Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавр
(ступінь вищої освіти)

на тему РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ
DEVELOPMENT OF SOFTWARE FOR AUTOMATED TESTING OF A WEB APPLICATION

Виконав(ла): студент(ка) 4 курсу, групи КНТ-112
Спеціальності 121 Інженерія програмного
(код і найменування спеціальності)
забезпечення

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

ЯНІН О.Д.
(ПРІЗВИЩЕ та ініціали)
Керівник ГЛАДКОВА О.М.
(ПРІЗВИЩЕ та ініціали)
Рецензент ГОЛУБ Т.В.
(ПРІЗВИЩЕ та ініціали)

2026

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____Сергій СУББОТІН
“____” _____2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ЯНІНА Олександра Денисовича
(ПРІЗВИЩЕ, ім’я, по батькові)

1. Тема проєкту (роботи) Розроблення програмного забезпечення автоматизованого тестування вебзастосунку. Development of Software for Automated Testing of a Web Application

керівник проєкту (роботи) к.т.н., доцент, ГЛАДКОВА Ольга Миколаївна,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім’я, по батькові)
затверджені наказом закладу вищої освіти від “ 07 ” квітня 2026 року № 139
2. Строк подання студентом проєкту (роботи) 10 червня 2026 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Теоретичні основи тестування вебзастосунків. 2. Проєктування програмного забезпечення тестування вебзастосунку. 3. Розроблення програмного забезпечення тестування вебзастосунку. 4. Експериментальне дослідження результатів роботи програмного забезпечення.
5. Перелік графічного матеріалу (з з точним зазначенням обов’язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ГЛАДКОВА О.М., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викладач		

7. Дата видачі завдання « 20 » квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій	2 тиждень	Розділ 2
	розробки.		
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження	5 тиждень	Розділ 4
	програмного забезпечення.		
7	Оформлення пояснювальної записки та документів	6 тиждень	Додатки
	до неї.		
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка) _____ Олександр ЯНІН
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Ольга ГЛАДКОВА
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 70 с., 3 табл., 19 рис., 2 дод., 11 джерел.

АВТОТЕСТИ, SELENIUM WEBDRIVER, JAVA, UI-ТЕСТУВАННЯ, TESTNG, ВЕБЗАСТОСУНОК, ФУНКЦІОНАЛЬНЕ ТЕСТУВАННЯ.

Об’єкт дослідження – процеси автоматизованого тестування вебзастосунків із використанням технологій програмної інженерії.

Предмет дослідження – методи, програмні засоби та архітектурні підходи до тестування вебінтерфейсів із використанням Selenium WebDriver та тестового фреймворку.

Мета роботи – підвищення швидкості та стабільності перевірки функціональності вебзастосунків шляхом розроблення програмного забезпечення тестування вебінтерфейсу.

Матеріали, методи та технічні засоби: Selenium WebDriver, TestNG, Java, Maven, Page Object Model, інтегроване середовище розробки IntelliJ IDEA, браузер Google Chrome та ChromeDriver.

Результати. Розроблено програмне забезпечення тестування вебзастосунку з використанням Selenium WebDriver та TestNG. Реалізовано тестові сценарії для перевірки функціональності вебінтерфейсу.

Висновки. Розроблено програмне забезпечення тестування вебзастосунку, яке дозволяє скоротити час виконання регресійного тестування, зменшити вплив людського фактора та підвищити стабільність повторного запуску тестових сценаріїв.

Галузь використання – тестування вебзастосунків у процесах забезпечення якості програмного забезпечення в IT-компаніях та під час розроблення вебсистем.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 70 pages, 3 tables, 19 figures, 2 appendixes, 11 sources.

AUTOTESTS, SELENIUM WEBDRIVER, JAVA, UI TESTING, TESTNG, WEB APPLICATION, FUNCTIONAL TESTING.

Object of study is the processes of automated testing of web applications using software engineering technologies.

Subject of study is methods, software tools and architectural approaches to web interface testing using Selenium WebDriver and a testing framework.

The purpose of the work is to increase the speed and stability of web application functionality verification by developing software for web interface testing.

Materials, methods, and tools: Selenium WebDriver, TestNG, Java, Maven, Page Object Model, IntelliJ IDEA integrated development environment, Google Chrome browser and ChromeDriver.

Results. Software for testing a web application using Selenium WebDriver and TestNG has been developed. Test scenarios for verifying web interface functionality have been implemented.

Conclusions. Software for testing a web application has been developed, which makes it possible to reduce the execution time of regression testing, decrease the impact of the human factor, and improve the stability of repeated test scenario execution.

The field of use is testing of web applications in software quality assurance processes in IT companies and during web systems development.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

CI/CD – Continuous Integration / Continuous Delivery;

DOM – Document Object Model;

GUI – Graphical User Interface;

HTML – HyperText Markup Language;

HTTP – HyperText Transfer Protocol;

JSON – JavaScript Object Notation;

POM – Page Object Model;

QA – Quality Assurance;

REST – Representational State Transfer;

SQA – Software Quality Assurance;

UI – User Interface;

URL – Uniform Resource Locator;

XML – Extensible Markup Language.

ПЗ – програмне забезпечення;

ВСТУП

Сучасні вебзастосунки є невід'ємною складовою інформаційних систем у багатьох сферах діяльності, зокрема у електронній комерції, банківських сервісах, освітніх платформах та корпоративних системах. Зі зростанням складності вебзастосунків збільшується потреба у забезпеченні їх стабільності, надійності та коректності роботи [1].

Одним із ключових етапів забезпечення якості програмного забезпечення є тестування. Традиційне ручне тестування вебінтерфейсів потребує значних витрат часу, особливо у випадку повторного виконання регресійних перевірок після оновлення функціональності системи. Крім того, ручне тестування супроводжується ризиком виникнення людських помилок та складністю масштабування тестових сценаріїв [2].

У сучасній розробці програмного забезпечення все більшого поширення набуває автоматизоване тестування вебзастосунків, яке дозволяє прискорити процес перевірки функціональності, зменшити навантаження на QA-фахівців та підвищити стабільність повторного запуску тестів [3].

Одним із найбільш популярних інструментів автоматизації тестування вебінтерфейсів є Selenium. Selenium WebDriver дозволяє автоматизувати взаємодію з браузером, виконувати перевірку елементів вебінтерфейсу та реалізовувати складні сценарії тестування вебзастосунків [4].

Для підвищення підтримуваності та масштабованості автоматизованих тестів широко використовується архітектурний підхід Page Object Model, який дозволяє відокремлювати логіку взаємодії зі сторінками вебзастосунку від тестових сценаріїв [5].

Актуальність теми роботи полягає у необхідності підвищення швидкості та стабільності перевірки функціональності вебзастосунків шляхом використання автоматизованого тестування вебінтерфейсу.

Об'єкт дослідження – процеси автоматизованого тестування вебзастосунків із використанням технологій програмної інженерії.

Предмет дослідження – методи, програмні засоби та архітектурні підходи до автоматизації тестування вебінтерфейсів із використанням Selenium WebDriver та тестового фреймворку.

Мета роботи – підвищення швидкості та стабільності перевірки функціональності вебзастосунків шляхом розроблення програмного забезпечення автоматизованого тестування вебінтерфейсу.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати сучасні методи тестування вебзастосунків;
- дослідити можливості автоматизованого UI-тестування;
- провести аналіз сучасних інструментів автоматизації тестування;
- розробити архітектуру програмного забезпечення тестування;
- реалізувати тестові сценарії;
- провести експериментальне дослідження розробленого програмного забезпечення.

Практична значимість роботи полягає у розробленні програмного забезпечення тестування вебзастосунку, яке дозволяє автоматизувати перевірку функціональності вебінтерфейсу та скоротити час виконання регресійного тестування.

1 ТЕОРЕТИЧНІ ОСНОВИ ТЕСТУВАННЯ ВЕБЗАСТОСУНКІВ

1.1 Особливості тестування вебзастосунків

Сучасні вебзастосунки є одними з найбільш поширених типів програмного забезпечення та використовуються у багатьох сферах діяльності, зокрема в електронній комерції, банківських системах, освітніх платформах та корпоративних сервісах. Вебзастосунки функціонують у середовищі браузера та взаємодіють із серверною частиною через мережеві протоколи, що створює додаткові вимоги до забезпечення якості програмного забезпечення [1].

Тестування вебзастосунків є важливим етапом процесу розроблення програмного забезпечення та спрямоване на перевірку правильності роботи функціональних можливостей, стабільності роботи інтерфейсу користувача та коректності взаємодії між компонентами системи. Основною метою тестування є виявлення помилок та забезпечення відповідності програмного продукту функціональним вимогам.

Особливістю вебзастосунків є залежність від браузера, операційної системи та мережевого середовища. Один і той самий вебзастосунок може поводитися по-різному у різних браузерах, що потребує додаткових перевірок сумісності. Крім того, вебзастосунки мають динамічний інтерфейс користувача, який змінюється залежно від дій користувача та отриманих даних із сервера [2].

Основними видами тестування вебзастосунків є:

- функціональне тестування;
- UI-тестування;
- регресійне тестування;
- інтеграційне тестування;
- навантажувальне тестування;
- тестування сумісності браузерів.

Функціональне тестування спрямоване на перевірку відповідності функціональних можливостей вебзастосунку визначеним вимогам. UI-тестування використовується для перевірки коректності роботи елементів інтерфейсу користувача, а регресійне тестування дозволяє переконатися у відсутності нових помилок після внесення змін у систему.

У сучасній розробці програмного забезпечення тестування вебзастосунків є важливою складовою процесу забезпечення якості та використовується практично в усіх етапах життєвого циклу програмного продукту [3].

1.2 Методи функціонального тестування вебзастосунків

Функціональне тестування є одним із основних методів перевірки вебзастосунків та спрямоване на контроль правильності роботи функціональних можливостей програмного забезпечення. Під час функціонального тестування перевіряється відповідність роботи вебзастосунку визначеним вимогам та очікуваній поведінці системи [4].

Основою функціонального тестування є перевірка взаємодії користувача з вебзастосунком через графічний інтерфейс або програмні інтерфейси. У процесі тестування аналізуються результати виконання певних сценаріїв роботи користувача.

До основних методів функціонального тестування належать:

- тестування авторизації та аутентифікації;
- тестування форм введення;
- тестування навігації;
- тестування роботи з даними;
- перевірка повідомлень про помилки;
- тестування бізнес-логіки системи.

Одним із важливих напрямів є регресійне тестування, яке виконується після оновлення програмного забезпечення. Його основною метою є

перевірка того, що внесені зміни не порушили роботу вже реалізованого функціоналу [5].

Під час функціонального тестування використовуються тестові сценарії, які описують послідовність дій користувача та очікувані результати роботи системи. Наприклад, для вебзастосунку електронної комерції тестовий сценарій може включати:

- авторизацію користувача;
- додавання товару до кошика;
- оформлення замовлення;
- перевірку повідомлення про успішне завершення операції.

Традиційне ручне тестування таких сценаріїв потребує значних витрат часу, особливо у випадку регулярного повторного запуску тестів. Саме тому у сучасних проєктах широко використовуються програмні засоби тестування вебзастосунків.

1.3 Тестування вебінтерфейсів

Тестування вебінтерфейсів є одним із найбільш важливих напрямів забезпечення якості вебзастосунків. Основною метою UI-тестування є перевірка коректності взаємодії користувача з елементами інтерфейсу вебсторінки [6].

Під час тестування вебінтерфейсу перевіряються:

- поля введення;
- кнопки;
- форми;
- меню навігації;
- повідомлення системи;
- таблиці;
- елементи керування.

Тестування вебінтерфейсу виконується шляхом моделювання дій користувача у браузері. Для цього програмне забезпечення тестування взаємодіє з DOM-структурою вебсторінки та виконує перевірку елементів інтерфейсу.

Однією з основних проблем тестування вебінтерфейсів є динамічність сучасних вебсторінок. Елементи можуть змінювати свої ідентифікатори, розташування або завантажуватися асинхронно після отримання даних із сервера. Це ускладнює процес створення стабільних тестових сценаріїв [7].

Для підвищення підтримуваності тестів використовується архітектурний підхід Page Object Model. Даний підхід дозволяє відокремити логіку взаємодії зі сторінками вебзастосунку від тестових сценаріїв, що спрощує супровід програмного забезпечення тестування.

У сучасних вебзастосунках UI-тестування активно використовується для:

- регресійного тестування;
- перевірки основних бізнес-сценаріїв;
- тестування сумісності браузерів;
- перевірки стабільності інтерфейсу після оновлень.

1.4 Аналіз інструментів тестування вебзастосунків

Для тестування вебзастосунків існує велика кількість сучасних програмних засобів, які дозволяють автоматизувати перевірку функціональності вебінтерфейсу та скоротити час виконання тестових сценаріїв.

Серед найбільш поширених інструментів тестування вебзастосунків можна виділити:

- Selenium;
- Cypress;
- Playwright;

- TestNG;
- JUnit.

Одним із найбільш популярних інструментів є Selenium WebDriver, який дозволяє керувати браузером та виконувати тестові сценарії у середовищі реального браузера [8].

Основними перевагами Selenium є:

- підтримка різних браузерів;
- підтримка різних мов програмування;
- можливість інтеграції з тестовими фреймворками;
- підтримка Page Object Model;
- масштабованість тестових сценаріїв.

Для реалізації програмного забезпечення тестування у даній роботі обрано Selenium WebDriver у поєднанні з мовою програмування Java та фреймворком TestNG.

TestNG використовується для запуску тестових сценаріїв, групування тестів, формування звітів, виконання assertions, керування послідовністю тестування.

Для керування залежностями та збирання проєкту використовується Maven, а середовищем розробки є IntelliJ IDEA.

Порівняння основних інструментів тестування наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння інструментів тестування вебзастосунків

Інструмент	Мова	Підтримка браузерів	Особливості
Selenium	Java, Python, C#	Так	Висока гнучкість
Cypress	JavaScript	Частково	Простота використання
Playwright	JavaScript, Java	Так	Висока швидкість
TestNG	Java	Ні	Організація тестів

За результатами аналізу для реалізації роботи було обрано Selenium WebDriver та TestNG, оскільки вони забезпечують достатню гнучкість, підтримку браузерів та можливість реалізації складних тестових сценаріїв.

1.5 Висновки до розділу 1

У першому розділі було розглянуто особливості тестування вебзастосунків та проаналізовано сучасні методи перевірки функціональності вебінтерфейсів.

Досліджено основні види тестування вебзастосунків, зокрема функціональне, UI- та регресійне тестування. Встановлено, що тестування вебінтерфейсів є важливою складовою забезпечення якості сучасного програмного забезпечення.

Проведено аналіз сучасних інструментів тестування вебзастосунків та обґрунтовано вибір Selenium WebDriver, Java та TestNG для реалізації програмного забезпечення тестування вебінтерфейсу.

Результати проведеного аналізу стали основою для проєктування архітектури програмного забезпечення тестування вебзастосунку, що буде розглянуто у наступному розділі.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

2.1 Архітектура програмного забезпечення тестування

Проєктування структури програмного забезпечення є важливим етапом розроблення систем тестування вебзастосунків. Від структури програмного забезпечення залежить зручність супроводу тестових сценаріїв, можливість масштабування системи та стабільність виконання тестів.

У роботі реалізовано модульну структуру програмного забезпечення тестування вебзастосунку. Такий підхід дозволяє розділити систему на окремі компоненти, кожен з яких виконує власні функції. Структура системи наведена на рисунку 2.1 складається з таких основних модулів:

- модуль запуску браузера;
- модуль взаємодії з вебелементами;
- модуль тестових сценаріїв;
- модуль Page Object;
- модуль перевірок;
- модуль формування звітності.

Рисунок 2.1 – Структура розробленого програмного забезпечення тестування вебзастосунку

Модуль запуску браузера відповідає за ініціалізацію Selenium WebDriver та налаштування середовища виконання тестів. Для виконання тестування використовується браузер Google Chrome та ChromeDriver.

Модуль взаємодії з вебелементами забезпечує пошук елементів інтерфейсу вебсторінки за допомогою локаторів:

- id;
- className;
- xpath;
- cssSelector.

Модуль тестових сценаріїв містить реалізацію функціональних перевірок вебзастосунку. У межах роботи реалізовано тестові сценарії:

- авторизації користувача;
- додавання товару до кошика;
- видалення товару;
- оформлення замовлення;
- виходу з облікового запису.

Модуль перевірок використовується для виконання assertions та аналізу результатів тестування. Перевірки дозволяють контролювати:

- коректність URL;
- наявність вебелементів;
- текст повідомлень;
- відображення кнопок;
- успішність виконання сценарію.

Модуль формування звітності забезпечує створення звітів про результати виконання тестів.

На рисунку 2.2 представлено архітектуру програмного забезпечення тестування вебзастосунку. Центральною частиною системи є програмне забезпечення тестування, яке включає модулі тестових сценаріїв, Page Object

Model, службові утиліти та модуль взаємодії з браузером через Selenium WebDriver.

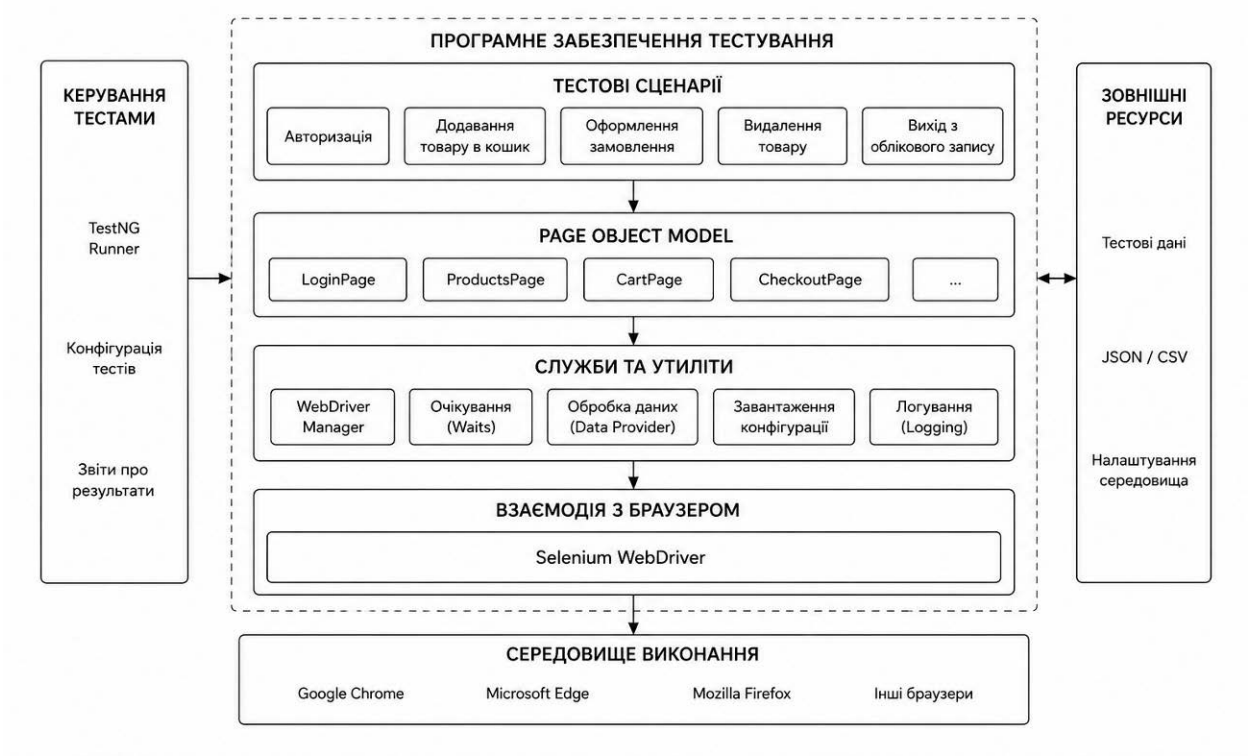


Рисунок 2.2 – Архітектура системи

У верхній частині схеми розташовано тестові сценарії, які реалізують перевірку основних функцій вебзастосунку, зокрема авторизацію користувача, додавання товару до кошика, оформлення замовлення, видалення товару та вихід із облікового запису. Для організації взаємодії з вебсторінками використовується архітектурний підхід Page Object Model, що містить окремі класи сторінок вебзастосунку.

Модуль служб та утиліт забезпечує підтримку роботи системи тестування, включаючи керування WebDriver, механізми очікування елементів, обробку тестових даних, завантаження конфігурацій та логування результатів виконання тестів. Взаємодія з браузерами здійснюється за допомогою Selenium WebDriver.

У лівій частині схеми представлено компоненти керування тестуванням, які включають TestNG Runner, конфігурацію тестів та

формування звітів про результати виконання сценаріїв. У правій частині відображено зовнішні ресурси, зокрема тестові дані, файли JSON і CSV, а також налаштування середовища виконання.

У нижній частині схеми наведено середовище виконання тестів, яке включає браузери Google Chrome, Microsoft Edge, Mozilla Firefox та інші браузери, що підтримуються Selenium WebDriver.

2.2 Використання Page Object Model

Одним із найбільш поширених архітектурних підходів до організації програмного забезпечення тестування вебзастосунків є Page Object Model. Даний підхід використовується для підвищення підтримуваності та повторного використання тестового коду [5].

Основна ідея Page Object Model полягає у створенні окремих класів для кожної вебсторінки або логічного компонента вебзастосунку. Кожен Page Object містить:

- локатори вебелементів;
- методи взаємодії зі сторінкою;
- логіку роботи з інтерфейсом.

У межах роботи було створено такі Page Object класи:

- LoginPage;
- ProductsPage;
- CartPage;
- CheckoutPage.

Наприклад, клас LoginPage містить:

- поля введення логіна та пароля;
- кнопку входу;
- метод виконання авторизації.

Використання Page Object Model дозволяє:

- уникнути дублювання коду;

- спростити підтримку тестів;
- ізолювати зміни інтерфейсу;
- покращити структуру проєкту.

Структуру взаємодії тестових сценаріїв із Page Object наведено на рисунку 2.2.

Завдяки використанню Page Object Model тестові сценарії стають більш структурованими та зрозумілими, а внесення змін у структуру вебзастосунку не потребує редагування всіх тестів.

2.3 Методи пошуку вебелементів на сторінці

Під час реалізації програмного забезпечення тестування важливим етапом є визначення способів пошуку елементів вебінтерфейсу. Для взаємодії з кнопками, полями введення, посиланнями, списками та іншими елементами сторінки Selenium WebDriver використовує систему локаторів.

Локатор — це спосіб ідентифікації HTML-елемента на вебсторінці для подальшої взаємодії з ним у тестовому сценарії. Основні методи пошуку зображені на рисунку 2.3.

У межах роботи було використано декілька основних методів пошуку вебелементів.

Локатор id доцільно використовувати для елементів, які мають унікальний ідентифікатор у DOM-структурі. Такий підхід є одним із найбільш надійних і швидких, оскільки забезпечує однозначне знаходження елемента на сторінці. Саме тому у межах роботи цей локатор використовувався для кнопки авторизації, полів введення логіна та пароля.

Пошук за атрибутом id є одним із найбільш стабільних і швидких способів знаходження елементів. Приклад: driver.findElement(By.id("login-button")). Такий підхід використовується у випадках, коли HTML-елемент має унікальний ідентифікатор.

Рисунок 2.3 – Основні методи пошуку вебелементів у Selenium WebDriver

Локатор className ефективний у випадках, коли необхідно працювати з групою однотипних елементів, наприклад зі списком товарів, картками або блоками інтерфейсу. Проте через повторюваність класів на сторінці він не завжди забезпечує однозначний пошук. Метод дозволяє знайти елемент за CSS-класом. Приклад: driver.findElement(By.className("inventory_item")). Використовується для роботи з групою однотипних елементів.

Локатор name зручний для взаємодії з полями введення форм, де присутній відповідний HTML-атрибут. Його доцільно застосовувати у формах авторизації, пошуку або реєстрації. Пошук за атрибутом name часто використовується для полів введення. Приклад: driver.findElement(By.name("username")).

Локатор XPath доцільно використовувати для складних або вкладених елементів, коли інші атрибути відсутні або є неунікальними. Він дозволяє будувати гнучкий шлях до елемента через DOM-структуру сторінки, однак є більш чутливим до змін верстки. Пошук за XPath дозволяє будувати складні шляхи до елементів HTML-документа. Приклад:

driver.findElement(By.xpath("//button[@id='login-button']")). XPath доцільно використовувати у випадках, коли інші локатори відсутні або недостатньо унікальні.

Локатор CSS Selector є гнучким та швидким способом пошуку елементів. Його зручно використовувати для кнопок, меню, навігаційних блоків, списків товарів та елементів зі складною структурою вкладеності. CSS Selector є гнучким способом пошуку елементів за атрибутами або структурою DOM. Приклад: driver.findElement(By.cssSelector(".shopping_cart_link")). Цей метод є одним із найпоширеніших через високу швидкість виконання.

Локатор Link Text застосовується переважно для роботи з гіперпосиланнями та елементами навігації, де пошук виконується за текстовим вмістом посилання. Пошук за Link Text застосовується для пошуку гіперпосилань за текстом. Приклад: driver.findElement(By.linkText("Logout")).

Раціональний вибір локатора безпосередньо впливає на стабільність роботи тестового сценарію, швидкість його виконання та зручність подальшого супроводу програмного забезпечення тестування.

У процесі розроблення програмного забезпечення тестування вебзастосунку основними були обрані локатори id, CSS Selector та XPath.

Перевага надавалася локатору id, оскільки він забезпечує стабільний пошук елементів і є найменш залежним від змін структури сторінки.

CSS Selector використовувався для елементів списків, кнопок навігації та блоків інтерфейсу.

XPath застосовувався у випадках, коли необхідно було знайти вкладені елементи або елементи без унікального ідентифікатора.

Такий підхід дозволив забезпечити стабільність тестових сценаріїв, зменшити кількість помилок під час пошуку вебелементів та спростити подальшу підтримку програмного забезпечення тестування.

2.3 UML-моделювання системи

Для проєктування програмного забезпечення тестування вебзастосунку було використано UML-моделювання. UML-діаграми дозволяють візуально представити структуру системи та взаємодію між її компонентами.

У межах роботи було створено Activity Diagram. UML-діаграма наведено на рисунку 2.3. Activity Diagram відображає послідовність виконання тестових сценаріїв та логіку взаємодії з вебзастосунком:

- запуск браузера;
- авторизація користувача;
- виконання тестових сценаріїв;
- формування звітності.

Рисунок 2.3 – Activity Diagram процесу тестування авторизації

Використання UML-моделювання дозволило структуровано описати архітектуру програмного забезпечення та спростити подальшу реалізацію тестових сценаріїв.

2.4 Алгоритм виконання тестових сценаріїв

Алгоритм роботи системи наведено на рисунку 2.4.

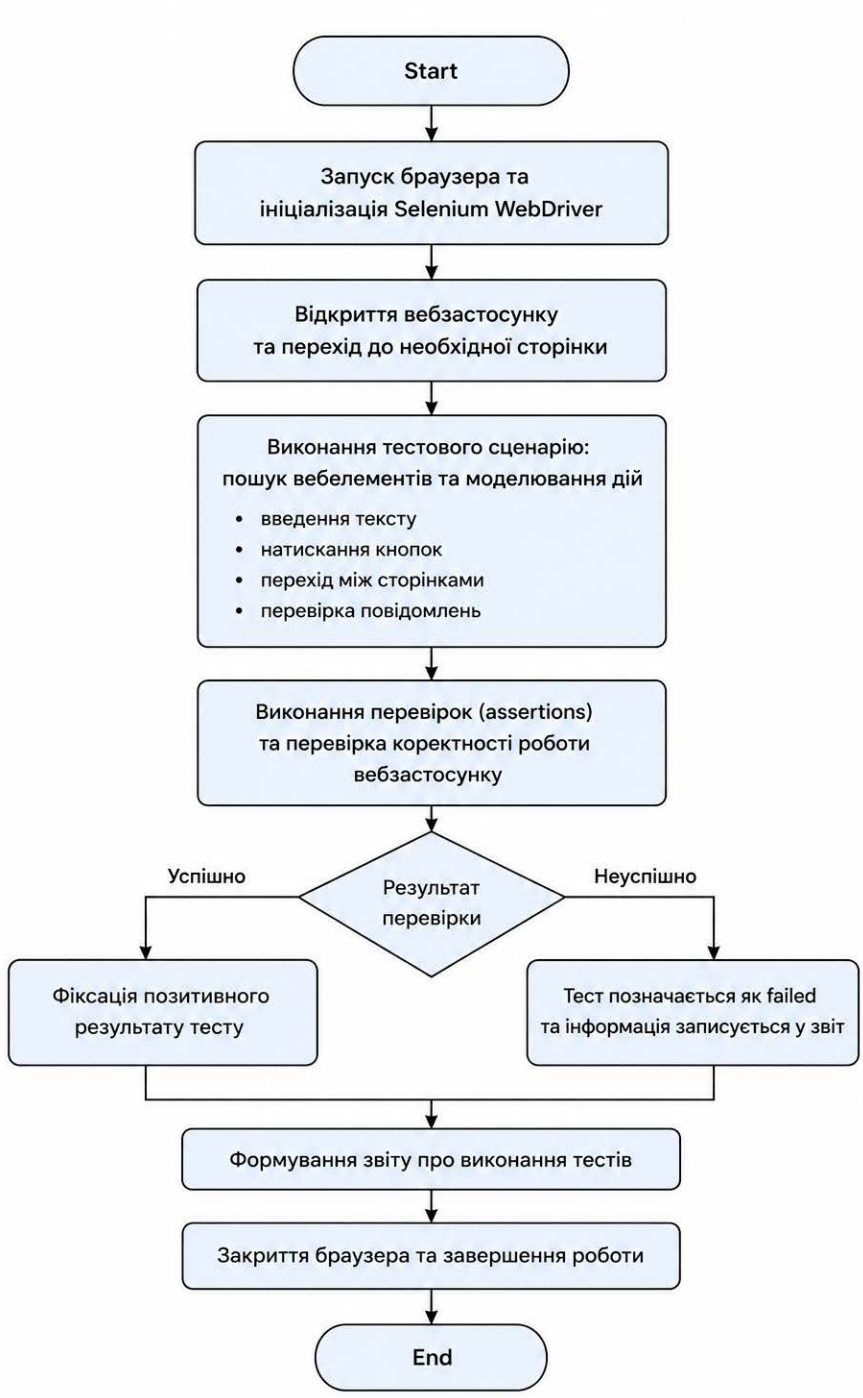


Рисунок 2.4 – Алгоритм виконання тестових сценаріїв

Алгоритм роботи програмного забезпечення тестування вебзастосунку складається з кількох послідовних етапів.

На першому етапі виконується запуск браузера та ініціалізація Selenium WebDriver. Після цього система відкриває вебзастосунок та переходить до необхідної сторінки.

Далі виконується пошук вебелементів за допомогою локаторів та моделювання дій користувача:

- введення тексту;
- натискання кнопок;
- перехід між сторінками;
- перевірка повідомлень.

Після завершення сценарію виконуються assertions, які перевіряють коректність роботи вебзастосунку.

У разі успішного виконання тесту система фіксує позитивний результат. Якщо перевірка не проходить, тест позначається як failed та інформація записується у звіт.

Запропонований алгоритм дозволяє реалізувати стабільне виконання тестових сценаріїв та забезпечує повторне використання програмного коду.

2.5 Висновки до розділу 2

У другому розділі було виконано проєктування програмного забезпечення тестування вебзастосунку.

Розроблено модульну архітектуру системи, яка включає модулі запуску браузера, взаємодії з вебелементами, виконання тестових сценаріїв та формування звітності.

У роботі використано архітектурний підхід Page Object Model, що дозволив покращити структуру програмного забезпечення та спростити підтримку тестових сценаріїв.

Також було виконано UML-моделювання системи та розроблено алгоритм виконання тестових сценаріїв.

Результати проєктування стали основою для реалізації програмного забезпечення тестування вебзастосунку, що буде розглянуто у наступному розділі.

3 РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

3.1 Налаштування середовища розробки для реалізації програмного забезпечення тестування вебзастосунку

Розроблення програмного забезпечення тестування вебзастосунку потребує попередньої підготовки програмного середовища, яке забезпечує створення, запуск та виконання тестових сценаріїв. Коректно налаштоване середовище є основою стабільної роботи тестів та дозволяє виконувати багаторазовий запуск перевірок без ручного втручання.

Для реалізації програмного забезпечення у роботі було використано мову програмування Java, оскільки вона широко застосовується для створення UI-тестів та має велику кількість готових бібліотек для тестування вебзастосунків. Java забезпечує об’єктно-орієнтований підхід до розробки та дозволяє зручно структурувати тестовий проєкт.

Як середовище розробки було використано IntelliJ IDEA. Дане IDE забезпечує зручне написання коду, навігацію між класами, автоматичне форматування, роботу з бібліотеками та запуск тестових сценаріїв безпосередньо з редактора [9].

Для керування залежностями проєкту застосовано Apache Maven. Використання Maven дозволило автоматизувати підключення необхідних бібліотек, спростити збірку проєкту та забезпечити зручне керування конфігурацією тестового середовища [10].

Файл pom.xml містить залежності, необхідні для запуску тестів, зокрема:

- Selenium WebDriver;
- TestNG;
- WebDriverManager;
- Maven Surefire Plugin.

Фрагмент конфігурації pom.xml наведено у додатку А.:

Для взаємодії з браузером використовувався Selenium WebDriver. Він забезпечує керування браузером через програмний код та дозволяє імітувати дії користувача:

- відкриття вебсторінки;
- введення тексту;
- натискання кнопок;
- вибір елементів зі списків;
- перевірку стану сторінки.

Для запуску тестування було використано браузер Google Chrome та драйвер ChromeDriver, який виступає посередником між Selenium WebDriver та браузером [11].

ChromeDriver дозволяє:

- запускати браузер автоматично;
- відкривати заданий URL;
- взаємодіяти з HTML-елементами сторінки;
- закривати браузер після завершення сценарію.

Для автоматичного керування версією драйвера було використано бібліотеку WebDriverManager, що дозволяє уникнути ручного налаштування драйвера у системі.

Ініціалізація драйвера виконується наступним чином:

```
WebDriverManager.chromedriver().setup();
WebDriver driver = new ChromeDriver();
```

Для запуску тестових сценаріїв у межах роботи було використано TestNG. Даний фреймворк дозволяє:

- групувати тестові сценарії;
- виконувати запуск окремих тестів або повного набору;
- виконувати assertions;
- отримувати звіти про проходження тестів;
- визначати статус виконання тесту.

Після запуску тестів система автоматично формує звіт із результатами виконання перевірок.

Таким чином, сформоване середовище розробки забезпечило повний цикл створення програмного забезпечення тестування вебзастосунку – від написання тестового коду до запуску перевірок та отримання результатів тестування.

3.2 Розроблення структури тестового проєкту

Після налаштування програмного середовища наступним етапом стала побудова структури тестового проєкту. Організація структури програмного забезпечення є важливою складовою розроблення, оскільки впливає на читабельність коду, зручність супроводу та можливість подальшого розширення тестового покриття.

У межах роботи програмне забезпечення тестування вебзастосунку було реалізовано як окремий Java-проєкт із логічним поділом на пакети відповідно до функціонального призначення компонентів. Такий підхід дозволив розділити код за зонами відповідальності та спростити роботу з тестовими сценаріями.

Структура проєкту включає кілька основних пакетів (рис. 3.1):

- base — базова логіка запуску браузера та ініціалізації WebDriver;
- pages — класи сторінок вебзастосунку, реалізовані відповідно до Page Object Model;
- tests — тестові сценарії перевірки функціональності;
- utils — службові методи та допоміжні функції;
- resources — конфігураційні файли та тестові дані.

У пакеті base розміщено базовий клас, який відповідає за створення WebDriver-сесії, відкриття браузера, перехід за URL вебзастосунку та завершення роботи браузера після виконання тесту. Саме через цей клас виконується початкова конфігурація середовища тестування. Ініціалізація

драйвера виконується через створення екземпляра ChromeDriver, а завершення роботи — через виклик методу driver.quit(). Фрагмент реалізації наведено у додатку Б.

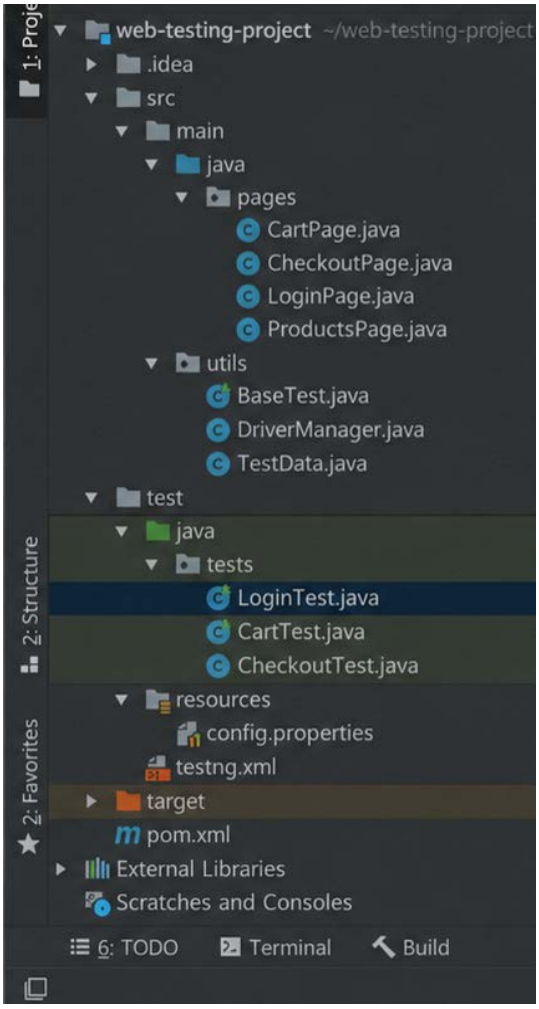


Рисунок 3.1 – Структура проєкту

Пакет pages реалізує шаблон Page Object Model. Для кожної сторінки вебзастосунку створено окремий клас із локаторами вебелементів та методами взаємодії з ними. Це дозволяє відокремити логіку роботи зі сторінкою від логіки тестового сценарію.

У межах роботи були створені окремі Page Object класи для:

- сторінки авторизації;
- сторінки списку товарів;
- сторінки кошика;

- сторінки оформлення замовлення.

Кожен клас містить локатори основних елементів сторінки та набір методів для взаємодії з ними. Наприклад, метод авторизації реалізує введення логіна, введення пароля та натискання кнопки Login. Детальна реалізація наведена у додатку Б.

Пакет tests містить набір тестових сценаріїв перевірки вебзастосунку. Саме тут реалізовано послідовність дій користувача та перевірки очікуваного результату. Кожен тест відповідає окремому функціональному сценарію.

У межах роботи були сформовані сценарії перевірки:

- входу користувача в систему;
- відкриття сторінки товарів;
- додавання товару до кошика;
- видалення товару з кошика;
- оформлення замовлення;
- завершення користувацької сесії.

Для перевірки коректності виконання сценаріїв використовувалися assertions TestNG. Перевірка виконувалася через аналіз URL сторінки, стану вебелементів, наявності повідомлень або змін в інтерфейсі після виконання користувацької дії.

У пакеті utils реалізовано допоміжні методи, які використовуються повторно в різних тестах. До таких методів належать:

- очікування появи вебелемента на сторінці;
- перевірка доступності елемента;
- отримання тексту елемента;
- робота зі скролом сторінки;
- допоміжні перевірки стану елементів.

Виділення службової логіки в окремий пакет дозволило уникнути дублювання коду та спростило структуру тестових сценаріїв.

Окремо в проєкті використовується директорія **resources**, де зберігаються параметри конфігурації, тестові дані та налаштування

середовища виконання. Зокрема там можуть зберігатися базова URL-адреса вебзастосунку, тестові облікові записи та додаткові параметри запуску.

Загалом побудована структура тестового проєкту забезпечує:

- зручну навігацію між файлами;
- розділення логіки за функціональними блоками;
- повторне використання коду;
- зручність внесення змін;
- можливість масштабування тестового покриття.

Запропонована структура була використана як основа для подальшої реалізації тестових сценаріїв і забезпечила зручну організацію програмного забезпечення тестування вебзастосунку.

3.3 Реалізація класів Page Object для взаємодії з вебінтерфейсом

Одним із ключових етапів розроблення програмного забезпечення тестування вебзастосунку стала реалізація класів Page Object. Використання даного підходу дозволило структурувати взаємодію з вебінтерфейсом та відокремити логіку роботи зі сторінками від логіки тестових сценаріїв.

У роботі використано шаблон Page Object Model, відповідно до якого кожна окрема сторінка вебзастосунку представлена окремим Java-класом. Такий клас містить опис елементів сторінки та методи взаємодії з ними.

Основною перевагою даного підходу є те, що тестовий сценарій працює не напряму з HTML-елементами, а через відповідний Page Object клас. Це підвищує читабельність коду та значно спрощує подальшу підтримку тестового проєкту.

У межах роботи було реалізовано окремі класи для основних сторінок вебзастосунку:

- сторінка авторизації;
- сторінка каталогу товарів;
- сторінка кошика;

– сторінка оформлення замовлення.

Кожен Page Object клас містить:

– локатори вебелементів;

– методи взаємодії з елементами;

– службові дії користувача в межах сторінки.

Наприклад, для сторінки авторизації реалізовано методи:

– введення логіна;

– введення пароля;

– натискання кнопки входу;

– перевірка успішної авторизації.

У процесі роботи користувач взаємодіє зі сторінкою через послідовність дій: відкриття форми авторизації, введення облікових даних та натискання кнопки Login. У програмному забезпеченні ці дії реалізовано окремими методами відповідного Page Object класу. Фрагмент реалізації наведено у додатку Б та представлено на картинці 3.2.

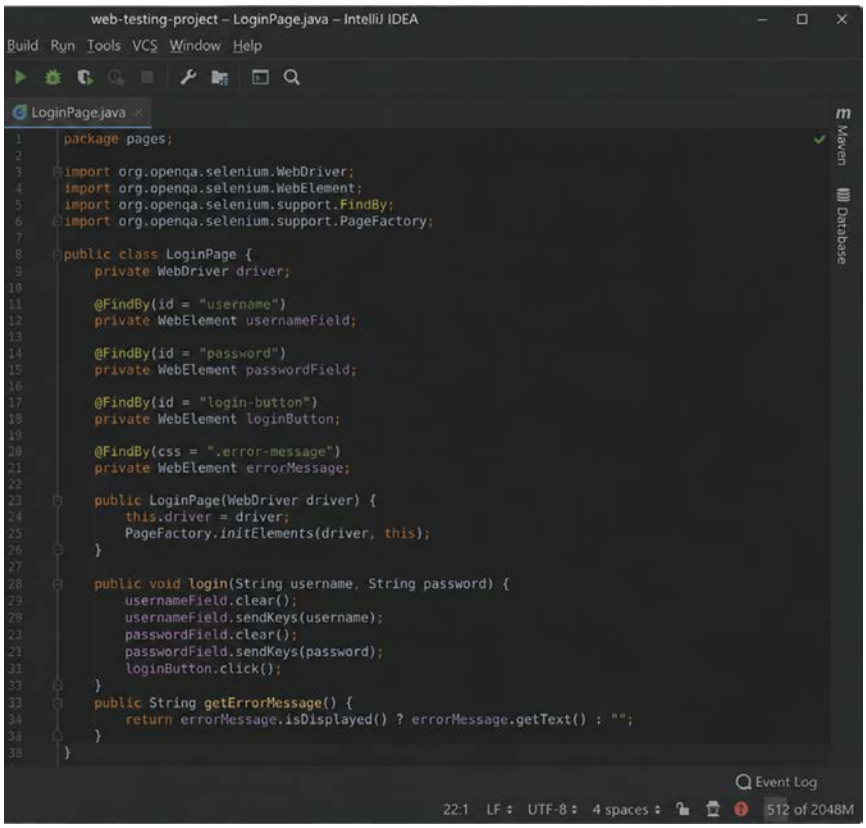


Рисунок 3.2 – Фрагмент реалізації Page Object класу сторінки Login

Для сторінки каталогу товарів було реалізовано методи:

- отримання списку товарів;
- відкриття картки товару;
- додавання товару до кошика;
- перевірка відображення кнопки Add to cart;
- перевірка зміни стану кнопки після додавання.

Для сторінки кошика створено окремі методи перегляду вмісту кошика, видалення товару та переходу до оформлення замовлення.

Для сторінки оформлення замовлення реалізовано взаємодію з формою введення даних користувача. Вона включає роботу з полями імені, прізвища, поштового індексу та кнопкою завершення оформлення покупки.

Пошук елементів у Page Object класах здійснювався через локатори Selenium WebDriver. У процесі реалізації переважно використовувалися:

- By.id();
- By.cssSelector();
- By.xpath().

Найчастіше застосовувався локатор By.id(), оскільки він забезпечує швидке та стабільне знаходження елементів інтерфейсу. У випадках складнішої DOM-структури використовувався By.xpath(). Для кнопок, блоків списків та елементів навігації використовувався By.cssSelector().

Важливою перевагою використання Page Object Model стало зменшення дублювання коду. Якщо певний вебелемент використовується у кількох тестах, його описується один раз у відповідному класі сторінки, після чого метод викликається повторно з різних тестових сценаріїв.

Такий підхід дозволяє:

- централізовано зберігати локатори елементів;
- швидко оновлювати код при зміні інтерфейсу;
- повторно використовувати методи взаємодії;
- спростити написання нових тестових сценаріїв.

У разі зміни структури сторінки достатньо змінити локатор або метод лише в одному Page Object класі без редагування всіх тестів, які використовують цей елемент.

Таким чином реалізація Page Object класів стала основою взаємодії програмного забезпечення з вебінтерфейсом та забезпечила гнучку, структуровану і масштабовану архітектуру тестового проєкту. Реалізовані класи були використані у тестових сценаріях перевірки функціональності вебзастосунку, що розглядаються у наступному підрозділі.

3.4 Реалізація тестових сценаріїв перевірки функціональності вебзастосунку

Після формування структури проєкту та реалізації Page Object класів наступним етапом стала розробка тестових сценаріїв перевірки функціональності вебзастосунку. Саме тестові сценарії є основною частиною програмного забезпечення тестування, оскільки забезпечують перевірку поведінки системи з точки зору користувача.

Метою реалізації тестових сценаріїв була перевірка основних функціональних можливостей вебзастосунку та контроль коректності роботи користувацького інтерфейсу після виконання типових дій.

У межах роботи було реалізовано набір сценаріїв, що охоплюють основні дії користувача під час роботи із вебзастосунком.

До реалізованих перевірок належать:

- перевірка авторизації користувача;
- перевірка входу з некоректними даними;
- перевірка відкриття каталогу товарів;
- перевірка додавання товару до кошика;
- перевірка видалення товару з кошика;
- перевірка переходу до оформлення замовлення;
- перевірка завершення користувацької сесії.

Першим реалізованим сценарієм стала перевірка авторизації користувача. У ході виконання тесту система відкриває сторінку входу, заповнює поле логіна та пароля тестовими даними, після чого виконує натискання кнопки Login. Після виконання входу перевіряється URL-адреса відкритої сторінки та відображення каталогу товарів.

Окремо реалізовано негативний сценарій входу з неправильними обліковими даними. У цьому випадку після введення некоректного логіна або пароля система перевіряє наявність повідомлення про помилку авторизації. Це дозволяє перевірити правильність обробки помилкових даних на рівні інтерфейсу.

Наступним сценарієм стала перевірка додавання товару до кошика. У процесі тестування система знаходить товар у каталозі та виконує натискання кнопки Add to cart. Після цього перевіряється зміна стану кнопки та оновлення індикатора кількості товарів у кошику.

Також реалізовано сценарій видалення товару з кошика. У даному випадку виконується відкриття сторінки кошика, пошук доданого товару та натискання кнопки видалення. Після виконання дії перевіряється зміна стану кошика або відсутність товару в переліку.

Для перевірки завершення користувацького сценарію реалізовано тест оформлення замовлення. Він включає:

- перехід до сторінки checkout;
- введення користувацьких даних у форму;
- підтвердження замовлення;
- перевірку успішного завершення операції.

Результатом виконання цього сценарію є відображення повідомлення про успішне завершення покупки.

Кожен тестовий сценарій реалізовано як окремий тестовий метод у фреймворку TestNG. Для запуску перевірки використовувалась анотація @Test, а перевірка очікуваного результату виконувалася за допомогою assertions. Фрагменти реалізації тестових сценаріїв наведені у додатку Б.

Під час виконання перевірок аналізувалися такі результати:

- URL відкритої сторінки;
- текст повідомлень;
- наявність або відсутність вебелемента;
- зміна стану кнопок;
- кількість товарів у кошику;
- успішність переходу між сторінками.

Використання окремих тестових сценаріїв дозволило охопити перевіркою основні функціональні можливості вебзастосунку та забезпечити повторне виконання перевірок після змін у програмному продукті.

Розроблений набір тестів може використовуватись як для локального запуску перевірок під час розроблення, так і для регулярного регресійного тестування вебзастосунку після оновлення функціональності.

Таким чином реалізовані тестові сценарії забезпечили практичну перевірку працездатності створеного програмного забезпечення тестування та стали основою подальшого формування звітності про результати виконання тестів, що буде розглянуто у наступному підрозділі.

3.5 Формування звітів про результати тестування

Важливою складовою програмного забезпечення тестування вебзастосунку є формування звітів про результати виконання тестових сценаріїв. Наявність звітності дозволяє отримати узагальнену інформацію про проходження тестів, швидко виявляти помилки та оцінювати загальний стан функціональності вебзастосунку після перевірки.

Після завершення запуску тестів система автоматично формує звіт із результатами виконання кожного сценарію. У межах роботи для цього використовувався вбудований механізм звітності фреймворку TestNG.

Звіт створюється після завершення тестового запуску та містить інформацію щодо виконання кожного тестового сценарію окремо.

У сформованому звіті відображаються:

- загальна кількість виконаних тестів;
- кількість успішно виконаних перевірок;
- кількість тестів, що завершилися помилкою;
- кількість пропущених сценаріїв;
- тривалість виконання тестового запуску;
- статус проходження кожного окремого тесту.

Статус результату подається у стандартному форматі:

- Passed — сценарій виконано успішно;
- Failed — під час виконання виявлено помилку;
- Skipped — сценарій пропущено.

Окрім загального статусу виконання, звіт дозволяє деталізовано переглядати результати окремого сценарію та визначати, на якому саме етапі виникла помилка.

У випадку неуспішного проходження перевірки звіт фіксує:

- назву тестового сценарію;
- текст помилки;
- місце зупинки виконання;
- результат assertion;
- час завершення тесту.

Це значно спрощує подальший аналіз дефекту та пришвидшує пошук причин некоректної роботи вебзастосунку.

Під час виконання роботи сформовані звіти використовувались для аналізу результатів тестування сценаріїв:

- авторизації користувача;
- роботи з каталогом товарів;
- додавання товару до кошика;
- оформлення замовлення;
- завершення сесії користувача.

Після кожного запуску результати зберігалися у звітному файлі та використовувалися для повторного аналізу стабільності виконання тестів.

Формування звітності дало можливість оцінювати:

- повноту проходження тестового набору;
- стабільність повторного запуску перевірок;
- наявність дефектів у функціональності вебзастосунку;
- середній час виконання перевірки.

Отримані результати використовувалися для подальшого експериментального аналізу та порівняння часу ручного й програмного тестування, що буде розглянуто у четвертому розділі роботи.

Приклад сформованого звіту за результатами виконання тестового запуску наведено на рисунку 3.4.

Загалом реалізований механізм звітності дозволив автоматизувати фіксацію результатів перевірки вебзастосунку та забезпечив зручний інструмент аналізу проходження тестових сценаріїв.

3.6 Висновки до розділу 3

У третьому розділі було розглянуто процес розроблення програмного забезпечення тестування вебзастосунку та практичну реалізацію основних компонентів тестового проєкту.

Виконано налаштування середовища розробки, що включає використання мови програмування Java, Selenium WebDriver, TestNG, Maven, IntelliJ IDEA та браузера Google Chrome. Сформоване середовище забезпечило можливість створення, запуску та виконання тестових сценаріїв перевірки вебінтерфейсу.

У процесі роботи було розроблено структуру тестового проєкту з поділом на окремі функціональні модулі. Такий підхід дозволив розмежувати логіку запуску браузера, взаємодію зі сторінками вебзастосунку, тестові сценарії та допоміжні службові методи.

Реалізовано Page Object класи для основних сторінок вебзастосунку. Використання Page Object Model дало можливість структурувати взаємодію з вебелементами, спростити супровід тестового коду та підвищити повторне використання окремих методів у межах різних тестових сценаріїв.

Також було створено набір тестових сценаріїв перевірки функціональності вебзастосунку. Реалізовані перевірки охоплюють основні дії користувача, зокрема авторизацію в системі, роботу з каталогом товарів, додавання товару до кошика, оформлення замовлення та завершення користувацької сесії.

Окремо реалізовано механізм формування звітності про результати виконання тестів. Сформовані звіти дозволяють аналізувати проходження перевірок, виявляти помилки та оцінювати стабільність роботи тестового набору.

Отримані результати підтвердили працездатність розробленого програмного забезпечення тестування вебзастосунку та можливість його практичного використання для перевірки вебінтерфейсів у процесі забезпечення якості програмного забезпечення.

Результати, отримані у процесі реалізації та запуску тестів, є основою для подальшого експериментального дослідження та аналізу ефективності використання програмного забезпечення тестування, що буде розглянуто у наступному розділі.

РОЗДІЛ 4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ РОБОТИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

4.1 Організація експериментального дослідження

Для перевірки працездатності розробленого програмного забезпечення тестування вебзастосунку було проведено експериментальне дослідження. Його метою стала практична оцінка роботи тестових сценаріїв, перевірка стабільності повторного запуску та аналіз отриманих результатів тестування.

Експеримент проводився після завершення реалізації тестового проєкту та включав запуск набору тестових сценаріїв у браузері Google Chrome з подальшим аналізом результатів виконання.

Для проведення дослідження використовувалося таке програмне середовище:

- мова програмування Java;
- Selenium WebDriver;
- TestNG;
- Maven;
- IntelliJ IDEA;
- Google Chrome;
- ChromeDriver.

Об’єктом перевірки виступав вебзастосунок із реалізованим користувацьким інтерфейсом, що містить типові функціональні сценарії взаємодії користувача із системою.

У межах експерименту було виконано запуск тестових сценаріїв перевірки:

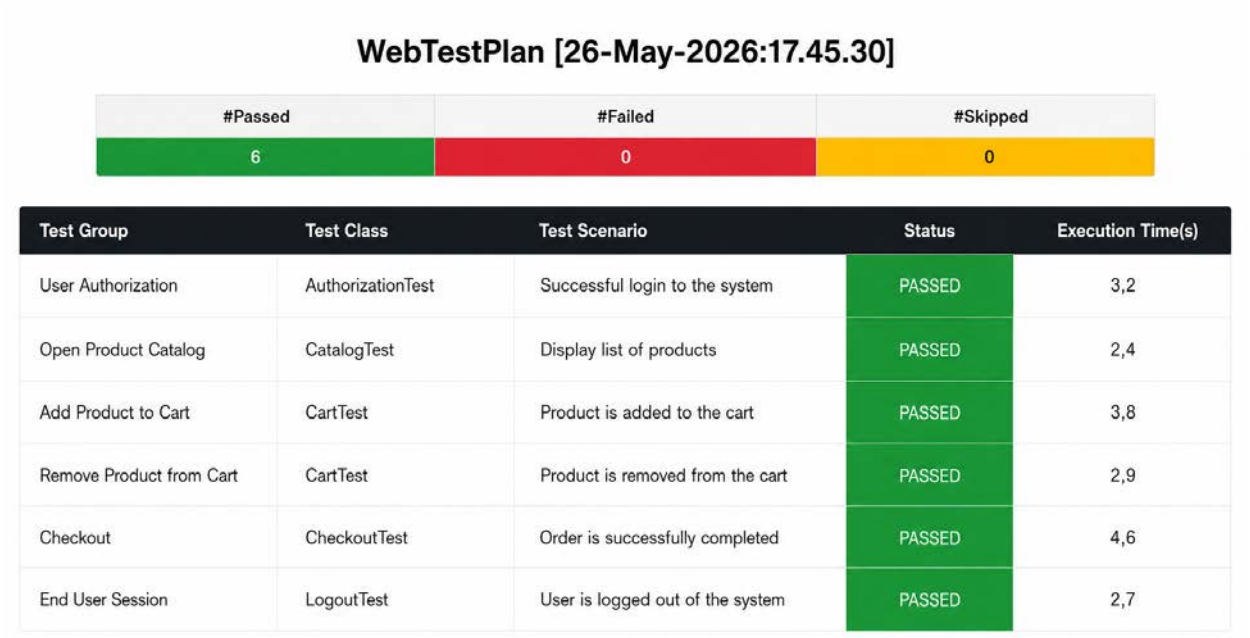
- авторизації користувача;
- відкриття каталогу товарів;
- додавання товару до кошика;

- видалення товару з кошика;
- оформлення замовлення;
- завершення користувацької сесії.

У процесі дослідження фіксувалися:

- загальна кількість тестів;
- кількість успішних проходжень;
- кількість помилок під час виконання;
- середня тривалість запуску;
- підсумкові результати у звіті TestNG.

Результати запуску тестів відображені на рисунку 4.1 та використовувалися для подальшого аналізу.



WebTestPlan [26-May-2026:17.45.30]

#Passed	#Failed	#Skipped
6	0	0

Test Group	Test Class	Test Scenario	Status	Execution Time(s)
User Authorization	AuthorizationTest	Successful login to the system	PASSED	3,2
Open Product Catalog	CatalogTest	Display list of products	PASSED	2,4
Add Product to Cart	CartTest	Product is added to the cart	PASSED	3,8
Remove Product from Cart	CartTest	Product is removed from the cart	PASSED	2,9
Checkout	CheckoutTest	Order is successfully completed	PASSED	4,6
End User Session	LogoutTest	User is logged out of the system	PASSED	2,7

Рисунок 4.1 – Результати тесту зі звіту TestNG

Проведене експериментальне дослідження дозволяє оцінити практичну придатність створеного програмного забезпечення тестування та визначити його можливість використання у задачах перевірки функціональності вебзастосунків.

4.2 Аналіз результатів виконання тестових сценаріїв

Після завершення запуску тестового набору було проведено аналіз отриманих результатів виконання сценаріїв перевірки функціональності вебзастосунку. Основною метою аналізу стало визначення працездатності розробленого програмного забезпечення тестування, перевірка стабільності проходження сценаріїв та оцінка коректності роботи вебінтерфейсу після виконання користувацьких дій.

У межах експериментального запуску було виконано серію тестів, що охоплюють основний функціонал вебзастосунку. Запуск здійснювався в однаковому програмному середовищі з використанням одного браузера та однакових тестових даних.

Під час аналізу оцінювалися такі показники:

- кількість виконаних тестових сценаріїв;
- кількість успішних проходжень;
- кількість сценаріїв із помилками;
- стабільність повторного запуску;
- середній час виконання одного тесту;
- загальна тривалість виконання тестового набору.

У результаті тестового запуску було виконано **6 тестових сценаріїв**, що перевіряють основні дії користувача у вебзастосунку.

До них належать:

- вхід користувача до системи;
- відкриття каталогу товарів;
- додавання товару до кошика;
- видалення товару з кошика;
- оформлення замовлення;
- завершення роботи користувача із застосунком.

За результатами запуску всі реалізовані сценарії завершилися успішно та отримали статус Passed. Помилок під час виконання перевірок зафіксовано не було.

У процесі аналізу встановлено, що:

- вебелементи коректно знаходяться локаторами;
- переходи між сторінками виконуються стабільно;
- кнопки та форми коректно реагують на дії користувача;
- результати assertions відповідають очікуваним;
- тестові сценарії повторно виконуються без зміни результату.

Повторний запуск тестів проводився декілька разів. У кожному випадку сценарії демонстрували однаковий результат проходження без випадкових помилок або нестабільної поведінки.

Окремо було проаналізовано час виконання перевірок. Середній час виконання одного сценарію залишався стабільним, а загальна тривалість повного тестового запуску була значно меншою порівняно з аналогічною ручною перевіркою.

За результатами аналізу встановлено, що розроблене програмне забезпечення тестування забезпечує:

- стабільне виконання тестового набору;
- коректну перевірку функціональності вебзастосунку;
- повторюваність результатів запуску;
- швидке отримання результатів перевірки;
- зручний аналіз статусів проходження сценаріїв.

Отримані результати підтверджують працездатність реалізованих тестових сценаріїв та можливість використання розробленого програмного забезпечення під час перевірки вебінтерфейсів у практичних задачах тестування.

Узагальнені результати виконання тестового запуску наведено у таблиці 4.1, а приклад сформованого звіту за результатами проходження тестів — на рисунку 4.1.

Таблиця 4.1 – Результати виконання тестових сценаріїв

Назва тестового сценарію	Очікуваний результат	Отриманий результат	Статус	Час, с
Авторизація користувача	Успішний вхід у систему	Відкрито сторінку каталогу товарів	Passed	3,2
Відкриття каталогу товарів	Відображення списку товарів	Список товарів відображено	Passed	2,4
Додавання товару до кошика	Товар додано до кошика	Кошик оновлено	Passed	3,8
Видалення товару з кошика	Товар видалено	Кошик оновлено	Passed	2,9
Оформлення замовлення	Замовлення успішно завершено	Відображено повідомлення про завершення	Passed	4,6
Завершення користувацької сесії	Виконано вихід із системи	Відкрито сторінку авторизації	Passed	2,7

Аналіз даних, наведених у таблиці 4.1, показує, що всі тестові сценарії були виконані успішно та завершилися зі статусом Passed. Середній час виконання одного тестового сценарію становив близько 3–4 секунд, що підтверджує стабільність роботи розробленого програмного забезпечення тестування та можливість його використання для регулярної перевірки функціональності вебзастосунку.

4.3 Порівняння часу ручного та програмного тестування вебзастосунку

Одним із важливих критеріїв оцінювання результатів розробленого програмного забезпечення тестування є порівняння часу виконання перевірок під час ручного тестування та при запуску тестових сценаріїв у браузері.

Для проведення порівняльного аналізу було виконано однаковий набір перевірок двома способами:

- вручну користувачем у браузері;
- за допомогою розробленого програмного забезпечення тестування.

До порівняння було включено такі функціональні сценарії:

- авторизація користувача;
- відкриття каталогу товарів;
- додавання товару до кошика;
- видалення товару;
- оформлення замовлення;
- завершення користувацької сесії.

Під час ручної перевірки користувач самостійно відкривав сторінки вебзастосунку, взаємодіяв з елементами інтерфейсу та перевіряв очікуваний результат виконання кожної дії. Під час програмної перевірки аналогічна послідовність дій виконувалася через Selenium WebDriver після запуску тестового набору. Для кожного сценарію фіксувався час виконання від початку перевірки до отримання результату.

Отримані результати наведено в таблиці 4.2.

Таблиця 4.2 – Порівняння часу ручного та програмного тестування

Назва тестового сценарію	Час ручної перевірки, с	Час виконання тесту, с
Авторизація користувача	18	3,2
Відкриття каталогу товарів	12	2,4
Додавання товару до кошика	20	3,8
Видалення товару з кошика	16	2,9
Оформлення замовлення	35	4,6
Завершення користувацької сесії	10	2,7
Загальний час	111	19,6

З отриманих результатів видно, що загальний час ручного виконання перевірок становив 111 секунд, тоді як запуск того самого набору сценаріїв за допомогою програмного забезпечення тестування зайняв 19,6 секунди.

Таким чином використання розробленого програмного забезпечення дозволило суттєво скоротити тривалість перевірки функціональності вебзастосунку.

Під час аналізу було встановлено, що:

- тестові сценарії виконуються швидше за ручну перевірку;
- повторний запуск перевірок не потребує додаткового втручання користувача;
- результати перевірки формуються автоматично;
- зменшується кількість рутинних повторюваних дій;
- скорочується загальний час регресійної перевірки.

Особливо помітною різниця є у випадку повторного запуску повного тестового набору після внесення змін у вебзастосунок. У ручному режимі така перевірка потребує значних витрат часу, тоді як запуск тестового набору дозволяє отримати результат у короткий проміжок часу.

Отримані результати підтверджують, що використання розробленого програмного забезпечення тестування дозволяє прискорити процес перевірки функціональності вебзастосунку та знизити трудомісткість повторюваного тестування.

4.4 Практичне застосування та оцінка результатів розробленого програмного забезпечення

Після проведення тестових запусків та аналізу отриманих результатів було виконано оцінювання практичного застосування розробленого програмного забезпечення тестування вебзастосунку.

Основною метою створення програмного забезпечення було спрощення перевірки функціональності вебінтерфейсу, скорочення часу

виконання повторюваних перевірок та підвищення стабільності тестового процесу під час роботи з вебзастосунком.

Проведене дослідження показало, що створене програмне забезпечення може використовуватись як практичний інструмент перевірки вебінтерфейсів під час розроблення та супроводу вебзастосунків.

До основних результатів практичного застосування розробленого рішення можна віднести:

- скорочення часу перевірки функціональності вебзастосунку;
- зменшення кількості повторюваних ручних дій;
- швидке повторне виконання тестових сценаріїв після внесення змін у систему;
- можливість багаторазового запуску перевірок без зміни тестового коду;
- підвищення стабільності перевірки основного функціоналу вебзастосунку;
- спрощення контролю якості під час оновлення вебінтерфейсу.

Розроблене програмне забезпечення дозволяє перевіряти вебзастосунок у режимі, наближеному до реальної взаємодії користувача з браузером. Усі сценарії виконуються послідовно в браузері та повторюють типові дії користувача під час роботи із системою.

Важливою перевагою є можливість повторного запуску тестів після оновлення функціональності вебзастосунку без потреби повторно виконувати ручну перевірку всіх сценаріїв.

Окремо слід відзначити, що реалізований підхід дозволяє швидко адаптувати програмне забезпечення до змін інтерфейсу. У випадку оновлення структури сторінки зміни вносяться лише до відповідного Page Object класу без необхідності переписування всього тестового набору.

Розроблене програмне забезпечення тестування може бути використане:

– у процесі локального тестування вебзастосунку під час розроблення;

– для повторної перевірки функціоналу після внесення змін;

– під час регресійного тестування;

– у процесах контролю якості програмного забезпечення;

– як частина процесу тестування у вебпроєктах малого та середнього масштабу.

Практичне значення отриманих результатів полягає також у можливості подальшого розширення створеного рішення. До системи можуть бути додані:

– перевірки інших сторінок вебзастосунку;

– тестування нових користувацьких сценаріїв;

– запуск у декількох браузерах;

– формування розширених HTML-звітів;

– інтеграція у CI/CD-процеси.

Таким чином результати проведеного дослідження підтвердили практичну придатність розробленого програмного забезпечення тестування вебзастосунку та можливість його використання для перевірки функціональності вебінтерфейсів у задачах забезпечення якості програмного забезпечення.

4.5 Висновки до розділу 4

У четвертому розділі було проведено експериментальне дослідження результатів роботи розробленого програмного забезпечення тестування вебзастосунку та виконано оцінювання його практичного використання.

У межах дослідження було організовано запуск тестових сценаріїв перевірки основної функціональності вебзастосунку, зокрема авторизації користувача, роботи з каталогом товарів, додавання товару до кошика, оформлення замовлення та завершення користувацької сесії.

За результатами виконання тестового набору встановлено, що всі реалізовані сценарії були виконані успішно та завершилися без помилок. Проведений аналіз підтвердив стабільність роботи тестових сценаріїв, коректність проходження перевірок та повторюваність отриманих результатів під час повторного запуску.

Виконано порівняння часу ручної перевірки функціональності вебзастосунку та часу запуску розробленого програмного забезпечення тестування. Отримані результати показали скорочення тривалості виконання перевірок, що підтверджує доцільність використання програмного підходу під час тестування вебінтерфейсів.

Також було проведено оцінювання практичного застосування розробленого рішення. Встановлено, що створене програмне забезпечення може використовуватись для перевірки функціональності вебзастосунків під час розроблення, регресійного тестування та контролю якості вебінтерфейсу після внесення змін у систему.

Отримані результати підтвердили працездатність розробленого програмного забезпечення тестування вебзастосунку, його стабільність у роботі та можливість практичного використання для перевірки вебінтерфейсів у процесах забезпечення якості програмного забезпечення.

ВИСНОВКИ

У дипломній роботі вирішено завдання розроблення програмного забезпечення тестування вебзастосунку, призначеного для перевірки функціональності вебінтерфейсу шляхом моделювання дій користувача у браузері.

У процесі виконання роботи було проведено аналіз сучасних підходів до тестування вебзастосунків, досліджено особливості функціонального та UI-тестування, а також розглянуто сучасні програмні засоби перевірки вебінтерфейсів. За результатами аналізу обґрунтовано використання Selenium WebDriver, Java та TestNG для реалізації програмного забезпечення тестування.

У роботі спроєктовано архітектуру тестового проєкту, сформовано структуру програмного забезпечення та реалізовано взаємодію з вебелементами сторінки із застосуванням підходу Page Object Model. Такий підхід дозволив відокремити логіку роботи зі сторінками вебзастосунку від тестових сценаріїв, спростити структуру проєкту та покращити підтримуваність програмного коду.

У практичній частині роботи реалізовано набір тестових сценаріїв перевірки основної функціональності вебзастосунку. Розроблені сценарії охоплюють перевірку авторизації користувача, взаємодії з каталогом товарів, роботи з кошиком, оформлення замовлення та завершення користувацької сесії. Для аналізу результатів виконання тестів реалізовано механізм формування звітності про проходження перевірок.

У межах експериментального дослідження виконано запуск тестового набору та проаналізовано результати проходження сценаріїв. Усі реалізовані перевірки були виконані успішно, що підтвердило працездатність створеного програмного забезпечення.

Також проведено порівняння часу ручного та програмного тестування вебзастосунку. За результатами дослідження встановлено, що використання

розробленого програмного забезпечення дозволяє скоротити час перевірки функціональності вебінтерфейсу, зменшити кількість повторюваних ручних дій та забезпечити стабільне повторне виконання тестових сценаріїв.

Практичне значення одержаних результатів полягає у можливості використання створеного програмного забезпечення для перевірки вебзастосунків у процесах забезпечення якості програмного забезпечення, під час розроблення вебсистем, регресійного тестування та перевірки основного користувацького функціоналу після внесення змін у програмний продукт.

Отримані результати підтверджують досягнення поставленої мети роботи — підвищення швидкості та стабільності перевірки функціональності вебзастосунків шляхом розроблення програмного забезпечення тестування вебінтерфейсу.

Перспективою подальшого розвитку роботи є розширення тестового покриття новими сценаріями перевірки, реалізація запуску тестів у різних браузерах, удосконалення механізму звітності та інтеграція програмного забезпечення у процеси безперервної перевірки якості вебзастосунків.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. The Selenium Browser Automation Project. URL: https://www.selenium.dev/documentation/ (date of access: 27.05.2026).

2. WebDriver. URL: https://www.selenium.dev/documentation/webdriver/ (date of access: 27.05.2026).

3. Best 10 Automated Software Testing Tools in 2026. URL: https://www.browserstack.com/guide/best-automated-software-testing-tools (date of access: 27.05.2026).

4. Page Object Pattern. URL: https://martinfowler.com/bliki/PageObject.html (date of access: 27.05.2026).

5. TestNG Documentation. URL: https://testng.org/ (date of access: 27.05.2026).

6. Apache Maven Project. URL: https://maven.apache.org/ (date of access: 27.05.2026).

7. JetBrains IntelliJ IDEA Documentation. URL: https://www.jetbrains.com/idea/documentation/ (date of access: 27.05.2026).

8. ChromeDriver Documentation. URL: https://developer.chrome.com/docs/chromedriver/ (date of access: 27.05.2026).

9. W3C DOM Standard. URL: https://dom.spec.whatwg.org/ (date of access: 27.05.2026).

10. Document Object Model (DOM). URL: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model (date of access: 27.05.2026).

11. What is ChromeDriver?. URL: https://developer.chrome.com/docs/chromedriver (date of access: 27.05.2026).

ДОДАТОК А

Текст програми

А.1 Текст файлу pom.xml

```
<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
         https://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.webtesting</groupId>
    <artifactId>web-testing-project</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>

        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>4.18.1</version>
        </dependency>

        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>7.9.0</version>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>io.github.bonigarcia</groupId>
            <artifactId>webdrivermanager</artifactId>
            <version>5.7.0</version>
        </dependency>

    </dependencies>

    <build>
        <plugins>

            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>3.2.5</version>
                <configuration>
                    <suiteXmlFiles>

<suiteXmlFile>src/test/resources/testng.xml</suiteXmlFile>
                    </suiteXmlFiles>
                </configuration>
            </plugin>

        </plugins>
    </build>

</project>
```

А.2 Текст файлу BaseTest.java

```
package base;

import io.github.bonigarcia.wdm.WebDriverManager;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;

import java.time.Duration;

public class BaseTest {

    protected WebDriver driver;

    @BeforeMethod
    public void setUp() {
        WebDriverManager.chromedriver().setup();

        ChromeOptions options = new ChromeOptions();
        options.addArguments("--start-maximized");

        driver = new ChromeDriver(options);
        driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
        driver.get("https://www.saucedemo.com/");
    }

    @AfterMethod
    public void tearDown() {
        if (driver != null) {
            driver.quit();
        }
    }
}
```

А.3 Текст файлу LoginPage.java

```
package pages;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;

public class LoginPage {

    private WebDriver driver;

    private By usernameInput = By.id("user-name");
    private By passwordInput = By.id("password");
    private By loginButton = By.id("login-button");
    private By errorMessage = By.cssSelector("[data-test='error']");

    public LoginPage(WebDriver driver) {
        this.driver = driver;
    }

    public void enterUsername(String username) {
```

```
        driver.findElement(usernameInput).clear();
        driver.findElement(usernameInput).sendKeys(username);
    }

    public void enterPassword(String password) {
        driver.findElement(passwordInput).clear();
        driver.findElement(passwordInput).sendKeys(password);
    }

    public void clickLoginButton() {
        driver.findElement(loginButton).click();
    }

    public void login(String username, String password) {
        enterUsername(username);
        enterPassword(password);
        clickLoginButton();
    }

    public boolean isErrorMessageDisplayed() {
        return driver.findElement(errorMessage).isDisplayed();
    }

    public String getErrorMessageText() {
        return driver.findElement(errorMessage).getText();
    }
}
```

A.4 Текст файлу ProductsPage.java

```
package pages;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;

public class ProductsPage {

    private WebDriver driver;

    private By pageTitle = By.className("title");
    private By firstProductAddButton = By.id("add-to-cart-sauce-labs-
backpack");
    private By firstProductRemoveButton = By.id("remove-sauce-labs-
backpack");
    private By cartButton = By.className("shopping_cart_link");
    private By cartBadge = By.className("shopping_cart_badge");
    private By menuButton = By.id("react-burger-menu-btn");
    private By logoutButton = By.id("logout_sidebar_link");

    public ProductsPage(WebDriver driver) {
        this.driver = driver;
    }

    public boolean isProductsPageOpened() {
        return driver.findElement(pageTitle).getText().equals("Products");
    }

    public void addProductToCart() {
        driver.findElement(firstProductAddButton).click();
    }
```

```
    public void removeProductFromCart() {
        driver.findElement(firstProductRemoveButton).click();
    }

    public void openCart() {
        driver.findElement(cartButton).click();
    }

    public boolean isCartBadgeDisplayed() {
        return driver.findElement(cartBadge).isDisplayed();
    }

    public String getCartBadgeText() {
        return driver.findElement(cartBadge).getText();
    }

    public void logout() {
        driver.findElement(menuButton).click();
        driver.findElement(logoutButton).click();
    }
}
```

A.5 Текст файлу CartPage.java

```
package pages;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;

public class CartPage {

    private WebDriver driver;

    private By cartTitle = By.className("title");
    private By cartItem = By.className("cart_item");
    private By removeButton = By.id("remove-sauce-labs-backpack");
    private By checkoutButton = By.id("checkout");

    public CartPage(WebDriver driver) {
        this.driver = driver;
    }

    public boolean isCartPageOpened() {
        return driver.findElement(cartTitle).getText().equals("Your Cart");
    }

    public boolean isProductDisplayedInCart() {
        return driver.findElement(cartItem).isDisplayed();
    }

    public void removeProduct() {
        driver.findElement(removeButton).click();
    }

    public void clickCheckout() {
        driver.findElement(checkoutButton).click();
    }
}
```

А.6 Текст файлу CheckoutPage.java

```
package pages;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;

public class CheckoutPage {

    private WebDriver driver;

    private By firstNameInput = By.id("first-name");
    private By lastNameInput = By.id("last-name");
    private By postalCodeInput = By.id("postal-code");
    private By continueButton = By.id("continue");
    private By finishButton = By.id("finish");
    private By completeHeader = By.className("complete-header");

    public CheckoutPage(WebDriver driver) {
        this.driver = driver;
    }

    public void fillCustomerInformation(String firstName, String lastName,
String postalCode) {
        driver.findElement(firstNameInput).sendKeys(firstName);
        driver.findElement(lastNameInput).sendKeys(lastName);
        driver.findElement(postalCodeInput).sendKeys(postalCode);
    }

    public void clickContinue() {
        driver.findElement(continueButton).click();
    }

    public void clickFinish() {
        driver.findElement(finishButton).click();
    }

    public String getCompleteMessage() {
        return driver.findElement(completeHeader).getText();
    }
}
```

А.7 Текст файлу LoginTest.java

```
package tests;

import base.BaseTest;
import org.testng.Assert;
import org.testng.annotations.Test;
import pages.LoginPage;
import pages.ProductsPage;

public class LoginTest extends BaseTest {

    @Test
    public void testLoginSuccess() {
        LoginPage loginPage = new LoginPage(driver);
        ProductsPage productsPage = new ProductsPage(driver);
```

```
        loginPage.login("standard_user", "secret_sauce");

        Assert.assertTrue(productsPage.isProductsPageOpened());
    }

    @Test
    public void testLoginWithInvalidData() {
        LoginPage loginPage = new LoginPage(driver);

        loginPage.login("invalid_user", "invalid_password");

        Assert.assertTrue(loginPage.isErrorMessageDisplayed());
    }
}
```

A.8 Текст файлу ProductsTest.java

```
package tests;

import base.BaseTest;
import org.testng.Assert;
import org.testng.annotations.Test;
import pages.LoginPage;
import pages.ProductsPage;

public class ProductsTest extends BaseTest {

    @Test
    public void testOpenProductsPage() {
        LoginPage loginPage = new LoginPage(driver);
        ProductsPage productsPage = new ProductsPage(driver);

        loginPage.login("standard_user", "secret_sauce");

        Assert.assertTrue(productsPage.isProductsPageOpened());
    }

    @Test
    public void testAddProductToCart() {
        LoginPage loginPage = new LoginPage(driver);
        ProductsPage productsPage = new ProductsPage(driver);

        loginPage.login("standard_user", "secret_sauce");
        productsPage.addProductToCart();

        Assert.assertTrue(productsPage.isCartBadgeDisplayed());
        Assert.assertEquals(productsPage.getCartBadgeText(), "1");
    }
}
```

A.9 Текст файлу CartTest.java

```
package tests;

import base.BaseTest;
import org.testng.Assert;
```

```
import org.testng.annotations.Test;
import pages.CartPage;
import pages.LoginPage;
import pages.ProductsPage;

public class CartTest extends BaseTest {

    @Test
    public void testRemoveProductFromCart() {
        LoginPage loginPage = new LoginPage(driver);
        ProductsPage productsPage = new ProductsPage(driver);
        CartPage cartPage = new CartPage(driver);

        loginPage.login("standard_user", "secret_sauce");
        productsPage.addProductToCart();
        productsPage.openCart();

        Assert.assertTrue(cartPage.isCartPageOpened());
        Assert.assertTrue(cartPage.isProductDisplayedInCart());

        cartPage.removeProduct();
    }
}
```

A.10 Текст файлу CheckoutTest.java

```
package tests;

import base.BaseTest;
import org.testng.Assert;
import org.testng.annotations.Test;
import pages.CartPage;
import pages.CheckoutPage;
import pages.LoginPage;
import pages.ProductsPage;

public class CheckoutTest extends BaseTest {

    @Test
    public void testCheckout() {
        LoginPage loginPage = new LoginPage(driver);
        ProductsPage productsPage = new ProductsPage(driver);
        CartPage cartPage = new CartPage(driver);
        CheckoutPage checkoutPage = new CheckoutPage(driver);

        loginPage.login("standard_user", "secret_sauce");
        productsPage.addProductToCart();
        productsPage.openCart();
        cartPage.clickCheckout();

        checkoutPage.fillCustomerInformation("Ivan", "Ivanenko", "69000");
        checkoutPage.clickContinue();
        checkoutPage.clickFinish();

        Assert.assertEquals(checkoutPage.getCompleteMessage(), "Thank you for
your order!");
    }
}
```

A.11 Текст файлу LogoutTest.java

```
package tests;

import base.BaseTest;
import org.testng.Assert;
import org.testng.annotations.Test;
import pages.LoginPage;
import pages.ProductsPage;

public class LogoutTest extends BaseTest {

    @Test
    public void testLogout() {
        LoginPage loginPage = new LoginPage(driver);
        ProductsPage productsPage = new ProductsPage(driver);

        loginPage.login("standard_user", "secret_sauce");
        productsPage.logout();

        Assert.assertTrue(driver.getCurrentUrl().contains("saucedemo.com"));
    }
}
```

A.12 Текст файлу testng.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">

<suite name="WebTestPlan" verbose="1">

    <test name="Web Application Testing">
        <classes>
            <class name="tests.LoginTest"/>
            <class name="tests.ProductsTest"/>
            <class name="tests.CartTest"/>
            <class name="tests.CheckoutTest"/>
            <class name="tests.LogoutTest"/>
        </classes>
    </test>

</suite>
```

ДОДАТОК Б

Слайди презентації

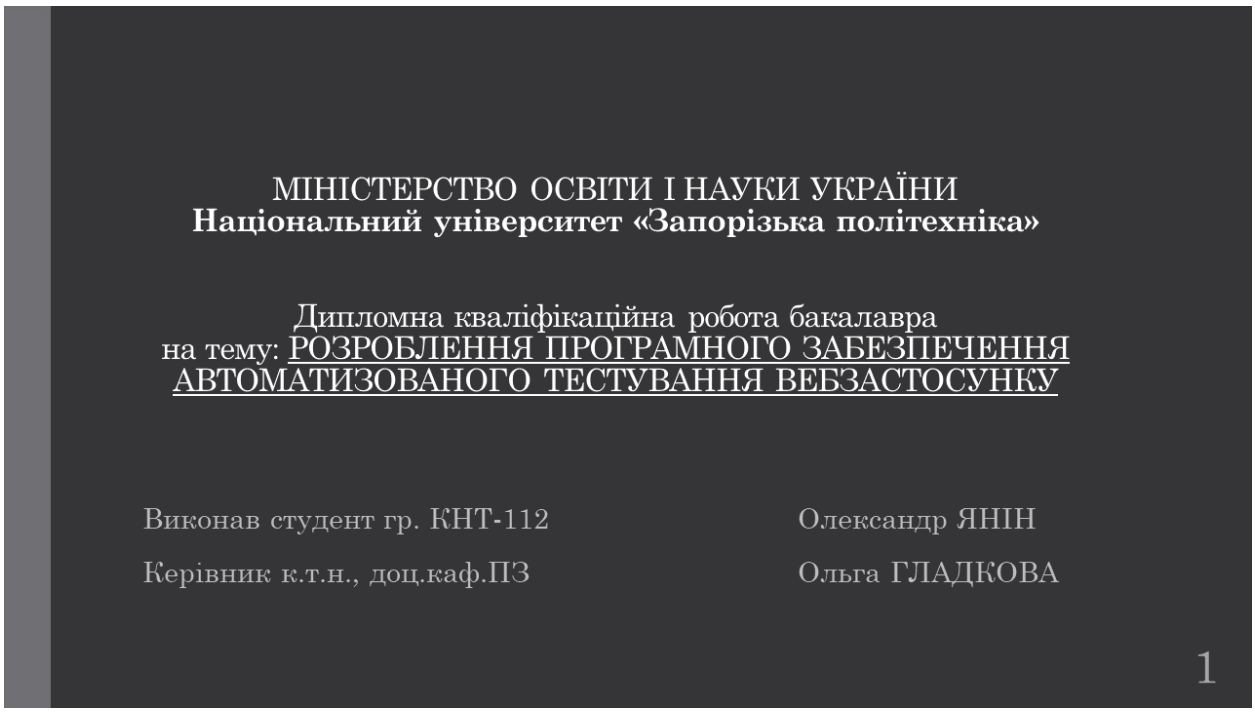


Рисунок Б.1 – Слайд 1

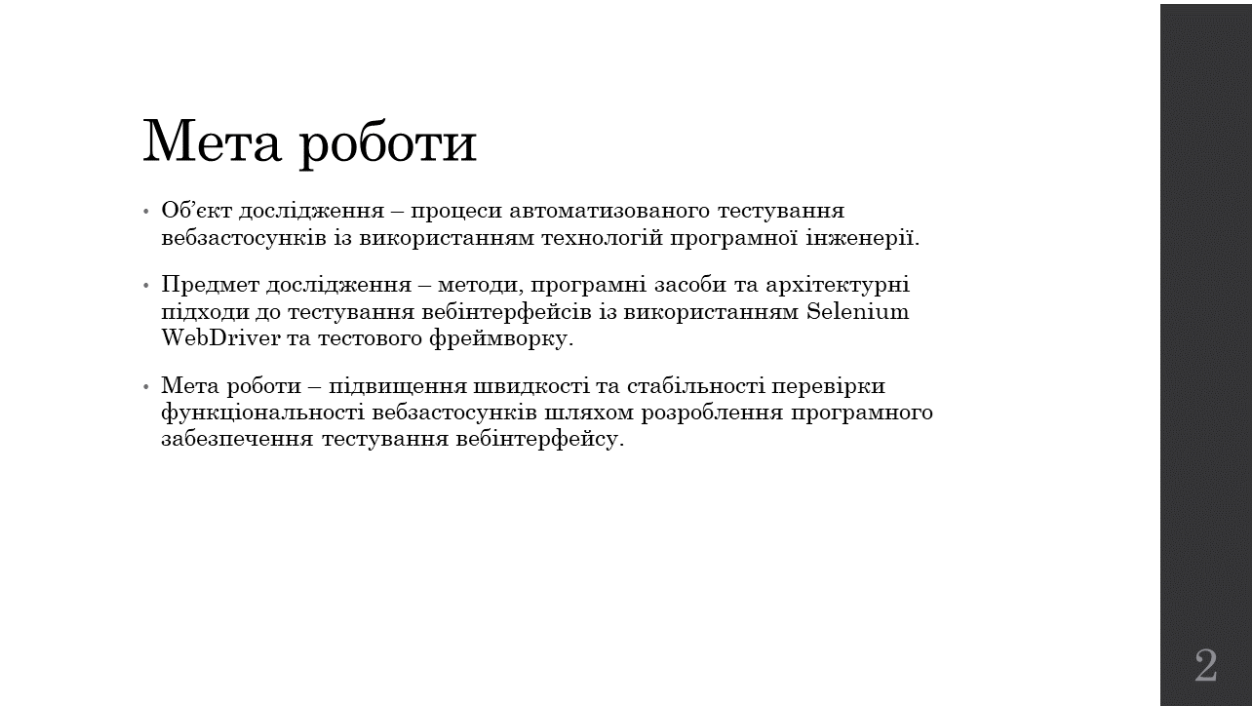


Рисунок Б.2 – Слайд 2

Вибір інструментарію розробки

Інструмент	Мова	Підтримка браузерів	Особливості
Selenium	Java, Python, C#	Так	Висока гнучкість
Cypress	JavaScript	Частково	Простота використання
Playwright	JavaScript, Java	Так	Висока швидкість
TestNG	Java	Ні	Організація тестів

3

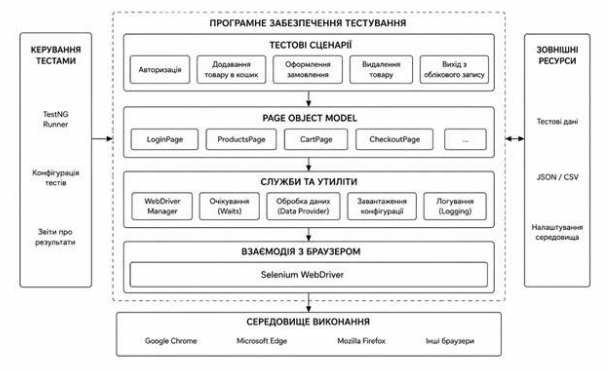
Рисунок Б.3 – Слайд 3

Вибір методів пошуку елементів

4

Рисунок Б.4 – Слайд 4

Архітектура системи



5

Рисунок Б.5 – Слайд 5

Структура програмного забезпечення

6

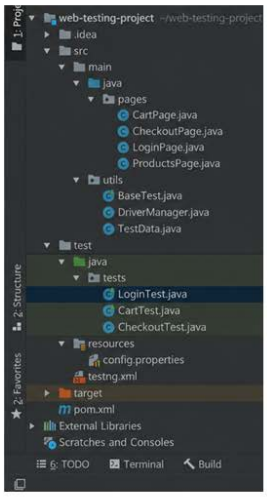
Рисунок Б.6 – Слайд 6

Алгоритм виконання тестових сценаріїв

7

Рисунок Б.7 – Слайд 7

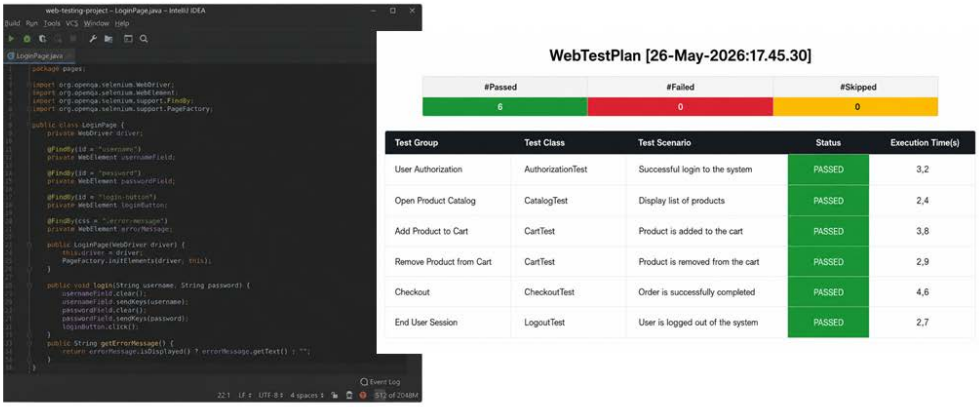
Структура проєкту в IntelliJ IDEA



8

Рисунок Б.8 – Слайд 8

Результати тестування



WebTestPlan [26-May-2026:17.45.30]

#Passed	#Failed	#Skipped
6	0	0

Test Group	Test Class	Test Scenario	Status	Execution Time(s)
User Authorization	AuthorizationTest	Successful login to the system	PASSED	3,2
Open Product Catalog	CatalogTest	Display list of products	PASSED	2,4
Add Product to Cart	CartTest	Product is added to the cart	PASSED	3,8
Remove Product from Cart	CartTest	Product is removed from the cart	PASSED	2,9
Checkout	CheckoutTest	Order is successfully completed	PASSED	4,6
End User Session	LogoutTest	User is logged out of the system	PASSED	2,7

9

Рисунок Б.9 – Слайд 9

Аналіз результатів виконання тестових сценаріїв

Назва тестового сценарію	Час ручної перевірки, с	Час виконання тесту, с
Авторизація користувача	18	3,2
Відкриття каталогу товарів	12	2,4
Додавання товару до кошика	20	3,8
Видалення товару з кошика	16	2,9
Оформлення замовлення	35	4,6
Завершення користувацької сесії	10	2,7
Загальний час	111	19,6

10

Рисунок Б.10 – Слайд 10

Висновки

Розроблено програмне забезпечення тестування вебзастосунку, яке дозволяє скоротити час виконання регресійного тестування, зменшити вплив людського фактора та підвищити стабільність повторного запуску тестових сценаріїв.

Отримані результати показали, що використання розробленого програмного забезпечення дозволяє скоротити час перевірки функціональності вебзастосунку порівняно з ручним виконанням аналогічних перевірок, а також забезпечує стабільність повторного запуску тестів і зменшує вплив людського фактора.

Рисунок Б.11 – Слайд 11