Міністерство освіти і науки України
Кам’янець-Подільський національний університет імені Івана Огієнка
Фізико-математичний факультет
Кафедра комп’ютерних наук

Кваліфікаційна робота бакалавра

з теми: «**Дослідження технології Selenium для автоматизації тестування веб-сайту**»

Виконав: здобувач вищої освіти групи KN1-B21
спеціальності 122 Комп’ютерні науки
Перепелюк Дмитро Андрійович
Керівник: Моцик Ростислав Васильович,
кандидат педагогічних наук,
доцент кафедри комп’ютерних наук

Рецензент: Оптасюк Сергій Васильович,
кандидат фізико-математичних наук,
доцент, завідувач кафедри фізики

м. Кам’янець-Подільський – 2025 р.

АНОТАЦІЯ

Тема роботи: “Дослідження технології Selenium для автоматизації тестування веб-сайту”

Виконавець: Перепелюк Дмитро Андрійович, здобувач вищої освіти групи KN1-B21, спеціальність 122 Комп’ютерні науки, Кам’янець-Подільський національний університет імені Івана Огієнка

Науковиий керівник: Моцик Ростислав Васильович, кандидат педагогічних наук, доцент, доцент кафедри комп’ютерних наук.

Кваліфікаційна робота присвячена дослідженню можливостей використання технології Selenium WebDriver для автоматизації тестування веб-сайтів. Метою дослідження є аналіз принципів роботи Selenium, порівняння його з іншими інструментами автоматизованого тестування та реалізація практичної частини — створення тестового фреймворку з перевіркою ключового функціоналу веб застосунку.

Об’єктом дослідження є процес автоматизованого тестування веб-застосунків.

Предметом дослідження — технологія Selenium WebDriver, архітектура інструменту, підхід Page Object Model та принципи створення стабільних автотестів.

У роботі застосовано методи аналізу технічної документації, порівняльного аналізу інструментів, модульного тестування та збору результатів через систему звітності Allure. Реалізовано тестування авторизації, реєстрації та обробки помилок у веб-інтерфейсі.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У результаті дослідження створено автоматизовану тестову систему на базі Selenium WebDriver та Pytest, яка демонструє ефективність автоматизації при перевірці веб-сайтів.

Ключові слова: автоматизація тестування, Selenium WebDriver, веб-сайт, автотести, Python, Pytest, Page Object Model, Allure.

ANNOTATION

Thesis topic: "Research of Selenium Technology for Website Testing Automation".

Author: Dmytro Andriiovych Perepeliuk, student of group KN1-B21, specialty 122 Computer Science, Kamianets-Podilskyi National Ivan Ohiienko University.

Supervisor: Rostyslav Vasylovych Motsyk Candidate of Pedagogical Sciences, Associate Professor, Associate Professor at the Department of Computer Science.

The bachelor's qualification thesis is devoted to the study of Selenium WebDriver technology for automating the testing of web applications. The purpose of the research is to analyze the architecture and functionality of Selenium, compare it with other test automation tools, and implement a practical test framework for verifying key website functionality.

The object of the research is the process of automated testing of web applications.

The subject of the research is Selenium WebDriver technology, its architecture, the Page Object Model approach, and principles of writing stable automated tests.

The study applies methods of documentation analysis, tool comparison, test design, and result processing using Allure reports. Automated test scenarios were developed for login, registration, and error handling pages.

The thesis consists of an introduction, three main chapters, conclusions, a list of references, and appendices. As a result, a working automated testing system was implemented using Selenium WebDriver and Pytest, demonstrating the effectiveness of automation in modern web testing.

Keywords: test automation, Selenium WebDriver, web application, automated tests, Python, Pytest, Page Object Model, Allure.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення

Баг – помилка, дефект

Selenium – набір інструментів для автоматизованого тестування веб-додатків, який включає WebDriver, Selenium Grid та інші інструменти.

Test Case – тестовий випадок (послідовність дій для перевірки функціональності).

Test Suite — набір тестів.

CI/CD – Continuous Integration/Continuous Deployment, безперервна інтеграція і безперервне розгортання.

POM – Page Object Model, модель об'єкта сторінки (шаблон проектування для автоматизації тестів).

UI – User Interface, користувацький інтерфейс.

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій якість програмного забезпечення (ПЗ) відіграє ключову роль. Користувачі очікують, що веб-сайти та веб-додатки працюватимуть стабільно, швидко та без помилок. Саме тому тестування стало невід'ємною частиною життєвого циклу розробки програмного забезпечення.

Серед усіх видів тестування особливе місце займає автоматизоване тестування, яке дозволяє значно скоротити час перевірки, підвищити точність результатів та зменшити навантаження на команду тестувальників. Одним із найпоширеніших і найпотужніших інструментів для автоматизації веб-тестування є Selenium — безкоштовна, багатоплатформна технологія з відкритим кодом, яка підтримує різні мови програмування, браузери та операційні системи.

Актуальність теми полягає у тому, що автоматизація тестування стає стандартом для сучасної розробки, а знання інструментів, таких як Selenium, є важливою складовою професійної підготовки спеціалістів з комп'ютерних наук.

Об'єктом дослідження є процес автоматизованого тестування веб-застосунків із використанням сучасних технологій та фреймворків.

Предметом дослідження є технологія Selenium WebDriver, її архітектурні особливості, супровідні підходи (Page Object Model, BDD), засоби інтеграції в автоматизовані пайплайни, та ефективність їх використання у реальному середовищі розгортання.

Метою кваліфікаційної роботи є вивчення можливостей та принципів роботи технології Selenium, а також на практиці реалізувати автоматизовані тести для перевірки функціоналу веб-сайту. Для досягнення мети були сформульовані наступні **завдання**:

1. Ознайомитися з теоретичними основами тестування та класифікацією його видів.

2. Проаналізувати сучасні інструменти автоматизованого тестування.

3. Дослідити структуру, архітектуру та функціональні можливості Selenium.

4. Розробити та реалізувати автоматизовані тести з використанням Selenium WebDriver.

5. Провести запуск тестів, зібрати результати та сформулювати висновки щодо ефективності обраного підходу.

Методи дослідження: аналіз наукових і технічних джерел (статей, документації, стандартів), емпіричне тестування в контрольованому середовищі, моделювання автоматизованого фреймворку, використання метрик для кількісної оцінки ефективності.

Наукова новизна роботи полягає в поєднанні архітектурного патерну POM і поведінкового підходу BDD із паралельним запуском у Docker-контейнеризованому середовищі. Запропонована методика дозволяє підвищити стабільність тестування, забезпечити масштабованість тестів і їхню адаптацію до CI/CD процесів сучасної розробки.

Практичне значення роботи полягає в тому, що розроблений фреймворк може бути використаний для побудови системи автоматизованого тестування у реальних проєктах, зокрема в середовищі середніх і великих IT-команд. Використані інструменти є відкритими та можуть бути легко інтегровані в пайплайни з метою автоматичного запуску перевірок якості.

Структура роботи: кваліфікаційна робота складається зі вступу, трьох розділів, висновків та списку використаних джерел. У першому розділі виконано теоретичний аналіз методів тестування та огляд інструментів. Другий розділ присвячений дослідженню можливостей Selenium. У третьому

розділі наведена практична реалізація тестування веб-застосунку за допомогою Selenium WebDriver.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКІВ

1.1 Основні поняття тестування програмного забезпечення

Визначення тестування програмного забезпечення. Тестування програмного забезпечення є важливою частиною життєвого циклу розробки програмного забезпечення, яка передбачає перевірку та підтвердження того, чи працює програмний додаток так, як очікувалося. Він забезпечує надійність, коректність, безпеку та високу продуктивність програмного забезпечення для веб, мобільних додатків, хмарних та CI/CD конвеєрів у DevOps [1]. За допомогою тестування можна надати незалежну оцінку якості програмного забезпечення.

Основні завдання тестування ПЗ включать:

- Виявлення дефектів. Виявлення помилок у коді, інтерфейсі чи логіці роботи програми.
- Перевірка відповідності вимогам, забезпечує, що ПЗ відповідає функціональним і нефункціональним вимогам, зазначеним у технічній документації.
- Оцінка якості, перевіряє стабільність, продуктивність, безпеку та зручність використання продукту.
- Забезпечення надійності, забезпечує коректну роботу ПЗ у різних умовах експлуатації, включаючи різні платформи, операційні системи, з’єднання до мережі.

Тестування веб застосунків має унікальні особливості, зумовлені їх архітектурою та умовами використання. Веб застосунки працюють у різних браузерах, операційних системах та пристроях, що вимагає комплексного підходу до тестування, який включає всі аспекти веб застосунків: перевірку

кросбраузерності, адаптивності дизайну, а також продуктивності в умовах різного з'єднання до інтернету.

Місце тестування в життєвому циклі ПЗ

Життєвий цикл розробки програмного забезпечення (SDLC) охоплює етапи від збору вимог до підтримки продукту. Тестування інтегрується в SDLC на різних рівнях залежно від моделі розробки. У традиційній моделі Waterfall тестування є окремим етапом після кодування, тоді як у Agile і DevOps воно виконується безперервно, інтегруючись у кожну ітерацію. Життєвий цикл тестування програмного забезпечення (STLC) є підмножиною SDLC і включає такі етапи:

- Аналіз вимог. Визначення тестованих функцій на основі специфікацій.
- Планування тестів. Розробка стратегії та вибір інструментів.
- Розробка тестів. Створення тестових випадків і сценаріїв.
- Налаштування середовища. Підготовка апаратного та програмного забезпечення
- Виконання тестів. Запуск тестів і фіксація результатів.
- Завершення тестування. Аналіз результатів і підготовка звітів.

У V-моделі SDLC кожна фаза розробки відповідає певному рівню тестування (наприклад, вимоги — приймальне тестування, кодування — одиничне тестування), що забезпечує систематичний підхід до якості.

Важливо, що процес розробки ПЗ неможливий без контролю якості. Воно дозволяє виявити дефекти на ранніх стадіях, усунути проблеми до релізу та підвищити довіру замовника до кінцевого продукту.

Роль тестувальника

Тестувальник відіграє ключову роль у забезпеченні якості ПЗ, беручи участь у всіх етапах SDLC. Основні обов'язки тестувальника включають аналіз вимог, розробку тестових випадків, виконання тестів, документування

дефектів і співпрацю з розробниками для їх усунення. У сучасних методологіях, таких як Agile, тестувальники також беруть участь у плануванні спринтів і переглядах проєктів, сприяючи ранньому виявленню проблем. Тестувальник повинен володіти технічними знаннями, аналітичними навичками та розумінням бізнес-вимог, щоб забезпечити відповідність продукту очікуванням користувачів.

1.2 Класифікація видів тестування

Класифікація тестування. Тестування класифікується за кількома критеріями як за такими, рівнями:

1. Модульне тестування для перевірки окремих компонентів або модулів ПЗ. Наприклад, тестування функції обробки даних у веб-додатку.
2. Інтеграційне тестування для перевірки взаємодії між модулями. Наприклад, тестування коректності обміну даними між фронтендом і бекендом.
3. Системне тестування для перевірки системи в цілому на відповідність вимогам. Наприклад, тестування всіх функцій веб-додатку для онлайн-магазину.
4. Приймальне тестування для перевірки відповідності ПЗ очікуванням замовника. Наприклад, перевірка, чи відповідає веб-додаток специфікації перед релізом.

За способом виконання класифікується як:

1. Ручне тестування полягає у проведенні тестування тестувальником вручну без використання скриптів. Наприклад, перевірка зручності інтерфейсу веб-додатку.
2. Автоматизоване тестування виконується за допомогою спеціалізованих інструментів. Наприклад, автоматизована перевірка форми входу за допомогою Selenium.

За доступом до коду класифікується як:

1. Black-box. Тестування без доступу до коду, зосереджене на поведінці системи. Наприклад, перевірка коректності відображення сторінки без аналізу її коду.
2. White-box. Тестування з доступом до коду для аналізу його структури. Наприклад, перевірка логіки функції в коді.
3. Grey-box. Комбінація підходів, де тестувальник має частковий доступ до коду. Наприклад, тестування API із розумінням структури запитів.

Згідно з рекомендаціями ISTQB, ефективне тестування передбачає комбінацію різних видів тестування для забезпечення максимального покриття вимог [2].

1.3 Автоматизація тестування: цілі, переваги, обмеження

Визначення автоматизованого тестування. Автоматизоване тестування – це процес виконання тестів за допомгою спеціалізованих інструментів і скриптів без прямої участі людини і порівняння результатів, тоді як ручне тестування виконується тестувальником вручну. Цей підхід є особливо важливим для веб застосунків, де велика кількість сценаріїв використання, часте оновлення коду та потреба у швидкому випуску нових версій.

Основні завдання автоматизації тестування:

- Підвищення ефективності. Автоматизація значно скорочує час, необхідний для виконання тестів, порівняно з ручним тестуванням, що прискорює цикл розробки.
- Забезпечення повторюваності. Автоматизовані тести дозволяють відтворювати однакові тестові сценарії з незмінними умовами, що є важливим для регресійного тестування.
- Мінімізація людських помилок. Автоматизовані скрипти виключають помилки, спричинені людським фактором, такі як неуважність або втома тестувальника.

- Покриття складних сценаріїв. Автоматизація забезпечує можливість тестування складних сценаріїв, таких як навантажувальні тести чи перевірка поведінки системи при великій кількості одночасних запитів.

Автоматизація тестування має низку переваг порівняно з ручним тестуванням:

- Швидкість виконання. Автоматизовані тести дозволяють оперативно отримувати результати, що особливо важливо для проєктів із короткими циклами розробки (наприклад, у методології Agile).
- Економія ресурсів. Зменшується потреба в залученні великої кількості тестувальників, що знижує витрати.
- Повторне використання тестів. Розроблені тестові скрипти можуть застосовуватися до різних версій ПЗ, що забезпечує економію часу при регресійному тестуванні.
- Висока точність. Автоматизовані тести забезпечують стабільні та відтворювані результати, що знижує ймовірність пропуску дефектів.

Доцільність автоматизації. Автоматизація доцільна, коли тести виконуються часто (наприклад, регресійне тестування), є потреба в швидкому виконанні або коли тести складно виконати вручну (наприклад, навантажувальні тести). Показник повернення інвестицій (ROI) залежить від частоти виконання тестів і складності їх створення. Наприклад, автоматизація регресійних тестів економить час, якщо проєкт має тривалий життєвий цикл. Проте цей підхід має і свої недоліки, тому що він не може повністю замінити ручне тестування в оцінці таких аспектів як зручність інтерфейсу, естетика дизайну чи відповідність очікуванням користувача. Крім того, написання, налаштування та підтримка автоматизованих тестів потребує значних початкових витрат часу, ресурсів та кваліфікованих спеціалістів.

1.4 Інструменти автоматизованого тестування

Сучасний ринок пропонує великий вибір інструментів для автоматизації тестування веб-застосунків:

● Selenium – один із найпоширеніших інструментів для автоматизації функціонального тестування веб-додатків. Він підтримує кілька мов програмування (Java, Python, C#, JavaScript) і дозволяє створювати тести для різних браузерів, таких як Chrome, Firefox і Edge. Selenium WebDriver забезпечує гнучкість у створенні складних тестових сценаріїв [3].

● Cypress – сучасний інструмент, який працює безпосередньо в браузері, що забезпечує швидке виконання тестів і зручний дебагінг. Cypress особливо популярний для тестування веб-додатків, побудованих на фреймворках, таких як React чи Angular [4].

● JMeter – інструмент для тестування продуктивності, який дозволяє моделювати велику кількість одночасних користувачів і оцінювати продуктивність веб-додатку. Наприклад, JMeter може симулювати 10 000 одночасних запитів до сервера [5].

● TestComplete – комерційний інструмент, який підтримує автоматизацію тестування як веб-, так і десктопних додатків. Він пропонує зручний інтерфейс для створення тестів.

● Postman – інструмент для автоматизації тестування API, що є важливим для веб-додатків із клієнт-серверною архітектурою. Postman дозволяє створювати тести для перевірки API-запитів і відповідей [6].

Вибір інструменту залежить від типу тестування, технічних вимог, бюджету та рівня кваліфікації команди. Наприклад, Selenium є універсальним і безкоштовним, тоді як TestCompelete використовується для великих комерційних проєктів.

1.5 Тестування веб-застосунків

Особливості веб-застосунків. Веб-додатки відрізняються від десктопних і мобільних додатків своєю клієнт-серверною архітектурою, залежністю від браузерів і динамічним характером інтерфейсу. На відміну від десктопних додатків, які працюють у контрольованому середовищі, веб-додатки

потребують тестування в різних браузерах і на різних пристроях. Порівняно з мобільними додатками, веб-додатки мають ширший спектр конфігурацій (браузери, ОС), але менш жорсткі вимоги до апаратних ресурсів.

Проблеми тестування веб-додатків

- Зміна DOM. Динамічні зміни структури сторінки через JavaScript ускладнюють тестування.
- Сумісність із браузерами. Різні браузери (Chrome, Firefox, Safari) можуть по-різному інтерпретувати HTML/CSS.
- Взаємодія з JavaScript/AJAX. Асинхронні запити ускладнюють синхронізацію тестів.
- Кросбраузерне тестування. Потреба перевірки на різних браузерах і пристроях.

Висновки до розділу 1

Тестування програмного забезпечення є невід'ємною частиною розробки веб-додатків, забезпечуючи їхню якість, надійність і відповідність вимогам. Воно охоплює функціональне, нефункціональне, регресійне, тестування сумісності, безпеки та локалізації, кожен із яких відіграє важливу роль у виявленні дефектів. Автоматизація тестування підвищує ефективність цього процесу, зменшує людські помилки та забезпечує повторюваність тестів. Інструменти, такі як Selenium, Cypress, JMeter, TestComplete і Postman, надають широкі можливості для автоматизації. Проте автоматизація має обмеження, зокрема неможливість повного заміщення ручного тестування в оцінці суб'єктивних аспектів. Поєднання ручного та автоматизованого тестування є оптимальним підходом для забезпечення якості веб-додатків.

РОЗДІЛ 2. ТЕХНОЛОГІЯ SELENIUM ЯК ІНСТРУМЕНТ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ

2.1 Історія та еволюція Selenium

Selenium з’явився у 2004 році, коли Джейсон Хаггінс, інженер компанії ThoughtWorks, створив інструмент "JavaScriptTestRunner" для автоматизації тестування внутрішнього веб-додатка, написаного на Python і Plone. Згодом інструмент був перейменований на Selenium Core і став відкритим програмним забезпеченням, що сприяло його швидкому розвитку.

Еволюція Selenium включає кілька ключових етапів:

- Selenium Core (2004). Початкова версія, яка використовувала JavaScript для взаємодії з веб-елементами. Обмеження політики "same origin" ускладнювали тестування на різних доменах.
- Selenium RC (Remote Control, 2006). Ввела серверний компонент для керування браузерами, дозволяючи тестування на різних платформах і мовах програмування.
- Selenium WebDriver (2007). Розроблений Саймоном Стюартом, став основою Selenium 2.0, пропонуючи простіший і ефективніший API.
- Selenium 4 (2020). Додано підтримку Chrome DevTools Protocol (CDP) для розширених функцій, таких як моніторинг продуктивності.

Прийняття стандарту W3C WebDriver забезпечило сумісність із браузерами. У Selenium 4 з’явилася підтримка WebDriver BiDi, що замінила застарілий JSON Wire Protoco. Завдяки активній спільноті Selenium став зрілим інструментом, який продовжує розвиватися.

2.2 Архітектура Selenium: WebDriver, Selenium Grid, IDE

Selenium складається з трьох основних компонентів:

Selenium WebDriver

Selenium WebDriver є основним компонентом фреймворку, який забезпечує прямий програмний доступ до веб-браузерів. Він дозволяє

автоматизувати дії користувача, такі як введення тексту, натискання кнопок або навігація по сторінках. WebDriver підтримує основні браузери, зокрема Google Chrome, Mozilla Firefox, Microsoft Edge і Safari, через відповідні драйвери (наприклад, ChromeDriver, GeckoDriver) [1]. Однією з ключових особливостей WebDriver є підтримка кількох мов програмування, таких як Java, Python, C#, Ruby та JavaScript, що робить його універсальним інструментом для команд із різними технічними стеками [2]. Наприклад, тестувальник може створити сценарій на Python для автоматизації заповнення веб-форми та перевірки результатів.

Selenium Grid

Selenium Grid призначений для паралельного виконання тестів на кількох машинах або віртуальних середовищах. Він складається з центрального вузла (Hub), який координує розподіл тестів, і кількох вузлів (Nodes), які виконують тести в різних комбінаціях браузерів і операційних систем [3]. Це дозволяє значно скоротити час виконання великих тестових наборів і забезпечує кросбраузерну та кросплатформну перевірку. Наприклад, Grid може одночасно запускати тести на Chrome у Windows 10 і Firefox на macOS, що є важливим для забезпечення сумісності веб-додатків [1].

Selenium IDE

Selenium IDE — це плагін для браузерів, який дозволяє записувати та відтворювати взаємодії користувача з веб-інтерфейсом без необхідності написання коду. Цей інструмент підтримується в браузерах Chrome, Firefox і Edge і є особливо корисним для швидкого створення тестових сценаріїв або відтворення помилок [4]. Наприклад, тестувальник може записати послідовність дій, таких як вибір елемента зі списку або введення даних у форму, і зберегти їх як тестовий сценарій. Хоча Selenium IDE менш потужний порівняно з WebDriver, він знижує бар’єр входження для тестувальників-початківців.

Архітектура Selenium забезпечує комплексний підхід до автоматизації тестування, дозволяючи адаптувати інструмент до різних потреб — від швидкого створення тестів до масштабного паралельного тестування.
Selenium підтримує більшість популярних мов програмування та браузерів, що робить його універсальним інструментом.

2.3 Основні можливості Selenium

Selenium WebDriver надає широкий функціонал для автоматизації:

● Навігація. Методи. get(url), navigate (). to(url), refresh(), back(), forward() керують сторінками [13, p. 1].

● Взаємодія з елементами. findElement (By.id("id")), click(), sendKeys ("text") дозволяють працювати з веб-елементами [14, p. 2].

● Очікування. Неявні (implicitlyWait) та явні (WebDriverWait) очікування синхронізують тести з асинхронними діями [15, p. 1].

● Управління вікнами, фреймами, алертами. WebDriver підтримує перемикання між вікнами, фреймами та обробку спливаючих вікон [16, p. 1].

● Робота з куками, скріншотами, проксі. Функції для управління куками, створення скріншотів і налаштування проксі [17, p. 1].
Приклад коду на Python:

```
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

# Ініціалізація драйвера
driver = webdriver.Chrome()

# Навігація до веб-сторінки
driver.get("https://example.com")

# Знаходження елемента та введення тексту
element = driver.find_element(By.ID, "username")
element.send_keys("testuser")
```

```
# Явне очікування для кнопки
submit_button = WebDriverWait(driver, 10).until(
    EC.presence_of_element_located((By.ID, "submit"))
)
submit_button.click()

# Закриття браузера
driver.quit()
```

Цей код демонструє базову автоматизацію веб-додатка.

2.4 Переваги та недоліки Selenium

Selenium є одним із найпоширеніших інструментів автоматизації завдяки своїм численним перевагам, але також має певні обмеження, які слід враховувати при виборі фреймворку для тестування.

Розкриємо переваги Selenium серед найпоширеніших інструментів автоматизації:

1. Відкритий код. Selenium є безкоштовним інструментом із відкритим кодом, що знижує витрати на тестування та сприяє його широкому застосуванню.
2. Кросбраузерна підтримка. Інструмент підтримує тестування на всіх основних браузерах, що забезпечує перевірку сумісності веб-додатків.
3. Мультиязикова підтримка. Selenium дозволяє створювати тести на різних мовах програмування, що робить його доступним для команд із різними технічними компетенціями.
4. Активна спільнота. Велика спільнота розробників і тестувальників забезпечує регулярні оновлення, документацію та підтримку через форуми та ресурси.
5. Інтеграція з іншими інструментами. Selenium легко інтегрується з фреймворками, такими як TestNG, JUnit, і системами CI/CD, що полегшує автоматизацію тестування.

6. Паралельне виконання. Завдяки Selenium Grid тести можуть виконуватися паралельно, що значно скорочує час тестування.
7. Кросплатформність. Selenium працює на Windows, macOS, Linux і підтримує мобільні платформи через інтеграцію з Appium.

Недоліки включать у себе:

1. Обмежена підтримка мобільних додатків. Selenium не підтримує тестування нативних мобільних додатків без використання додаткових інструментів, таких як Appium.
2. Складність налаштування. Налаштування Selenium Grid і драйверів для різних браузерів може бути складним і вимагати значних зусиль.
3. Вимоги до програмування. Створення надійних тестів потребує знань програмування, що може бути викликом для тестувальників без технічної підготовки.
4. Бриткість тестів. Зміни в інтерфейсі веб-додатків можуть призводити до нестабільності тестів, що вимагає їх частого оновлення.
5. Повільність виконання. Для великих тестових наборів Selenium може бути повільнішим порівняно з іншими інструментами, такими як Cypress або Playwright.
6. Обмежена звітність. Вбудовані можливості звітності є базовими, і для створення детальних звітів необхідні сторонні інструменти, такі як Allure.

Незважаючи на згадані недоліки, переваги Selenium (відкритий код, багатомовність, гнучкість інтеграції) залишають його провідним інструментом для автоматизації веб-тестування в багатьох проєктах. Особливо це стосується корпоративних рішень, де критично важливо забезпечити широку підтримку браузерів та зберігати можливість масштабування тестового середовища.

2.5 Інтеграція Selenium з іншими інструментами (TestNG, CI/CD, Allure)

Інтеграція Selenium з іншими інструментами значно розширює його функціональність, дозволяючи оптимізувати процеси тестування, автоматизації та звітності.

Інтеграція з TestNG

TestNG є популярним фреймворком для організації та виконання тестів, який широко використовується разом із Selenium. Основні переваги інтеграції включають:

- **Структурування тестів**, що дозволяють групувати тести за функціональністю, пріоритетом або іншими критеріями, що полегшує їх керування.
- **Паралельне виконання** для підтримки паралельного запуску тестів скорочує час виконання великих тестових наборів.
- **Параметризація**, що дозволяє передавати параметри до тестів, що є корисним для даних-орієнтованого тестування.
- **Звітність**, що допомогає генерувати детальні HTML-звіти про результати виконання тестів, включаючи статистику успішних і невдалих тестів.

Приклад коду для інтеграції Selenium із TestNG:

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.Test;

public class SeleniumTestNGExample {
   @Test
   public void testGoogleSearch() {
      WebDriver driver = new ChromeDriver();
      driver.get("https://www.google.com");
      driver.quit();
   }
}
```

Інтеграція з CI/CD

Selenium легко інтегрується з системами континуальної інтеграції та розгортання, такими як Jenkins, GitLab CI/CD і GitHub Actions. Це дозволяє автоматизувати запуск тестів після кожного коміту коду, що сприяє ранньому виявленню помилок і забезпечує швидку доставку якісного програмного забезпечення [9]. Наприклад, у Jenkins можна налаштувати пайплайн, який запускає Selenium-тести після кожного пушу в репозиторій.

Приклад конфігурації Jenkins для виконання тестів:

```
pipeline {
    agent any
    stages {
        stage('Test') {
            steps {
                sh 'mvn test'
            }
        }
    }
}
```

Інтеграція з Allure

Allure — це інструмент для створення інтерактивних і деталізованих звітів про виконання тестів. Інтеграція Selenium із Allure через TestNG або JUnit дозволяє генерувати звіти, які включають скріншоти, логи та детальну статистику. Allure підтримує анотації, такі як @Epic, @Feature, які допомагають структурувати звіти та полегшують аналіз результатів [10].

Приклад інтеграції Allure з Selenium і TestNG:

```
import io.qameta.allure.Description;
import io.qameta.allure.Severity;
import io.qameta.allure.SeverityLevel;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.Test;

public class AllureSeleniumTest {
    @Test
    @Description("Тестування пошуку Google")
    @Severity(SeverityLevel.NORMAL)
    public void testGoogleSearch() {
        WebDriver driver = new ChromeDriver();
```

```
        driver.get("https://www.google.com");
        driver.quit();
    }
}
```

Ці інтеграції значно підвищують ефективність Selenium, дозволяючи створювати структуровані, автоматизовані та прозорі процеси тестування.

2.6 Порівняння з іншими фреймворками (Cypress, Playwright, Puppeteer)

Selenium є лідером у сфері автоматизації тестування, але конкуренти, такі як Cypress, Playwright і Puppeteer, пропонують альтернативні підходи, які можуть бути кращими в певних сценаріях.

Selenium

- Переваги: Підтримка кількох мов програмування, кросбраузерна сумісність, велика спільнота, інтеграція з CI/CD і Appium для мобільного тестування [1].
- Обмеження: Складність налаштування, бриткість тестів, повільність для великих тестових наборів [6, с. 3].

Cypress

- Переваги: Швидке виконання, автоматичне чекання елементів, інтуїтивний API, ідеально для фронтенд-тестування в JavaScript-проектах [11].
- Обмеження: Обмежена підтримка браузерів (Chrome, Electron), підтримує лише JavaScript/TypeScript, не підходить для мобільних додатків [11].

Playwright

- Переваги: Підтримка кількох браузерів, швидке виконання, автоматичне чекання елементів, сучасний API, підтримка кількох мов [12].
- Обмеження: Менша спільнота порівняно з Selenium, інструмент ще розвивається [12].

Puppeteer

- Переваги: Висока швидкість, високорівневий API, ідеально для Chrome/Chromium, підходить для вебскрейпінгу та генерації PDF [11].
- Обмеження: Обмежена підтримка браузерів (лише Chrome/Chromium), більше орієнтований на автоматизацію, ніж на тестування [11].

Таблиця 1: Порівняння Selenium, Cypress, Playwright і Puppeteer

Фреймворк	Мови програмування	Підтримка браузерів	Швидкість	Основне використання
Selenium	Java, Python, C#, Ruby, JavaScript	Chrome, Firefox, Safari, Edge	Повільніше	Кросбраузерне тестування, CI/CD
Cypress	JavaScript, TypeScript	Chrome, Electron	Швидше	Фронтенд-тестування
Playwright	JavaScript, Python, Java, C#	Chrome, Firefox, Safari, Edge	Швидше	Кросбраузерне тестування
Puppeteer	JavaScript	Chrome, Chromium	Швидше	Вебскрейпінг, автоматизація

Порівняння за ключовими аспектами

- Використання: Selenium є універсальним рішенням для кросбраузерного тестування, тоді як Cypress оптимальний для JavaScript-проектів, Playwright пропонує сучасний підхід, а Puppeteer — для специфічних завдань автоматизації в Chrome.
- Швидкість: Cypress і Playwright швидші завдяки автоматичному чеканню елементів, тоді як Selenium може бути повільнішим через необхідність явного очікування.
- Легкість використання: Cypress і Playwright мають більш інтуїтивні API, тоді як Selenium вимагає додаткових зусиль для налаштування.
- Підтримка мов: Selenium підтримує найбільше мов, за ним йде Playwright, тоді як Cypress і Puppeteer обмежені JavaScript.

Висновки до розділу 2

Selenium є потужним і універсальним інструментом для автоматизації тестування веб-додатків, який завдяки своїй архітектурі (WebDriver, Grid, IDE) підтримує широкий спектр сценаріїв тестування. Його переваги, такі як безкоштовність, кросбраузерна підтримка та інтеграція з іншими інструментами, роблять його популярним вибором для великих команд і проектів із різноманітними технічними вимогами [1, 2]. Інтеграція з TestNG, CI/CD і Allure дозволяє створювати структуровані, автоматизовані та прозорі процеси тестування, що підвищує ефективність розробки програмного забезпечення.

Однак Selenium має обмеження, зокрема складність налаштування, бриткість тестів і повільність для великих тестових наборів. У порівнянні з Cypress, Playwright і Puppeteer, Selenium є більш універсальним, але поступається в швидкості та простоті використання. Вибір між цими інструментами залежить від специфіки проекту: Selenium оптимальний для кросбраузерного тестування, тоді як Cypress і Playwright краще підходять для швидкого тестування фронтенду, а Puppeteer — для автоматизації в Chrome.

Таким чином, Selenium залишається важливим інструментом у сфері автоматизації тестування, але його ефективність залежить від правильного налаштування, вибору інтеграцій і врахування потреб проекту. Для оптимального використання необхідно ретельно оцінити вимоги до тестування, ресурси команди та бажаний рівень автоматизації.

РОЗДІЛ 3 МЕТОДИКА ТА РЕАЛІЗАЦІЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ВЕБ САЙТУ З ВИКОРИСТАННЯМ SELENIUM

3.1 Підготовка до тестування

Тестований веб-додаток. Для практичної реалізації автоматизованого тестування було обрано веб-додаток "The Internet" з відкритим вихідним кодом, який є тестовим середовищем для перевірки функціональності електронної комерції. Основні сторінки, що тестувалися, включають:

- Головна сторінка: Відображення каталогу товарів.
- Сторінка входу (Login): Форма для аутентифікації користувача.
- Сторінка реєстрації (Register): Форма для створення нового облікового запису.
- Сторінка профілю (My Account): Інтерфейс для редагування даних користувача.

Перевірялися функціональні аспекти, такі як коректність входу в систему, реєстрація користувача, зміна даних профілю та обробка помилок при введенні невалідних даних.

Використана мова та бібліотеки. Для автоматизації тестування використано мову програмування Python у поєднанні з бібліотекою Selenium WebDriver. Додатково застосовувалися:

- pytest: Для організації та виконання тестів.
- webdriver-manager: Для автоматичного керування драйверами браузерів.
- allure-pytest: Для створення деталізованих звітів про результати тестування.

Встановлення середовища

Налаштування тестового середовища включало такі кроки:

1. Встановлення Python. Використано Python версії 3.9, завантажений із офіційного сайту (https://www.python.org/).

2. Встановлення бібліотек. За допомогою менеджера пакетів pip встановлено бібліотеки: *pip install selenium pytest webdriver-manager allure-pytest*
3. Налаштування браузера та драйвера. Для тестування використано браузер Google Chrome. Драйвер ChromeDriver автоматично завантажувався за допомогою webdriver-manager, що усуває необхідність ручного оновлення драйверів.
4. Перевірка середовища: Виконано тестовий запуск простого скрипту для підтвердження коректності налаштувань.

Структура проєкту. Проєкт організовано за принципами модульності для забезпечення зручності підтримки та розширення:

- Папка tests. Містить тестові скрипти, наприклад, test_login.py, test_register.py, які описують сценарії тестування.
- Папка pages. Зберігає класи Page Object для кожної сторінки (наприклад, LoginPage, RegisterPage), що відповідають за взаємодію з елементами сторінки.
- Папка utils. Містить допоміжні утиліти, наприклад, функції для роботи з конфігураційними файлами чи генерації тестових даних.
- Папка reports. Зберігає звіти про виконання тестів, зокрема файли Allure Report.

Приклад коду для запуску браузера. Нижче наведено код для ініціалізації браузера з використанням Selenium:

```
from selenium import webdriver
from selenium.webdriver.chrome.service
import Service
from webdriver_manager.chrome import
 ChromeDriverManager


class Driver:
    def __init__(self):
        options =
 webdriver.ChromeOptions()

        service = Service(executable_path=
ChromeDriverManager().install())
        self.driver = webdriver.Chrome(
service=service, options=options)

    def get_driver(self):
        return self.driver

    def close(self):
        self.driver.quit()
```

Цей код ініціалізує браузер Chrome, максимізує вікно та відкриває головну сторінку тестового сайту.

3.2 Вибір сценаріїв тестування

Об’єкти тестування. Для тестування обрано такі сценарії:

- Вхід у систему (логін). Перевірка коректності аутентифікації з валідними та невалідними даними.
- Робота з випадаючим списком.
- Робота з JavaScript-алертами.

- Перевірка повідомлення про помилку: Перевірка відображення повідомлень про помилки при некоректному введенні даних.

Обґрунтування вибору сценаріїв. Ці сценарії демонструють різні типи взаємодії з UI: введення даних, натискання кнопок, очікування, роботу з алертами. Вони є хорошою основою для оцінки стабільності та коректності автотестів.

3.3 Реалізація тестів

Усі тести структуровано згідно з POM. Наприклад, для логіну створено клас LoginPage, який містить метод login(username, password), а також get_message() для перевірки результату. Аналогічно реалізовано класи AlertsPage, DropdownPage.

Приклад позитивного тесту логіну:

Приклад тесту prompt-алерта:

```python
from pages.alerts_page import AlertsPage
import time


def test_js_alert(driver):
    alerts = AlertsPage(driver)
    alerts.open()
    alerts.trigger_alert()
    assert alerts.get_result() == "You successfully clicked an alert"


def test_js_confirm_accept(driver):
    alerts = AlertsPage(driver)
    alerts.open()
    alerts.trigger_confirm_accept()
    assert alerts.get_result() == "You clicked: Ok"


def test_js_confirm_dismiss(driver):
    alerts = AlertsPage(driver)
    alerts.open()
    alerts.trigger_confirm_dismiss()
    assert alerts.get_result() == "You clicked: Cancel"


def test_js_prompt_enter_text(driver):
    alerts = AlertsPage(driver)
    alerts.open()
    alerts.trigger_prompt_enter_text("Дмитро")
    assert alerts.get_result() == "You entered: "


def test_js_prompt_cancel(driver):
    alerts = AlertsPage(driver)
    alerts.open()
    alerts.trigger_prompt_cancel()
    assert alerts.get_result() == "You entered: null"
```

Код за патерном Page Object

Приклад класу LoginPage:

```
from pages.base_page import BasePage
from selenium.webdriver.common.by import
 By


class LoginPage(BasePage):
    URL = "http://localhost:8080/login"
    # Локатори
    USERNAME_FIELD = (By.XPATH,
"//input[@id='username']")
    PASSWORD_FIELD = (By.XPATH,
"//input[@id='password']")
    LOGIN_BUTTON = (By.XPATH,
"//button[@class='radius']")
    ERROR_FIELD = (By.XPATH,
"//div[@id='flash']")

    def open(self):
        self.driver.get(self.URL)

    def login(self, email, password):
        self.type(*self.USERNAME_FIELD,
text=email)
        self.type(*self.PASSWORD_FIELD,
text=password)
        self.click(*self.LOGIN_BUTTON)

    def get_flash(self):
        return self.get_text(*self
.ERROR_FIELD)
```

Виконання тестів

Тести включають такі кроки:

1. Відкриття сторінки входу.
2. Введення даних (валідних або невалідних).
3. Натиснення кнопки "Log in".
4. Перевірка результату (успішний вхід, повідомлення про помилку).

3.4 Запуск тестів і результати

Запуск тестів. Тести запускалися за допомогою pytest через консоль: pytest tests/ --alluredir=reports/allure-results

Після запуску команда allure serve дозволяє переглянути результати в інтерактивному звіті. Усі позитивні сценарії (логін, випадаючий список, алерти) були виконані успішно. У випадку помилки — звіт Allure додає скріншоти та логи для аналізу. Результати тестів підсумовано в таблиці:

Тест	**Статус**	**Тривалість (с)**	**Скріншот при помилці**
Успішний вхід	Пройдено	2.5	-
Невірний пароль	Пройдено	2.7	-
Перевірка JavaScript-алерт	Пройдено	3.1	-
Перевірка dropdown меню	Помилка	3.0	-

Звіт Allure. Звіт Allure містив детальну інформацію про кожен тест, включаючи статус, тривалість, логи та скріншоти помилок.

3.5 Інтеграція з CI/CD

Автоматичний запуск тестів. Фреймворк було інтегровано з GitHub Actions, що дозволило реалізувати автоматичний запуск тестів після кожного коміту в репозиторій. У конфігураційному файлі tests.yml налаштовано встановлення Python та залежностей, запуск тестів і збереження Allure-звіту. Такий підхід забезпечує безперервну інтеграцію, підвищує стабільність коду та забезпечує раннє виявлення дефектів.

Приклад конфігурації GitHub Actions

Файл .github/workflows/tests.yml:

```
name: Run Tests
on: [push]
jobs:
    test:
        runs-on: ubuntu-latest
        steps:
            - uses: actions/checkout@v3
            - name: Set up Python
              uses:
actions/setup-python@v4
              with:
                  python-version: '3.9'
            - name: Install dependencies
              run:
pip install -r requirements.txt
            - name: Run tests
              run:
pytest tests/ --alluredir=reports/allure-r
esults
            - name: Upload Allure report
              uses:
actions/upload-artifact@v3
              with:
                  name: allure-results
                  path:
reports/allure-results
```

Висновки до розділу 3

У рамках цього розділі реалізовано сучасний фреймворк для автоматизованого тестування сайту "The Internet" з відкритим кодом. Завдяки використанню Selenium, Pytest, структури Page Object Model та інтеграції з Allure і GitHub Actions вдалося побудувати гнучку, масштабовану й зручну у використанні систему. Проведені тести охоплюють основні функції взаємодії з інтерфейсом користувача, демонструючи ефективність автоматизованого підходу до контролю якості веб-застосунків. Основні труднощі включали налаштування драйверів, вибір стабільних локаторів і керування неявними очікуваннями. Selenium виявився зручним інструментом завдяки гнучкості, підтримці різних браузерів і широкій документації. У майбутньому можна розширити сценарії тестування, додавши перевірку інших сторінок, а також оптимізувати тести шляхом використання явних очікувань і паралельного виконання.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи було досліджено процес автоматизації тестування веб-сайтів за допомогою технології Selenium.

У першому розділі проведено теоретичний аналіз основ тестування програмного забезпечення, наведено класифікацію його видів, розглянуто переваги автоматизації та проаналізовано сучасні інструменти для автоматизованого тестування, зокрема Selenium, Cypress, Playwright та інші. Було обґрунтовано вибір Selenium як найбільш універсального та популярного фреймворку для перевірки веб-додатків.

У другому розділі детально описано архітектуру Selenium WebDriver, його можливості, підтримувані мови програмування та браузери. Показано, як реалізуються ключові функції взаємодії з веб-елементами, навігація, очікування та інші важливі аспекти автоматизації. Наведено порівняльну характеристику Selenium з альтернативними інструментами.

У третьому розділі реалізовано практичну частину: створено тестовий проєкт, налаштовано середовище, розроблено автоматизовані сценарії перевірки функціоналу веб-сайту (логін, реєстрація, перевірка помилок), проведено запуск тестів та проаналізовано результати. Було використано принцип Page Object Model для структурування тестів та Allure — для візуалізації звітності.

Отже, мету дослідження досягнуто. Selenium підтвердив свою ефективність як інструмент для автоматизації тестування веб-додатків, а реалізований набір тестів може бути корисним прикладом для подальшого розвитку системи автоматичного контролю якості в реальних проєктах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. GeeksforGeeks. Software Testing Tutorial [Електронний ресурс]. Режим доступу: https://www.geeksforgeeks.org/software-testing-tutorial/
2. International Software Testing Qualifications Board (ISTQB). Standard Glossary of Terms used in Software Testing. Version 3.2, 2018. [Електронний ресурс]. Режим доступу: https://glossary.istqb.org/
3. SeleniumHQ. The Selenium Browser Automation Project [Електронний ресурс]. Режим доступу: https://www.selenium.dev/
4. Cypress Documentation [Електронний ресурс]. Режим доступу: https://docs.cypress.io/
5. Apache JMeter User’s Manual [Електронний ресурс]. Режим доступу: https://jmeter.apache.org/usermanual/index.html
6. Postman Learning Center [Електронний ресурс]. Режим доступу: https://learning.postman.com/
7. GitHub Actions Documentation [Електронний ресурс]. Режим доступу: https://docs.github.com/en/actions
8. Allure Framework Documentation [Електронний ресурс]. Режим доступу: https://docs.qameta.io/allure/
9. Selenium WebDriver with Python Documentation [Електронний ресурс]. Режим доступу: https://selenium-python.readthedocs.io/
10. Software Testing Help. Selenium WebDriver Tutorial [Електронний ресурс]. Режим доступу: https://www.softwaretestinghelp.com/selenium-tutorial-1/
11. Microsoft. Playwright Testing Framework [Електронний ресурс]. Режим доступу: https://playwright.dev/
12. Puppeteer Documentation – Tools for Headless Chrome [Електронний ресурс]. Режим доступу: https://pptr.dev/

13. OpenJS Foundation. WebDriver API W3C Specification [Електронний ресурс]. Режим доступу: https://www.w3.org/TR/webdriver2/
14. Python Software Foundation. Official Python Documentation [Електронний ресурс]. Режим доступу: https://docs.python.org/3/
15. Pytest Framework Documentation [Електронний ресурс]. Режим доступу: https://docs.pytest.org/
16. Docker Documentation [Електронний ресурс]. Режим доступу: https://docs.docker.com/
17. Jenkins User Documentation [Електронний ресурс]. Режим доступу: https://www.jenkins.io/doc/
18. Selenium Blog. What’s New in Selenium 4 [Електронний ресурс]. Режим доступу: https://www.selenium.dev/blog/

ДОДАТКИ

ДОДАТОК А

ЛІСТИНГИ КОДУ РЕАЛІЗАЦІЇ ТЕСТІВ

Фрагменти коду тестів

```python
from pages.login_page import LoginPage
import time


def test_valid_login(driver):
    login = LoginPage(driver)
    login.open()
    login.login("tomsmith", "SuperSecretPassword!")
    time.sleep(10)
    assert "Ви успішно увійшли!" in login.get_flash()


def test_invalid_login(driver):
    login = LoginPage(driver)
    login.open()
    login.login("wrong", "wrongpass")
    assert "Ваш юзернейм невірний" in login.get_flash()
```

Рис. А.1 Приклад тесту логіну

```python
from pages.alerts_page import AlertsPage
import time


def test_js_alert(driver):
    alerts = AlertsPage(driver)
    alerts.open()
    alerts.trigger_alert()
    assert alerts.get_result() == "You successfully clicked an alert"


def test_js_confirm_accept(driver):
    alerts = AlertsPage(driver)
    alerts.open()
    alerts.trigger_confirm_accept()
    assert alerts.get_result() == "You clicked: Ok"


def test_js_confirm_dismiss(driver):
    alerts = AlertsPage(driver)
    alerts.open()
    alerts.trigger_confirm_dismiss()
    assert alerts.get_result() == "You clicked: Cancel"


def test_js_prompt_enter_text(driver):
    alerts = AlertsPage(driver)
    alerts.open()
    alerts.trigger_prompt_enter_text("Дмитро")
    assert alerts.get_result() == "You entered: "


def test_js_prompt_cancel(driver):
    alerts = AlertsPage(driver)
    alerts.open()
    alerts.trigger_prompt_cancel()
    assert alerts.get_result() == "You entered: null"
```

Рис. А.2 Приклад тесту промпт-алерту

```
from pages.base_page import BasePage
from selenium.webdriver.common.by import
 By


class LoginPage(BasePage):
    URL = "http://localhost:8080/login"
    # Локатори
    USERNAME_FIELD = (By.XPATH,
"//input[@id='username']")
    PASSWORD_FIELD = (By.XPATH,
"//input[@id='password']")
    LOGIN_BUTTON = (By.XPATH,
"//button[@class='radius']")
    ERROR_FIELD = (By.XPATH,
"//div[@id='flash']")

    def open(self):
        self.driver.get(self.URL)

    def login(self, email, password):
        self.type(*self.USERNAME_FIELD,
text=email)
        self.type(*self.PASSWORD_FIELD,
text=password)
        self.click(*self.LOGIN_BUTTON)

    def get_flash(self):
        return self.get_text(*self
.ERROR_FIELD)
```

Рис. А.3 Приклад класу LoginPage

ДОДАТОК Б

ПРИЛАД КОНФІГУРАЦІЇ GITHUB ACTIONS

```yaml
name: Run Tests
on: [push]
jobs:
    test:
        runs-on: ubuntu-latest
        steps:
            - uses: actions/checkout@v3
            - name: Set up Python
              uses:
actions/setup-python@v4
              with:
                  python-version: '3.9'
            - name: Install dependencies
              run:
pip install -r requirements.txt
            - name: Run tests
              run:
pytest tests/ --alluredir=reports/allure-r
esults
            - name: Upload Allure report
              uses:
actions/upload-artifact@v3
              with:
                  name: allure-results
                  path:
reports/allure-results
```

Рис. Б.1 Конфігурація GitHub Actions