основні принципи функціонування вебзастосунків, актуальні аспекти автоматизації тестування, підходи до побудови CI/CD пайплайнів та особливості взаємодії різних інструментів у сучасному DevOps-середовищі. Особливу увагу приділено методам тестування інтерфейсів користувача та захищеності вебдодатків.

У ході реалізації було створено набір автотестів на мові Python із використанням бібліотеки Selenium WebDriver. Було протестовано основні функціональні елементи головної сторінки сайту exam.nuwm.edu.ua, включаючи доступність розділів, перевірку на наявність повідомлень про помилки при некоректних діях користувача та відповідність очікуваному інтерфейсу.

Під час реалізації підрозділу 4.3 проведено базову перевірку на безпеку: протестовано обмеження доступу до захищених сторінок без авторизації, здійснено спробу XSS-ін'єкції, а також перевірено коректність обробки сторінки з кодом помилки 404. Це дозволило виявити базовий рівень захищеності ресурсу та впевнитись у коректному реагуванні системи на спроби несанкціонованого доступу.

У підрозділі 4.4 проведено базове тестування продуктивності вебзастосунку за допомогою Selenium. Попри те, що Selenium не є вузькоспеціалізованим інструментом навантажувального тестування, проведений експеримент дозволив оцінити середній час відгуку сторінки під час повторного завантаження, що дало первинне уявлення про стабільність роботи ресурсу.

Підрозділ 4.5 був присвячений масштабуванню та розширенню системи автоматизованого тестування. Було впроваджено інтеграцію з Allure для формування детальних візуалізованих звітів про виконання тестів. Це дало змогу

автоматично збирати та зберігати результати у вигляді артефактів, які доступні після завершення CI/CD пайплайну. Такий підхід значно підвищує прозорість процесу тестування, полегшує аналіз результатів і підтримку тестів у майбутньому.

Крім того, було виконано налаштування GitLab CI/CD конфігурації у файлі .gitlab-ci.yml, що дозволило повністю автоматизувати запуск тестів у репозиторії. Внесення змін до цього файлу дозволило одночасно зберігати історію виконання тестів та результати звітів.

У результаті виконаної роботи можна стверджувати, що поставлені завдання були реалізовані повністю. Сформована система дозволяє:

- автоматизовано перевіряти функціональність вебінтерфейсу;
- перевіряти базову безпеку сайту без втручання вручну;
- оцінювати продуктивність сайту при повторних зверненнях;
- отримувати зручні звіти про виконання тестів;
- масштабувати тестову систему для розширення обсягу перевірок.

Таким чином, реалізована система є ефективною основою для подальшого вдосконалення якості та безпеки вебресурсів, а також є актуальною практикою для сучасного підходу до DevOps і забезпечення якості ПЗ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Selenium. *Офіційна документація*. URL: https://www.selenium.dev/documentation/ (дата звернення: 15.02.2025).
2. GitLab CI/CD Pipelines. URL: https://docs.gitlab.com/ee/ci/ (дата звернення: 15.02.2025).
3. GitLab Runner. *Офіційна документація*. URL: https://docs.gitlab.com/runner/ (дата звернення: 15.02.2025).
4. Allure Framework Documentation. URL: https://docs.qameta.io/allure/ (дата звернення: 17.02.2025).
5. Python Selenium WebDriver API. URL: https://selenium-python.readthedocs.io/ (дата звернення: 19.02.2025).
6. Інструкція по написанню GitLab CI/CD для Python-проєктів. *Real Python*. URL: https://realpython.com/python-gitlab-ci-cd/ (дата звернення: 20.02.2025).
7. Стаття: «Continuous Integration and Deployment using GitLab CI/CD and Selenium». *BrowserStack Guide*. URL: https://www.browserstack.com/guide/selenium-gitlab-ci-integration (дата звернення: 22.02.2025).
8. Офіційна сторінка Google ChromeDriver. URL: https://sites.google.com/chromium.org/driver/ (дата звернення: 24.02.2025).
9. Приклад написання автотестів на Python із використанням pytest. URL: https://docs.pytest.org/en/latest/ (дата звернення: 01.03.2025).
10. CI/CD в Python з GitLab (відео і блог). *Towards Data Science*. URL: https://towardsdatascience.com/how-to-set-up-ci-cd-pipeline-using-gitlab-on-python-project-3909b0f9f8b5 (дата звернення: 10.03.2025).
11. Звіти Allure у GitLab CI/CD — GitHub репозиторій прикладу. URL: https://github.com/marketplace/actions/allure-report (дата звернення: 14.04.2025).

12. Публікація про автоматизацію UI-тестів. *Test Automation University / Applitools*. URL: https://testautomationu.applitools.com/selenium-webdriver-tutorial-java/ (дата звернення: 16.04.2025).
13. Стаття про безпечне тестування (SQL Injection, XSS). *OWASP*. URL: https://owasp.org/www-community/attacks/ (дата звернення: 20.04.2025).