Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: «Розробка фреймворку для інтеграційного тестування сайту JetIQ з використанням технології Selenium WebDriver»

Виконав: студент IV курсу
групи 2ПІ–18б
спеціальності
121 – «Інженерія програмного забезпечення»
(шифр і назва напряму підготовки, спеціальності)

Симон А. Д.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Романюк О. В.
(прізвище та ініціали)

Рецензент: к.т.н., доц. каф. КН Колодний В.В.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри _____Романюк О.Н.
«____» _____ 2022 р.

ВНТУ – 2022

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., проф. О. Н. Романюк
_____
25 березня 2022 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Симону Андрію Дмитровичу

1. Тема роботи: «Розробка фреймворку для інтеграційного тестування сайту JetIQ з використанням технології Selenium WebDriver».

Керівник роботи: к.т.н., доц. кафедри ПЗ Романюк О. В. затверджені наказом вищого навчального закладу від 24 березня 2022 року № 66.

2. Термін подання студентом роботи 13 червня 2022 року.

3. Вихідні дані до роботи: метод написання тестових сценаріїв – технологія Cucumber; алгоритми реалізації – технологія Selenium WebDriver; вхідні дані – тестові параметри; вихідні дані – результати тестування, логи, скріншоти; середовище розробки – IntelliJ IDEA; мова розробки – Java; операційна система – Windows 10 та Linux; Git система – GitLab; CI/CD пайплайн – GitLab.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз галузі автоматизованого інтеграційного тестування; розробка тестових сценаріїв, матриці трасування, архітектури та CI/CD схеми фреймворку; розробка програмних компонент для автоматизованого тестового фреймворку; інструкція користувача та проведення тестування сайту JetIQ; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: титульний слайд; актуальність теми; мета, об'єкт та предмет дослідження; задачі дослідження; новизна одержаних результатів; практична цінність одержаних результатів; Selenium WebDriver; вимоги, які необхідно протестувати; список тест-кейсів; матриця трасування; архітектура фреймворку; алгоритм роботи тестового сценарію; блок-схема CI/CD пайплайну; застосування фреймворку: тестування сайту JetIQ локально;

застосування фреймворку: тестування сайту JetIQ за допомогою CI/CD; апробація та публікації результатів роботи; фінальний слайд.

6. Консультанти розділів бакалаврської дипломної роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1–4	к.т.н Романюк О. В., доц. кафедри ПЗ	25.03.2022	10.06.2022

7. Дата видачі завдання 25 березня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз галузі автоматизованого інтеграційного тестування	26.03.2022 – 08.04.2022	Виконано
2	Розробка тестових сценаріїв, матриці трасування та архітектури фреймворку	09.04.2022 – 20.04.2022	Виконано
3	Розробка алгоритму роботи тестового сценарію та CI/CD схеми фреймворку	21.04.2022 – 30.04.2022	Виконано
4	Розробка програмних компонент для автоматизованого тестового фреймворку	01.05.2022 – 19.05.2022	Виконано
5	Розробка інструкції користувача та проведення тестування сайту JetIQ	20.05.2022 – 29.05.2022	Виконано
6	Оформлення матеріалів до захисту БДР	30.05.2022 – 10.06.2022	Виконано

Студент _____ **Симон А. Д.**
(підпис) (прізвище та ініціали)

Керівник бакалаврської дипломної роботи _____ **Романюк О. В.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 109 сторінок формату А4, на яких є 37 рисунків, 8 таблиць, список використаних джерел містить 29 найменувань.

У бакалаврській дипломній роботі проведено детальний аналіз технологій автоматизованого тестування, розглянуто особливості їх застосування при розробці програмного забезпечення. Встановлено об’єкт, предмет, завдання та методи дослідження. Сформульовано мету дослідження – підвищення ефективності тестування сайту JetIQ шляхом створення фреймворку для його інтеграційного тестування. Для реалізації мети було розроблено вимоги, які необхідно протестувати, тест-кейси, матрицю трасування, архітектуру фреймворку, алгоритм роботи фреймворку, програмний продукт, CI/CD пайплайн.

Запропоновано архітектуру тестового фреймворку для інтеграційного тестування, яка дала можливість спростити процес налаштування та запуску тестових сценаріїв і підвищити якість тестування. Удосконалено алгоритм роботи тестового сценарію, який дозволив спростити процес написання тестів, скоротити час проходження тестів та зменшити обсяги задіяної пам’яті. Розроблено CI/CD пайплайн для віддаленого запуску тестів на віртуальній ізольованій машині.

Створено програмний додаток для інтеграційного тестування сайту JetIQ з використанням мови програмування Java, Selenium WebDriver та Cucumber технологій, середовища розробки IntelliJ IDEA, Git та CI/CD, пайплайн – GitLab. В результаті виконання бакалаврської дипломної роботи розроблено тестовий фреймворк, за допомогою якого було проведено інтеграційне тестування сайту JetIQ як локально, так і на віддаленій машині.

Отримані в бакалаврській дипломній роботі результати можна використати для автоматизованого тестування сайту JetIQ.

Ключові слова: тестування, BDD методологія, тестовий фреймворк, CI/CD.

ABSTRACT

The Bachelor's thesis consists of 109 A4 pages, which have 37 Figures, 8 tables, the list of sources used contains 29 titles.

In The Bachelor's thesis, a detailed analysis of Automated Testing Technologies was carried out, and the features of their application in software development were considered. The object, subject, tasks and methods of research are established. The aim of the study is to improve the efficiency of testing the JetIQ site by creating a test framework for its integration testing. To achieve this goal, we developed requirements that need to be tested, test cases, a trace Matrix, a framework architecture, a framework algorithm, a software product, and a CI/CD pipeline.

We propose a test framework architecture for integration testing, which made it possible to simplify the process of configuring and running test scenarios and improve the quality of testing. The algorithm of the test script has been improved, which made it possible to simplify the process of writing tests, reduce the time required to pass tests, and reduce the amount of memory used. Developed CI / CD pipeline for remote testing on an isolated virtual machine.

Created a software application for integration testing of the JetIQ site using the programming language Java, Selenium WebDriver and Cucumber technologies, IntelliJ IDEA development environment, Git and CI/CD, pipeline – GitLab. As a result of completing the Bachelor's thesis, a test framework was developed, which was used to conduct integration testing of the JetIQ site both locally and on a remote machine.

The results obtained in The Bachelor's thesis can be used for automated testing of the JetIQ website.

Keywords: testing, BDD methodology, test framework, CI/CD.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сьогодні неможливо уявити розробку будь-якого програмного продукту без якісного та глибокого тестування. Тестування – це процес технічного дослідження, призначений для виявлення інформації про якість продукту відносно контексту, в якому його мають використовувати [1, 2].

Основний підхід тестування полягає в оцінці та максимізації значущості всіх етапів життєвого циклу розроблення програмного забезпечення для досягнення необхідного рівня якості, продуктивності, доступності [3]. Саме тому розробники приділяють велику увагу перевірці працездатності веб-додатків як в локальному середовищі, так і за його межами.

Раніше все це робилося буквально вручну: десятки людей тестували сотні сценаріїв у всіх існуючих браузерах, виявляючи неполадки, і намагались визначити причини їх виникнення [4, 5]. Така робота вимагала багато часу та кваліфікованих працівників, що дуже негативно впливало на бізнес [6, 7].

Особливо ця проблема стосується інтеграційного тестування – стадії тестування ПЗ, при якій окремі модулі програми комбінуються та тестуються разом у взаємодії [8, 9]. Метою інтеграційного тестування є верифікувати вимоги з функціональності, продуктивності, надійності до основних компонентів програми. Одним із видів інтеграційного тестування є UI-тестування, яке вручну проводити не завжди зручно та доцільно [10].

Таким чином виникла потреба у автоматизації тестових сценаріїв. Автоматизація дозволяє суттєво скоротити час, затрачений на стадію тестування, що є вигідно для бізнесу і якості продукт. Проте такий підхід потребує ще більш кваліфікованих працівників і несе за собою використання відповідних інструментів та технологій для автоматизованого тестування [11].

Тому розробка фреймворку для інтеграційного тестування сайтів, зокрема сайту JetIQ, який є зрозумілим і простим у використанні та дозволяє розробляти автоматизовані тестові сценарії, є досить актуальною задачею.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета і завдання дослідження. Метою даної роботи є підвищення ефективності тестування сайту JetIQ шляхом створення тестового фреймворку для його інтеграційного тестування.

Відповідно до поставленої мети в бакалаврській дипломній роботі потрібно вирішити такі **задачі дослідження:**

– проаналізувати сучасний стан розвитку технологій для автоматизації інтеграційного та UI-тестування;

– проаналізувати існуючі технології автоматизації тестів для розробки фреймворку;

- розробити перелік вимог, які необхідно протестувати;
- розробити тестові сценарії;
- розробити матрицю трасування;
- розробити гнучку архітектуру фреймворку згідно з принципами ООП;
- розробити архітектурні компоненти фреймворку;
- розробити алгоритм роботи тестового сценарію;
- розробити CI/CD схему;
- провести інтеграційне та UI-тестування сайту JetIQ;
- розробити інструкцію користувача.

Об’єкт дослідження – процес автоматизованого інтеграційного та UI-тестування.

Предмет дослідження – методи та засоби для інтеграційного автоматизованого тестування сайту JetIQ.

Методи дослідження. У бакалаврській дипломній роботі використовувалися: загальна теорія тестування програмного забезпечення для створення тест-кейсів та матриці трасування, теорія алгоритмів для розробки та вдосконалення алгоритмів; комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Новизна отриманих результатів.

1. Удосконалено архітектуру тестового фреймворку для інтеграційного тестування, у якій на відміну від традиційної, веб-драйвер вбудовано у проект і забезпечена можливість проведення тестування у декількох браузерах, що дало можливість спростити процес налаштування та запуску тестових сценаріїв і підвищити якість тестування.

2. Удосконалено алгоритм роботи тестового сценарію, який на відміну від оригінального алгоритму, використовує можливості технології Cucumber, що дозволило писати тести українською мовою і спростити процес написання тестів; та паттерн PageObject і хеш мапу ScenarioContext, що дало можливість скоротити час проходження тестів та зменшити обсяги пам'яті, необхідні для зберігання тестових даних.

Практична цінність одержаних результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі теоретичних положень запопоновано алгоритм та розроблено програмні засоби для тестового фреймворку, за допомогою якого можна проводити інтеграційне тестування сайту JetIQ.

Особистий внесок здобувача. Усі наукові результати отримано здобувачем самостійно. У працях, опублікованих у співавторстві, автору належать: особливості застосування Selenium технології при розробці фреймворку для автоматизованого UI-тестування [12], використання технології WebDriver для розробки тестового автоматизованого фреймворку [13].

Апробація матеріалів бакалаврської дипломної роботи. Основні матеріали бакалаврської кваліфікаційної роботи доповідались на:

– L Науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії (2022 р., м. Вінниця);

– LI Науково-технічна конференція підрозділів Вінницького національного технічного університету (2022 р., м. Вінниця).

Публікації. За тематикою дослідження опубліковано 2 наукових праці у збірниках матеріалів конференцій.

1 АНАЛІЗ ГАЛУЗІ АВТОМАТИЗОВАНОГО ІНТЕГРАЦІЙНОГО ТЕСТУВАННЯ

1.1 Аналіз технологій автоматизації тестування

В процесі створення програмних продуктів нерідкі помилки і дефекти – це цілком очікуване і нормальне явище, а в умовах обмежених тимчасових ресурсів і високих вимог до якості програмних продуктів неминуче виникає необхідність в організації ефективного контролю і управління всім процесом тестування. Контроль якості програмного забезпечення неможливий сьогодні без автоматизації всіх задач тестування [1].

Ручне тестування є витратним за часом, трудомістким і часто монотонним процесом. Воно приводить до виникнення проблем, особливо при обмежених ресурсах і жорстких термінах. Якщо вам потрібно поліпшити тестування додатків для перевірки коректності їх роботи, важливо рухатися у бік автоматизації всіх ручних задач тестування [2].

Інструментальні засоби автоматизації записують взаємодію користувачів з програмним продуктом, а сформовані на цій основі сценарії використовуються для подальших тестів, тобто автоматизація тестування дозволяє оптимізувати якість складних програмних продуктів ефективним по вартості способом за прийнятний час. Це допомагає швидше випустити програмне забезпечення більш високої якості.

Процес автоматизації тестування ділиться на три етапи [3]:

– Запис. Сценарій тестування може включати точки верифікації (verification points) для перевірки відповіді системи і зробити сценарії тестування залежними від даних, щоб виконувати один і той же сценарій з різними наборами вхідних даних.

– Поліпшення шляхом додавання коду з різноманітними функціями Типові зміни сценаріїв тестування – умовне галуження, рефакторинг і обробка виняткових ситуацій.

– Відтворення. Виконання сценаріїв, що емалюють дії, які виконував

користувач додатку при записі тесту. Розбіжності реєструються, і тестувальник може зробити висновок про те, чи добре функціонує система або регресійне тестування виявило проблеми.

Перші спроби «автоматизації» з'явилися в епоху операційних систем DOS і CP/M. Тоді вона полягала у видачі додатком команд через командний рядок і аналізі результатів. Трохи пізніше додалися віддалені виклики через API для роботи з мережі. Вперше про автоматизоване тестування згадується в книзі Фредеріка Брукса «Міфічний людино-місяць», де йдеться про перспективи використання модульного тестування. Але по-справжньому автоматизація тестування стала розвиватися тільки в 1980-х роках [4].

Існує два основних підходи до автоматизації тестування: тестування на рівні коду і GUI-тестування. До першого типу належить, зокрема, модульне тестування. До другого — імітація дій користувача за допомогою спеціальних тестових фреймворків.

Найпоширенішою формою автоматизації є тестування додатків через графічний інтерфейс користувача. Популярність такого виду тестування пояснюється двома факторами: по-перше, додаток тестується тим же способом, яким його буде використовувати людина, по-друге, можна тестувати додаток, не маючи при цьому доступу до вихідного коду.

Однією з головних проблем автоматизованого тестування є його трудомісткість: попри те, що воно дозволяє усунути частину рутинних операцій і прискорити виконання тестів, великі ресурси можуть витрачатися на оновлення самих тестів. Це відноситься до обох видів автоматизації. При рефакторінгу часто буває необхідно оновити і модульні тести, і зміна коду тестів може зайняти стільки ж часу, скільки і зміна основного коду. З іншого боку, при зміні інтерфейсу програми необхідно заново переписати всі тести, які пов'язані з оновленими вікнами, що при великій кількості тестів може відняти значні ресурси [5].

Отже, одним з головних завдань впровадження автоматизації в процес тестування є підвищення ефективності, збільшення охоплення та прискорення

тестування за умов постійного повтору тестових сценаріїв. Автотест можна запускати регулярно, в робочий і неробочий час. На виконання ручних тестів, знаходження і реєстрацію помилок у тестувальника в середньому йде близько дня. При автоматизації цей процес займе хвилини, а також дозволить знаходити помилки в коді на момент його внесення в репозиторій вихідного коду.

1.2 Аналіз існуючих інструментів для розробки тестових фреймворків

Тестування застосовується для визначення відповідності предмета випробування заданим специфікаціям. До завдань тестування не належить визначення причин невідповідності заданим вимогам (специфікаціям).

Тестування – необхідний етап створення продукту. Автоматизація дозволяє полегшити й прискорити цей процес. Але не всі види тестування потребують автоматизації, а тільки ті, які засновані на діях, що повторюються.

Вибір того чи іншого інструменту безпосередньо залежить від того, які вимоги пред’являються до тестових сценаріїв. У більшості випадків використовується відразу кілька інструментів, кожен з яких тестує свій рівень системної архітектури.

Використання інструмента автоматизації тестування дозволить записувати та реєструвати відтворений тест, а також повторно запускатись як і коли відбудуться послідовні випуски [6].

При їх виборі важливу роль відіграє наявність таких функцій:

– Простота використання та інтуїтивно зрозумілий інтерфейс

– Вбудована підтримка основних типів тестування — виробниче, регресійне, функціональне і т.п.

– Легкість в налагодженні та підтримці сценаріїв

– Формування інформативних звітів про тестування

Є список надійних і рекомендованих засобів тестування. Найбільш популярні з них наведено на рисунку 1.1. Вони дозволяють автоматизувати процес для різних продуктів і є невід'ємними допоміжними інструментами при інтеграційному.

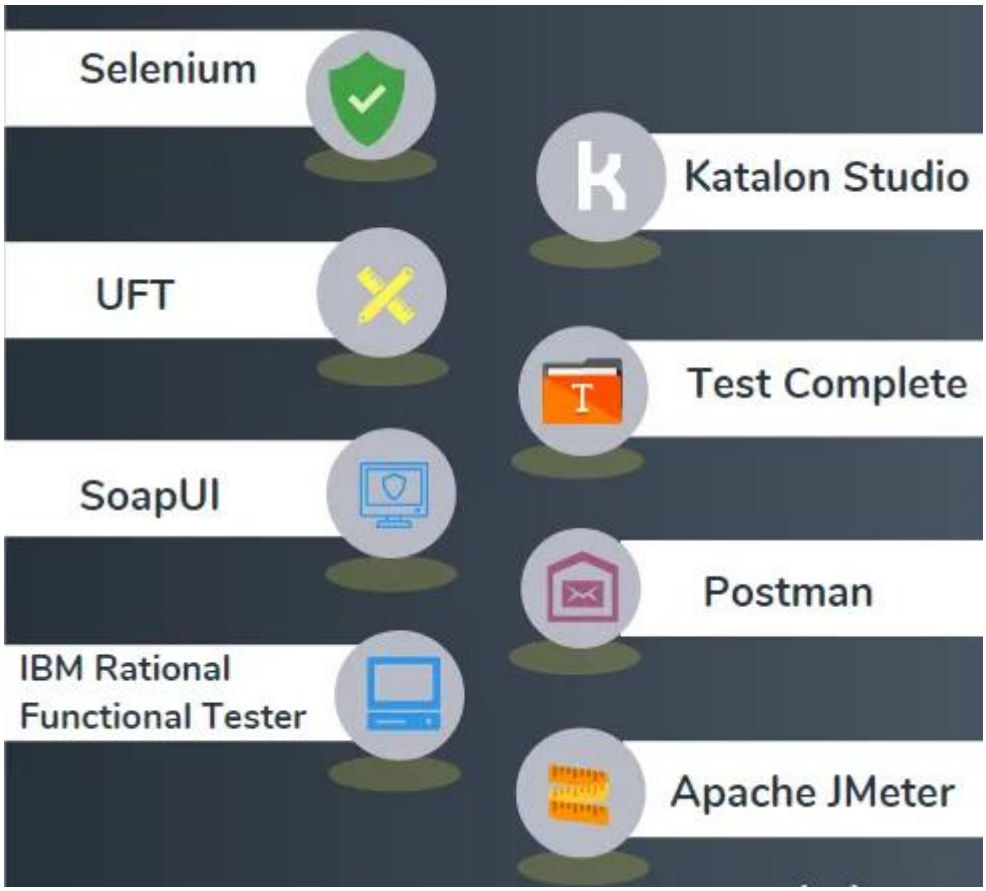


Рисунок 1.1 – Приклад інструментів для автоматизації тестування

Одним із видів інтеграційного тестування веб-додатків є UI-тестування. Тому далі розглянемо основні інструменти автоматизації інтеграційного UI-тестування:

– Selenium;
– TestingWhiz;
– HPE UFT;
– JMeter;
– Postman.

Selenium призначений для тестування сайтів і веб-додатків на різних операційних системах і браузерах [7]. Його перевага в тому, що тестувальники можуть обирати свою мову програмування для написання тестів. Selenium надає застосунок запису/відтворення, що дозволяє створювати тести вебзастосунків без вивчення мов програмування [8]. Інструмент також надає власну предметно-орієнтовану мову (Selenese) задля написання тестів на таких

мовах як C#, Groovy, Java, Perl, PHP, Python, Ruby та Scala. Найчастіше використовується для автоматизації Web-тестування [9]. У більшості випадків, коли говорять про Selenium, мають на увазі Selenium WebDriver [10, 11]. Основна частина розробки продукту зосереджена саме на цьому елементі. Автоматизація тестування в WebDriver часто порівнюється з водінням таксі (англ. Driver-водій). У водінні таксі і тестуванні беруть участь три складові: замовник (інженер-випробувач) – автомобіль (браузер) – водій таксі (WebDriver) [12, 13]. На рисунку 1.2 наведено детальну архітектуру технології.

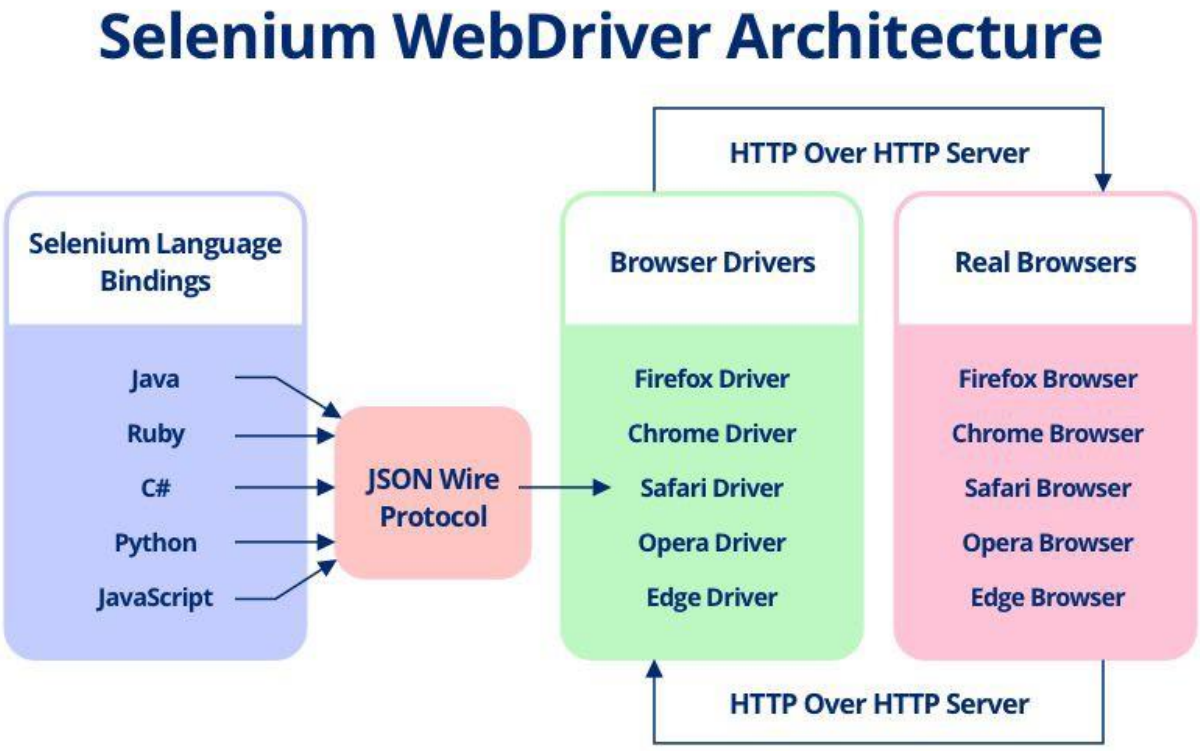


Рисунок 1.2 – Архітектура технології Selenium WebDriver

TestingWhiz дозволяє налагодити автоматизоване тестування для веб-продуктів і мобільних додатків, ПО, баз даних, програмних інтерфейсів додатків (API). Серед тестів підтримується регресійне і кросбраузерне види тестування [14]. TestingWhiz надає велику кількість сторонніх інтеграцій і функцій, щоб зробити автоматизацію тестування плавною і успішною. На рисунку 1.3 наведено інтерфейс додатку.

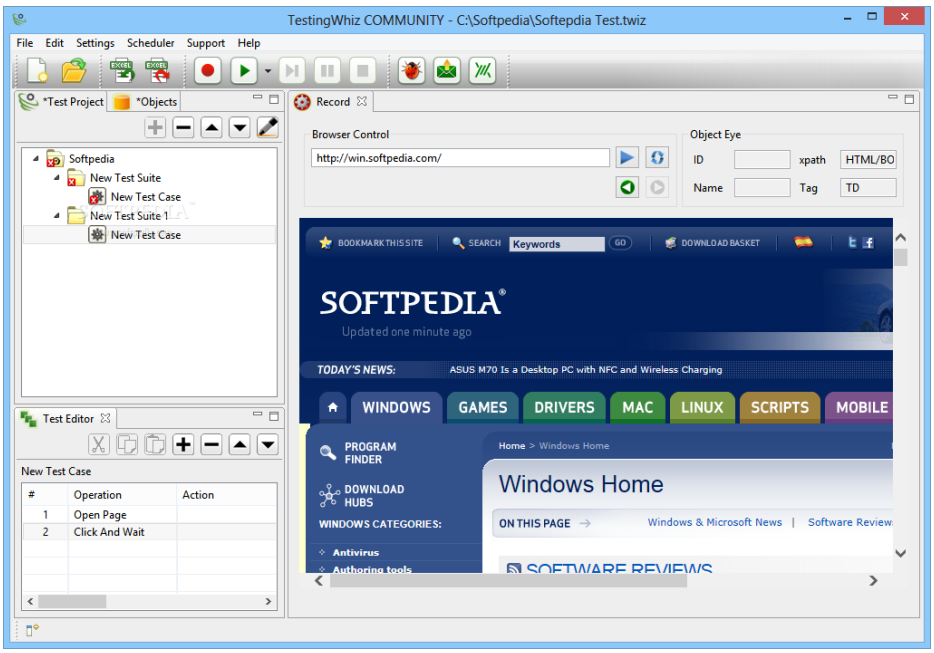


Рисунок 1.3 – Користувацький інтерфейс додатку TestingWhiz

HPE UFT – це ліцензований інструмент від Hewlett Packard. Призначений для функціонального тестування програмних додатків. Має вбудований механізм обробки багів, розпізнає смарт-об’єкти, контролює створюваний текст скрипта безпосередньо під час дій користувача [15]. Є одним із провідних інструментів автоматизації функціонального тестування, є флагманським продуктом компанії HP в своїй лінійці. На рисунку 1.4 наведено інтерфейс додатку.

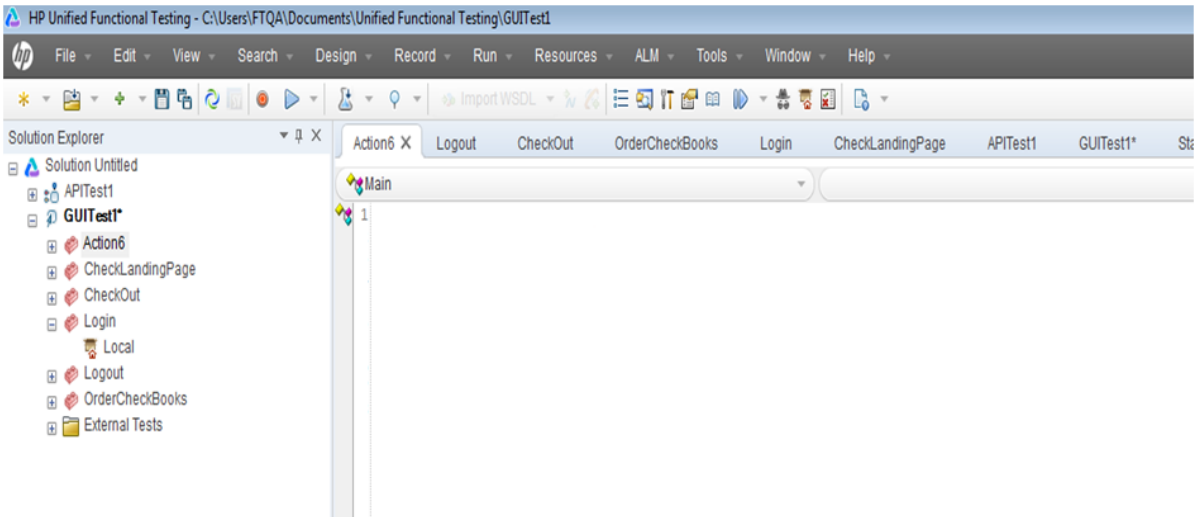


Рисунок 1.4 – Користувацький інтерфейс додатку HPE UFT

JMeter – це інструмент для проведення навантажувального тестування, що розробляється Apache Software Foundation. Хоча спочатку JMeter розроблявся як засіб тестування web-додатків, в даний час він здатний проводити навантажувальні тести для JDBC-з'єднань, FTP, LDAP, SOAP, JMS, POP3, IMAP, HTTP і TCP [16]. Цікава можливість створення великої кількості запитів за допомогою декількох комп'ютерів при управлінні цим процесом з одного з них. Організовано логування результатів тесту і різноманітна візуалізація результатів у вигляді діаграм, таблиць тощо [17]. На рисунку 1.5 наведено інтерфейс програми.

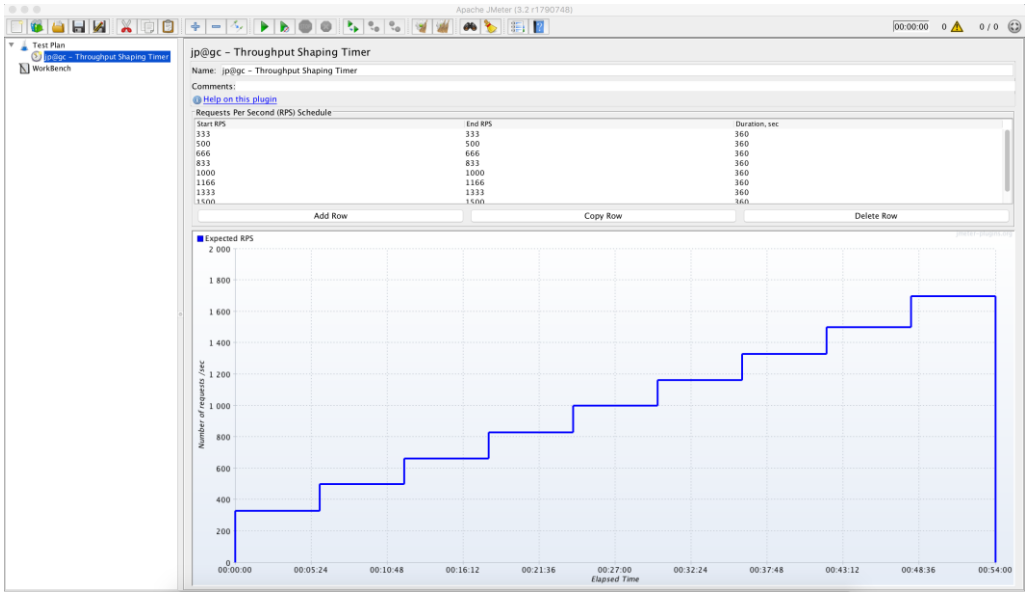


Рисунок 1.5 – Користувацький інтерфейс JMeter

Postman призначений для створення і тестування роботи програмних інтерфейсів додатків (API) і веб-сайтів, а також для відправки запитів на сервер. Завдяки графічному інтерфейсу можна легко налаштовувати всі необхідні дані для проведення тестів [18]. Будь-який розробник або тестувальник, відкривши колекцію, зможе з легкістю розібратися в роботі сервісу. До всього іншого, Postman дозволяє проектувати дизайн API і створювати на його основі Mock-сервер. На рисунку 1.6 наведено інтерфейс програми.

Після аналізу існуючих інструментів автоматизації тестування визначено їхні переваги та недоліки. Результат порівняння зведено в таблицю 1.1.

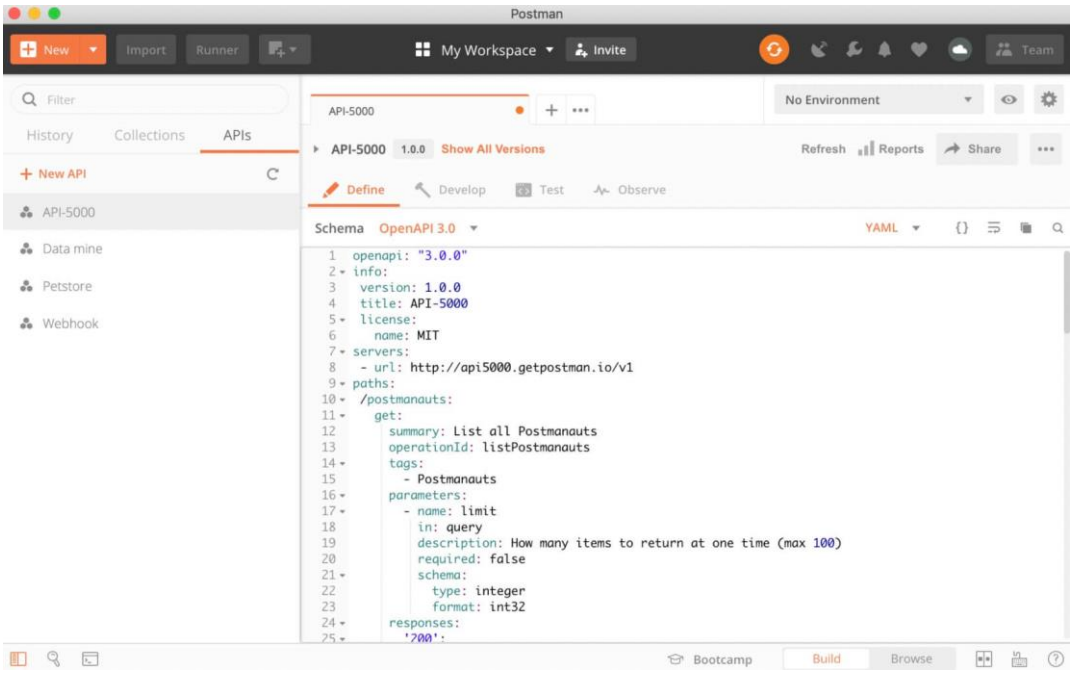


Рисунок 1.6 – Користувацький інтерфейс Postman

Таблиця 1.1 – Порівняльні характеристики програмних продуктів

Критерій	Selenium	Testing Whiz	HPE UFT	JMeter	Postman
Підтримка ООП мови програмування	1	1	1	1	0
Можливість десктопного UI-тестування	1	1	1	0	0
Безкоштовність повного функціоналу	1	0	0	1	1
Можливість інтегрування інших інструментів автоматизації	1	1	0	1	0
Мультибраузерність	1	1	1	1	1
Підсумковий результат	5	4	3	4	2

В результаті порівняння існуючих інструментів автоматизації тестування зроблено висновок, що розробка власного тестового фреймворку буде базуватись на технології Selenium.

1.3 Аналіз методів розв’язання поставленої задачі

Перш за все для написання автоматизованого тестового фреймворку знадобиться перелік вимог, які підлягають тестуванню. Потрібно занести їх до таблиці та надати кожній унікальний ID.

Після цього потрівно перейти до створення тестових сценаріїв, іншими словами – тест-кейси. Тест-кейс – це професійна документація тестувальника, послідовність дій спрямована на перевірку будь-якого функціоналу, що описує як прийти до фактичного результату [19]. Набір тест-кейсів називають тест-комплектом. Тест-кейси повинен допомогти провести перевірку продукту без ознайомлення з усією документацією. Написаний один раз, зручний в підтримці тест-кейс заощадить багато часу і сил тестувальникам.

Далі необхідно створити матрицю трасування і перевірити, чи всі вимоги покриті тест-кейсами.

Після цього тест-кейси потрібно їх автоматизувати. Далі розглянемо як в загальному проходить процес автоматизації тестів поетапно:

– Перший етап. Автотест запускається на основі проведених ручних тестів з перевіреними сценаріями. Для автоматизації процесу необхідно обрати інструмент тестування в залежності від типу продукту: веб-сайт, мобільний додаток, API, ПЗ і т.д. При його виборі звертаємо увагу на відповідність бюджету (скільки компанія готова заплатити за нього), на технічні характеристики й підтримку технологій, що використовуються в продукті; на вимоги до кваліфікації фахівців, що працюють з даними інструментом і на якість звітів, який генерує цей інструмент.

– Другий етап. Визначаємо обсяг автоматизації. Обсяги залежать від того, яка вироблена стратегія продукту. Якщо це основний продукт, то краще забезпечити максимальне покриття автоматичними тестами. Якщо це прототип, то тут велику роль відіграють терміни, а не якість продукту.

– Третій етап. Розробляємо стратегію і план автоматизації. Для цього проєктується інфраструктура для автоматизації (готуються необхідні стенди), затверджується графік запуску сценаріїв. Перед запуском автоматичних тестів

йде підготовка тестових даних. Після виконання тестів, аналізу звіту випробувань і фіксації помилок йде відстеження, виправлення і повторне тестування.

– Четвертий етап. Обслуговуємо тести для перевірки якості автоматизації. Це необхідно для підвищення ефективності вже наявних сценаріїв і при розробці нових.

Далі після розробки автоматизованого тестового фреймворку потрібно створити для нього CI/CD схему. CI/CD передбачає безперервну інтеграцію і доставку. Цей набір принципів призначений для підвищення зручності, частоти і надійності розгортання змін програмного забезпечення або продукту, а також він дає змогу проводити тестування на віддаленій віртуальній машині. Цілі CI/CD:

– Забезпечення послідовного і автоматизованого способу складання, упаковки і тестування продуктів або додатків;

– Автоматизація розгортання в різних оточеннях;

– Зведення до мінімуму помилок і проблем.

Поєднавши усі етапи розробки, в результаті отримаємо автоматизований тестовий фреймворк, в якому тести можна запускати як локально, так і на віддаленій віртуальній машині.

1.4 Постановка задач дослідження

Проаналізувавши питання розробки автоматизованого тестового фреймворку визначено наступні задачі:

– проаналізувати сучасний стан розвитку технологій для автоматизації інтеграційного UI-тестування;

– проаналізувати існуючі технології автоматизації тестів для розробки фреймворку;

– розробити перелік вимог, які необхідно протестувати;

– розробити тестові сценарії;

– розробити матрицю трасування;

- розробити гнучку архітектуру фреймворку згідно з принципами ООП;
- розробити архітектурні компоненти фреймворку;
- розробити алгоритм роботи тестового сценарію;
- розробити CI/CD схему;
- провести інтеграційне та UI-тестування сайту JetIQ;
- розробити інструкцію користувача.

Технічне завдання до роботи подано у Додатку А.

1.5 Висновки

У результаті проведеного аналізу досліджено особливості розробки автоматизованого тестового фреймворку для інтеграційного UI-тестування. Проаналізовано ряд існуючих інструментів та технологій, які призначені для автоматизації тестування. В результаті ретельного аналізу вирішено розробляти власний фреймворк на основі технології Selenium, оскільки вона відповідає усім поставленим вимогам та є кращою серед наведених технологій. Також встановлено основні завдання, які необхідно досягти для якісної розробки програмного продукту та порядок їх виконання.

2 РОЗРОБКА ТЕСТОВИХ СЦЕНАРІЇВ, МАТРИЦІ ТРАСУВАННЯ, АРХІТЕКТУРИ ТА CI/CD СХЕМИ ФРЕЙМВОРКУ

2.1 Розробка вимог, що підлягають тестуванню та тестових сценаріїв

Тестування ПЗ – процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності, який здійснюють шляхом спостереження за його роботою в штучно створених ситуаціях і на обмеженому наборі тестів, обраних певним чином. Тому для створення тестових сценаріїв перш за все потрібно визначити ряд вимог, які повинні бути протестовані.

Вимоги описують логіку роботи продукту (функціональні вимоги), його зовнішній вигляд (користувацький інтерфейс), а також нефункціональні вимоги.

Для опису функціональних вимог часто використовуються користувацькі сценарії (use cases). У користувацьких сценаріях представлені варіанти того, як користувач може взаємодіяти з ПЗ.

Нефункціональні вимоги описують обмеження пов'язані з дизайном продукту та його реалізацією (продуктивність, безпека, надійність, сумісність, стандарти якості).

Вимоги повинні відповідати наступним характеристикам:

- правильність;
- завершеність;
- ясність;
- реальність;
- необхідність;
- простежуваність;
- коректність;
- узгодженість;
- модифікованість.

JetIQ – це веб-ресурс для взаємодії студентів та викладачів ВНТУ. Для цього складемо таблицю 2.1, яка містить базові функціональні можливості веб-ресурсу, які потрібно протестувати.

Таблиця 2.1 – Перелік вимог

ID	Назва
В-1	Вхід у кабінет студента
В-2	Перевірка роботи панелі меню “Матеріали”
В-2.1	Перевірка роботи панелі меню “Матеріали” – “Матеріали моїх дисциплін”
В-2.2	Перевірка роботи панелі меню “Матеріали” – “Матеріали моєї спеціальності”
В-2.3	Перевірка роботи панелі меню “Матеріали” – “Матеріали всіх дисципліни”
В-2.4	Перевірка роботи панелі меню “Матеріали” – “Мої силабуси”
В-3	Перевірка роботи панелі меню “Розклад”
В-4	Перевірка роботи панелі меню “Мої результати”
В-5.1	Перевірка роботи панелі меню “Повідомлення” – “Написати викладачу”
В-5.2	Перевірка роботи панелі меню “Повідомлення” – “Надіслати файли”
В-6.1	Перевірка роботи панелі меню “Смартфон” – “Android”
В-6.2	Перевірка роботи панелі меню “Смартфон” – “IOS 14+”
В-7	Вихід із кабінету студента

Для розробки автоматизованого тестового фреймворку перш за все необхідно створити ряд тестових сценаріїв (тест-кейси), які будуть покривати основний функціонал продукту (сайт JetIQ).

Тест-кейс (Test Case) – це сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації функції, що тестується або її частини. Тест кейси є важливою частиною процесу тестування.

Тест-кейс може мати багато атрибутів, серед яких основними є:

– ID (унікальний номер)

– Назва;

– Передумови;

– Кроки відтворення;

– Очікуваний результат.

Тест-кейси можуть бути:

– Позитивні (використовуються тільки коректні дані і перевіряють, чи правильно додаток виконує функцію);

– Негативні (використовуються як коректні, так і некоректні дані (мінімум 1 некоректний параметр) і ставить за мету перевірку виняткових ситуацій (спрацьовування валідаторів), а також перевіряє, що функція не виконується при спрацьовування валідатора).

Основні стани проходження тест кейсу:

– No run (не запущено);

– Passed (пройдено);

– Failed (помилка);

– Blocked (заблоковано);

– Not Completed (не завершено).

Обов'язкові вимоги до тест кейсів:

– відсутність залежності один від одного;

– чіткі формулювання та висока ймовірність виявлення помилки;

– наявність детальної та не надлишкової інформації;

– легка діагностика помилок;

– дослідження відповідної області додатку, виконання потрібних дій.

Набір тест-кейсів називають тест-комплектом. Іноді тест-набір плутають з тест-планом. Тест-план описує які роботи, як і коли повинні бути проведені в рамках тестування продукту, а також що необхідно для їх виконання.

Тест-кейси допомогають проводити перевірку продукту без ознайомлення з усією документацією. Також вони дозволяють структурувати і систематизувати підхід до тестування та обчислювати метрики тестового

покриття і вживати заходів щодо його збільшення. Написаний один раз, зручний в підтримці тест-кейс заощадить багато часу і сил тестувальникам.

Нижче наведено рисунки 2.1 - 2.2, на яких зображено створені тест-кейси для сайту JetIQ.

ID	Назва	Передумови	Кроки	Очікуючий результат
T-1	Кабінет Студента - Перевірка меню: Матеріали -> Матеріали моїх дисциплін	1. Відкрити сторінку сайту JetIQ 2. Увійти в кабінет студента 3. Натиснути на кнопку 'Матеріали'	1. Натиснути на кнопку 'Матеріали моїх дисциплін'	Відобразилась таблиця електронних матеріалів
T-2	Кабінет Студента - Перевірка меню: Матеріали -> Матеріали моєї спеціальності		1. Натиснути на кнопку 'Матеріали моєї спеціальності'	
T-3	Кабінет Студента - Перевірка меню: Матеріали -> Матеріали всіх дисциплін		1. Натиснути на кнопку 'Матеріали всіх дисциплін'	Відобразилась таблиця "Електронні навчальні матеріали"
T-4	Кабінет Студента - Перевірка меню: Матеріали -> Мої силабуси	1. Відкрити сторінку сайту JetIQ 2. Увійти в кабінет студента	1. Натиснути на кнопку 'Мої силабуси'	Відобразилась таблця матеріалів з посиланнями на них
T-5	Кабінет Студента - Перевірка меню: Мої Результати -> Індивідуальний план студента	1. Відкрити сторінку сайту JetIQ 2. Увійти в кабінет студента 3. Натиснути на кноку 'Мої Результати'	1. Натиснути на кноку 'Індивідуальний план студента'	Відкрилась сторінка із індивідуальним навчальним планом студента
T-6	Кабінет Студента - Перевірка меню: Мої Результати -> Електронний журнал успішності		1. Натиснути на кноку 'Електронний журнал успішності'	Відкрилась сторінка із електронним журналом успішності
T-7	Кабінет Студента - Перевірка меню: Мої Результати -> Журнал пропусків та заборгованостей		1. Натиснути на кноку 'Журнал пропусків' 2. Повернутись назад і натиснути на кноку 'Журнал заборгованостей'	1. Відкрилась сторінка із журналом пропусків 2. Відкрилась сторінка із журналом заборгованостей
T-8	Кабінет Студента - Вихід з власного кабінету	1. Відкрити сторінку сайту JetIQ 2. Увійти в кабінет студента	1. Натиснути на кноку 'Вихід'	Студент вийшов з власного кабінету

Рисунок 2.1 – Тест-кейси T-1 – T-8

ID	Назва	Передумови	Кроки	Очікуючий результат
T-9	Кабінет Студента - Перевірка інтерфейсів відправки файлів та повідомлень викладачу	1. Відкрити сторінку сайту JetIQ 2. Увійти в кабінет студента 3. Натиснути на кнопку 'Повідомлення'	1. Натиснути на кнопку 'Відправити повідомлення' 2. Повернутись на вкладку назад і натиснути на кнопку 'Надіслати файл'	1. Відобразився інтферейс 'Повідомлення викладачу' 2. Відобразився інтферейс 'Надіслати файл викладачу'
T-10	Кабінет Студента - Відправка пустих повідомлень та повідомлень без файлів викладачу		1. Натиснути на кнопку 'Відправити повідомлення' 2. Натиснути на кнопку 'Відправити' 3. Повернутись на вкладку назад і натиснути на кнопку 'Надіслати файл' 4. Натиснути на кнопку 'Відправити'	1. Відобразився інтферейс 'Повідомлення викладачу' 2. Відобразилось повідомлення про помилку відправки пустого повідомлення 3. Відобразився інтферейс 'Надіслати файл викладачу' 4. Відобразився попап із помилкою відправки файла
T-11	Кабінет Студента - Перевірка меню: Смартфон -> Android та IOS	1. Відкрити сторінку сайту JetIQ 2. Увійти в кабінет студента 3. Навести на кнопку 'Смартфон'	1. Натиснути на кнопку 'Андроїд' 2. Повернутись назад та натиснути на кнопку 'IOS'	1. Відкрилась сторінка додатка у Play Market 2. Відкрилась сторінка додатка у Apple Store
T-12	Кабінет Студента - Перевірка меню: розклад	1. Відкрити сторінку сайту JetIQ 2. Увійти в кабінет студента	1. Натиснути на кнопку 'Розклад'	Відобразилась таблиця 'Розклад'

Рисунок 2.2 – Тест-кейси T-8 – T-12

2.2 Розробка матриці трасування

Матриця трасування – це двовимірна таблиця, що містить відповідність функціональних вимог продукту (functional requirements) і підготовлених тестових сценаріїв (test cases) [20]. Матриця трасування показує взаємозв’язки між тест-кейсами та вимогами.

На перетині відповідних рядка і стовпця ставиться відмітка, що позначає, що дана вимога покривається даними тест-кейсом. Таким чином, таблиця дає візуальне відображення двох параметрів: наявність в системі вимог, які ще не покриті (якщо у вимоги немає жодного перетину з тест-кейсами (достатня умова); чи є в системі надмірне тестування – якщо вимоги має кілька перетинів (необхідна умова).

Матриця трасування дозволяє:

– візуалізувати актуальний стан реалізації;

– розбивати вимоги на більш атомарні і структурувати їх;

– відстежувати, чи є вимоги, на які ще не запланована розробка (пропуск реалізації);

– відстежувати, чи реалізовано вимогу в даний момент;

– відстежувати, чи покрита вимога тест-кейсом (пропуск тестування);

– наочно відображати пріоритезацію вимог.

Варіанти зв'язків у матриці трасування:

– 1 до 1 (атомарна вимога, яка покривається одним тест-кейсом, даний тест-кейс покриває тільки цю вимогу);

– 1 до n (вимога, яка покривається декількома тест-кейсами, дані тест-кейси покривають тільки цю вимогу);

– n до n (вимога, яка покривається декількома тест-кейсами, дані тест-кейси покривають це та інші вимоги).

Щодо зв'язку n до n, коли одна вимога в матриці трасованості покривається декількома тестами, це може говорити про надмірність тестування. У такому випадку треба проаналізувати, наскільки вимога атомарна. Якщо виконанням всіх тест-кейсів забезпечується повнота покриття, а самі тест-кейси не дублюють один одного, то це не буде надмірним тестуванням. Оцінка покриття також розраховується окремо для кожного модуля або фічі.

На основі вимог та тест-кейсів, які наведено у розділах 2.1 та 2.2 створимо матрицю трасування та занесемо дані у таблицю 2.2.

Таблиця 2.2 – Матриця трасування

ID	Т-1	Т-2	Т-3	Т-4	Т-5	Т-6	Т-7	Т-8	Т-9	Т-10	Т-11	Т-12
В-1	+	+	+	+	+	+	+	+	+	+	+	+
В-2	+	+	+	+								
В-2.1	+											
В-2.2		+										
В-2.3			+									
В-2.4				+								
В-3												+
В-4					+	+	+					
В-5.1									+	+		
В-5.2									+	+		
В-6.1											+	
В-6.2											+	
В-7								+				

Отже, базуючись на побудованій матриці трасування можна зробити висновок, що кожна вимога покрита щонайменше одним тест-кейсом.

2.3 Розробка архітектури тестового фреймворку

Архітектура тестового фреймворку – це структура та опис того, як всередині влаштовані компоненти і як вони один з одним взаємодіють [21]. Відсутність архітектури при проектуванні тестового фреймворку призводить до численних переписувань коду, що в свою чергу вимагає серйозних витрат часу. Наслідок всього цього – втрата прибутку і невдоволення замовників [22].

Перейдемо безпосередньо до важливих та необхідних частин архітектури тестового фреймворку. Умовно її можна поділити на декілька частин:

– Тестовий шар (тестові сценарії)

– Шар імплементації на основі вимог бізнес логіки (реалізація тестових сценаріїв у коді)

– Ядро (базові функції, тестові ранери тощо)

Приклад такої архітектури наведено на рисунку 2.3.

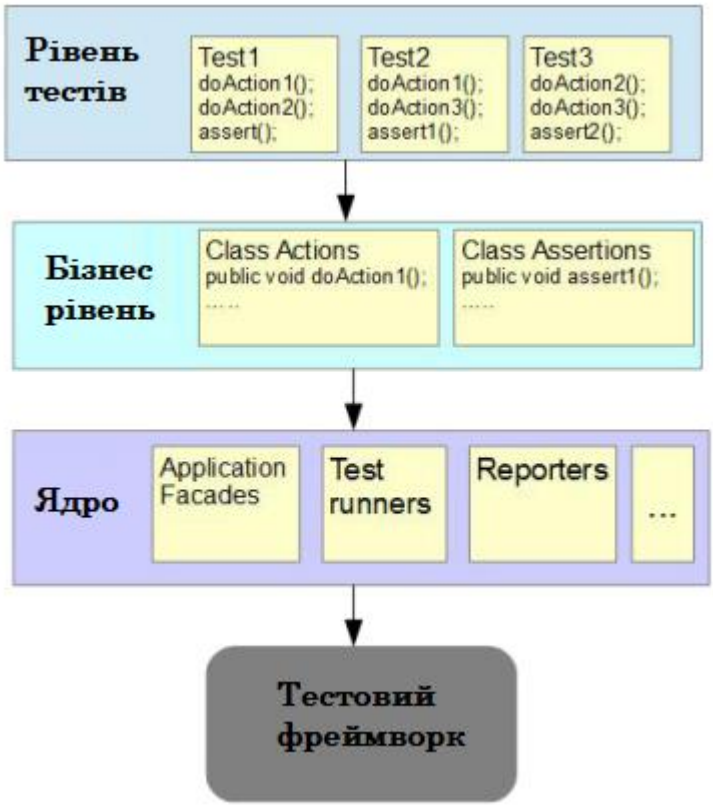


Рисунок 2.3 – Загальний приклад архітектури тестового фреймворку

Оскільки в основі фреймворку буде Selenium технологія, вирішено застосовувати патерн Page Object. Розглянемо цей патерн більш детальніше.

PageObject – шаблон проєктування, що використовується при написанні автоматизованих тестів, який дає змогу абстрагуватись від окремих елементів HTML і інкапсулювати їх у функції доступу до елементів інтерфейсу вищого рівня, як їх бачить користувач [23]. PageObject є об'єктом ООП, і містить методи, на основі яких створюється DSL для керування застосунком на основі якої пишуть варіанти тестування. Page Object зазвичай містить лише код для доступу до елементів керування і не містить ніяких тестових припущень [24]. Єдині перевірки, які здійснюються під час створення об'єкта – це те, що

інтерфейс та елементи керування на ньому завантажились і відобразились коректно. З елементами керування можна або взаємодіяти, або отримувати від них інформацію [25]. Наприклад, галочка може відображатись у змінну типу boolean. Головні переваги патерну:

– Розділення логіки роботи та представлення

– Зменшення дублювання коду для пошуку елементів керування застосунком

– При змінах інтерфейсу, що не зачіпають логіки, потрібно буде змінити лише PageObject, а не логіку тестів

Тестові сценарії в фреймворку будуть представлені на основі методологіїі BDD. Керована поведінкою розробка (BDD) – процес розробки програмного забезпечення, що виник з керованої тестами розробки (TDD). BDD поєднує основні засади та техніки TDD з ідеями предметно-орієнтованого проектування та об'єктно-орієнтованого дизайну з метою надати командам розробників та менеджменту спільні інструменти для співпраці під час розробки програмного забезпечення.

Керована поведінкою розробка, це розширення керованої тестами розробки, яка використовує прості предметно-орієнтовані мови програмування. Ці мови перетворюють запити природною мовою у виконувані тести. Результатом є більш тісний зв'язок з критеріями прийнятності для конкретної функції та тестами, які використовуються для перевірки цієї функціональності. Це є природним продовженням тестування TDD в цілому. BDD фокусується на наступному:

– Коли розпочати процеси

– Що тестувати та що не тестувати

– Скільки тестувати за один раз

– Як зрозуміти, чому тести пройшли неуспішно

Виходячи з цих питань, BDD вимагає, щоб імена тестів були цілими реченнями, які починаються з дієслова в умовному способі і слідували бізнес цілям. Опис приймальних тестів повинно вестися гнучкою мовою розповідей

користувача, наприклад: “Як [роль того, чиї бізнес інтереси задовольняються] я хочу, щоб [визначення функціональності так, як вона повинна працювати], для того щоб [визначення вигоди]”. Критерії приймання повинні бути описані через сценарій, який реалізує користувач, щоб досягти результату. На рисунку 2.4 зображена схема BDD методології.

Рисунок 2.4 – Схема BDD методології розробки ПЗ

Розробка через тестування (англ. test-driven development, TDD) – техніка розробки програмного забезпечення, яка ґрунтується на повторенні дуже коротких циклів розробки: спочатку пишеться тест, що покриває бажану зміну, потім пишеться код, який дозволить пройти тест, і під кінець проводиться рефакторинг нового коду до відповідних стандартів.

Тому базуючись на цьому, спроєктовано архітектуру майбутнього тестовго фреймворку, який поєднує у собі ООП принципи, патерн Page Object та BDD методологію розкобки ПЗ. Схема зображена на рисунку 2.5.

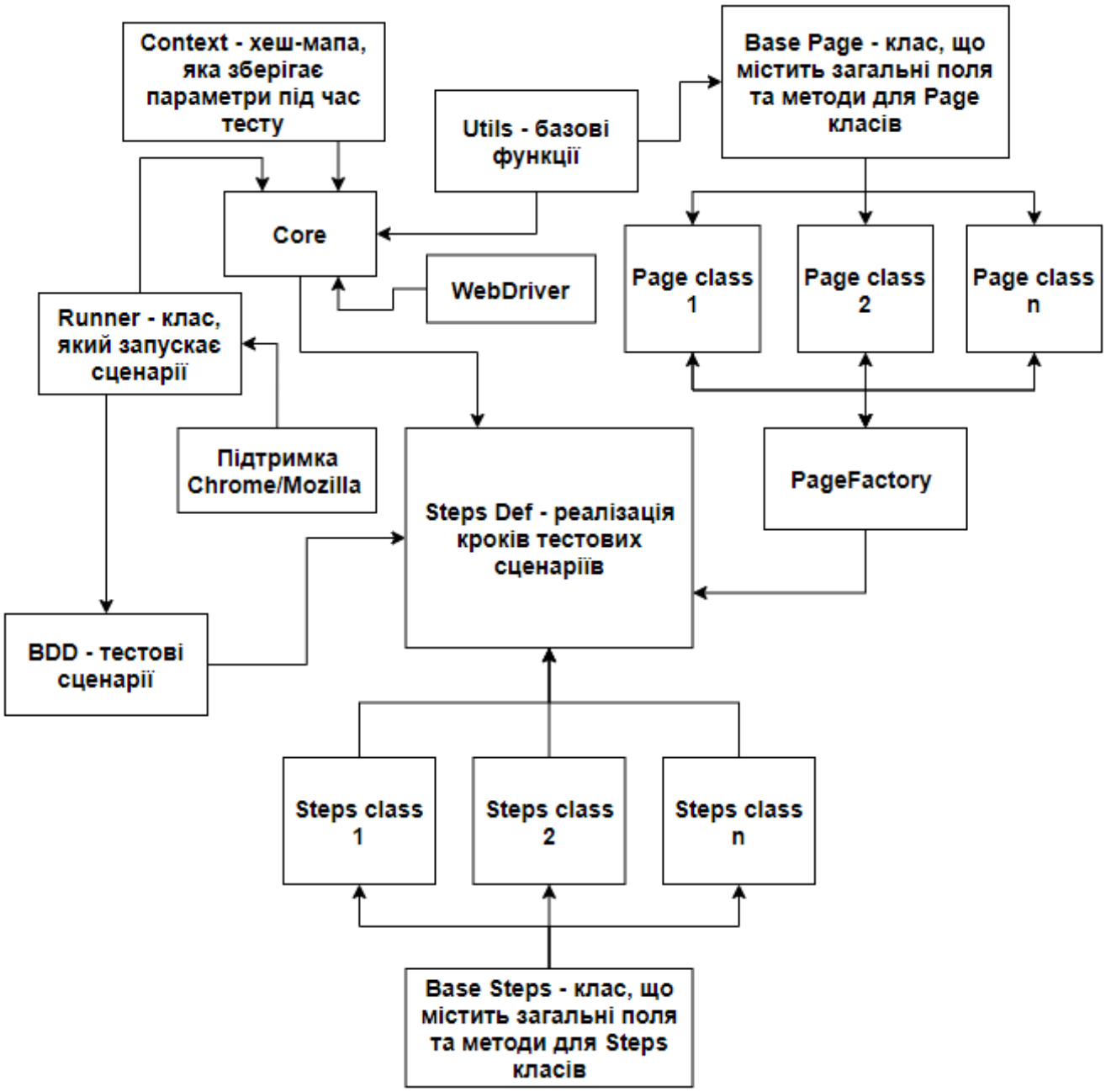


Рисунок 2.5 – Архітектура тестового фреймворку

Отже, в результаті маємо загальну архітектурну схему тестового фреймворку, яка відповідає усім поставленим вимогам і на основі якої буде відбуватись безпосередньо розробка.

2.4 Розробка алгоритму роботи тестового сценарію

Алгоритм – це набір інструкцій, які описують порядок дій виконавця, щоб досягти результату розв'язання задачі за скінченну кількість дій; система

правил виконання дискретного процесу, яка досягає поставленої мети за скінченний час. Для візуалізації алгоритмів часто використовують блок-схеми.

Кожен алгоритм передбачає існування початкових (вхідних) даних та в результаті роботи призводить до отримання певного результату. Робота кожного алгоритму відбувається шляхом виконання послідовності кроків, а процес їхнього виконання називають алгоритмічним процесом.

Для комп'ютерних програм алгоритм є списком деталізованих інструкцій, що реалізують процес обчислення, який, починаючи з початкового стану, відбувається через послідовність логічних станів, яка завершується кінцевим станом. Перехід з попереднього до наступного стану не обов'язково детермінований — деякі алгоритми можуть містити елементи випадковості.

Тестовий сценарій може містити тестові процедури і вкладені тестові сценарії. Крім того, тестовий сценарій, аналогічно тестовій процедурі, повинен повертати певне значення, яке однозначно визначає результат виконання тестової процедури.

Якщо тестовий сценарій є головним, то результат його виконання є кінцевим результатом. Рішення про те, яким має бути повертається значення, приймається самим тестовим сценарієм на основі результатів виконання всіх вхідних в нього сценаріїв і процедур.

Тестовий сценарій повинен бути здатний визначати результат його виконання на підставі результатів виконання всіх тестових процедур, які він викликає.

Крім того, будь-який тестовий сценарій повинен перевіряти виконання умов, необхідних для запуску всіх вкладених тестових сценаріїв і процедур. Такими умовами можуть бути правильність параметрів, переданих тестовому сценарію, наявність необхідних файлів, існування і стан тестованих об'єктів тощо.

Таким чином, на основі наведених вище міркувань був розроблений загальний алгоритм роботи тестового сценарію, блок-схема якого зображена на малюнку 2.6.

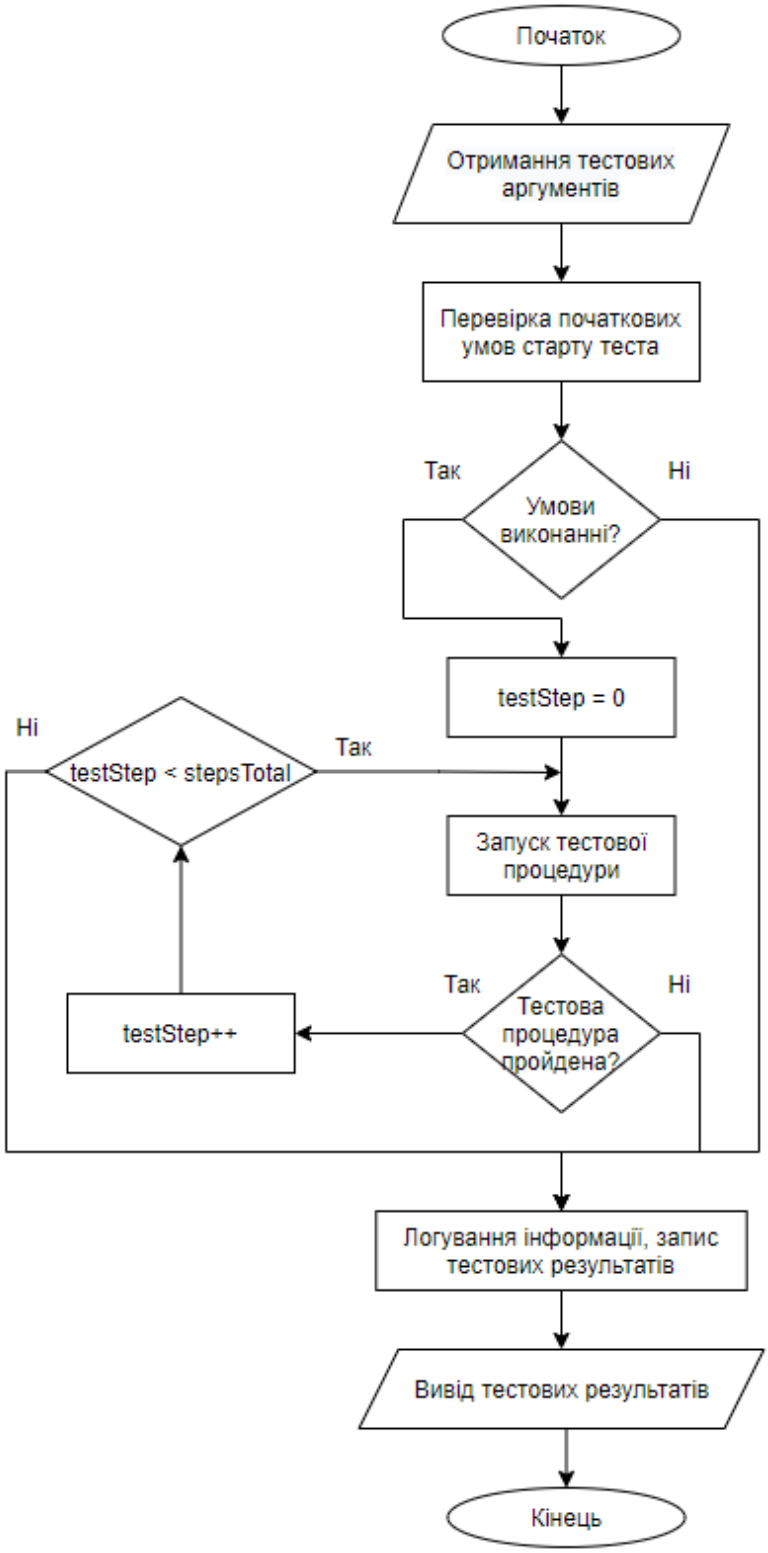


Рисунок 2.6 – Загальна блок-схема алгоритму роботи тестового сценарію

Алгоритм роботи можна коротко описати наступним чином: користувач запускає тестовий сценарій. Далі в тест передаються тестові аргументи та дані. Потім відбувається перевірка усіх умов, необхідних для початку тестування. У

випадку непроходження хоча б однієї з вимог тест вважається не пройденим, виводиться результат. Якщо усі умови успішно пройдено, то відбувається проходження тесту крок за кроком. Якщо хоча б один крок буде не пройденим, то тест буде вважатись непройденим також. Коли усі кроки будуть пройдені (або хоча б один не пройдений) відбувається логування інформації та вивід тестових результатів користувачу. На цьому тест закінчується.

Отже, було розроблено блок-схему алгоритму роботи тестового сценарію. Порівняно із аналогами, розроблений алгоритм дозволяє писати тестові сценарії українською мовою, зменшити обсяги памяті, що необхідні для тестування та підвищити швидкість і якість тестування. На основі цього алгоритму будуть розроблятись автоматизовані тестові сценарії.

2.5 Розробка CI/CD схеми

У розробці програмного забезпечення, CI/CD [26] – це комбінація безперервної інтеграції (continuous integration) і безперервного розгортання (continuous delivery або continuous deployment) програмного забезпечення в процесі розробки. CI/CD об'єднує розробку, тестування та розгортання програми. На даний момент програмісти прагнуть застосовувати CI/CD практично для всіх завдань.

На рисунку 2.7 зображено загальну схему CI/CD.

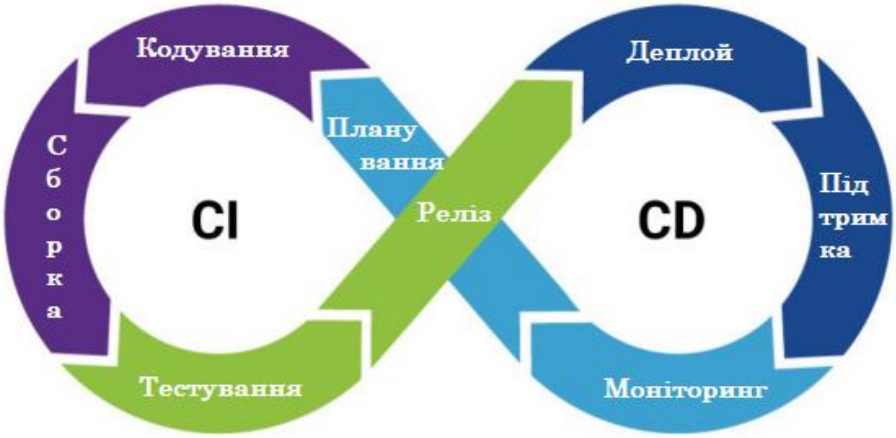


Рисунок 2.7 – Загальна базова схема технології CI/CD

Мета полягає в тому, щоб збільшити раннє виявлення дефектів, підвищити продуктивність і забезпечити більш швидкі цикли випуску. Цей процес відрізняється від традиційних методів, коли набір оновлень програмного забезпечення інтегрувався в один великий пакет перед розгортанням новішої версії [27].

Сучасні методи DevOps включають в себе безперервну розробку, безперервне тестування, безперервну інтеграцію, безперервне розгортання і безперервний моніторинг програмних додатків протягом усього життєвого циклу розробки. Практика CI/CD становить основу сучасних операцій DevOps.

Для того, щоб створити CI/CD потрібно буде підкючити проект до Git системи. Git – розподілена система керування версіями файлів та спільної роботи. Проєкт створив Лінус Торвальдс для керування розробкою ядра Linux, а сьогодні підтримується Джуніо Хамано (англ. Junio C. Hamano). На рисунку 2.8 зображено схему роботи Git.

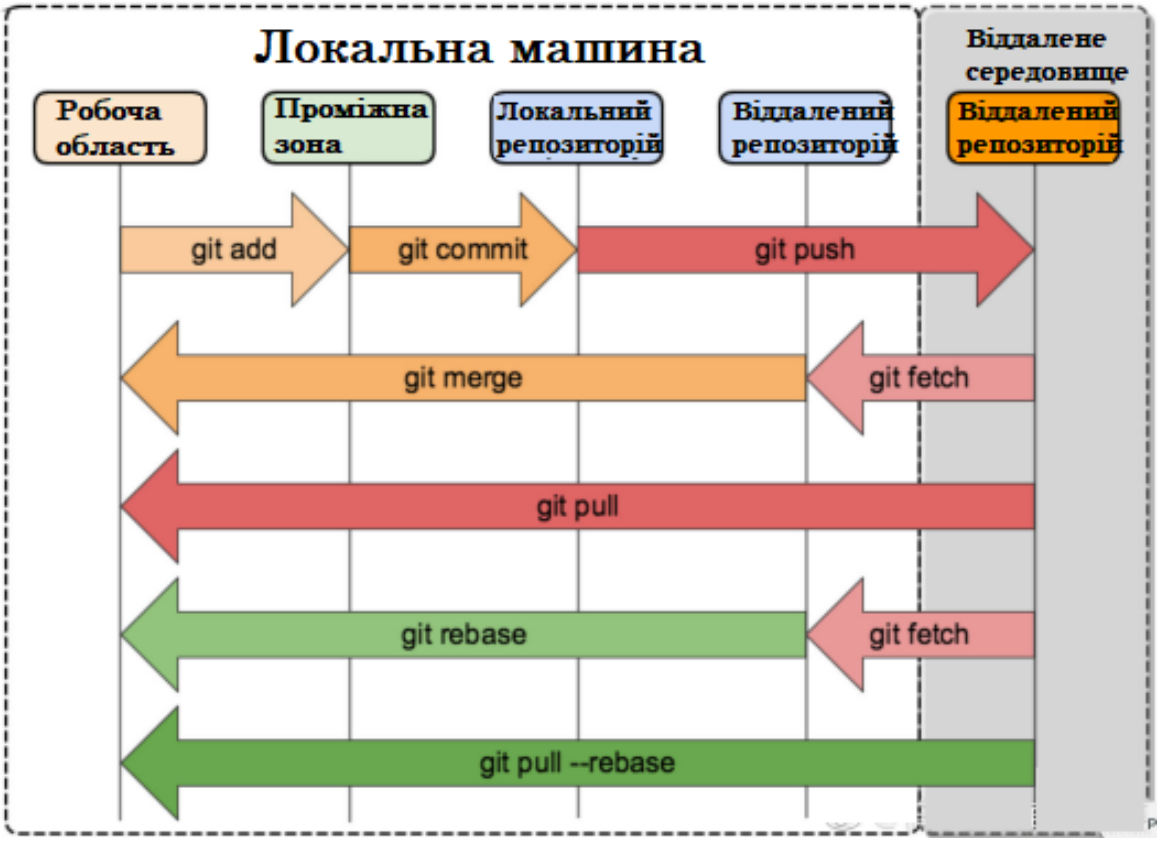


Рисунок 2.8 – Загальна базова схема технології Git

Git є однією з найефективніших, надійних і високопродуктивних систем керування версіями, що надає гнучкі засоби нелінійної розробки, що базуються на відгалуженні і злитті гілок. Для забезпечення цілісності історії та стійкості до змін заднім числом використовуються криптографічні методи, також можлива прив'язка цифрових підписів розробників до тегів і комітів.

Система спроєктована як набір програм, спеціально розроблених з врахуванням їхнього використання у скриптах. Це дозволяє зручно створювати спеціалізовані системи керування версіями на базі Git або користувацькі інтерфейси. Наприклад, Cogito є саме таким прикладом фронтенда до репозиторіїв Git. A StGit використовує Git для управління колекцією латок. Система має ряд користувацьких інтерфейсів: наприклад, gitk та git-gui розповсюджуються з самим Git.

На основі цього побудовано CI/CD схему, яка зображена на рисунку 2.9, де CI/CD Pipeline – це загальна “труба”, Stage – стадії, а роль джоб виконують круги – саме в них запускаються тести відповідно до тестових тегів (кожен тест має спеціальний тег).

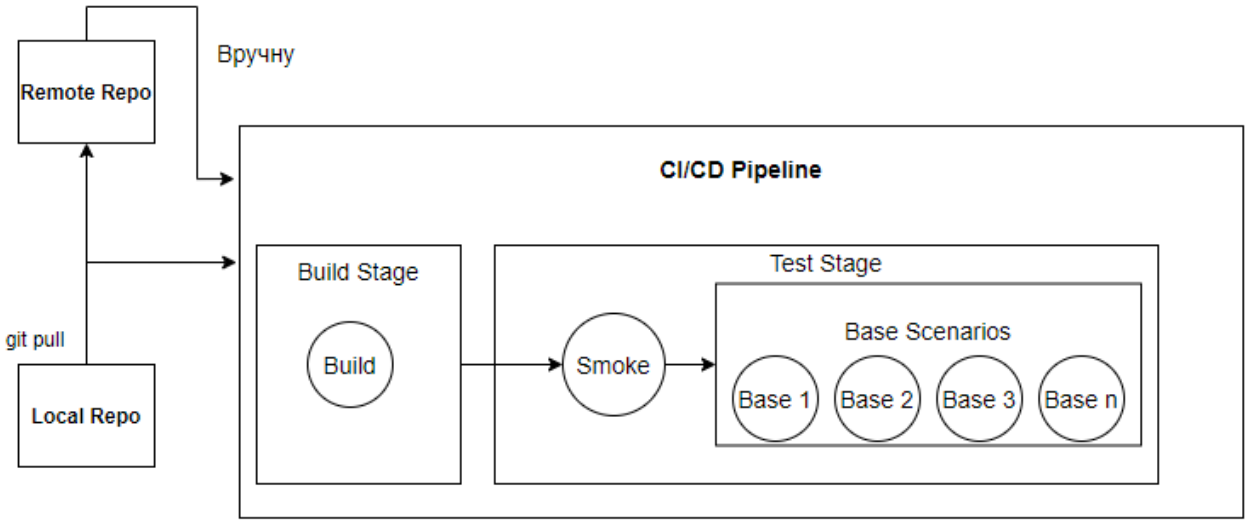


Рисунок 2.9 – CI/CD схема пайплайну для тестового фреймворку

Віддалений доступ до репозиторіїв Git забезпечується git-демоном, SSH або HTTP сервером. TCP-сервіс git-daemon входить у дистрибутив Git і є разом

з SSH найпоширенішим і надійним методом доступу. Метод доступу HTTP, хоч має низку обмежень, дуже популярний в контрольованих мережах, тому що дозволяє використання наявних конфігурацій мережевих фільтрів.

Отже, в результаті маємо загальну CI/CD схему для тестового фреймворку, яка відповідає усім поставленим вимогам, містить дві стадії: “Build” (зборка проєкту) та “Test” (тестування). Пайплайн запускається при кожному пуші коду до віддаленого репозиторію. Також користувач може вручну запустити пайплайн на сторінці репозиторію у GitLab.

2.6 Висновки

У другому розділі проведено детальну підготовку до розробки тестового фреймворку поетапно. Наведено перелік вимог, які необхідно протестувати. Також створено 12 тест-кейсів, за допомогою яких можна протестувати веб-ресурс JetIQ. При розробці фреймворку вони будуть автоматизовані і їх можна буде запускати як локально, так і на віддаленій віртуальній машині. На основі вимог і тест-кейсів створено матрицю трасування. Аналізуючи дані з цієї матриці, дійшли висновку, що кожна вимога покривається щонайменше одним тест-кейсом і вимоги, які б не покривались жодним тест-кейсом, відсутні. Створено загальну архітектуру програми. Вона базується на технології Selenium, патерні Page Object та базових принципах ООП. Було розроблено блок-схему алгоритму роботи тестового сценарію. На основі цього алгоритму будуть розроблятись автоматизовані тестові сценарії. Також створено загальну схему CD/CD пайплайну і в ході планування вирішено підключити проєкт до Git системи.

3 РОЗРОБКА ПРОГРАМНИХ КОМПОНЕНТ ДЛЯ АВТОМАТИЗОВАНОГО ТЕСТОВОГО ФРЕЙМВОРКУ

3.1 Варіантний аналіз і обґрунтування вибору мови програмування

Перед початком розробки будь-якого програмного забезпечення дуже важливим є питання вибору мови програмування, технологій та засобів розробки. Для цього слід перед початком написання програми чітко визначити вимоги до неї та визначитись, які функції в ній будуть реалізовані.

Для реалізації програми знадобиться ООП мова програмування, яка підтримує Selenium технологію. Розглянемо найбільш популярні мови програмування, які використовуються для розробки автоматизованих тестів, а саме C#, Java, Python та JavaScript.

C# – сучасна, об'єктно-орієнтована та строготипізована мова програмування, що дозволяє розробникам створювати безліч типів безпечних і надійних програм на базі екосистеми .NET. C# відноситься до широко відомого сімейства мов C і є близьким за синтаксисом і структрою до мов C, C++, Java або JavaScript. C# надає мовні конструкції для безпосередньої підтримки об’єктно-орієнтованої концепції роботи. Завдяки цьому C# підходить для створення та застосування програмних компонентів.

Мова C# була розроблена з урахуванням сильних і слабких особливостей інших мов, зокрема Java і C++. Специфікація мови C# була написана Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде. Андерс Хейлсберг відомий у світі програмування як автор компілятора Turbo Pascal і лідер команди, яка створила Delphi.

Java – сильно типізована об'єктно-орієнтована мова програмування. Бібліотеки з відкритим кодом полегшують використання Java у всьому світі. Apache, Google та інші організації розробили велику кількість потужних бібліотек, що полегшує та пришвидшує розробку програм. Часто варто пошукати в Google допоміжні бібліотеки, перш ніж писати свій власний код.

Існує велика вірогідність того, що подібний функціонал уже розроблено, протестовано й відкрито для використання.

Іншою важливою особливістю технології Java є гнучка система безпеки, в рамках якої виконання програми повністю контролюється віртуальною машиною. Будь-які операції, які перевищують встановлені повноваження програми , викликають негайне переривання. Також додає безпеки те, що методи перевірки автентичності засновані на шифруванні з відкритим ключем.

Java створювалася як мова для розподіленого програмування: дані та програми можуть спільно використовуватись кількома комп'ютерами, що підвищує продуктивність і ефективність праці.

Python – інтерпретована об'єктно-орієнтована мова програмування високого рівня із строгою динамічною типізацією. Дана мова програмування в першу чергу орієнтована на підвищення продуктивності розробника за рахунок простоти свого коду та зрозумілості документації. Це скриптова мова, яка застосовується для вирішення широкого спектру завдань. Найчастіше Python застосовують в роботі з великими даними і при розробці сайтів і мобільних ігор. Він підходить і для створення десктопних і мобільних додатків.

JavaScript (JS) – динамічна, об'єктно-орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується для створення сценаріїв вебсторінок, що надає можливість на боці клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд вебсторінки. JavaScript класифікують як прототипну (підмножина об'єктно-орієнтованої), скриптову мову програмування з динамічною типізацією.

Окрім прототипної, JavaScript також частково підтримує інші парадигми програмування (імперативну та частково функціональну) і деякі відповідні архітектурні властивості, зокрема: динамічна та слабка типізація, автоматичне

керування пам'яттю, прототипне наслідування, функції як об'єкти першого класу.

Порівняння мов програмування наведено в таблиці 3.1.

Таблиця 3.1 – Порівняння мов програмування

Критерій	C#	Java	Python	JavaScript
Об’єктно-орієнтованість	1	1	0.5	0.5
Простота синтаксису	1	1	1	1
Обширність та доступність документації	0,5	1	0,5	0,5
Інструменти для легкого аналізу даних	1	1	0,5	0,5
Підтримка Selenium WebDriver	1	1	1	1
Підсумковий результат	4,5	5	3,5	3,5

Згідно таблиці 3.1, можна зробити висновок, що мова програмування Java підходить найкраще серед усіх розглянутих мов, так як задовольняє усі потреби, які можуть виникнути в процесі розробки тестового фреймворку. Вона є простою у використанні та має зрозумілий синтаксис, а також, що є дуже важливим, підтримує роботу із Selenium WebDriver технологією.

3.2 Вибір середовища розробки

Не менш важливим, ніж питання вибору мови програмування, на якій буде реалізовано програму, є питання вибору інтегрованого середовища розробки.

Інтегрованим середовищем розробки (IDE) називають комплексне програмне рішення для розробки програмного забезпечення, яке зазвичай складається з редактора початкового коду, інструментів для автоматизації складання та відлагодження програм. Розглянемо найбільш популярні інтегровані середовища розробки, які використовуються для реалізації

програмних продуктів на мові програмування Java, а саме Eclipse та IntelliJ IDEA.

Eclipse – вільне модульне інтегроване середовище розробки програмного забезпечення. Розробляється і підтримується Eclipse Foundation і включає проєкти, такі як платформа Eclipse, набір інструментів для програмістів на мові Java, системи контролю версій, конструктори GUI тощо. Написаний в основному на Java, може бути використаний для розробки застосунків на Java і, за допомогою різних плагінів, на інших мовах програмування, включаючи Ada, C, C++, C#, COBOL, Fortran, Perl, PHP, Python, R, Ruby (включно з каркасом Ruby on Rails), Scala, Clojure та Scheme. Середовища розробки зокрема включають Eclipse ADT (Ada Development Toolkit) для Ada, Eclipse CDT для C/C++, Eclipse JDT для Java, Eclipse PDT для PHP.

IntelliJ IDEA – комерційне інтегроване середовище розробки для різних мов програмування (Java, Python, Scala, PHP тощо) від компанії JetBrains. Система поставляється у вигляді урізаної по функціональності безкоштовної версії «Community Edition» і повнофункціональної комерційної версії «Ultimate Edition», для якої активні розробники відкритих проектів мають можливість отримати безкоштовну ліцензію. Сирцеві тексти Community-версії поширюються рамках ліцензії Apache 2.0. Бінарні збірки підготовлені для Linux, Mac OS X і Windows.

Community версія середовища IntelliJ IDEA підтримує інструменти (у вигляді плагінів) для проведення тестування TestNG і JUnit, системи контролю версій CVS, Subversion, Mercurial і Git, засоби складання Maven, Ant, Gradle, мови програмування Java, Scala, Clojure, Groovy і Dart. Підтримується розробка застосунків для мобільної платформи Android. До складу входить модуль візуального проектування GUI-інтерфейсу Swing UI Designer, XML-редактор, редактор регулярних виразів, система перевірки коректності коду, система контролю за виконанням завдань і доповнення для імпорту та експорту проектів з Eclipse. Доступні засоби інтеграції з системами

відстеження помилок JIRA, Trac, Redmine, Pivotal Tracker, GitHub, YouTrack, Lighthouse.

Порівняння середовищ розробки наведено в таблиці 3.2.

Таблиця 3.2 – Порівняння середовищ розробки

Критерій	Eclipse	IntelliJ IDEA
Підтримка Java	1	1
Підтримка Selenium	0,5	1
Обширність та доступність документації	0,5	1
Можливості базового функціоналу	1	1
Простота у використанні	0,5	1
Підсумковий результат	3,5	5

Згідно таблиці 3.2, можна зробити висновок, що середовище розробки IntelliJ IDEA підходить найкраще серед усіх розглянутих, так як задовольняє усі потреби, які можуть виникнути в процесі розробки тестового фреймворку. Воно є простим та зрозумілим у використанні, а також підтримує роботу із Selenium WebDriver технологією та мовою програмування Java.

3.3 Реалізація тестових сценаріїв за допомогою BDD підходу

У мові програмування Java для реалізації BDD підходу застосовують Cucumber – середовище автоматичного тестування, яке підтримує BDD (Behavior Driven Development), тобто розробку, керовану поведінкою. Перед модульним тестуванням або інтеграційним тестуванням етапи тестування та інформація про перевірку заздалегідь визначаються загальною мовою, щоб усі могли зрозуміти етапи та мету кожного етапу модульного тестування та інтеграційного тестування. А ті, хто пише модульні тести та інтеграційні тести, можуть писати код на основі заздалегідь написаного фреймворку для досягнення мети розробки, заснованої на поведінці.

Тестові сценарії знаходяться у файлах із розширенням .feature. У такому файлі можна написати сценарій покроково, потім потрібно реалізувати ці кроки в коді. Після цього сценарій можна запускати. На рисунку 3.1 та 3.2 зображено готовий тестовий сценарій в одному із таких файлів і реалізація одного із кроків сценарію в коді відповідно.

```
Feature: Базові тестові сценарії для сайту JetIQ

  Background:
    Given Відкрити домашню сторінку JetIQ
    And Натиснути на кноку 'Вхід'
    And Ввести 'valid' значення в поле логіну
    And Натиснути на кнопку 'Далі'
    And Ввести 'valid' значення в поле паролю
    And Натиснути на кноку 'Ввійти'

  @StudentSchedule
  Scenario: Happy Pass: Кабінет Студента - Перевірка меню: розклад
    When Натиснути на кнопку 'Розклад'
    Then Відобразилась таблиця 'Розклад'
```

Рисунок 3.1 – Готовий тестовий сценарій, створений за домопогою Cucumber

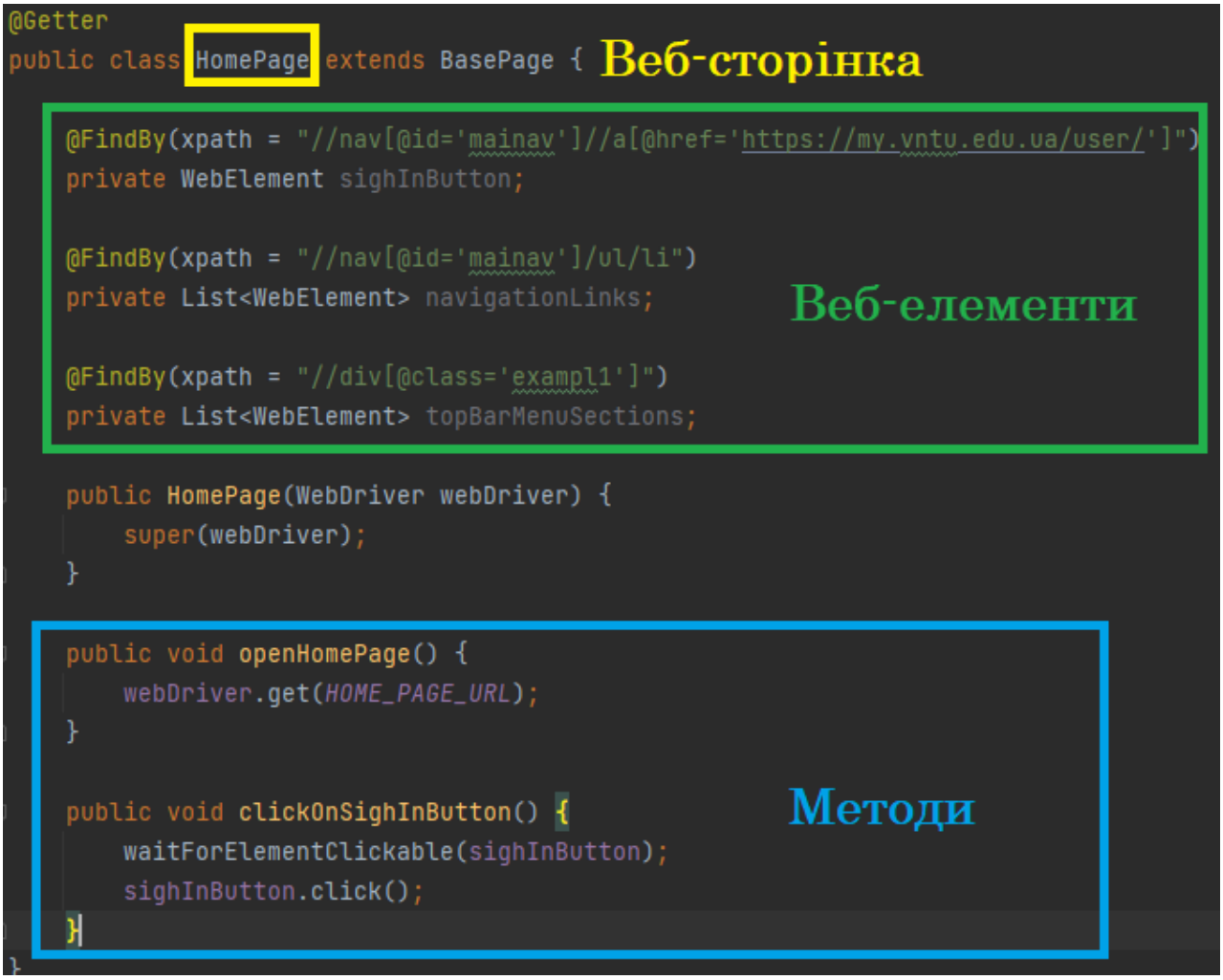
```
@When("Відкрити домашню сторінку JetIQ")
public void openHomePage() {
    homePage.openHomePage();
    homePage.waitForPageLoadComplete();
    log.info("Home page was opened");
}
```

Рисунок 3.2 – Реалізація одного із кроків сценарію в коді

Отже, по такому принципу було створено 7 файлів, містять 12 тест-кейсів, які було розроблено у розділі 2.2. Далі необхідно створити .java класи Steps, які будуть реалізовувати кроки із цих файлів в коді.

3.4 Програмна реалізація на основі Selenium технології

Тестові сценарії готові то ж тепер потрібно створити для них реалізацію в коді і тоді вони цілком можуть вважатись автоматизованими. Код був написаний на мові програмування Java відповідно до усіх стандартів та вимог ООП та програмування в цілому із застосуванням патерну Page Object. Далі наведено частини коду тестового фреймворку того чи іншого компоненту (рисунки 3.4 – 3.9).




```
@Getter
public class HomePage extends BasePage {

    @FindBy(xpath = "//nav[@id='mainav']//a[@href='https://my.vntu.edu.ua/user/']")
    private WebElement sighInButton;

    @FindBy(xpath = "//nav[@id='mainav']/ul/li")
    private List<WebElement> navigationLinks;

    @FindBy(xpath = "//div[@class='exampl1']")
    private List<WebElement> topBarMenuSections;

    public HomePage(WebDriver webDriver) {
        super(webDriver);
    }

    public void openHomePage() {
        webDriver.get(HOME_PAGE_URL);
    }

    public void clickOnSighInButton() {
        waitForElementClickable(sighInButton);
        sighInButton.click();
    }
}
```

Рисунок 3.4 – Лістинг коду класу веб-сторінки

На рисунку 3.4 наведено приклад класу однієї із веб-сторінок. Жовтим кольором виділено назву класу, зеленим – поля, які представляються собою веб-елементи (або колекції з веб-елементів), які знаходяться на цій веб-сторінці, а синім – методи взаємодії з цими веб-елементами.

```java
@Slf4j
public class AssertsSteps extends BaseSteps {

    @Then("Навігаційні посилання відображаються на сторінці")
    public void checkNavLinks() {
        log.info("Start checking navigation links...");
        for (WebElement webElement : homePage.getNavigationLinks())
            assertTrue(homePage.isElementDisplayed(webElement));
        }
        log.info("Navigation links are displayed on page");
    }
```

Рисунок 3.5 – Лістинг коду класу реалізації кроків

На рисунку 3.5 показано приклад реалізації одного з кроків за допомогою спеціальної анотації @Then, яка належить технології Cucumber.

```java
@AllArgsConstructor
public class PageFactoryManager {

    private final WebDriver webDriver;

    public HomePage getHomePage() { return new HomePage(webDriver); }

    public SignInPage getSignInPage() { return new SignInPage(webDriver); }

    public StudentHomePage getUserHomePage() { return new StudentHomePage(webDriver); }

    public MaterialsPage getMaterialsPage() { return new MaterialsPage(webDriver); }

    public MessagesFilesPage getMessagesFilesPage() { return new MessagesFilesPage(webDriver); }

    public MyResultsPage getMyResultsPage() {
        return new MyResultsPage(webDriver);
    }
}
```

Рисунок 3.6 – Лістинг коду класу, що створює екземпляри класів веб-сторінок

На рисунку 3.6 показано приклад реалізації PageFactory патерну, який є удосконаленим видом PageObject патерну. Даний клас являє собою так звану “фабрику”, яка створює екземпляри класів сторінок. PageFactory патерн дещо схожий до Factory патерну [28].

```
Feature: Базові тестові сценарії для сайту JetIQ

  Background:
    Given Відкрити домашню сторінку JetIQ
    And Натиснути на кноку 'Вхід'
    And Ввести 'valid' значення в поле логіну
    And Натиснути на кнопку 'Далі'
    And Ввести 'valid' значення в поле паролю
    And Натиснути на кноку 'Ввійти'

  @StudentMaterials
  Scenario: Happy Pass: Кабінет Студента - Перевірка меню: Матеріали -> Матеріали моїх дисциплін
    When Натиснути на кнопку 'Матеріали'
    And Натиснути на кнопку 'Матеріали моїх дисциплін'
    Then Відобразилась таблця електронних матеріалів
    When Обрати диспипліну під номером 1
    Then Відобразилась таблця матеріалів з посиланнями на них

  @StudentMaterials
  Scenario: Happy Pass: Кабінет Студента - Перевірка меню: Матеріали -> Матеріали моєї спеціальності
    When Натиснути на кнопку 'Матеріали'
    And Натиснути на кнопку 'Матеріали моєї спеціальності'
    And Обрати будь який семестр
    Then Відобразилась таблця електронних матеріалів
    When Обрати диспипліну під номером 1
    Then Відобразилась таблця матеріалів з посиланнями на них

  @StudentMaterials
  Scenario: Happy Pass: Кабінет Студента - Перевірка меню: Матеріали -> Матеріали всіх дисциплін
    When Натиснути на кнопку 'Матеріали'
    And Натиснути на кнопку 'Матеріали всіх дисцпиплін'
    Then 'Електронні навчальні матеріали' table is appeared

  @StudentMaterials
  Scenario: Happy Pass: Кабінет Студента - Перевірка меню: Матеріали -> Мої силабуси
    When Натиснути на кнопку 'Мої силабуси'
    Then Відобразилась таблця матеріалів з посиланнями на них
```

Рисунок 3.7 – Реалізація тест-кейсів Т-1 – Т-4

На рисунку 3.7 показано приклад реалізації тестових сценаріїв за допомогою Cucumber технології. Кожен сценарій має власний тег.

```
@SneakyThrows
@After(order = 2)
public void scenarioReporter(Scenario scenario) {
    if (scenario.isFailed()) {
        String fileName = OffsetDateTime.now().toEpochSecond() + ".png";
        log.info("Scenario is failed. Start making a screenshot...");
        File screenshot = ((TakesScreenshot) webDriver).getScreenshotAs(OutputType.FILE);
        FileUtils.copyFile(screenshot, new File( pathname: "target/generated-test-sources/" + fileName));
    }
}
```

Рисунок 3.8 – Лістинг коду методу scenarioReporter

На рисунку 3.8 зображено метод scenarioReporter(), який робить скріншот сторінки, у випадку коли сценарій є не пройденим. Метод написаний із застосуванням можливостей Cucumber технології. Скріншот зберігається у тимчасову папку тестових результатів. Даний метод є дуже корисним при дебазі та пошуку багів чи помилок.

```
@Before(order = 2)
public void setUpPageFactory() {
    PageFactoryManager pageFactoryManager = new PageFactoryManager(webDriver);
    scenarioContext.put(HOME_PAGE, pageFactoryManager.getHomePage());
    scenarioContext.put(SIGN_IN_PAGE, pageFactoryManager.getSignInPage());
    scenarioContext.put(USER_HOME_PAGE, pageFactoryManager.getUserHomePage());
    scenarioContext.put(MATERIALS_PAGE, pageFactoryManager.getMaterialsPage());
    scenarioContext.put(MESSAGES_FILES_PAGE, pageFactoryManager.getMessagesFilesPage());
    scenarioContext.put(MY_RESULTS_PAGE, pageFactoryManager.getMyResultsPage());
    log.info("PageFactory was started successfully");
}
```

Рисунок 3.9 – Лістинг коду одного із хуків

Метод setUpPageFactory є одним із хуком. Даний метод поміщає усі екземляри класів сторінок у контекст, щоб в подальшому їх можна було зручно використовуватию.

Хуки – це такі методи, які запускаються до та після виконання тестового сценарію за допопогою відповідних анотацій: @Before та @After. Значення змінної ‘order’ означає пріоритет виконання хуку. Хуки виконуються завжди, незалежно від результатів тестування [29].

3.5 Налаштування CI/CD пайплайну

Як було зазначено раніше, для конфігурації CI/CD потрібно додати до проекту Git. Тому прийнято рішення створити віддалений репозиторій за допомогою сервісу GitLab – сайту та системи керування репозиторіями програмного коду для Git. Також він має додаткові можливості: власна вікі та система відстеження помило.

Віддалений репозиторій зображений на рисунку 3.10.

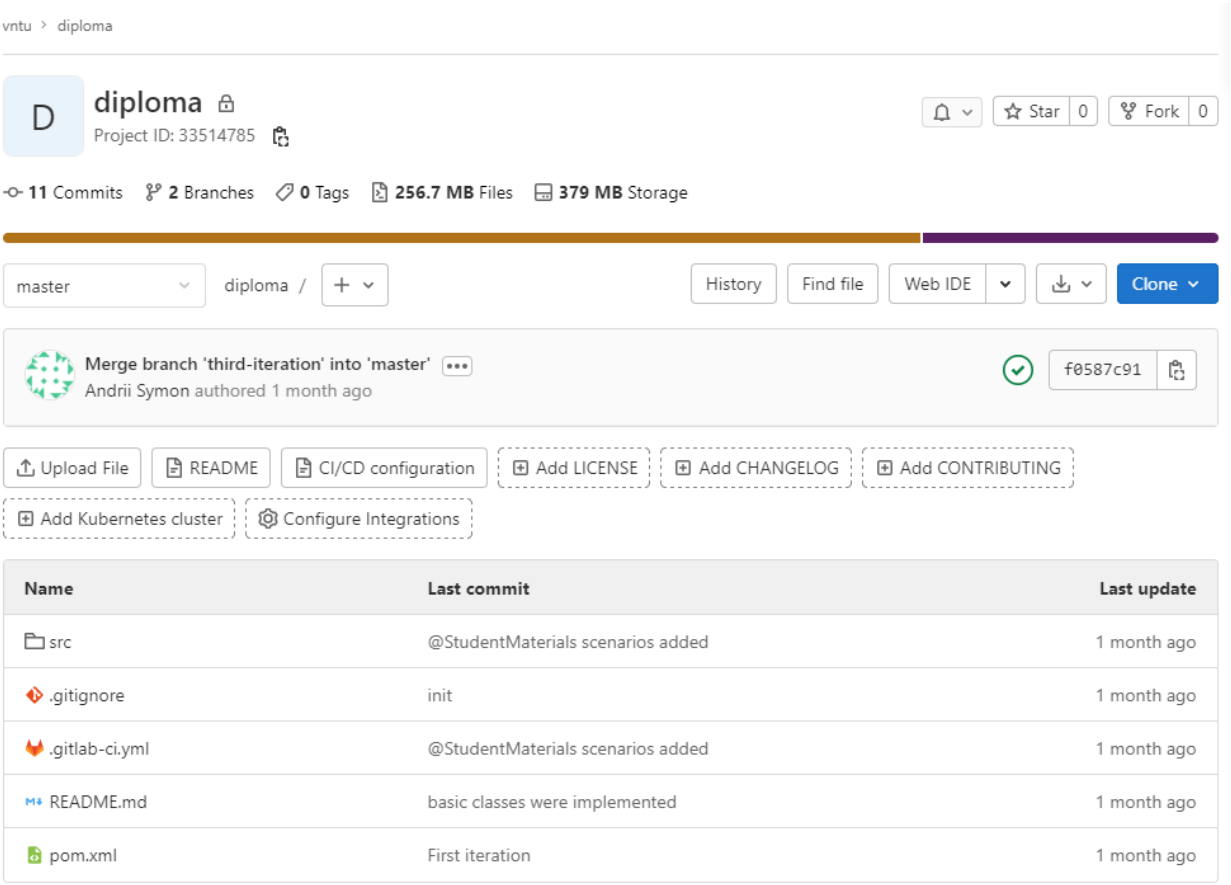


Рисунок 3.10 – Віддалений репозиторій на GitLab

Далі створено .gitlab-ci.yml файл, на основі якого будується CI/CD пайплайн. Проте перед цим потрібно сконфігурувати віддалену віртуальну машину, адже саме на ній відбувається тестування в CI/CD. Тому вирішено використовувати публічний Docker-image “markhobson/maven-chrome:latest” та GitLab Shared Runner на базі ОС Linux.

Далі було створено 3 стадії пайплайну: Build, Smoke Test та Base Test. На першій стадії відбувається побудова проекту, далі смок тестування (базові сценарії без яких, у разі непроходження, неможливе подальше тестування) і в кінці запуск усіх інших тестів.

Слід зазначити, що при непроходженні першої чи другої стадії весь пайплайн зупиняється і помічається як “Failed”. Якщо непроходить третя стадія частково або повністю – “Passed with warnings”. Якщо всі три стадії будуть пройдені, то пайплайн помічається як “Passed”.

Також додано до конфігурації можливість запускати пайплайн вручну з сайту GitLab. По замовчуванню пайплайн стартує після кожної відправки коміту в віддалений репозиторій.

На рисунках 3.11 та 3.12 зображено лістинг коду, що містить Docker-image, стадії, скрипт запуску Cucumber-сценаріїв та джоби, які запускають відповідні сценарії за допомогою спеціальних тегів.

```
image: markhobson/maven-chrome:latest

stages:
  - Build
  - Smoke Tests
  - Base Tests

.run_cucumber_tests: &run_cucumber_tests |
  mvn -f ${CI_PROJECT_DIR}/pom.xml \
  -Dcucumber.filter.tags="${TEST_TAG}" test

Build Project:
  stage: Build
  allow_failure: false
  script:
    - mvn -f ${CI_PROJECT_DIR}/pom.xml -Dmaven.test.skip=true clean package
```

Рисунок 3.11 – Лістинг частини коду у .gitlab-ci.yml файлі, що містить Docker-image, стадії, скрипт запуску Cucumber-сценаріїв

На рисунку 3.11 видно docker image, який є основою CI/CD контейнеру, в якому відбується тестування. Іншими словами, це заздалегіть налаштоване закрите тестове середовище, в якому відбувається віддалений запуск джоби. Нижче бачимо назви стадій, їх всього три. Далі показано скрипт, який запускає тестові сценарії. Даний скрипт базується одразу на кількох технологіяї: Maven, JUnit, Cucumber. Нижче бачимо Build Project джобу, яка відповідає за зборку проєкту. Джоба належить до стадії Build і не може бути не пройденою, про що свідчить поле allow_failure: false.

```yaml
Smoke:
  variables:
    TEST_TAG: "@Smoke"
  stage: Smoke Tests
  allow_failure: false
  script:
    - *run_cucumber_tests

Student Materials:
  variables:
    TEST_TAG: "@StudentMaterials and not @Disabled"
  stage: Base Tests
  allow_failure: true
  script:
    - *run_cucumber_tests
```

Рисунок 3.12 – Лістинг частини коду у .gitlab-ci.yml файлі, що містить джоби

GitLab CI/CD пайплайн зображено на рисунку 3.13. На рисунку добре видно пайплайн, стадії та джоби. На рисунку 3.14 зображено статус повністю пройденого пайплайну.

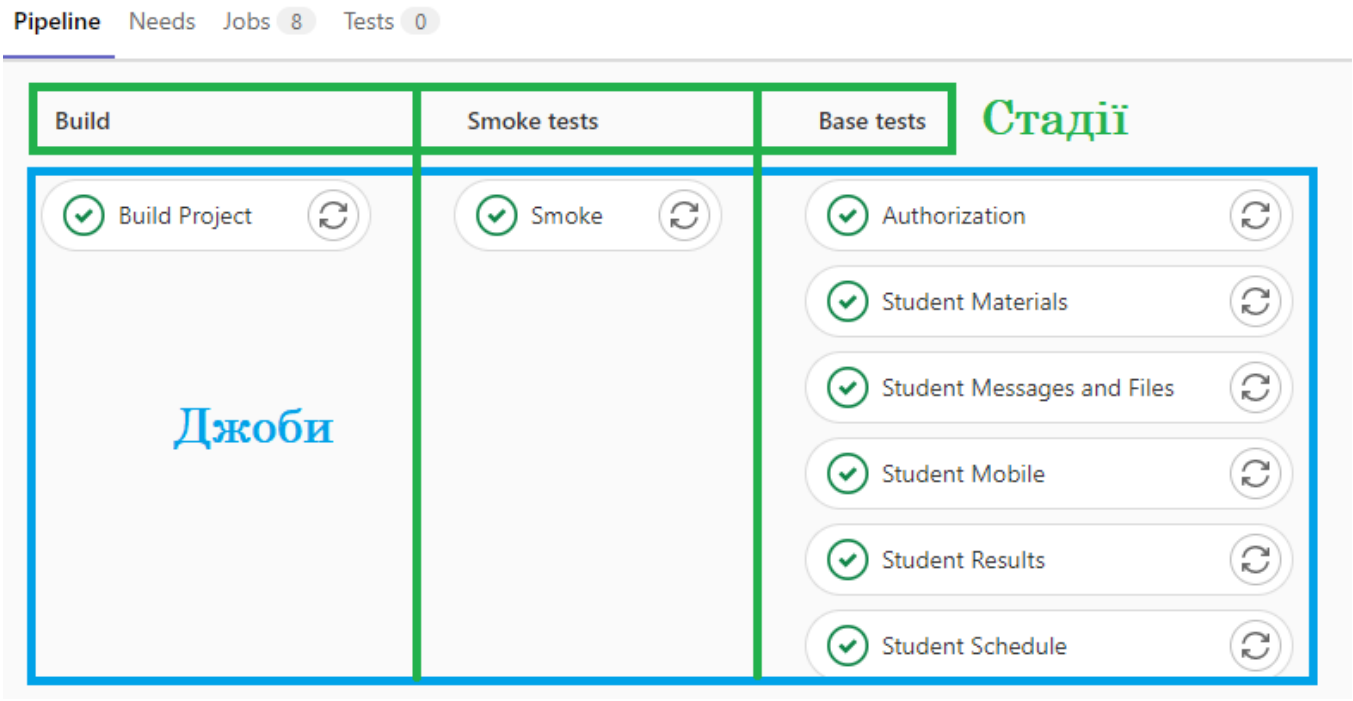


Рисунок 3.12 – GitLab CI/CD пайплайн

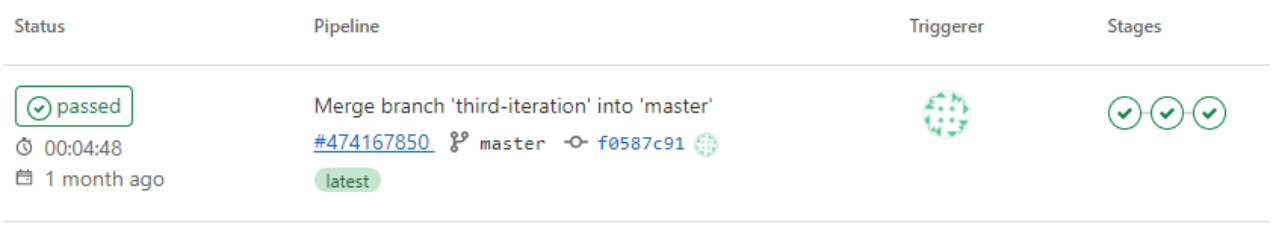


Рисунок 3.13 – Пройдений пайплайн

Як бачимо, на рисунках 3.12 та 3.13 CI/CD пайплайн є пройденим, про що свідчить зелений колір джоб та статус Passed. Це свідчить про те, що усі тести працюють як потрібно і ніяких багів чи помилок виявлено не було. Повний лістинг коду наведено у додатку В.

3.6 Висновки

Отже, у результаті розробки тестового фреймворку здіснено аналіз та вибір мови програмування і середовища розробки. Створено автоматизовані тестові сценарії за допомогою Cucumber інструменту. Реалізовано програму на основі Selenium технології та Page Object патерну. Налаштовано Git та GitLab CI/CD пайплайн для тестування на віддаленій машині у віртуальному середовищі.

4 ІНСТРУКЦІЯ КОРИСТУВАЧА ТА ПРОВЕДЕННЯ ТЕСТУВАННЯ САЙТУ JETIQ

4.1 Інструкція користувача

Оскільки тестовий фреймворк підтримує проведення тестування як на локальній машині, так і на віддаленій за допогою CI/CD пайплайну, прийнятно рішення створити покрокову інструкцію для користувачів. Інструкція повинна містити усю необхідну інформацію для запуску тестів.

Для локального тестування користувач має два варіанти дій. Розглянемо їх більш детально:

– Cucumber підхід. Для цього користувачу потрібно відкрити будь-який .feature файл. Знаходяться вони за таким шляхом: ‘src/test/recources/features”. Кожен такий файл підписаний відповідно до переліку тестових сценаріїв, які він містить (рисунок 4.1).

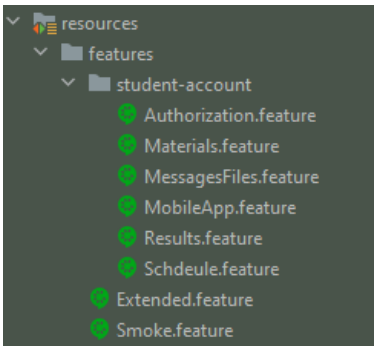


Рисунок 4.1 – Список .feature файлів, які містять тестові сценарії

Далі потрібно налаштувати конфігурацію. Для цього треба відкрити відповідне меню (рисунок 4.2).

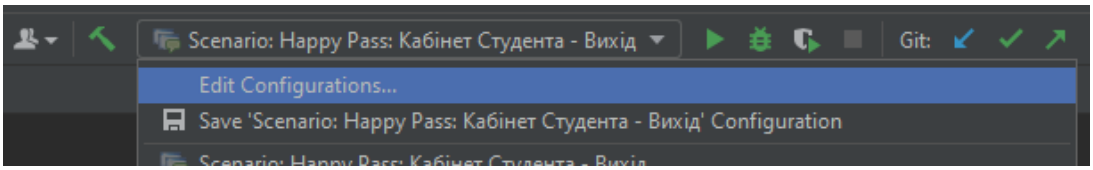


Рисунок 4.2 – Меню конфігурація Cucumber ранеру

У вікні “Run/Debug Configuration” вести в поле “Environments variables” значення “LOCAL_RUN=true” (рисунок 4.3).

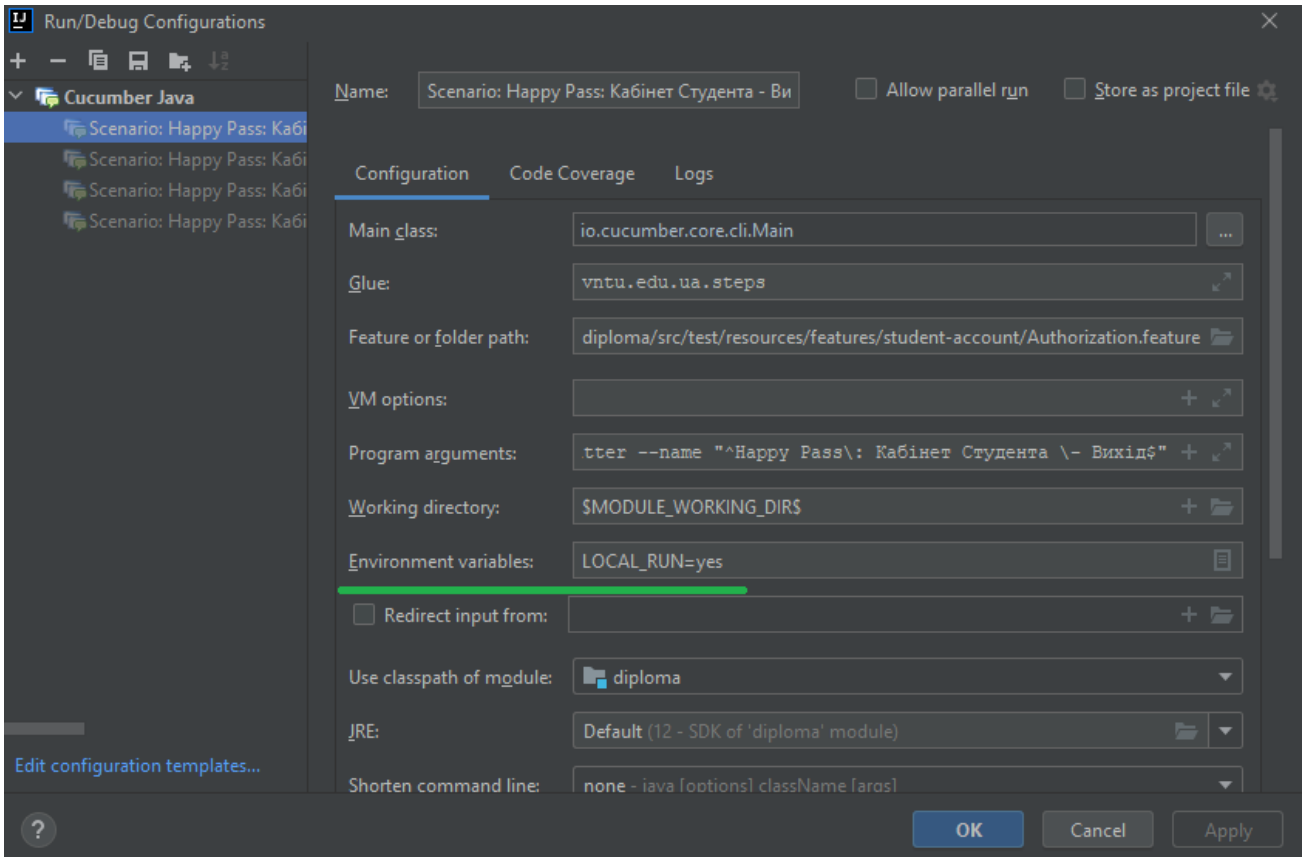


Рисунок 4.3 – Вікно Run/Debug Configuration

Конфігурацію для запуску тесту на основі Cucumber технології зроблено, тепер користувач повинен запустити сценарій. Для цього у .feature файлі, який він відкрив попередньо, треба натиснути на зелений трикутник (рисунок 4.4). Після цього автотест розпочне роботу, внизу відкриється діалогове вікно, в якому буде виводитись інформація щодо перебігу тестування.

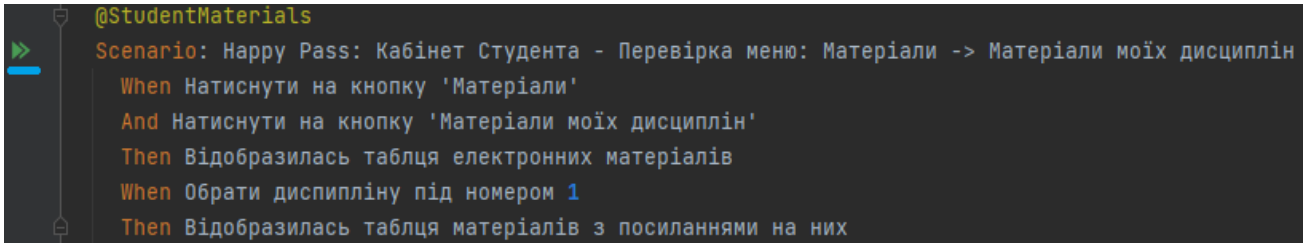


Рисунок 4.4 – Кнопка запуску тестового сценарію

– JUnit підхід. Для цього користувачу потрібно відкрити Runner.java клас, який знаходиться за наступним шляхом: “src/test/java/runner”. Далі у поле “tags” потрібно ввести тег сценарію, який користувач хоче запустити (рисунок 4.5). Приклад тегу сценарію наведено на рисунку 4.4 (жовтим кольором). Конфігурацію потрібно налаштувати по прикладу Cucumber підходу (рисунки 4.2 та 4.3). Далі потрібно запустити ранер, натиснувши на зелений трикутник (рисунок 4.5).

Рисунок 4.5 – JUnit ранер

Далі розглянемо віддалений варіант тестування, який реалізований GitLab CI/CD.

Перш за все користувачу потрібно перейти за посиланням на віддалений репозиторій на GitLab та відкрити меню зліва CI/CD – Pipelines. Важливо, що перед цим йому потрібно запросити доступ до віддаленого репозиторію, оскільки в цілях безпеки було прийнятно рішення закрити доступ звичайним користувачам запускати пайплайн. Далі на цій сторінці потрібно натиснути на синю кнопку “Run Pipeline”, яка розташована у правому верхньому кутку. На новій сторінці потрібно знову натиснути на кнопку “Run Pipeline”. Додаткові тестові параметри вводити не потрібно.

Якщо все було зроблено правильно, повинна була з'явитись сторінка, що зображена на рисунку 4.6.

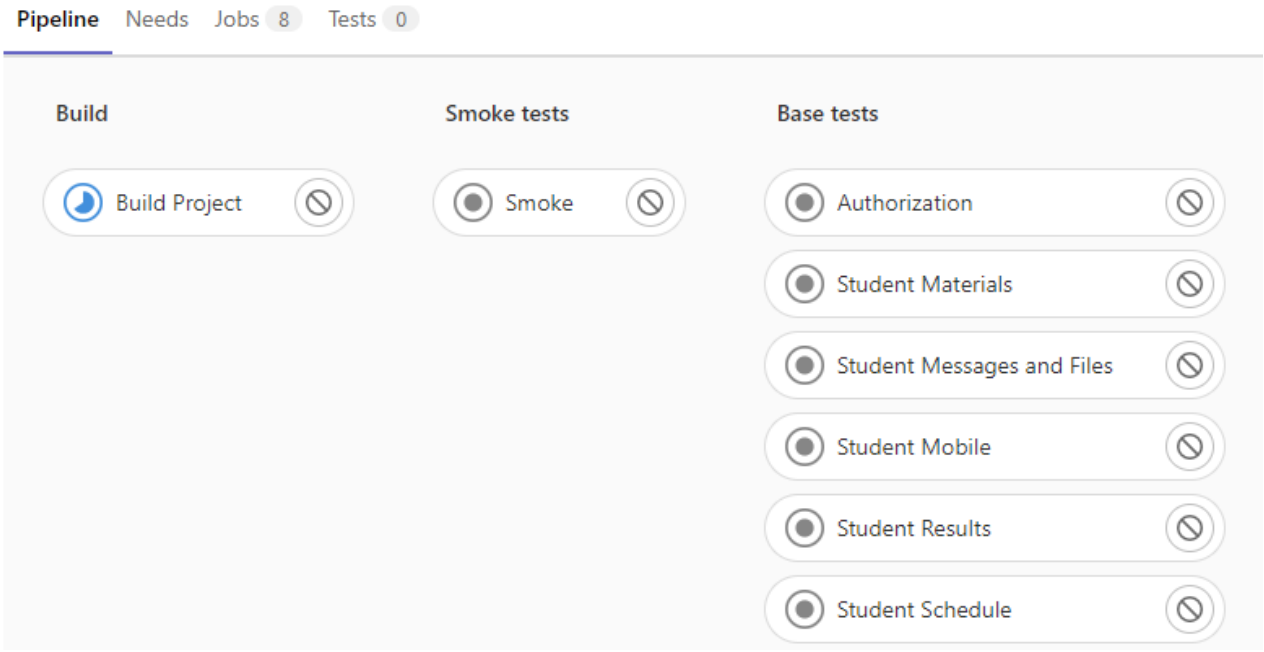


Рисунок 4.6 – Сторінка CI/CD пайплайну

На рисунку 4.6 бачимо, що пайплайн успішно стартував. Якщо Build Project джоба виявиться червоною після виконання, слід відкрити її та подивитись логи джоби, можливо користувач допустив помилку. У випадку, коли Build Project джоба виявиться зеленою, користувач все зробив правильно і CI/CD пайлайн стартував коректно і готовий до проведення тестування.

4.2 Проведення локального тестування

Для проведення локального тестування обрано Cucumber підхід. Нижче наведно технічні характеристики локальної машини.

Таблиця 4.1 – Конфігурація локальної машини

Процесор	AMD Ryzen 5 1600
Об’єм оперативної пам’яті	16 GB DDR4
Об’єм жорсткого диску	1 ТБ (вільно 32%)
Графічний пристрій	AMD Radeon RX580 4GB
Операційна система	Windows 10 64-bit

Результати тестування занесено до таблиці 4.2. Список тест-кейсів наведено у розділі 2.2.

Таблиця 4.2 – Результати локального тестування

Тест-кейс	Статус	Час (с)
Т-1	Пройдено	8.5
Т-2	Пройдено	8.9
Т-3	Пройдено	8.3
Т-4	Пройдено	8.4
Т-5	Пройдено	10.3
Т-6	Пройдено	9.5
Т-7	Пройдено	9.4
Т-8	Пройдено	8.5
Т-9	Пройдено	8.5
Т-10	Пройдено	8.9
Т-11	Пройдено	11.4
Т-12	Пройдено	9.0

Приклад проходження сценаріїв наведено на рисунку 4.7.

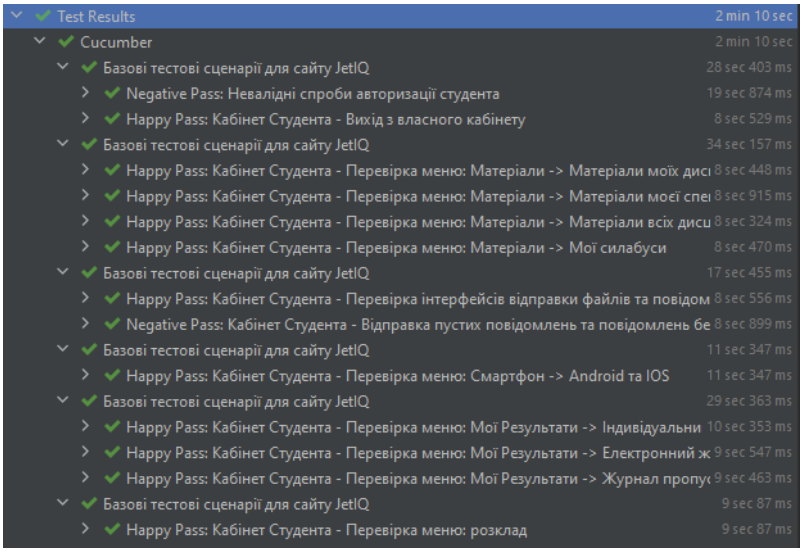


Рисунок 4.7 – Приклад проходження автотесту

Слід зазначити, що кожен .feature файл запущено окремо один від одного, щоб краще було аналізувати результати тестування. В результаті усі тестові сценарії є пройдені. Загальний час проходження базових тестів становить 109.6 секунд, середній – 9.1. Оскільки усі тести пройдено, то можна вважати, що тестування пройшло успішно і нових багів заводити не потрібно.

4.3 Проведення тестування за допомогою CI/CD пайплайну

Для проведення тестування на віддаленій машині скористаємось інструкціює користувача, яка наведена у розділі 4.1. Віддалена машина базується на ОС Linux.

Варто зазначити, що пайплайн містить 8 джоб, серед яких 6 джоб, які містять автоматизовані тест-кейси. Тестові сценарії розподілені по джобами завдяки тегам для зручності. Кожна джоба має свою унікальну навзу, яка відповідає набору тестів, що до неї входять.

На рисунку 4.8 зображено приклад результатів роботи однієї із джоб.

```
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 49.262 s
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:11 min
[INFO] Finished at: 2022-04-09T16:44:09Z
[INFO] ------------------------------------------------------------------------
Cleaning up project directory and file based variables
Job succeeded
```

Рисунок 4.8 – Приклад результатів роботи джоби

Результати тестування зображано на рисунку 4.9 та занесено до таблиці 4.3. Список тест-кейсів наведено у розділі 2.2.

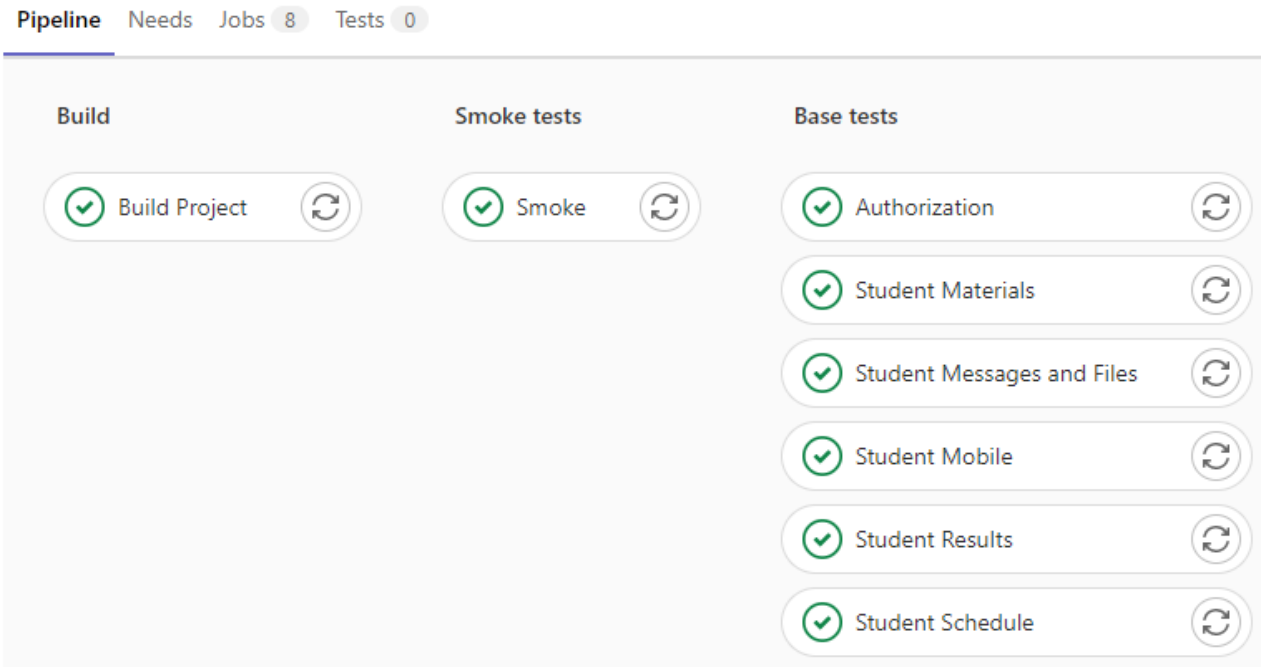


Рисунок 4.9 – Результат проходження тестування на віддаленій машині

Таблиця 4.3 – Результати тестування на віддаленій віртуальній машині

Тест-кейси	Статус	Час (с)
Т-1 Т-2 Т-3 Т-4	Пройдено	71.5
Т-5 Т-6 Т-7	Пройдено	68.1
Т-8	Пройдено	31.4
Т-9 Т-10	Пройдено	50.4
Т-11	Пройдено	43.2
Т-12	Пройдено	40.5

Отже бачимо, що усі тестові сценарії пройдені. Загальний час проходження базових тестів становить 305.1 секунд, середній – 25.4. Оскільки усі тести пройдено, то можна вважати, що тестування пройшло успішно і нових багів заводити не потрібно.

Також слід зазначити, що загальний і середній час тестування збільшившся. Це пояснюється тим, що в кожній джоб перед стартом набору

тестових сценаріїв відбувається зборка проєкту, оскільки кожна джоба є незалежною одна від одної і є по суті незалежним ізольованим середовищем.

4.4 Висновки

У четвертому розділі наведено чітку покрокову інструкцію користувача. Завдяки цій інструкцій будь-який користувач має змогу запустити тести в себе локально або на віддаленій машині.

Також проведено загальне тестування сайту JetIQ на власній локальній машині та за допомогою CI/CD пайплайну. В ході тестування усі сценарії були пройдені, а нових багів виявлено не було.

Створений автоматизований тестовий фреймворк показав, що здатен проводити регресійне та інтеграційне тестування сайту JetIQ.

ВИСНОВКИ

У бакалаврській дипломній роботі розроблено автоматизований тестовий фреймворк на базі технології Selenium WebDriver, який призначений підвищення ефективності інтеграційного тестування сайту JetIQ.

Досліджено особливості розробки автоматизованого тестового фреймворку для інтеграційного UI-тестування. Проаналізовано ряд існуючих інструментів та технологій, які призначені для автоматизації тестування. В результаті ретельного аналізу вирішено розробляти тестовий фреймворк на основі технології Selenium.

Визначено перелік вимог, які необхідно протестувати. Створено 12 тест-кейсів. На основі вимог і тест-кейсів створено матрицю трасування. Аналізуючи дані з цієї матриці, дійшли висновку, що кожна вимога покривається щонайменше одним тест-кейсом і вимоги, які б не покривались жодним тест-кейсом, відсутні. Створено загальну архітектуру та алгоритм роботи програми. Архітектура базується на технології Selenium, патерні Page Object та базових принципах ООП. Також створено загальну схему CD/CD пайплайну і в ході планування вирішено підключити проєкт до Git системи.

У результаті розробки тестового фреймворку здіснено аналіз та вибір мови програмування і середовища розробки. Створено автоматизовані тестові сценарії за допомогою Cucumber інструменту українською мовою. Реалізовано програму на основі Selenium технології та Page Object патерну. Налаштовано Git та GitLab CI/CD пайплайн для тестування на віддаленій машині у віртуальному середовищі.

Наведено чітку покрокову інструкцію користувача. Також проведено загальне тестування сайту JetIQ на локальній машині та за допомогою CI/CD пайплайну. В ході тестування усі сценарії були пройдені, нових багів виявлено не було.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Автоматизоване тестування [Електронний ресурс] – Режим доступу: https://studopedia.com.ua/1_151546_avtomatizovane-testuvannya.html

2. Мушка П., «The ART of Automation», 2019.

3. Автоматизоване тестування [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Автоматизоване_тестування

4. Герг Н., «Test Automation using Selenium WebDriver with Java», 2014.

5. Бульба В., «Керівництво по співбесіді з інженером з автоматизації тестування», 2020.

6. Інструменти автоматизації тестування [Електронний ресурс] – Режим доступу: https://uk.education-wiki.com/9205738-automation-testing-tools

7. Мішальк Б., «Selenium», Springer, 2018.

8. Selenium [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Selenium

9. Бандана О., «Selenium Testing Interview Q&A», 2018.

10. Кочіаро К., «Selenium Framework Design in DDT», Packt Publishing, 2018.

11. «Selenium — лютый набор инструментов для разработчиков» [Електронний ресурс] – Режим доступу: https://promdevelop.com/technologies/selenium/

12. Симон А. Д., "Особливості застосування Selenium технології при розробці фреймворку для автоматизованого UI-тестування", на науково-технічній конференції підрозділів Вінницького національного технічного університету, Вінниця, 2022.

13. Симон А. Д., "Використання технології WebDriver для розробки тестового автоматизованого фреймворку", на XXI Всеукраїнській науково-технічній конференції молодих вчених, аспірантів і студентів, Одеса, 2022.

14. Автоматизуємо тестування: коли, навіщо і кому це потрібно [Електронний ресурс] – Режим доступу: https://newline.tech/test-automation-when-why-and-who-needs-it_uk/

15. Micro Focus Unified Functional Testing [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Micro_Focus_Unified_Functional_Testing#-:~:text=Micro%20Focus%20Unified%20Functional%20Testing,of%20the%20application%20under%20test

16. Чандрасекара Ч. «Hands-On Functional Test Automation», 2019.

17. Apache JMeter [Електронний ресурс] – Режим доступу: https://jmeter.apache.org/

18. Postman [Електронний ресурс] – Режим доступу: https://www.postman.com/

19. Тест-кейси [Електронний ресурс] – Режим доступу: https://training.qatestlab.com/blog/course-materials/preconditions-test-cases/

20. «Матрица трассабилити» [Електронний ресурс] – Режим доступу: https://habr.com/ru/company/simbirsoft/blog/412677/

21. Хосе Б., «Test Automation: A manager's guide», 2021.

22. «Архитектура. Основные элементы.» [Електронний ресурс] – Режим доступу: https://github.com/COMAQA/Selenium-Webdriver-lectures/blob/master/-page_object_pattern_arhitektura_testovogo_proekta/arhitektura_osnovnie_elementi_primeri.md

23. Ангелов А. «Design Patterns for High-Quality Automated Tests: Clean Code for Bulletproof Tests», 2021.

24. Худа А., «Cucumber Interview Questions and Answers», 2020.

25. PageObject [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/PageObject

26. «Автоматизированное тестирование для CI/CD» [Електронний ресурс] – Режим доступу: https://www.jetbrains.com/ru-ru/teamcity/ci-cd-guide/automated-testing/

27. Хакет М. «Agile Test Automation», 2019.

28. Пінакін А., «Page Object Model using Selenium WebDriver & Java», 2018.

29. Керована поведінкою розробка [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Керована_поведінкою_розробка

ДОДАТКИ

Додаток А. Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедрою ПЗ
д.т.н., проф. О. Н. Романюк
_____
31 березня 2022 р.

Технічне завдання
на бакалаврську дипломну роботу «Розробка фреймворку для інтеграційного тестування сайту JetIQ з використанням технології Selenium WebDriver» за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
_______ к.т.н., доц. Романюк О. В.
31 березня 2022 р.
Виконав:
_____студент гр. 2ПІ-18б Симон А. Д.
31 березня 2022 р.

Вінниця – 2022 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка фреймворку для інтеграційного тестування сайту JetIQ з використанням технології Selenium WebDriver».

Галузь застосування – програмне забезпечення для автоматизованого інтеграційного тестування.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ № 66 від 24 березня 2022 р. ректора по ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Метою даної роботи є підвищення ефективності тестування сайту JetIQ шляхом створення тестового фреймворку для його інтеграційного тестування.

Призначення роботи – розробка програмного продукту для автоматизованого інтеграційного тестування сайту JetIQ.

4. Вихідні дані для проведення НДР

Основні літературні джерела, на основі яких буде виконуватись БДР:

1. Бандана О., «Selenium Testing Interview Q&A», 2018.

2. Кочіаро К., «Selenium Framework Design in DDT», Packt Publishing, 2018.

3. Бульба В., «Керівництво по співбесіді з інженером з автоматизації тестування», 2020.

4. Худа А., «Cucumber Interview Questions and Answers», 2020.

5. Пінакін А., «Page Object Model using Selenium WebDriver & Java», 2018.

5. Технічні вимоги

Метод написання тестових сценаріїв – технологія Cucumber; алгоритми реалізації – технологія Selenium WebDriver; вхідні дані – тестові параметри; вихідні дані – результати тестування, логи, скріншоти; середовище розробки – IntelliJ IDEA; мова розробки – Java; операційна система – Windows 10 та Linux; Git система – GitLab; CI/CD пайплайн – GitLab.

6. Конструктивні вимоги.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до БДР;
– технічне завдання;
– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз галузі автоматизованого інтеграційного тестування	26.03.2022 – 08.04.2022	Виконано
2	Розробка тестових сценаріїв, матриці трасування та архітектури фреймворку	09.04.2022 – 20.04.2022	Виконано
3	Розробка алгоритму роботи тестового сценарію та CI/CD схеми фреймворку	21.04.2022 – 30.04.2022	Виконано
4	Розробка програмних компонент для автоматизованого тестового фреймворку	01.05.2022 – 19.05.2022	Виконано
5	Розробка інструкції користувача та проведення тестування сайту JetIQ	20.05.2022 – 29.05.2022	Виконано
6	Оформлення матеріалів до захисту БДР	30.05.2022 – 10.06.2022	Виконано

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської дипломної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття бакалаврської дипломної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б. Протокол перевірки кваліфікаційної роботи
на наявність текстових запозичень

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Розробка фреймворку для інтеграційного тестування сайту JetIQ з використанням технології Selenium WebDriver»

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Романюк О. В.

Оригінальність	**81,2%**
Схожість	18,8%

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Симон А. Д.

Керівник роботи _____ Романюк О. В.

Додаток В. Лістинг програми

```
@Smoke
Feature: Тестові смок сценарії для сайту JetIQ

  Background:
    Given Відкрити домашню сторінку JetIQ

   Scenario: Happy Pass: Перевірка роботи домашньої сторінки
     Then Навігаційні посилання відображаються на сторінці
     And Розділи меню верхньої панелі відображаються на сторінці

  Scenario: Happy Pass: Вхід студента у власний кабінет
    And Натиснути на кноку 'Вхід'
    And Ввести 'valid' значення в поле логіну
    And Натиснути на кнопку 'Далі'
    And Ввести 'valid' значення в поле паролю
    And Натиснути на кноку 'Ввійти'
    Then Студента звати 'Симон Андрій Дмитрович'


Feature: Базові тестові сценарії для сайту JetIQ

  Background:
    Given Відкрити домашню сторінку JetIQ
    And Натиснути на кноку 'Вхід'

  @Authorization
  Scenario Outline: Negative Pass: Невалідні спроби авторизації студента
    When Ввести '<typeFirst>' значення в поле логіну
    And Натиснути на кнопку 'Далі'
    And Ввести '<typeSecond>' значення в поле паролю
    And Натиснути на кноку 'Ввійти'
    Then Вискочило повідомлення про неправильний логін або пароль

    Examples:
      | typeFirst | typeSecond |
      | invalid   | invalid    |
      | valid     | invalid    |
      | invalid   | valid      |

  @Authorization
  Scenario: Happy Pass: Кабінет Студента - Вихід з власного кабінету
    When Ввести 'valid' значення в поле логіну
    And Натиснути на кнопку 'Далі'
    And Ввести 'valid' значення в поле паролю
    And Натиснути на кноку 'Ввійти'
    Then Студента звати 'Симон Андрій Дмитрович'
    When Натиснути на кноку 'Вихід'
    Then Навігаційні посилання відображаються на сторінці
```

```
    And Розділи меню верхньої панелі відображаються на сторінці

Feature: Базові тестові сценарії для сайту JetIQ

  Background:
    Given Відкрити домашню сторінку JetIQ
    And Натиснути на кноку 'Вхід'
    And Ввести 'valid' значення в поле логіну
    And Натиснути на кнопку 'Далі'
    And Ввести 'valid' значення в поле паролю
    And Натиснути на кноку 'Ввійти'

  @StudentMaterials
  Scenario: Happy Pass: Кабінет Студента - Перевірка меню: Матеріали ->
Матеріали моїх дисциплін
    When Натиснути на кнопку 'Матеріали'
    And Натиснути на кнопку 'Матеріали моїх дисциплін'
    Then Відобразилась таблця електронних матеріалів
    When Обрати диспипліну під номером 1
    Then Відобразилась таблця матеріалів з посиланнями на них

  @StudentMaterials
  Scenario: Happy Pass: Кабінет Студента - Перевірка меню: Матеріали ->
Матеріали моєї спеціальності
    When Натиснути на кнопку 'Матеріали'
    And Натиснути на кнопку 'Матеріали моєї спеціальності'
    And Обрати будь який семестр
    Then Відобразилась таблця електронних матеріалів
    When Обрати диспипліну під номером 1
    Then Відобразилась таблця матеріалів з посиланнями на них

  @StudentMaterials
  Scenario: Happy Pass: Кабінет Студента - Перевірка меню: Матеріали ->
Матеріали всіх дисциплін
    When Натиснути на кнопку 'Матеріали'
    And Натиснути на кнопку 'Матеріали всіх дисцпиплін'
    Then 'Електронні навчальні матеріали' table is appeared

  @StudentMaterials
  Scenario: Happy Pass: Кабінет Студента - Перевірка меню: Матеріали ->
Мої силабуси
    When Натиснути на кнопку 'Мої силабуси'
    Then Відобразилась таблця матеріалів з посиланнями на них


Feature: Базові тестові сценарії для сайту JetIQ

  Background:
    Given Відкрити домашню сторінку JetIQ
    And Натиснути на кноку 'Вхід'
    And Ввести 'valid' значення в поле логіну
```

```
    And Натиснути на кнопку 'Далі'
    And Ввести 'valid' значення в поле паролю
    And Натиснути на кноку 'Ввійти'

  @StudentMessagesFiles
  Scenario: Happy Pass: Кабінет Студента - Перевірка інтерфейсів
відправки файлів та повідомлень викладачу
    When Натиснути на кнопку 'Повідомлення'
    And Натиснути на кнопку 'Відправити повідомлення'
    Then Відобразився інтферейс 'Повідомлення викладачу'
    When Повернутись на вкладку назад і натиснути на кнопку 'Надіслати
файл'
    Then Відобразився інтферейс 'Надіслати файл викладачу'

  @StudentMessagesFiles
  Scenario: Negative Pass: Кабінет Студента - Відправка пустих
повідомлень та повідомлень без файлів викладачу
    When Натиснути на кнопку 'Повідомлення'
    And Натиснути на кнопку 'Відправити повідомлення'
    Then Відобразився інтферейс 'Повідомлення викладачу'
    When Натиснути на кнопку 'Відправити'
    Then Відобразилось повідомлення про помилку відправки пустого
повідомлення
    When Повернутись на вкладку назад і натиснути на кнопку 'Надіслати
файл'
    Then Відобразився інтферейс 'Надіслати файл викладачу'
    When Натиснути на кнопку 'Відправити'
    Then Відобразився попап із помилкою відправки файла

Feature: Базові тестові сценарії для сайту JetIQ

  Background:
    Given Відкрити домашню сторінку JetIQ
    And Натиснути на кноку 'Вхід'
    And Ввести 'valid' значення в поле логіну
    And Натиснути на кнопку 'Далі'
    And Ввести 'valid' значення в поле паролю
    And Натиснути на кноку 'Ввійти'

  @StudentMobile
  Scenario: Happy Pass: Кабінет Студента - Перевірка меню: Смартфон ->
Android та IOS
    When Навести на кнопку 'Смартфон'
    And Натиснути на кнопку 'Андроїд'
    Then Відкрилась сторінка додатка у Play Market
    When Повернутись назад та натиснути на кнопку 'IOS'
    Then Відкрилась сторінка додатка у Apple Store

Feature: Базові тестові сценарії для сайту JetIQ

  Background:
```

```
    Given Відкрити домашню сторінку JetIQ
    And Натиснути на кноку 'Вхід'
    And Ввести 'valid' значення в поле логіну
    And Натиснути на кнопку 'Далі'
    And Ввести 'valid' значення в поле паролю
    And Натиснути на кноку 'Ввійти'
    When Натиснути на кноку 'Мої Результати'
    Then Відкрилась сторінка із результатами тестів і посиланнями

  @StudentResults
  Scenario: Happy Pass: Кабінет Студента - Перевірка меню: Мої Результати
-> Індивідуальний план студента
    When Натиснути на кноку 'Індивідуальний план студента'
    Then Відкрилась сторінка із індивідуальним навчальним планом студента

  @StudentResults
  Scenario: Happy Pass: Кабінет Студента - Перевірка меню: Мої Результати
-> Електронний журнал успішності
    When Натиснути на кноку 'Електронний журнал успішності'
    Then Відкрилась сторінка із електронним журналом успішності

  @StudentResults
  Scenario: Happy Pass: Кабінет Студента - Перевірка меню: Мої Результати
-> Журнал пропусків та заборгованостей
    When Натиснути на кноку 'Журнал пропусків'
    Then Відкрилась сторінка із журналом пропусків
    When Повернутись назад і натиснути на кноку 'Журнал заборгованостей'
    Then Відкрилась сторінка із журналом заборгованостей

Feature: Базові тестові сценарії для сайту JetIQ

  Background:
    Given Відкрити домашню сторінку JetIQ
    And Натиснути на кноку 'Вхід'
    And Ввести 'valid' значення в поле логіну
    And Натиснути на кнопку 'Далі'
    And Ввести 'valid' значення в поле паролю
    And Натиснути на кноку 'Ввійти'

  @StudentSchedule
  Scenario: Happy Pass: Кабінет Студента - Перевірка меню: розклад
    When Натиснути на кнопку 'Розклад'
    Then Відобразилась таблиця 'Розклад'
package vntu.edu.ua.runner;

import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
import org.junit.runner.RunWith;

import static vntu.edu.ua.utils.Constants.FEATURES;
import static vntu.edu.ua.utils.Constants.GLUE;
```

```
@RunWith(Cucumber.class)
@CucumberOptions(
        glue = GLUE,
        features = FEATURES,
        plugin = {"pretty", "html:target/cucumber"})
public class Runner { }

image: markhobson/maven-chrome:latest

stages:
  - Build
  - Smoke Tests
  - Base Tests
  - Extended Tests

.run_cucumber_tests: &run_cucumber_tests |
  mvn -f ${CI_PROJECT_DIR}/pom.xml \
  -Dcucumber.filter.tags="${TEST_TAG}" test

Build Project:
  stage: Build
  allow_failure: false
  script:
    - mvn -f ${CI_PROJECT_DIR}/pom.xml -Dmaven.test.skip=true clean
package

Smoke:
  variables:
    TEST_TAG: "@Smoke"
  stage: Smoke Tests
  allow_failure: false
  script:
    - *run_cucumber_tests

Student Materials:
  variables:
    TEST_TAG: "@StudentMaterials and not @Disabled"
  stage: Base Tests
  allow_failure: true
  script:
    - *run_cucumber_tests

Student Schedule:
  variables:
    TEST_TAG: "@StudentSchedule and not @Disabled"
  stage: Base Tests
  allow_failure: true
  script:
    - *run_cucumber_tests
```

```
Student Results:
  variables:
    TEST_TAG: "@StudentResults and not @Disabled"
  stage: Base Tests
  allow_failure: true
  script:
    - *run_cucumber_tests

Student Messages and Files:
  variables:
    TEST_TAG: "@StudentMessagesFiles and not @Disabled"
  stage: Base Tests
  allow_failure: true
  script:
    - *run_cucumber_tests

Student Mobile:
  variables:
    TEST_TAG: "@StudentMobile and not @Disabled"
  stage: Base Tests
  allow_failure: true
  script:
    - *run_cucumber_tests

Authorization:
  variables:
    TEST_TAG: "@Authorization and not @Disabled"
  stage: Base Tests
  allow_failure: true
  script:
    - *run_cucumber_tests

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>vntu.edu.ua</groupId>
  <artifactId>diploma</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <java.version>11</java.version>
    <encoding>UTF-8</encoding>

    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
    <maven-compiler-plugin.version>3.7.0</maven-compiler-plugin.version>
    <maven.surefire.version>2.22.1</maven.surefire.version>
```

```
</properties>

<dependencies>
  <dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>3.141.59</version>
  </dependency>
  <dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>6.8.1</version>
  </dependency>
  <dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-junit</artifactId>
    <version>6.8.1</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
  </dependency>
  <dependency>
    <groupId>io.github.bonigarcia</groupId>
    <artifactId>webdrivermanager</artifactId>
    <version>4.3.1</version>
  </dependency>
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.20</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.2.3</version>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>${maven-compiler-plugin.version}</version>
      <configuration>
        <source>${java.version}</source>
        <target>${java.version}</target>
```

```
            <encoding>${encoding}</encoding>
          </configuration>
        </plugin>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-surefire-plugin</artifactId>
          <version>${maven.surefire.version}</version>
          <configuration>
            <argLine>-Xmx32768m</argLine>
            <includes>
              <include>**/Runner.java</include>
            </includes>
          </configuration>
        </plugin>
      </plugins>
    </build>

</project>
```

```
package vntu.edu.ua.manager;

import lombok.AllArgsConstructor;
import org.openqa.selenium.WebDriver;
import vntu.edu.ua.pages.*;

@AllArgsConstructor
public class PageFactoryManager {

    private final WebDriver webDriver;

    public HomePage getHomePage() {
        return new HomePage(webDriver);
    }

    public SignInPage getSignInPage() {
        return new SignInPage(webDriver);
    }

    public StudentHomePage getUserHomePage() {
        return new StudentHomePage(webDriver);
    }

    public MaterialsPage getMaterialsPage() {
        return new MaterialsPage(webDriver);
    }

    public MessagesFilesPage getMessagesFilesPage() {
        return new MessagesFilesPage(webDriver);
    }

    public MyResultsPage getMyResultsPage() {
```

```
        return new MyResultsPage(webDriver);
    }
}

package vntu.edu.ua.pages;

import lombok.Getter;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.PageFactory;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

import java.util.ArrayList;
import java.util.List;

import static vntu.edu.ua.utils.Constants.*;

public abstract class BasePage {

    @Getter
    protected WebDriver webDriver;

    public BasePage(WebDriver webDriver) {
        this.webDriver = webDriver;
        PageFactory.initElements(webDriver, this);
    }

    public String getActiveTab(int tab) {
        List<String> tabs = new
ArrayList<>(webDriver.getWindowHandles());
        return tabs.get(tab);
    }

    public void waitForPageLoadComplete() {
        getWebDriverWait().until(driver -> ((JavascriptExecutor) driver)
                .executeScript(PAGE_LOAD_SCRIPT).equals(COMPLETE));
    }

    public void waitForElementVisibility(WebElement webElement) {

getWebDriverWait().until(ExpectedConditions.visibilityOf(webElement));
    }

    public void waitForElementClickable(WebElement webElement) {

getWebDriverWait().until(ExpectedConditions.elementToBeClickable(webEleme
nt));
    }
```

```
    public boolean isElementDisplayed(WebElement webElement) {
        return webElement.isDisplayed();
    }

    private WebDriverWait getWebDriverWait() {
        return new WebDriverWait(webDriver, DEFAULT_TIMEOUT);
    }
}
```

```
package vntu.edu.ua.pages;

import lombok.Getter;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;

import java.util.List;

import static vntu.edu.ua.utils.Constants.HOME_PAGE_URL;

@Getter
public class HomePage extends BasePage {

    @FindBy(xpath =
"//nav[@id='mainav']//a[@href='https://my.vntu.edu.ua/user/']")
    private WebElement sighInButton;

    @FindBy(xpath = "//nav[@id='mainav']/ul/li")
    private List<WebElement> navigationLinks;

    @FindBy(xpath = "//div[@class='exampl1']")
    private List<WebElement> topBarMenuSections;

    public HomePage(WebDriver webDriver) {
        super(webDriver);
    }

    public void openHomePage() {
        webDriver.get(HOME_PAGE_URL);
    }

    public void clickOnSighInButton() {
        waitForElementClickable(sighInButton);
        sighInButton.click();
    }
}
```

```
package vntu.edu.ua.pages;

import lombok.Getter;
import org.openqa.selenium.WebDriver;
```

```
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;

import java.util.List;

@Getter
public class MaterialsPage extends BasePage {


    @FindBy(xpath = "//h4/a[@href='/method/subj2.php']")
    private WebElement myDisciplinesMaterialsButton;

    @FindBy(xpath = "//h4/a[@href='/method/sem2.php']")
    private WebElement mySpecialtyMaterialsButton;

    @FindBy(xpath = "//h4/a[@href='/method/index2.php']")
    private WebElement allMaterialsButton;

    @FindBy(xpath = "//tr//td/a[not(contains(@href, 'teacher'))]")
    private List<WebElement> subjectsInMaterialTable;

    @FindBy(xpath = "//table//th[text()='Код.']")
    private WebElement teacherMaterialsTable;

    @FindBy(xpath = "//a[text()='4 курс 8 триместр ']")
    private WebElement searchBySemesterMaterials;

    @FindBy(xpath = "//table[@class='table table-hover table-
condensed']")
    private WebElement electronicMaterialsTable;

    @FindBy(xpath = "//h4[contains(text(), 'Ефективність')]")
    private WebElement allMaterialsTablePage;

    public MaterialsPage(WebDriver webDriver) {
        super(webDriver);
    }

    public void clickMyDisciplinesMaterialsButton() {
        waitForElementClickable(myDisciplinesMaterialsButton);
        myDisciplinesMaterialsButton.click();
    }

    public void clickOnSubjectsInMaterialTable(int value) {
        WebElement subject = subjectsInMaterialTable.get(value - 1);
        waitForElementVisibility(subject);
        waitForElementClickable(subject);
        subject.click();
    }

    public void clickOnMySpecialtyMaterialsButton() {
```

```
        waitForElementClickable(mySpecialtyMaterialsButton);
        mySpecialtyMaterialsButton.click();
    }

    public void clickOnSearchBySemesterMaterials() {
        waitForElementClickable(searchBySemesterMaterials);
        searchBySemesterMaterials.click();
    }

    public void clickOnAllMaterialsButton() {
        waitForElementClickable(allMaterialsButton);
        allMaterialsButton.click();
    }
}

package vntu.edu.ua.pages;

import lombok.Getter;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;

@Getter
public class MessagesFilesPage extends BasePage {

    @FindBy(xpath = "//a[@class='bd-linkbutton-2  bd-button-13  bd-own-
margins bd-content-element']")
    private WebElement sendMessageButton;

    @FindBy(xpath = "//h1[text()='Повідомлення викладачу у ПК і на
електронну пошту.']")
    private WebElement messageToProfPage;

    @FindBy(xpath = "//a[@class='bd-linkbutton-5  bd-button-52  bd-own-
margins bd-content-element']")
    private WebElement sendFileButton;

    @FindBy(xpath = "//form[@enctype='multipart/form-data']")
    private WebElement sendFilePage;

    @FindBy(xpath = "//input[@value='Надіслати']")
    private WebElement sendMessageToProfButton;

    @FindBy(xpath = "//font[contains(text(), 'Відсутній текст')]")
    private WebElement messageIsEmptyError;

    public MessagesFilesPage(WebDriver webDriver) {
        super(webDriver);
    }

    public void clickOnSendMessageButton() {
```

```
        waitForElementClickable(sendMessageButton);
        sendMessageButton.click();
    }

    public void clickOnSendFileButton() {
        webDriver.switchTo().window(getActiveTab(0));
        waitForElementClickable(sendFileButton);
        sendFileButton.click();
    }

    public void clickOnSendMessageToProfButton() {
        waitForElementClickable(sendMessageToProfButton);
        sendMessageToProfButton.click();
    }
}
```

```
package vntu.edu.ua.pages;

import lombok.Getter;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;

import java.util.List;

@Getter
public class MyResultsPage extends BasePage {

    @FindBy(xpath = "//th[text()='Назва тесту']")
    private WebElement tableWithTestResults;

    @FindBy(xpath = "//a[text()='Індивідуальний план студента']")
    private WebElement studentPlanButton;

    @FindBy(xpath = "//th[text()='Дисципліна']")
    private List<WebElement> studentPlanPage;

    @FindBy(xpath = "//a[text()='Електронний журнал успішності']")
    private WebElement logOfRateButton;

    @FindBy(xpath = "//td[text()='Модулі']")
    private List<WebElement> logOfRatesTable;

    @FindBy(xpath = "//a[text()='Журнал пропусків']")
    private WebElement passLogButton;

    @FindBy(xpath = "//td[text()='Дисципліна']")
    private WebElement passLogTable;

    @FindBy(xpath = "//a[text()='Журнал заборгованостей']")
    private WebElement debtLogButton;
```

```
    @FindBy(xpath = "//h1[contains(text(), 'Ceci')]")
    private WebElement debtLogInfo;

    public MyResultsPage(WebDriver webDriver) {
        super(webDriver);
    }

    public void clickOnStudentPlanButton() {
        waitForElementClickable(studentPlanButton);
        studentPlanButton.click();
    }

    public void clickOnLogOfRateButton() {
        waitForElementClickable(logOfRateButton);
        logOfRateButton.click();
    }

    public void clickOnPassLogButton() {
        waitForElementClickable(passLogButton);
        passLogButton.click();
        webDriver.switchTo().window(getActiveTab(2));
    }

    public void clickOnDebtLogButton() {
        webDriver.switchTo().window(getActiveTab(1));
        waitForElementClickable(debtLogButton);
        debtLogButton.click();
        webDriver.switchTo().window(getActiveTab(3));
    }
}
package vntu.edu.ua.pages;

import lombok.Getter;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;

@Getter
public class SignInPage extends BasePage {

    @FindBy(xpath = "//input[@id='user_field']")
    private WebElement usernameField;

    @FindBy(xpath = "//input[@id='pwd_field']")
    private WebElement passwordField;

    @FindBy(xpath = "//div[@class='login-header']")
    private WebElement loginHeader;

    @FindBy(xpath = "//button[@id='do-verify-login']")
```

```
    private WebElement nextButton;

    @FindBy(xpath = "//button[@id='do-verify-password']")
    private WebElement enterButton;

    @FindBy(xpath = "//div[@class='alert alert-warning']")
    private WebElement authErrorMessage;

    public SignInPage(WebDriver webDriver) {
        super(webDriver);
    }

    public void setValueToUsernameField(String value) {
        waitForElementVisibility(usernameField);
        waitForElementClickable(usernameField);
        usernameField.sendKeys(value);
    }

    public void setValueToPasswordField(String value) {
        waitForElementVisibility(passwordField);
        waitForElementClickable(passwordField);
        passwordField.sendKeys(value);
    }

    public void clickOnLoginHeader() {
        waitForElementClickable(loginHeader);
        loginHeader.click();
    }

    public void clickOnNextButton() {
        waitForElementClickable(nextButton);
        nextButton.click();
    }

    public void clickOnEnterButton() {
        waitForElementClickable(enterButton);
        enterButton.click();
    }
}

package vntu.edu.ua.pages;

import lombok.Getter;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.interactions.Actions;
import org.openqa.selenium.support.FindBy;

@Getter
public class StudentHomePage extends BasePage {
```

```
@FindBy(xpath = "//h2[contains(@class, 'bd-textblock-50')]")
private WebElement studentName;

@FindBy(xpath = "//a[@class='menufont' and text()='Матеріали']")
private WebElement materialsMenu;

@FindBy(xpath = "//a[text()='Мої силабуси']")
private WebElement mySilabusesMenu;

@FindBy(xpath = "//a[text()='Розклад']")
private WebElement scheduleMenu;

@FindBy(xpath = "//form[@name='myform']")
private WebElement scheduleTable;

@FindBy(xpath = "//a[text()='Мої результати']")
private WebElement myResultsMenu;

@FindBy(xpath = "//a[text()='Повідомлення']")
private WebElement messagesMenu;

@FindBy(xpath = "//a[text()='Смартфон']")
private WebElement mobileMenu;

@FindBy(xpath = "//a[text()='Android']")
private WebElement androidButton;

@FindBy(xpath = "//span[text()='JetIQ-Student']")
private WebElement playMarketPage;

@FindBy(xpath = "//a[text()='iOS 14+']")
private WebElement appleButton;

@FindBy(xpath = "//h1[text()='JetIQ-Student']")
private WebElement appleMarketPage;

@FindBy(xpath = "//a[text()='Вихід']")
private WebElement exitButton;

public StudentHomePage(WebDriver webDriver) {
    super(webDriver);
}

public void clickOnMaterialsMenu() {
    waitForElementClickable(materialsMenu);
    materialsMenu.click();
}

public void clickOnScheduleMenu() {
    waitForElementClickable(scheduleMenu);
    scheduleMenu.click();
```

```
        webDriver.switchTo().window(getActiveTab(1));
    }

    public void clickOnMySilabusesMenu() {
        Actions actions = new Actions(webDriver);
        actions.moveToElement(materialsMenu).build().perform();
        waitForElementClickable(mySilabusesMenu);
        mySilabusesMenu.click();
        webDriver.switchTo().window(getActiveTab(1));
    }

    public void clickOnMyResultsMenu() {
        waitForElementClickable(myResultsMenu);
        myResultsMenu.click();
        webDriver.switchTo().window(getActiveTab(1));
    }

    public void clickOnMessagesMenu() {
        waitForElementClickable(messagesMenu);
        messagesMenu.click();
    }

    public void clickOnMobileMenu() {
        waitForElementVisibility(mobileMenu);
        waitForElementClickable(mobileMenu);
        new
Actions(webDriver).moveToElement(mobileMenu).build().perform();
    }

    public void clickOnAndroidButton() {
        waitForElementVisibility(androidButton);
        waitForElementClickable(androidButton);
        androidButton.click();
    }

    public void clickOnAppleButton() {
        webDriver.switchTo().window(getActiveTab(0));
        clickOnMobileMenu();
        waitForElementVisibility(appleButton);
        waitForElementClickable(appleButton);
        appleButton.click();
    }

    public void clickOnExitButton() {
        waitForElementClickable(exitButton);
        exitButton.click();
    }
}

package vntu.edu.ua.steps;
```

```
import io.cucumber.java.en.And;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import lombok.extern.slf4j.Slf4j;
import org.openqa.selenium.WebElement;

import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

@Slf4j
public class AssertsSteps extends BaseSteps {

    @Then("Навігаційні посилання відображаються на сторінці")
    public void checkNavLinks() {
        log.info("Start checking navigation links...");
        for (WebElement webElement : homePage.getNavigationLinks()) {
            assertTrue(homePage.isElementDisplayed(webElement));
        }
        log.info("Navigation links are displayed on page");
    }

    @And("Розділи меню верхньої панелі відображаються на сторінці")
    public void checkTopBarMenuSections() {
        log.info("Start checking top bar menu sections...");
        for (WebElement webElement : homePage.getTopBarMenuSections()) {
            assertTrue(homePage.isElementDisplayed(webElement));
        }
        log.info("Top bar menu sections are displayed on page");
    }

    @Then("Студента звати {string}")
    public void checkStudentName(String studentName) {
        studentHomePage.waitForPageLoadComplete();

studentHomePage.waitForElementVisibility(studentHomePage.getStudentName()
);
        log.info("Start checking student name...");

assertTrue(studentHomePage.getStudentName().getText().replace("\n", "
").contains(studentName));
    }

    @Then("Вискочило повідомлення про неправильний логін або пароль")
    public void checkAuthErrorMessage() {

signInPage.waitForElementVisibility(signInPage.getAuthErrorMessage());
        log.info("Start checking error message...");

assertTrue(signInPage.getAuthErrorMessage().getText().contains("Неправиль
ний"));
    }
```

```
    @Then("Відобразилась таблця матеріалів з посиланнями на них")
    public void checkMaterialLinks() {
        materialsPage.waitForPageLoadComplete();

materialsPage.waitForElementVisibility(materialsPage.getTeacherMaterialsT
able());

assertTrue(materialsPage.isElementDisplayed(materialsPage.getTeacherMater
ialsTable()));
    }

    @Then("Відобразилась таблця електронних матеріалів")
    public void checkElectronicMaterialsTable() {
        materialsPage.waitForPageLoadComplete();

materialsPage.waitForElementVisibility(materialsPage.getElectronicMateria
lsTable());

assertTrue(materialsPage.isElementDisplayed(materialsPage.getElectronicMa
terialsTable()));
    }

    @Then("'Електронні навчальні матеріали' table is appeared")
    public void checkAllMaterialsTable() {
        materialsPage.waitForPageLoadComplete();

materialsPage.waitForElementVisibility(materialsPage.getAllMaterialsTable
Page());

assertTrue(materialsPage.isElementDisplayed(materialsPage.getAllMaterials
TablePage()));
    }

    @When("Відобразилась таблиця 'Розклад'")
    public void checkScheduleTable() {

studentHomePage.waitForElementClickable(studentHomePage.getScheduleTable(
));

assertTrue(studentHomePage.isElementDisplayed(studentHomePage.getSchedule
Table()));
    }

    @Then("Відобразився інтферейс 'Повідомлення викладачу'")
    public void checkSendMessageInterface() {

messagesFilesPage.getWebDriver().switchTo().window(messagesFilesPage.getA
ctiveTab(1));
        messagesFilesPage.waitForPageLoadComplete();
```

```
messagesFilesPage.waitForElementVisibility(messagesFilesPage.getMessageTo
ProfPage());

assertTrue(messagesFilesPage.isElementDisplayed(messagesFilesPage.getMess
ageToProfPage()));
    }

    @Then("Відобразився інтферейс 'Надіслати файл викладачу'")
    public void checkSendFileInterface() {

messagesFilesPage.getWebDriver().switchTo().window(messagesFilesPage.getA
ctiveTab(2));
        messagesFilesPage.waitForPageLoadComplete();

messagesFilesPage.waitForElementVisibility(messagesFilesPage.getSendFileP
age());

assertTrue(messagesFilesPage.isElementDisplayed(messagesFilesPage.getSend
FilePage()));
    }

    @Then("Відобразилось повідомлення про помилку відправки пустого
повідомлення")
    public void checkEmptyMessageError() {
        messagesFilesPage.waitForPageLoadComplete();

messagesFilesPage.waitForElementVisibility(messagesFilesPage.getMessageIs
EmptyError());

assertTrue(messagesFilesPage.isElementDisplayed(messagesFilesPage.getMess
ageIsEmptyError()));
    }

    @Then("Відобразився попап із помилкою відправки файла")
    public void checkEmptyFileError() {
        String popupMessage =
messagesFilesPage.getWebDriver().switchTo().alert().getText();
        assertTrue(popupMessage.contains("Не заповнені всі поля"));
    }

    @Then("Відкрилась сторінка додатка у Play Market")
    public void checkPlayMarketPage() {

studentHomePage.getWebDriver().switchTo().window(studentHomePage.getActiv
eTab(1));
        studentHomePage.waitForPageLoadComplete();

studentHomePage.waitForElementVisibility(studentHomePage.getPlayMarketPag
e());
```

```
assertTrue(studentHomePage.isElementDisplayed(studentHomePage.getPlayMark
etPage()));
    }

    @Then("Відкрилась сторінка додатка у Apple Store")
    public void checkAppleStorePage() {

studentHomePage.getWebDriver().switchTo().window(studentHomePage.getActiv
eTab(2));
        studentHomePage.waitForPageLoadComplete();

studentHomePage.waitForElementVisibility(studentHomePage.getAppleMarketPa
ge());

assertTrue(studentHomePage.isElementDisplayed(studentHomePage.getAppleMar
ketPage()));
    }

    @Then("Відкрилась сторінка із результатами тестів і посиланнями")
    public void checkTableWithTestResults() {
        myResultsPage.waitForPageLoadComplete();

myResultsPage.waitForElementVisibility(myResultsPage.getTableWithTestResu
lts());

assertTrue(myResultsPage.isElementDisplayed(myResultsPage.getTableWithTes
tResults()));
    }

    @Then("Відкрилась сторінка із індивідуальним навчальним планом
студента")
    public void checkStudentPlanPage() {
        myResultsPage.waitForPageLoadComplete();

myResultsPage.waitForElementVisibility(myResultsPage.getStudentPlanPage()
.get(0));
        assertFalse(myResultsPage.getStudentPlanPage().isEmpty());
        for (WebElement webElement : myResultsPage.getStudentPlanPage())
{
            assertTrue(myResultsPage.isElementDisplayed(webElement));
        }
    }

    @Then("Відкрилась сторінка із електронним журналом успішності")
    public void checkLogOfRatesTable() {
        myResultsPage.waitForPageLoadComplete();

myResultsPage.waitForElementVisibility(myResultsPage.getLogOfRatesTable()
.get(0));
        assertFalse(myResultsPage.getLogOfRatesTable().isEmpty());
```

```
        for (WebElement webElement : myResultsPage.getLogOfRatesTable())
{
            assertTrue(myResultsPage.isElementDisplayed(webElement));
        }
    }

    @Then("Відкрилась сторінка із журналом пропусків")
    public void checkPassLogTable() {
        myResultsPage.waitForPageLoadComplete();

myResultsPage.waitForElementVisibility(myResultsPage.getPassLogTable());

assertTrue(myResultsPage.isElementDisplayed(myResultsPage.getPassLogTable
()));
    }

    @Then("Відкрилась сторінка із журналом заборгованостей")
    public void checkDebtLogInfo() {
        myResultsPage.waitForPageLoadComplete();

myResultsPage.waitForElementVisibility(myResultsPage.getDebtLogInfo());

assertTrue(myResultsPage.isElementDisplayed(myResultsPage.getDebtLogInfo(
));
    }
}

package vntu.edu.ua.steps;

import vntu.edu.ua.pages.*;

import static vntu.edu.ua.utils.Context.*;

public abstract class BaseSteps {
    protected HomePage homePage;
    protected SignInPage signInPage;
    protected StudentHomePage studentHomePage;
    protected MaterialsPage materialsPage;
    protected MessagesFilesPage messagesFilesPage;
    protected MyResultsPage myResultsPage;

    public BaseSteps() {
        homePage = (HomePage) scenarioContext.get(HOME_PAGE);
        signInPage = (SignInPage) scenarioContext.get(SIGN_IN_PAGE);
        studentHomePage = (StudentHomePage)
scenarioContext.get(USER_HOME_PAGE);
        materialsPage = (MaterialsPage)
scenarioContext.get(MATERIALS_PAGE);
        messagesFilesPage = (MessagesFilesPage)
scenarioContext.get(MESSAGES_FILES_PAGE);
```

```
        myResultsPage = (MyResultsPage)
scenarioContext.get(MY_RESULTS_PAGE);
    }
}

package vntu.edu.ua.steps;

import io.cucumber.java.en.And;
import io.cucumber.java.en.When;
import lombok.extern.slf4j.Slf4j;

@Slf4j
public class ClickSteps extends BaseSteps {

    @When("Відкрити домашню сторінку JetIQ")
    public void openHomePage() {
        homePage.openHomePage();
        homePage.waitForPageLoadComplete();
        log.info("Home page was opened");
    }

    @And("Натиснути на кноку 'Вхід'")
    public void clickOnSighInButton() {
        homePage.clickOnSighInButton();
        log.info("Was clicked on 'Вхід' button");
    }

    @And("Натиснути на кнопку 'Далі'")
    public void clickOnNextButton() {
        signInPage.clickOnLoginHeader();
        signInPage.clickOnNextButton();
        log.info("Was clicked on 'Далі' button");
    }

    @And("Натиснути на кноку 'Ввійти'")
    public void clickOnEnterButton() {
        signInPage.clickOnEnterButton();
        log.info("Was clicked on 'Ввійти' button");
    }

    @When("Натиснути на кнопку 'Матеріали'")
    public void clickOnMaterialsButton() {
        studentHomePage.waitForPageLoadComplete();
        studentHomePage.clickOnMaterialsMenu();
        log.info("Was clicked on 'Матеріали' button");
    }

    @And("Натиснути на кнопку 'Матеріали моїх дисциплін'")
    public void clickOnMyDisciplineMaterialsButton() {
        materialsPage.clickMyDisciplinesMaterialsButton();
    }
```

```
    @When("Обрати диспипліну під номером {int}")
    public void clickOnDisciplineNameButton(int value) {
        materialsPage.waitForPageLoadComplete();
        materialsPage.clickOnSubjectsInMaterialTable(value);
    }

    @When("Натиснути на кнопку 'Матеріали моєї спеціальності'")
    public void clickOnMySpecialtyMaterialsButton() {
        materialsPage.clickOnMySpecialtyMaterialsButton();
    }

    @When("Обрати будь який семестр")
    public void chooseAnySemester() {
        materialsPage.clickOnSearchBySemesterMaterials();
    }

    @When("Натиснути на кнопку 'Матеріали всіх дисцпиплін'")
    public void clickOnAllMaterialsButton() {
        materialsPage.clickOnAllMaterialsButton();
    }

    @When("Натиснути на кнопку 'Мої силабуси'")
    public void clickOnMySilabusesButton() {
        studentHomePage.clickOnMySilabusesMenu();
    }

    @When("Натиснути на кнопку 'Розклад'")
    public void clickOnScheduleButton() {
        studentHomePage.clickOnScheduleMenu();
    }

    @When("Натиснути на кнопку 'Повідомлення'")
    public void clickOnMessagesMenu() {
        studentHomePage.clickOnMessagesMenu();
    }

    @When("Натиснути на кнопку 'Відправити повідомлення'")
    public void clickOnSendMessageButton() {
        messagesFilesPage.clickOnSendMessageButton();
    }

    @When("Повернутись на вкладку назад і натиснути на кнопку 'Надіслати
файл'")
    public void clickOnSendFileButton() {
        messagesFilesPage.clickOnSendFileButton();
    }

    @When("Натиснути на кнопку 'Відправити'")
    public void clickOnSendButton() {
        messagesFilesPage.clickOnSendMessageToProfButton();
```

```
    }

    @When("Навести на кнопку 'Смартфон'")
    public void clickOnMobileMenu() {
        studentHomePage.clickOnMobileMenu();
    }

    @When("Натиснути на кнопку 'Андроїд'")
    public void clickOnAndroidButton() {
        studentHomePage.clickOnAndroidButton();
    }

    @When("Повернутись назад та натиснути на кнопку 'IOS'")
    public void clickOnAppleButton() {
        studentHomePage.clickOnAppleButton();
    }

    @When("Натиснути на кноку 'Мої Результати'")
    public void clickOnMyResultsMenu() {
        studentHomePage.clickOnMyResultsMenu();
    }

    @When("Натиснути на кноку 'Індивідуальний план студента'")
    public void clickOnStudentPlanButton() {
        myResultsPage.clickOnStudentPlanButton();
    }

    @When("Натиснути на кноку 'Електронний журнал успішності'")
    public void clickOnLogOfRateButton() {
        myResultsPage.clickOnLogOfRateButton();
    }

    @When("Натиснути на кноку 'Журнал пропусків'")
    public void clickOnPassLogButton() {
        myResultsPage.clickOnPassLogButton();
    }

    @When("Повернутись назад і натиснути на кноку 'Журнал
заборгованостей'")
    public void clickOnDebtLogButton() {
        myResultsPage.clickOnDebtLogButton();
    }

    @When("Натиснути на кноку 'Вихід'")
    public void clickOnExitButton() {
        studentHomePage.waitForPageLoadComplete();
        studentHomePage.clickOnExitButton();
    }
}

package vntu.edu.ua.steps;
```

```
import io.cucumber.java.en.And;
import io.cucumber.java.en.When;
import lombok.extern.slf4j.Slf4j;

import static vntu.edu.ua.utils.Constants.*;

@Slf4j
public class FillFieldsSteps extends BaseSteps {

    @When("Ввести {string} значення в поле логіну")
    public void setUsernameToField(String valueType) {
        String value;
        if (valueType.equals(VALID)) {
            value = USERNAME;
        } else {
            value = INVALID_VALUE;
        }
        signInPage.setValueToUsernameField(value);
        log.info("Username was set");
    }

    @And("Ввести {string} значення в поле паролю")
    public void setPasswordToField(String valueType) {
        String value;
        if (valueType.equals(VALID)) {
            value = PASSWORD;
        } else {
            value = INVALID_VALUE;
        }
        signInPage.setValueToPasswordField(value);
        log.info("Password was set");
    }
}
```

```
package vntu.edu.ua.steps;

import io.cucumber.java.After;
import io.cucumber.java.Before;
import io.cucumber.java.Scenario;
import lombok.SneakyThrows;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.io.FileUtils;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import vntu.edu.ua.manager.PageFactoryManager;

import java.io.File;
```

```
import java.time.OffsetDateTime;

import static io.github.bonigarcia.wdm.WebDriverManager.chromedriver;
import static vntu.edu.ua.utils.Constants.*;
import static vntu.edu.ua.utils.Context.*;

@Slf4j
public class Hooks {

    private WebDriver webDriver;

    @Before(order = 1)
    public void setUpWebDriver() {
        chromedriver().setup();
        if (System.getenv(LOCAL_RUN) == null) {
            ChromeOptions chromeOptions = new ChromeOptions();
            chromeOptions.addArguments(NO_SANDBOX_OPTION);
            chromeOptions.addArguments(DISABLE_SETUID_OPTION);
            chromeOptions.addArguments(DISABLE_GPU_OPTION);
            chromeOptions.addArguments(HEADLESS_OPTION);
            chromeOptions.addArguments(DISABLE_DEV_SHM_OPTION);
            chromeOptions.addArguments(WINDOW_RESOLUTION_OPTION);
            chromeOptions.setBinary(GOOGLE_CHROME_LINUX_PATH);
            webDriver = new ChromeDriver(chromeOptions);
        } else {
            webDriver = new ChromeDriver();
            webDriver.manage().window().maximize();
        }
        scenarioContext.put(WEB_DRIVER, webDriver);

        log.info("WebDriver was started successfully");
    }

    @Before(order = 2)
    public void setUpPageFactory() {
        PageFactoryManager pageFactoryManager = new
PageFactoryManager(webDriver);
        scenarioContext.put(HOME_PAGE, pageFactoryManager.getHomePage());
        scenarioContext.put(SIGN_IN_PAGE,
pageFactoryManager.getSignInPage());
        scenarioContext.put(USER_HOME_PAGE,
pageFactoryManager.getUserHomePage());
        scenarioContext.put(MATERIALS_PAGE,
pageFactoryManager.getMaterialsPage());
        scenarioContext.put(MESSAGES_FILES_PAGE,
pageFactoryManager.getMessagesFilesPage());
        scenarioContext.put(MY_RESULTS_PAGE,
pageFactoryManager.getMyResultsPage());
        log.info("PageFactory was started successfully");
    }
```

```
    @SneakyThrows
    @After(order = 2)
    public void scenarioReporter(Scenario scenario) {
        if (scenario.isFailed()) {
            String fileName = OffsetDateTime.now().toEpochSecond() +
".png";
            log.info("Scenario is failed. Start making a screenshot...");
            File screenshot = ((TakesScreenshot)
webDriver).getScreenshotAs(OutputType.FILE);
            FileUtils.copyFile(screenshot, new File("target/generated-
test-sources/" + fileName));
        }
    }

    @After(order = 1)
    public void closeWebDriver() {
        webDriver.quit();
        log.info("WebDriver was closed successfully");
    }
}
```

```
package vntu.edu.ua.utils;

import static vntu.edu.ua.utils.Convertor.getDecodeValue;

public class Constants {

    public final static String HOME_PAGE_URL =
"https://jetiq.vntu.edu.ua/";

    public final static long DEFAULT_TIMEOUT = 60;

    public final static String PAGE_LOAD_SCRIPT = "return
document.readyState";
    public final static String COMPLETE = "complete";
    public final static String VALID = "valid";
    public final static String INVALID_VALUE = "INVALID_VALUE";
    public final static String LOCAL_RUN = "LOCAL_RUN";

    public final static String GLUE = "vntu.edu.ua.steps";
    public final static String FEATURES = "src/test/resources/features";

    public final static String USERNAME = getDecodeValue("MDQtMTgtMzk2");
    public final static String PASSWORD =
getDecodeValue("Q3VtZGVvMTg3Mg==");

    public final static String GOOGLE_CHROME_LINUX_PATH =
"/usr/bin/google-chrome";
    public final static String NO_SANDBOX_OPTION = "--no-sandbox";
    public final static String DISABLE_SETUID_OPTION = "--disable-setuid-
sandbox";
```

```
    public final static String DISABLE_GPU_OPTION = "--disable-gpu";
    public final static String HEADLESS_OPTION = "headless";
    public final static String DISABLE_DEV_SHM_OPTION = "--disable-dev-
shm-usage";
    public final static String WINDOW_RESOLUTION_OPTION = "--window-
size=1920x1080";
}
```

```
package vntu.edu.ua.utils;

import lombok.extern.slf4j.Slf4j;

import java.util.HashMap;

@Slf4j
public class Context {
    public static final HashMap<String, Object> scenarioContext = new
HashMap<>();

    public static final String WEB_DRIVER = "WEB_DRIVER";

    public static final String HOME_PAGE = "HOME_PAGE";
    public static final String SIGN_IN_PAGE = "SIGN_IN_PAGE";
    public static final String USER_HOME_PAGE = "USER_HOME_PAGE";
    public static final String MATERIALS_PAGE = "MATERIALS_PAGE";
    public static final String MESSAGES_FILES_PAGE =
"MESSAGES_FILES_PAGE";
    public static final String MY_RESULTS_PAGE = "MY_RESULTS_PAGE";
}
```

```
package vntu.edu.ua.utils;

import java.util.Base64;

public class Convertor {

    public static String getDecodeValue(String key) {
        return new String(Base64.getDecoder().decode(key));
    }
}
```

Додаток Г. Графічна частина

ГРАФІЧНА ЧАСТИНА

РОЗРОБКА ФРЕЙМВОРКУ ДЛЯ ІНТЕГРАЦІЙНОГО ТЕСТУВАННЯ САЙТУ JETIQ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ SELENIUM WEBDRIVER

Рисунок Г.1 – Титульний слайд

Рисунок Г.2 – Актуальність теми

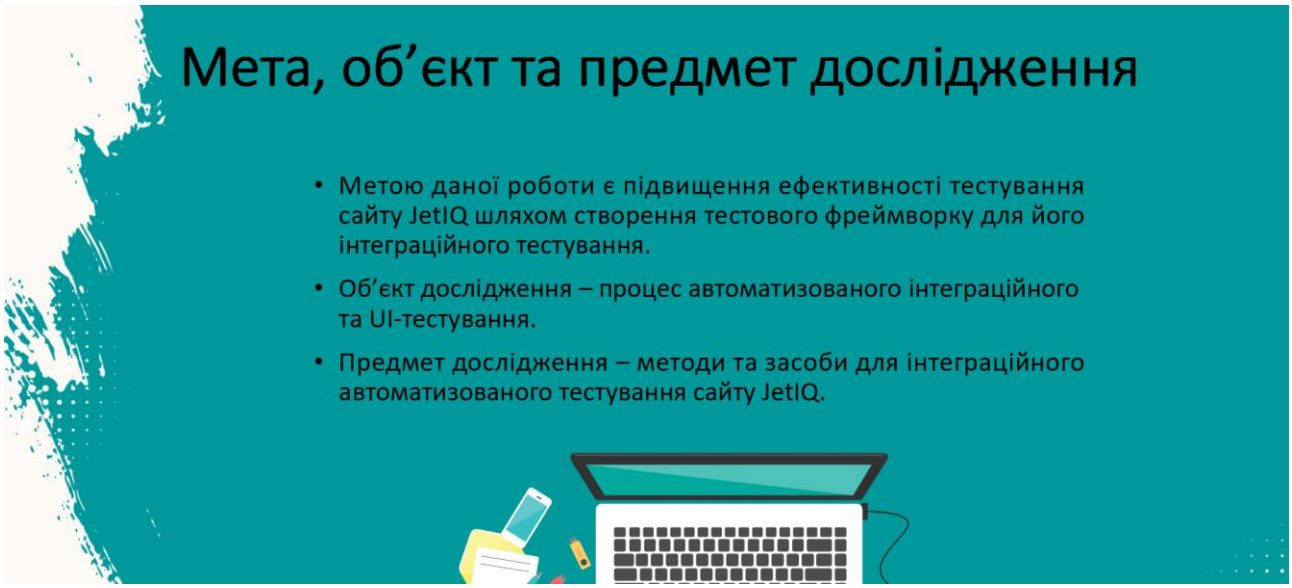


Рисунок Г.3 – Мета, об’єкт та предмет дослідження

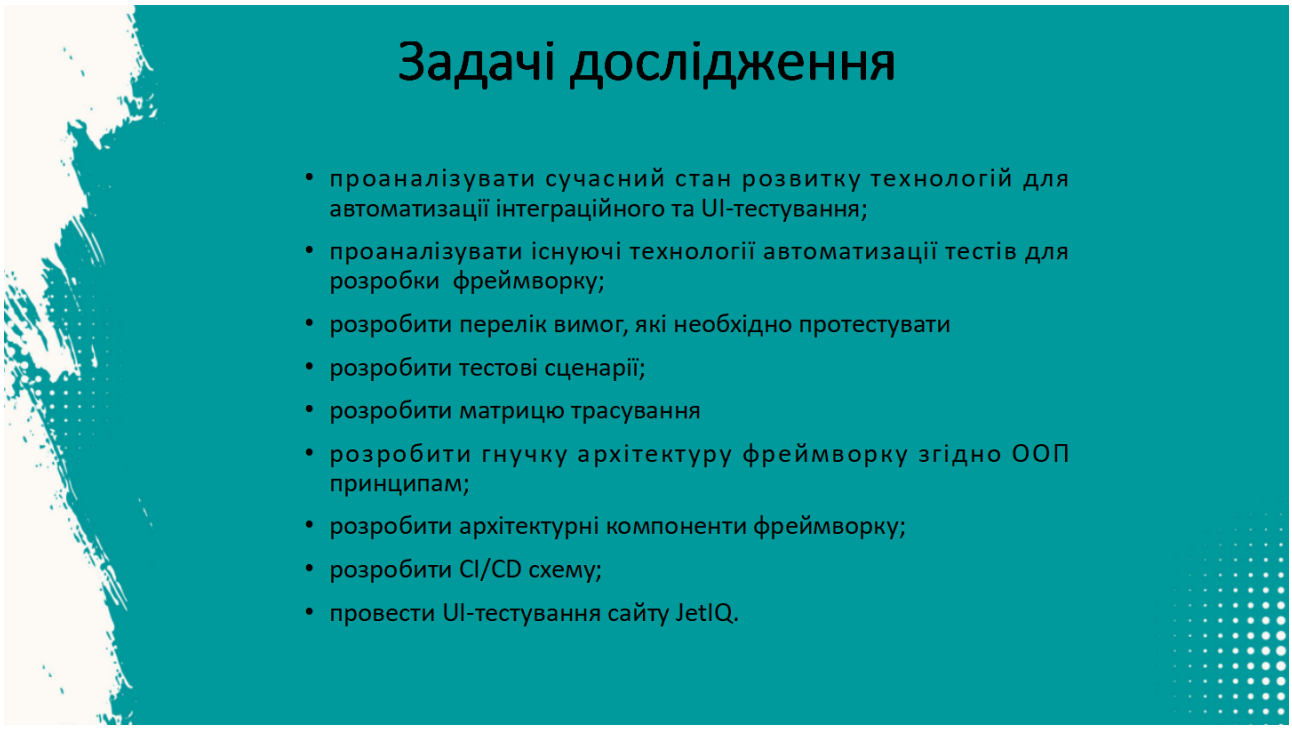


Рисунок Г.4 – Задачі дослідження

Рисунок Г.5 – Новизна одержаних результатів

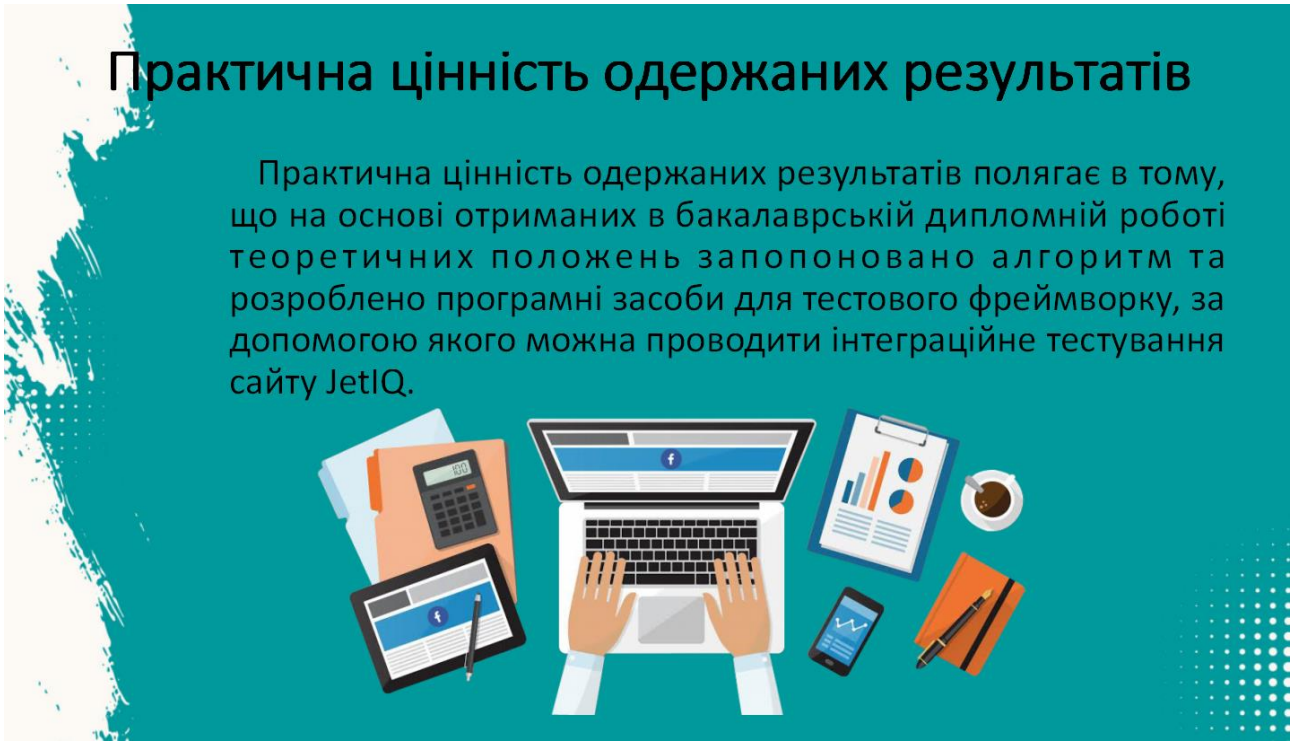


Рисунок Г.6 – Практична цінність одержаних результатів

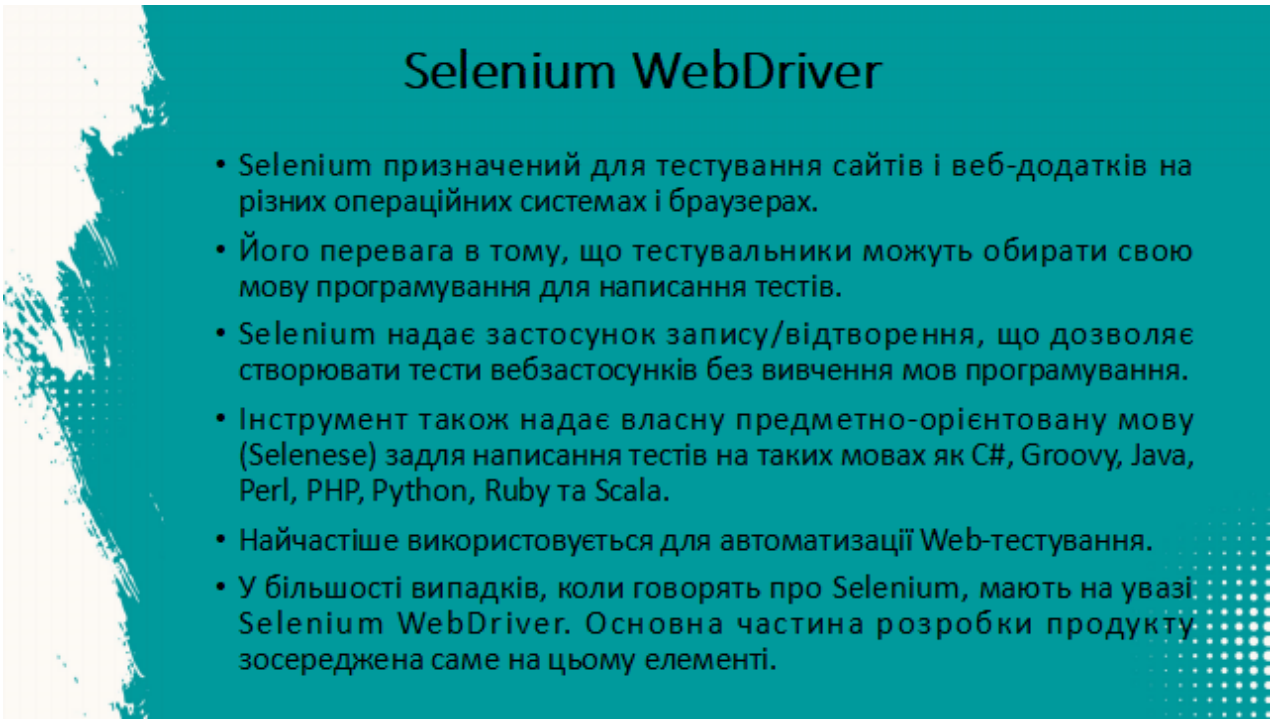


Рисунок Г.7 – Selenium WebDriver

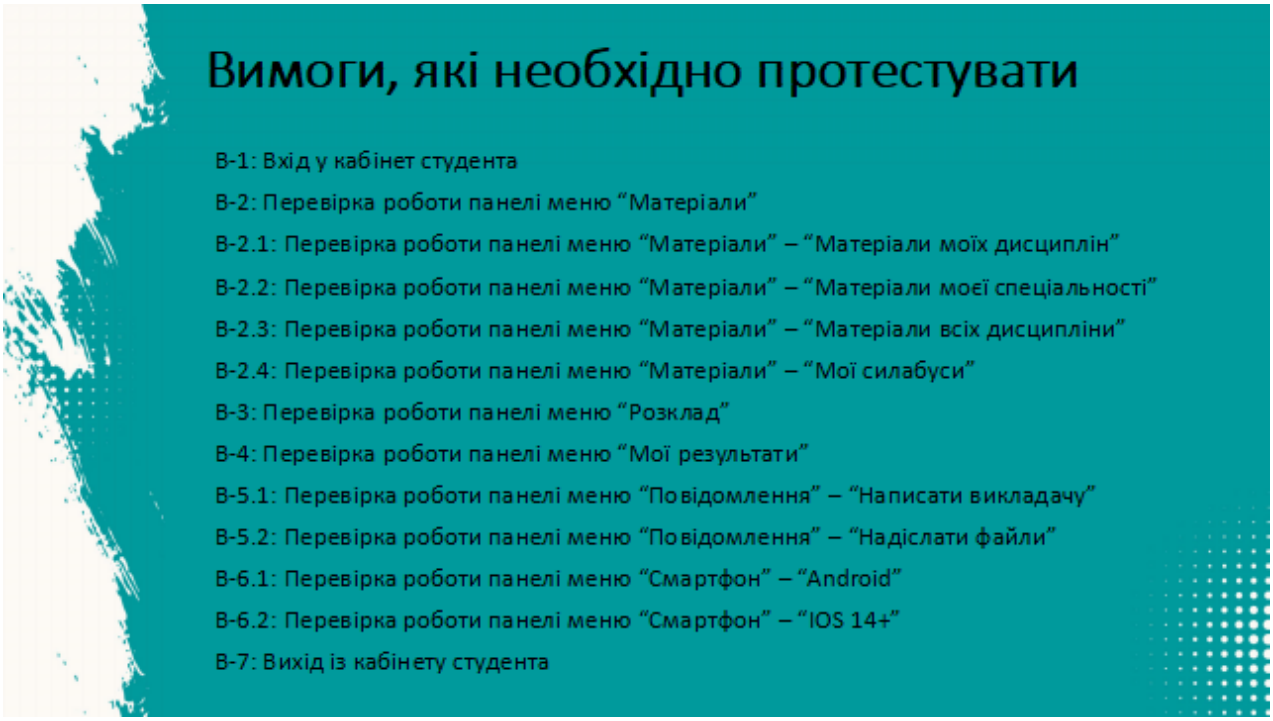


Рисунок Г.8 – Вимоги, які необхідно протестувати

ID	Назва	Передумови	Кроки	Очікуючий результат
T-1	Кабінет Студента - Перевірка меню: Матеріали -> Матеріали моїх дисциплін	1. Відкрити сторінку сайту JetIQ. 2. Увійти в кабінет студента. 3. Натиснути на кнопку "Матеріали"	1. Натиснути на кнопку "Матеріали моїх дисциплін"	Відображено таблиця електронних матеріалів
T-2	Кабінет Студента - Перевірка меню: Матеріали -> Матеріали моєї спеціальності		1. Натиснути на кнопку "Матеріали моєї спеціальності"	
T-3	Кабінет Студента - Перевірка меню: Матеріали -> Матеріали всіх дисциплін		1. Натиснути на кнопку "Матеріали всіх дисциплін"	Відображалась таблиця "Електронних навчальних матеріалів"
T-4	Кабінет Студента - Перевірка меню: Матеріали -> Мої силабуси	1. Відкрити сторінку сайту JetIQ. 2. Увійти в кабінет студента	1. Натиснути на кнопку "Мої силабуси"	Відображалась таблиця матеріалів з посиланнями на них
T-5	Кабінет Студента - Перевірка меню: Мої Результати -> Індивідуальний план студента		1. Натиснути на кнопку "Індивідуальний план студента"	Відкрилась сторінка із індивідуальним навчальним планом студента
T-6	Кабінет Студента - Перевірка меню: Мої Результати -> Електронний журнал успішності	1. Відкрити сторінку сайту JetIQ. 2. Увійти в кабінет студента 3. Натиснути на кнопку "Мої Результати"	1. Натиснути на кнопку "Електронний журнал успішності"	Відкрилась сторінка із електронним журналом успішності
T-7	Кабінет Студента - Перевірка меню: Мої Результати -> Журнал пропусків та заборгованостей		1. Натиснути на кнопку "Журнал пропусків" 2. Повернутись назад і натиснути на кнопку "Журнал заборгованостей"	1. Відкрилась сторінка із журналом пропусків 2. Відкрилась сторінка із журналом заборгованостей
T-8	Кабінет Студента - Вихід з власного кабінету	1. Відкрити сторінку сайту JetIQ. 2. Увійти в кабінет студента	1. Натиснути на кнопку "Вихід"	Студент вийшов з власного кабінету

ID	Назва	Передумови	Кроки	Очікуючий результат
T-9	Кабінет Студента - Перевірка інтерфейсів відправки файлів та повідомлень викладачу	1. Відкрити сторінку сайту JetIQ. 2. Увійти в кабінет студента 3. Натиснути на кнопку "Повідомлення"	1. Натиснути на кнопку "Відправити повідомлення" 2. Повернутись на вкладку назад і натиснути на кнопку "Надіслати файл"	1. Відображався інтерфейс "Повідомлення викладачу" 2. Відображався інтерфейс "Надіслати файл викладачу"
T-10	Кабінет Студента - Відправка пустих повідомлення та повідомлень без файлів викладачу		1. Натиснути на кнопку "Відправити повідомлення" 2. Натиснути на кнопку "Відправити" 3. Повернутись на вкладку назад і натиснути на кнопку "Надіслати файл" 4. Натиснути на кнопку "Відправити"	1. Відображався інтерфейс "Повідомлення викладачу" 2. Відображалось повідомлення про помилку відправки пустого повідомлення 3. Відображався інтерфейс "Надіслати файл викладачу" 4. Відображався попап із помилкою відправки файла
T-11	Кабінет Студента - Перевірка меню: Смартфон -> Android та iOS	1. Відкрити сторінку сайту JetIQ. 2. Увійти в кабінет студента 3. Навести на кнопку "Смартфон"	1. Натиснути на кнопку "Android" 2. Повернутись назад та натиснути на кнопку "iOS"	1. Відкрилась сторінка додатка у Play Market 2. Відкрилась сторінка додатка у Apple Store
T-12	Кабінет Студента - Перевірка меню: розклад	1. Відкрити сторінку сайту JetIQ. 2. Увійти в кабінет студента	1. Натиснути на кнопку "Розклад"	Відобразилась таблиця "Розклад"

Рисунок Г.9 – Список тест-кейсів

ID	T-1	T-2	T-3	T-4	T-5	T-6	T-7	T-8	T-9	T-10	T-11	T-12
B-1	+	+	+	+	+	+	+	+	+	+	+	+
B-2	+	+	+	+								
B-2.1	+											
B-2.2		+										
B-2.3			+									
B-2.4				+								
B-3												+
B-4					+	+	+					
B-5.1									+	+		
B-5.2									+	+		
B-6.1											+	
B-6.2											+	
B-7								+				

Рисунок Г.10 – Матриця трасування

Рисунок Г.11 – Архітектура фреймворку

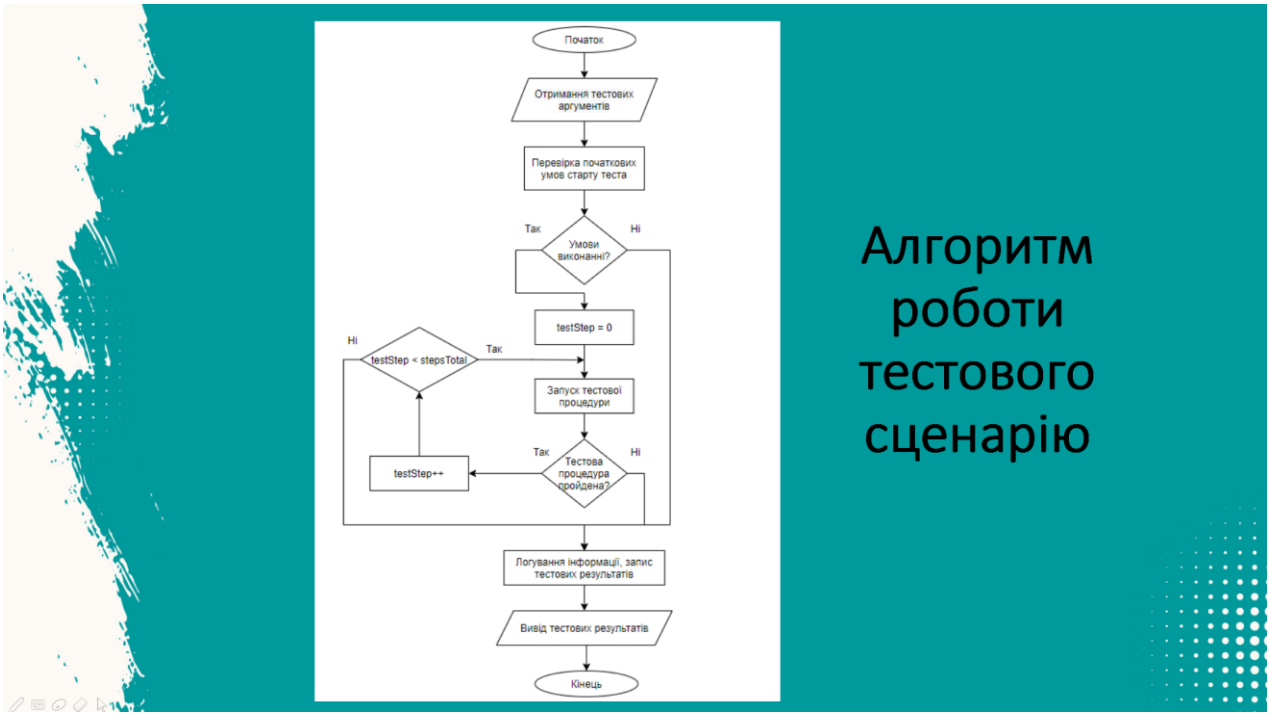


Рисунок Г.12 – Алгоритм роботи тестового сценарію

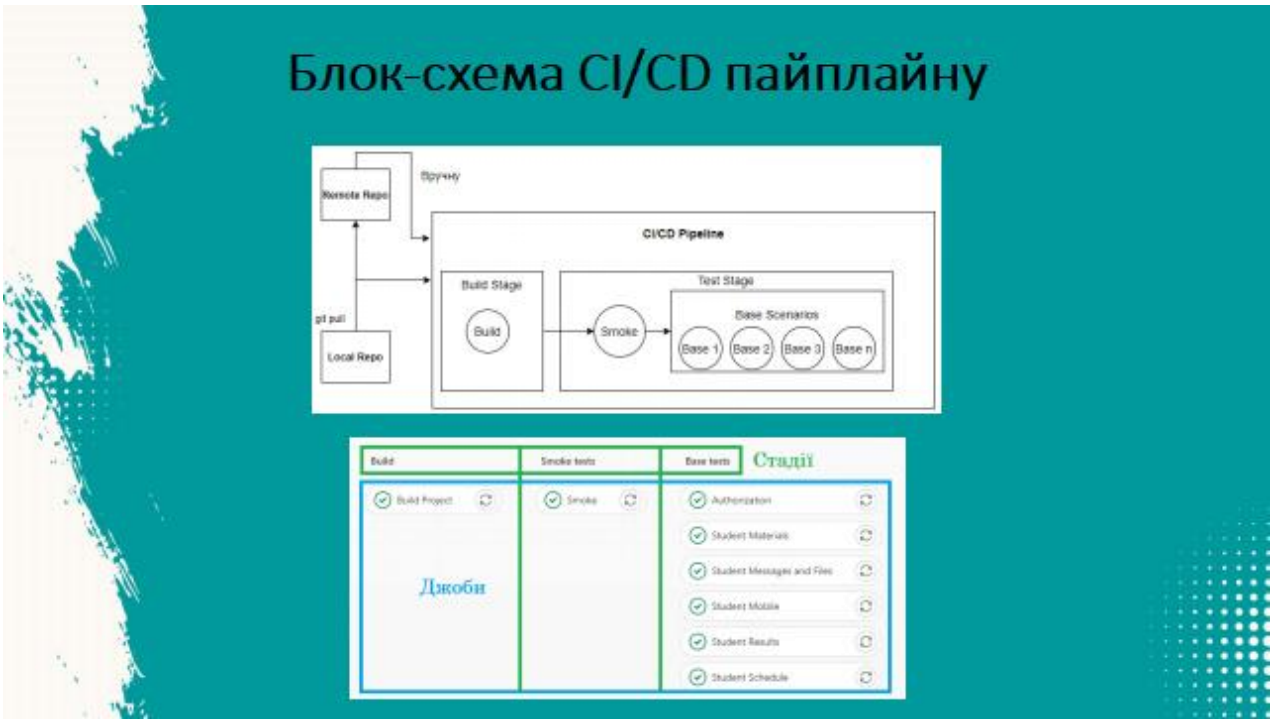


Рисунок Г.13 – Блок-схема CI/CD пайплайну

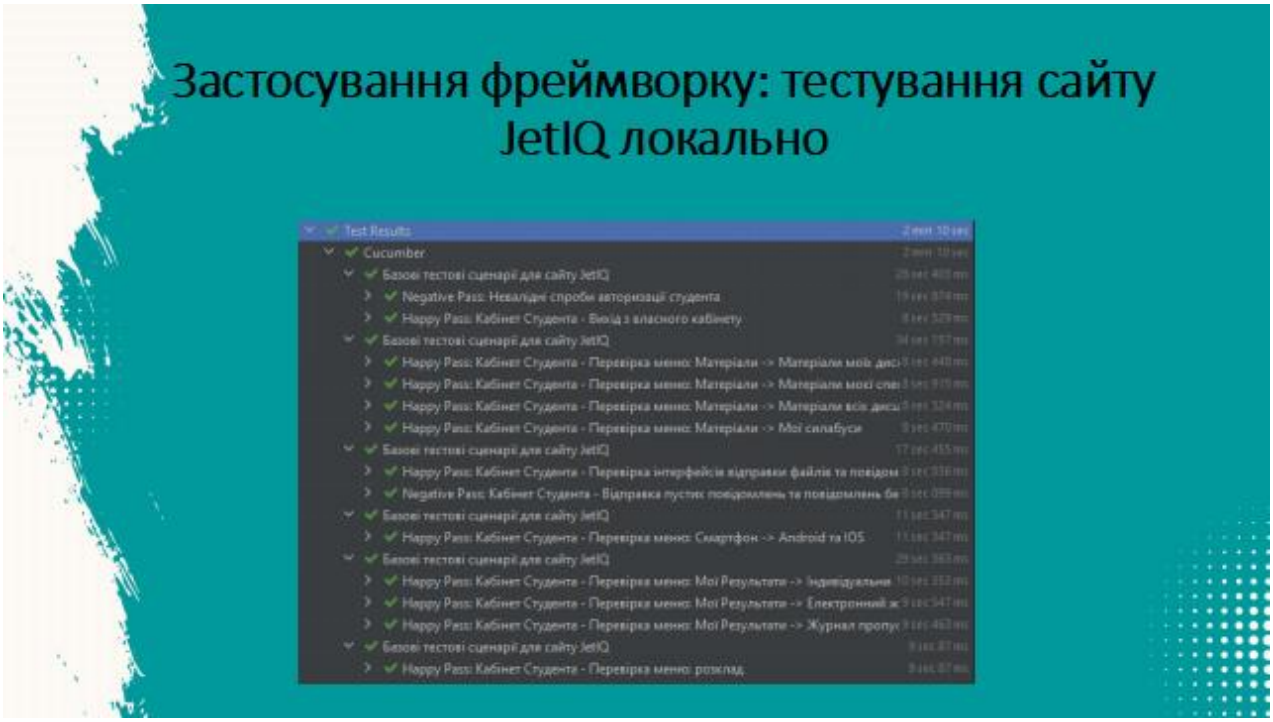


Рисунок Г.14 – Застосування фреймворку: тестування сайту JetIQ локально

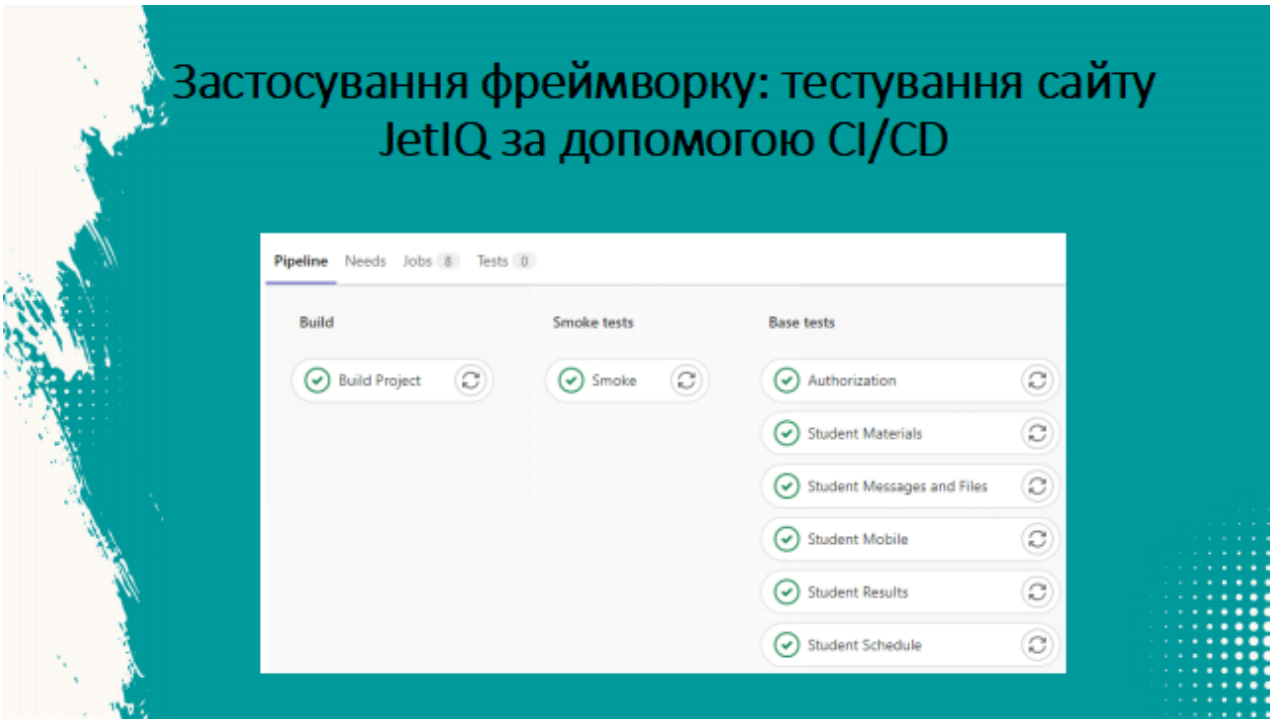


Рисунок Г.16 – Застосування фреймворку: тестування сайту JetIQ за допомогою CI/CD

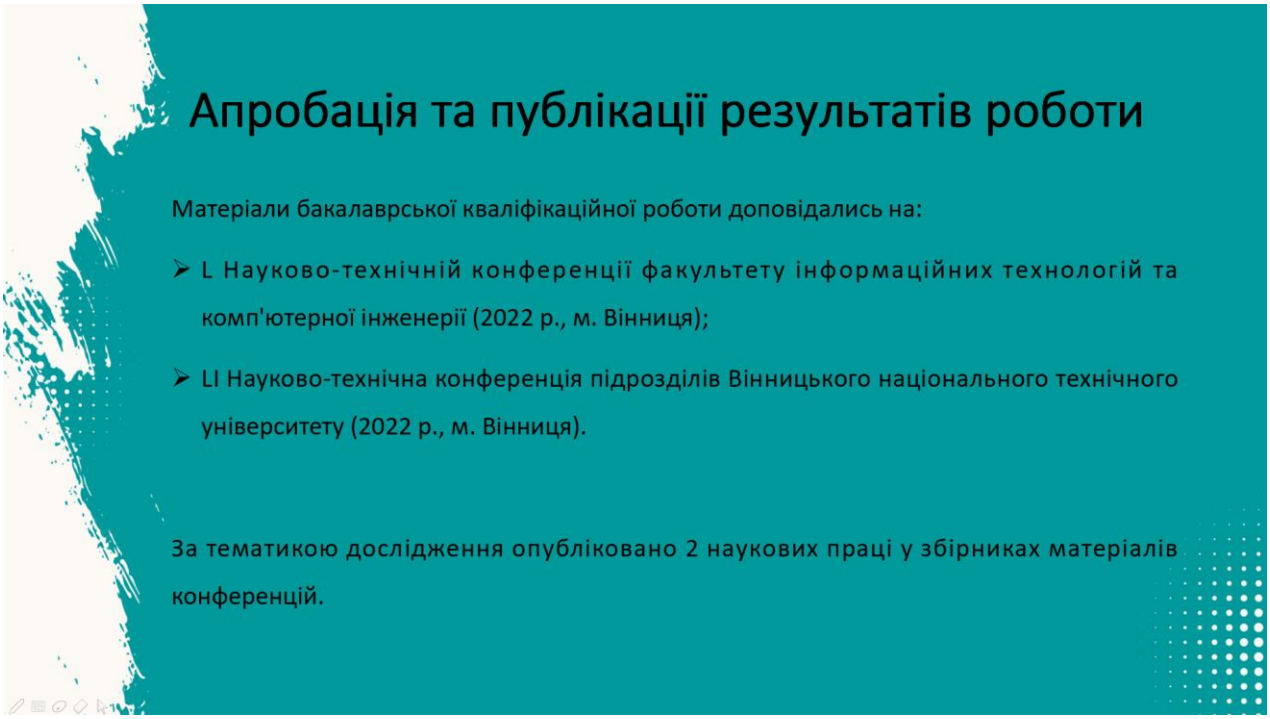


Рисунок Г.17 – Апробація та публікації результатів роботи

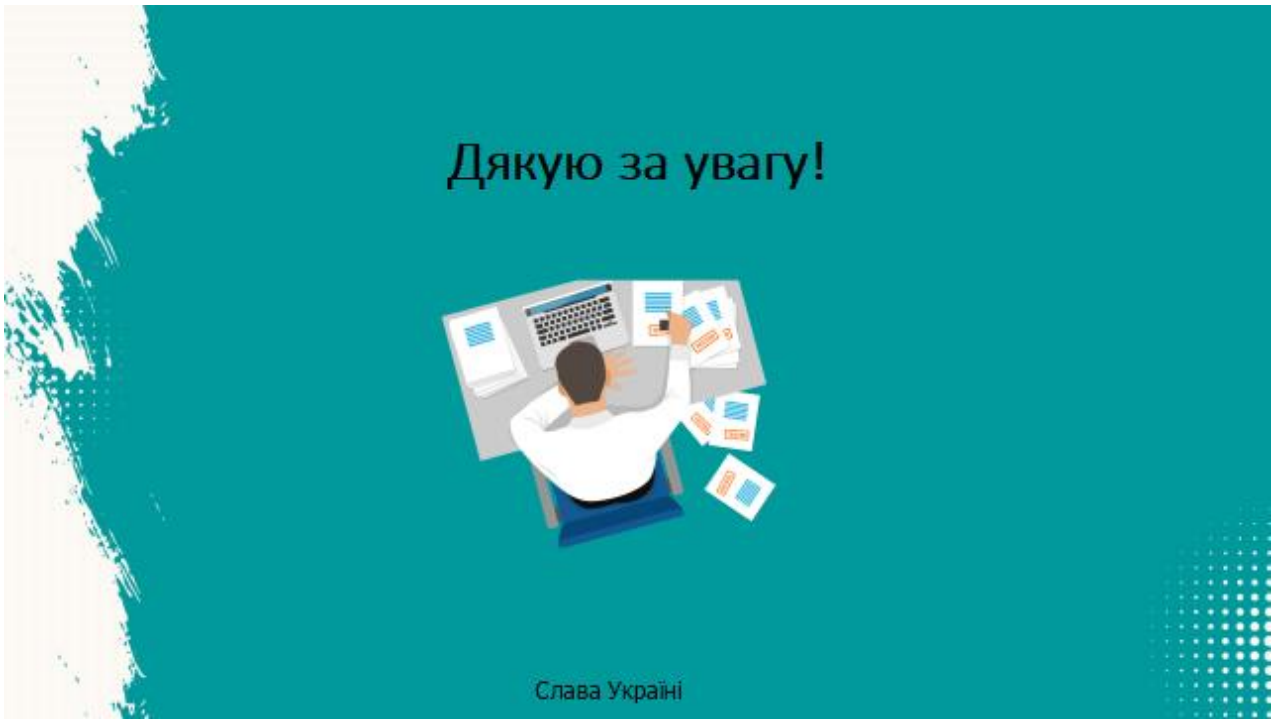


Рисунок Г.18 – Фінальний слайд