Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій
(повна назва)
Кафедра Інформаційно-вимірювальних технологій
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів автоматизованого тестування вебзастосунків
(тема)

Виконав:
здобувач 2 року навчання,
групи ЗЯм-24-1
Дмитро БІЛОУС
(власне ім'я, прізвище)
Спеціальність 175 – Інформаційно-вимірювальні технології
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Забезпечення якості
(повна назва освітньої програми)
Керівник доц. Олександр ДЕГТЯРЬОВ
(посада, власне ім'я, прізвище)

Допускається до захисту
Завідувач кафедри _____ Ігор ЗАХАРОВ
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій

Кафедра Інформаційно-вимірювальних технологій

Рівень вищої освіти другий (магістерський)

Спеціальність 175 – Інформаційно-вимірювальні технології
(код і повна назва)

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

Освітня програма Забепечення якості
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Білоусу Дмитру Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів автоматизованого тестування вебзастосунків затверджена наказом по університету від «07» листопада 2025 р. № 1011Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії «15» грудня 2025 р.
3. Вихідні дані до роботи Розробити та реалізувати систему автоматизованих тестів для вебзастосунку (інтернет-магазину на платформі WordPress + WooCommerce), застосовуючи методологію BDD. Виконати функціональне тестування, UI-тестування (E2E), негативне та регресійне тестування ключових бізнес-процесів (авторизація, кошик, оформлення замовлення). Використовувати мову програмування Python, бібліотеки Selenium WebDriver, Behave, систему звітності Allure та локальний сервер XAMPP (Apache, MySQL).
4. Перелік питань, що потрібно опрацювати в роботі 4.1 Вступ 4.2 Аналіз предметної області: загальні поняття тестування та життєвого циклу ПЗ, класифікація методів тестування 4.3 Огляд інструментів автоматизованого тестування та підходів BDD 4.4 Аналіз об'єкту та середовища тестування: характеристика вебзастосунку на WordPress, архітектура середовища, огляд функціональних можливостей та UI-елементів 4.5 Обґрунтування вибору стеку технологій (Python, Selenium, Behave) 4.6 Автоматизоване тестування вебзастосунку: ініціація проєкту та налаштування оточення 4.7 Складання сценаріїв тестування на мові Gherkin 4.8 Реалізація патерну Page Object Model (POM) та створення сторінок 4.9 Реалізація кроків тестування (step definitions) 4.10 Формування звітності тестування за допомогою Allure 4.11 Аналіз результатів та ефективності

впровадження автоматизації 4.12 Висновки 4.13 Перелік джерел посилання.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)
Слайди презентації кваліфікаційної роботи

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

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / термін виконання етапів роботи	Примітка
1.	Отримання завдання на кваліфікаційну роботу	10.11.2025	Виконано
2.	Аналіз предметної області, літератури та існуючих підходів до автоматизованого тестування	11.11.2025 –15.11.2025	Виконано
3.	Вибір та обґрунтування стеку технологій та налаштування локального середовища	16.11.2025	Виконано
4.	Розробка архітектури тестового проєкту та створення Page Object Model (POM) класів	17.11.2025 – 20.11.2025	Виконано
5.	Написання сценаріїв тестування мовою Gherkin для основних бізнес-процесів	21.11.2025-25.11.2025	Виконано
6.	Реалізація кроків тестування та налагодження автотестів	26.11.2025 – 30.11.2025	Виконано
7.	Виконання тестів, налаштування генерації звітів Allure та аналіз результатів	01.12.2025	Виконано
8.	Оформлення пояснювальної записки згідно з вимогами нормоконтролю	02.12.2025 – 05.12.2025	Виконано
9.	Підготовка графічного матеріалу та презентації для захисту роботи	06.12.2025 –08.12.2025	Виконано
10.	Подання роботи керівнику та отримання відгуку	09.12.2025	Виконано
11.	Подання роботи на рецензування та до ЕК	12.12.2025	Виконано

Дата видачі завдання «10» листопада 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Олександр ДЕГТЯРЬОВ
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи магістра: 95 с., 27 рис., 15 табл., 17 джерел.

АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ, BEHAVIOR-DRIVEN DEVELOPMENT, SELENIUM, WORDPRESS, ІНТЕРНЕТ-МАГАЗИН, ЗАБЕЗПЕЧЕННЯ ЯКОСТІ, АНАЛІЗ.

Об’єктом досліджень кваліфікаційної роботи є процес автоматизованого тестування функціональності вебзастосунків.

Предметом досліджень кваліфікаційної роботи є методи та засоби автоматизованого тестування вебзастосунків.

Метою досліджень є дослідження та практичне застосування сучасних підходів до автоматизованого тестування вебзастосунків.

Область застосування – автоматизація процесів забезпечення якості для вебзастосунків.

Результати роботи – розроблена та реалізована система автоматизованих тестів на базі методології BDD, що охоплює ключові функції вебсайту, такі як авторизація користувачів, робота з кошиком та оформлення замовлень.

ABSTRACT

Explanatory note of the master's qualification work: 95 p., 27 fig., 15 tables, 17 sources.

AUTOMATED TESTING, BEHAVIOR-DRIVEN DEVELOPMENT, SELENIUM, WORDPRESS, ONLINE STORE, QUALITY ASSURANCE, ANALYSIS.

The object of the qualification work research is the process of automated testing of web application functionality.

The subject of the qualification work research is the methods and tools of automated testing of web applications.

The purpose of the research is the study and practical application of modern approaches to automated testing of web applications.

Scope – automation of quality assurance processes for web applications.

Results of the work – a system of automated tests based on the BDD methodology was developed and implemented, covering key website functions such as user authorization, working with the shopping cart, and placing orders.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ПЗ – Програмне Забезпечення

API – Програмний Інтерфейс Додатку (Application Programming Interface)

BDD – Розвиток Орієнтований на Поведінку (Behavior-Driven Development)

CI/CD – Безперервна інтеграція/Безперервна доставка (Continuous Integration/Continuous Delivery)

CMS – Система Управління Контентом (Content Management System)

CRM – Управління Взаємовідносинами з Клієнтами (Customer Relationship Management)

CSS – Каскадні Таблиці Стилів (Cascading Style Sheets)

DOM – Модель Об’єкта Документа (Document Object Model)

E2E – Наскрізне тестування (End-to-End)

HTMP – Мова Розмітки Гіпертексту (Hyper Text Markup Language)

ISTQB – Міжнародна Рада з Кваліфікації Тестування Програмного Забезпечення (International Software Testing Qualifications Board)

JSON – Нотація Об’єктів JavaScript (JavaScript Object Notation)

POM – Модель Об’єкта Сторінки (Page Object Model)

SEO – Пошукова Оптимізація (Search Engine Optimization)

TDD – Розробка через тестування (Test-Driven Development)

UI – Інтерфейс Користувача (User Interface)

XAMPP – X - Cross-platform, A - Apache, M - MySQL/MariaDB, P - PHP, P - Perl

XML – Розширювана Мова Розмітки (eXtensible Markup Language)

X-Path – Мова XML-Шляхів (XML Path Language)

ВСТУП

Вебзастосунки складають невід'ємну частину повсякденного життя. Онлайн-магазини, освітні платформи, інтернет-банкінг, сервіси бронювання тощо – вебзастосунки, що обслуговують та оброблюють запити мільйонів користувачів щодня. Масштабування та ускладнення системи потребує високих вимог до її надійності, стійкості, функціональності, безпеки та стабільної роботи. У процесі життєвого циклу програмного забезпечення (ПЗ) забезпечення якості вебзастосунку займає ключовий етап, а тестування – критично важлива складова цього процесу.

Традиційні методи ручного (мануального) тестування, попри свою гнучкість, мають значні обмеження. Вони є трудомісткими, потребують багато часу та схильні до людських помилок. Для зменшення людського впливу, спрощення процесу тестування та економії часу в галузі розробки програмного забезпечення активно впроваджуються методи автоматизованого тестування. Завдяки автоматизованому методу скорочується час тестування, підвищується точність і повторюваність результатів, а виявлення помилок на початкових етапах розробки стає вагомою перевагою у порівнянні з мануальним тестуванням.

Автоматизоване тестування вебзастосунків охоплює широкий спектр підходів, інструментів і технологій. Behavior-Driven Development (BDD) – один із підходів, який спрямований на покращення комунікації між розробником, тестувальником та замовником. Опис поведінки системи у BDD має вигляд сценаріїв, що значно спрощує сприйняття очікувань і результатів тестування. Інструменти Selenium WebDriver, Behave та Allure у зв'язці дають можливість реалізувати цей підхід на практиці, забезпечуючи гнучке та ефективне тестування інтерфейсів користувача.

У кваліфікаційній роботі досліджуються методи автоматизованого тестування на прикладі проєкту – інтернет-магазину, створеного на платформі WordPress та розгорнутого на локальному сервері за допомогою XAMPP. У

межах проєкту було розроблено систему автоматизованих тестів, що охоплюють основні функції сайту: авторизацію користувачів, роботу з кошиком, оформлення замовлень, перевірка замовлень користувача, редагування особистих даних користувача на сторінці профілю.

Мета кваліфікаційної роботи полягає у досліджені та практичному застосуванні сучасних підходів до автоматизованого тестування вебзастосунків.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальні поняття тестування та життєвого циклу програмного забезпечення

Тестування програмного забезпечення – це технічний процес перевірки ПЗ, спрямований на виявлення помилок і недоліків. Воно дозволяє впевнитися, що система функціонує правильно, відповідає встановленим вимогам і задовольняє очікування користувачів [1].

Основні цілі тестування ПЗ:

– виявлення дефектів та помилок;
– забезпечення якості та надійності;
– перевірка відповідності вимогам;
– зниження ризиків;
– забезпечення задоволення користувачів;
– підвищення ефективності розробки;
– оцінка якості ПЗ.

Життєвий цикл програмного забезпечення – це сукупність окремих етапів робіт, що проводяться у заданому порядку протягом періоду часу, який починається з вирішення питання про розроблення ПЗ і закінчується припиненням використання ПЗ.

Цикл ПЗ може бути представлений у каскадній, V-подібній, інкрементній, спіральній, гнучкій та скрам моделях.

Каскадна або водоспадна модель (на рис. 1.1) характеризується тим, що кожен з етапів проєкту проводиться лише один раз у послідовному порядку, щоб перейти на наступний етап, потрібно завершити роботу над попереднім. Переваги каскадної моделі характеризуються тим, що всі етапи проєкту виконуються у чітко визначеній послідовності, що забезпечує організованість процесу. Така структурованість дозволяє точно встановлювати часові рамки для

завершення робіт, а також необхідні ресурси. Вимоги до системи визначаються лише на початковому етапі і не можуть бути змінені протягом життєвого циклу, що спрощує контроль за його реалізацією. Недоліками такого підходу є труднощі з формулюванням чітких вимог на початковому етапі, а також відсутність можливості змінювати їх у процесі розробки. Тестування починається лише на пізніх стадіях, що може призвести до виявлення критичних помилок надто пізно. Користувачі не мають змоги заздалегідь оцінити якість продукту, адже доступ до результату вони отримують лише після завершення всієї розробки.

Рисунок 1.1 – Каскадна (водоспадна) модель життєвого циклу ПЗ

V-подібна модель (рис. 1.2) є послідовником каскадної, так як з її допомогою можна усунути недоліки, які були раніше. Для того, щоб перейти на наступний етап, потрібно мати контроль над усіма процесами, а процес тестування можна розпочинати ще на стадії складання вимог до ПЗ. Серед переваг цієї моделі варто виділити сувору етапізацію, яка забезпечує чітку організацію процесу розробки. Тестування інтегрується вже на ранніх стадіях, що дозволяє мінімізувати ризики та своєчасно усувати потенційні проблеми.

Завдяки структурованому підходу вдосконалюється тайм-менеджмент, що сприяє кращому контролю за виконанням завдань у межах встановлених термінів. Серед недоліків моделі – зміна вимог після початку проєкту майже неможлива. Тривалий цикл розробки, який може тривати навіть кілька років, призводить до того, що на момент завершення продукт уже може не відповідати актуальним потребам замовника. В V-подібній моделі не передбачені окремі дії, спрямовані на систематичний аналіз ризиків.

Рисунок 1.2 – V-подібна модель життєвого циклу ПЗ

Інкрементна модель (рис. 1.3) характеризується розробкою ПЗ у лінійній послідовності етапів, але в кілька інкрементів (версій). Вимоги визначаються на початку роботи, а розробка проводиться у вигляді послідовності версій, кожна з яких є кінцевим і працездатним продуктом. До переваг можна віднести те, що замовник має можливість надавати зворотний зв’язок після кожної нової версії продукту. Цей підхід дозволяє вчасно переглядати та оцінювати ризики, пов’язані з витратами і дотриманням графіка виконання робіт. Ще однією перевагою є те, що замовник поступово звикає до нової технології, адаптуючись до змін без

різкого переходу. Проте існують і певні недоліки. Щоб правильно розподілити ітерації, функціональні вимоги до системи повинні бути повністю визначені ще на початку життєвого циклу, що не завжди можливо. Часті зміни ведуть до порушення структури системи, ускладнюючи її підтримку. Також терміни здачі проєкту можуть бути збільшені через обмеженість доступних ресурсів.

Рисунок 1.3 – Інкрементна модель життєвого циклу ПЗ

Спіральна модель (рис. 1.4) життєвого циклу розпочинається на етапі планування та розкручується з кожним наступним етапом. При виході з чергового витка розробник отримує готовий протестований прототип, це продовжується до тих пір, поки продукт не буде задовольняти усі сплановані вимоги. До основних переваг підходу належить те, що значна увага приділяється управлінню ризиками, що допомагає вчасно виявляти та усувати загрози для проєкту. Модель також передбачає можливість гнучкого проєктування, що дає змогу адаптуватися до змін і додавати новий функціонал навіть на фінальних етапах розробки. До недоліків можна віднести процес оцінки ризиків на кожному етапі та постійна взаємодія з замовником, його відгуки та побажання можуть стимулювати появу

нових ітерацій, що, у свою чергу, може призводити до затягування термінів розробки. Цей підхід частіше застосовується для великих і складних проєктів, що обмежує його ефективність у невеликих розробках.

Рисунок 1.4 – Спіральна модель життєвого циклу ПЗ

Гнучка модель (рис. 1.5) – це сукупність різних підходів до розробки ПЗ. Цій моделі притаманні такі властивості: використання ітеративної розробки та динамічне формування вимог. Кожна ітерація являє собою міні проєкт, а одна з фундаментальних ідей гнучкої моделі – взаємодія всередині команди і з замовником напряму. Перевагами цього підходу є швидке прийняття рішень завдяки постійним комунікаціям між членами команди та із замовником. Також спрощується робота з документацією – замість створення об’ємних технічних описів основна увага зосереджується на практичній реалізації продукту. Серед недоліків варто зазначити, що велика кількість конференцій, обговорень і узгоджень може уповільнити сам процес розробки. Також складно заздалегідь спланувати всі етапи, оскільки вимоги до продукту постійно змінюються в процесі роботи [2].

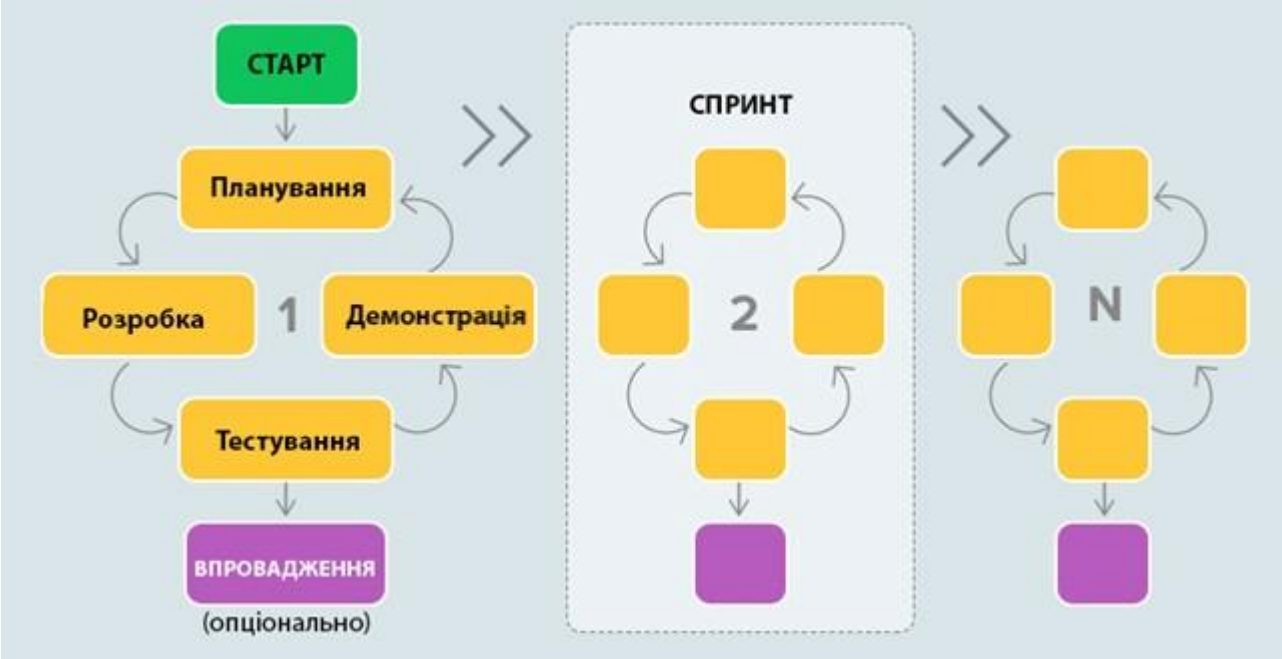

Рисунок 1.5 – Гнучка модель життєвого циклу ПЗ

Скрам модель (рис. 1.6) гнучка в плані розробки ПЗ, в ній акцент робиться на чіткому і якісному контролі розробки. Має рольовий підхід (Scrum Master, Product Owner, Team), де Scrum Master відповідає за успіх проєкту та є сполучним вузлом між менеджментом і командою. Product Owner несе відповідальність безпосередньо за процес розробки, може також ставити завдання і приймати остаточні рішення. Team – цілий «організм», результати роботи якого оцінюються не по кожному учаснику окремо, а по спільному показнику. Перевагами даного підходу є можливість отримання швидкого зворотного зв'язку від фахівців різних напрямів – дизайнерів, архітекторів, тестувальників та інших фахівців, які залучені до проєкту. Завдяки активній участі тестувальників у процесі розробки стає можливим оперативне додавання нового функціоналу, а також швидкий запуск продукту з базовими, але працездатними функціями. Команда, яка працює за цим підходом, зазвичай є самостійною. Серед недоліків варто відзначити, що через відсутність повноцінної документації деякі фахівці стають практично незамінними, оскільки лише вони володіють повним

обсягом знань про продукт. Також складно заздалегідь визначити точну дату завершення проєкту, оскільки планування відбувається на основі результатів попередніх ітерацій (спринтів) [3].

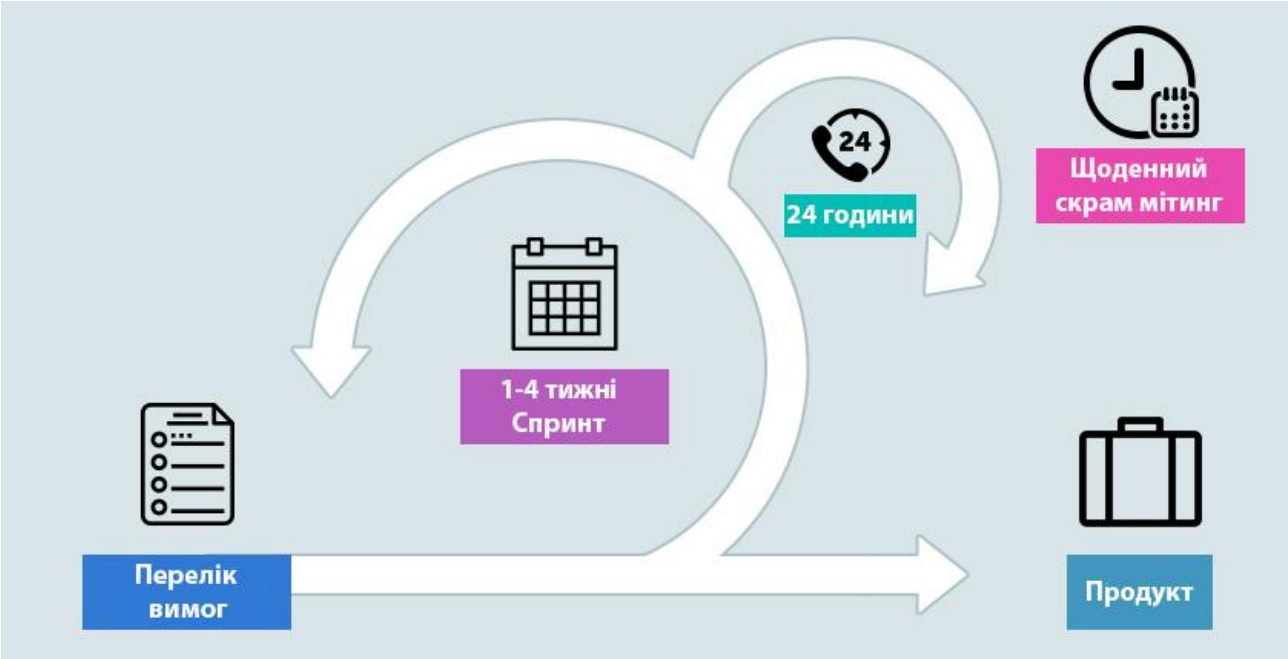

Рисунок 1.6 – Скрам модель життєвого циклу ПЗ

Підсумовуючи, робота тестувальника та підходів тестування у проєкті напряму залежить від заздалегідь обраної моделі життєвого циклу ПЗ.

1.2 Класифікація методів тестування

Серед методів тестування (або підходів) виділяють три основні: «Метод білої скриньки (White-box)», «Метод чорної скриньки (Black-box)» та «Метод сірої скриньки (Gray-box)» [4].

Тестування за методом «Білої скриньки» полягає в перевірці програмного забезпечення з урахуванням його внутрішньої будови. У цьому випадку тестувальник має повний доступ до коду та структури системи, що дозволяє створювати тест-кейси, виходячи з логіки, алгоритмів та потоків даних,

закладених у компонент або систему. Такий підхід орієнтований на виявлення внутрішніх дефектів, логічних помилок, неохоплених гілок коду та інших проблем, що неможливо виявити, не знаючи реалізації.

Тестування за методом «Чорної скриньки» передбачає перевірку як функціональних, так і нефункціональних характеристик програмного забезпечення без урахування його внутрішньої будови чи коду. У цьому підході система розглядається як непрозорий об’єкт, де тестувальників не знає, як вона влаштована, а фокусується лише на її зовнішній поведінці згідно з вимогами. Процес створення тестів ґрунтується на специфікаціях і очікуваних результатах, а не на програмній структурі.

Тестування за методом «Сірої скриньки» поєднує елементи як «Чорного», так і «Білого» підходів. Це означає, що тестувальник має обмежене розуміння внутрішньої структури ПЗ, наприклад, частковий доступ до архітектури, логіки роботи чи базових алгоритмів. На основі цих знань створюються більш релевантні сценарії перевірки, однак саме тестування здебільшого виконується з точки зору користувача, як у методі «Чорного скриньки». Такий гібридний підхід часто називають «Напівпрозорою скринькою», оскільки тестувальник володіє лише частиною інформації про систему. Він дозволяє досягти балансу між глибиною аналізу та зручністю реалізації тестів [5].

Згідно з ISTQB (International Software Testing Qualifications Board) вид тестування – це засіб чіткого визначення мети конкретного рівня для програми чи проєкту. Метою тестування може бути перевірка нефункціональних характеристик (надійність, зручність), структури або архітектури системи, а також перевірка змін – наприклад, виправлення дефекту (повторне тестування) чи виявлення випадкових збоїв (регресійне тестування). Існує чотири види тестування ПЗ:

- функціональне тестування;
- нефункціональне тестування;
- структурне тестування;
- тестування змін.

Функціональне тестування спрямоване на перевірку всіх функцій системи відповідно до документації. До елементів функціонального тестування належать: підготовка тестових даних відповідно до наданої документації, врахування бізнес-вимог як складової функціонального тестування, отримання результатів на основі специфікації, виконання тест-кейсів, а також аналіз фактичних та очікуваних результатів.

Нефункціональне тестування відрізняється від функціонального тим, що має інший підхід до складання тестів. Замість відповіді на запитання «Чи працює система?», вирішується «Як добре працює система?». Цей вид тестування специфікується на перевірці аспектів ПЗ, які можуть бути описані в документації, але не відносяться до функцій продукту. Підвиди нефункціонального тестування наведені в таблиці 1.1.

Таблиця 1.1 – Підвиди нефункціонального тестування

Назва підвиду	Опис підвиду
Тестування стабільності	Оцінка здатності додатку працювати без збоїв при тривалому використанні під стандартним навантаженням.
Юзабіліті тестування	Аналіз зручності взаємодії користувача з програмним забезпеченням.
Тестування ефективності	Перевірка оптимального використання ресурсів і обсягу коду при виконанні функцій.
Тестування ремонтопридатності	Оцінка простоти та швидкості обслуговування та підтримки працездатності системи.

Продовження таблиці 1.1

Назва підвиду	Опис підвиду
Перевірка портативності	Аналіз здатності ПЗ або його компонентів переноситись між різними середовищами.
Тестування «пра-витоків»	Оцінка якості технічної документації та вимог, які є базою для створення тестів.
Приймальне тестування	Контроль відповідності готового продукту критеріям приймання.
Тестування документації	Перевірка повноти та актуальності супровідної документації, створеної під час тестування.
Тестування витривалості системи	Оцінка поведінки системи при тривалому впливі високих навантажень.
Тестування навантаження	Аналіз реакції ПЗ на очікувану інтенсивність користування або операцій.
Тестування продуктивності	Вимірювання швидкості виконання операцій ПЗ або його модулів.
Тестування сумісності	Перевірка працездатності ПЗ у різних конфігураціях апаратного й програмного середовища.
Тестування безпеки	Оцінка рівня захищеності системи від зовнішніх та внутрішніх загроз.
Об’ємне тестування	Тестування роботи програмного забезпечення з великими обсягами даних у базі.

Кінець таблиці 1.1

Назва підвиду	Опис підвиду
Стрес тестування	Перевірка функціонування системи в екстремальних умовах, таких як нестача ресурсів.
Тестування швидкості відновлення	Оцінка часу, необхідного для повного відновлення ПЗ після збою або поломки.
Тестування локалізації	Контроль відповідності ПЗ культурним, мовним і регіональним стандартам користувачів.

Структурне тестування спрямоване на перевірку структури або компонентів системи. До методів структурного тестування відносяться:

– рядкове покриття;
– покриття шляху;
– покриття рішення;
– покриття умови.

Тестування змін поділяється на регресійне тестування та повторне тестування. Регресійне тестування виконується для впевненості в тому, що раніше реалізований функціонал залишається працездатним після внесення змін до системи, таких як оновлення версії або модифікація коду. Воно також дозволяє виявити нові помилки, які могли виникнути внаслідок цих змін. Повторне тестування застосовується для перевірки того, чи була помилка успішно усунута, та чи працює відповідний функціонал коректно після виправлення [6].

1.3 Автоматизоване тестування

Автоматизоване тестування – це частина процесу тестування на етапі контролю якості в процесі розробки ПЗ. Зазвичай використовує програмні засоби

(набір інструментів) для виконання тестів і перевірки результатів виконання. Значно скорочує час тестування та спрощує процес за рахунок заздалегідь прописаних кроків, може використовуватися багато разів у рамках одного або архітектурно однакових проєктів [7].

Автоматизоване тестування полягає у застосуванні спеціальних засобів для самостійного виконання набору тестових сценаріїв. На відміну від ручного тестування, де кожен крок перевірки виконує тестувальник, автоматизовані засоби можуть вводити тестові дані, перевіряти результати на відповідність очікуваним та формувати звіти без участі людини.

Незважаючи на ефективність, автоматизація потребує початкових фінансових і часових витрат. Під час життєвого циклу програмного забезпечення один і той самий набір тестів часто потрібно повторювати. Завдяки автоматизації створені одного разу тести можна запускати безліч разів, що зменшує потребу в ручному втручанні та підвищує ефективність. Важливо зазначити, що автоматизоване тестування не замінює повністю ручне, а лише зменшує його обсяг.

Реалізація автоматизованого тестування складається з двох кроків: пошук потрібного елемента за селектором та взаємодія з ним.

У вебдодатках елементи зазвичай ідентифікуються через DOM-структуру з використанням локаторів типу X-Path або CSS-селекторів. У випадку з десктопними та мобільними застосунками пошук елементів, як правило, базується на їхніх координатах на екрані. Після того як елемент знайдено, над ним виконується відповідна дія або проводиться перевірка.

DOM (Document Object Model) – програмний інтерфейс, який дозволяє програмам і скриптам, таким як JavaScript, отримувати доступ для вмісту, структури та стилю HTML, XML та інших документів, а також змінювати їх [8].

Селектор (або локатор) – шлях до елементу на сторінці [9].

1.4 Огляд інструментів автоматизованого тестування

Для реалізації практичної частини було використано стек з Python, Selenium WebDriver, Behave та Allure.

Selenium WebDriver – це програмний інструмент, який використовується для автоматизації тестування вебсайтів. Він працює як набір бібліотек, що підтримують різні мови програмування, як-от Java, C#, Python, Ruby, PHP та JavaScript, що дозволяє створювати автоматичні тести для перевірки функціональності вебзастосунків.

Selenium вміє працювати з браузерами Google Chrome, FireFox, Safari, Edge, Internet Explorer та навіть з браузерами без графічного інтерфейсу. Підтримує операційні системи Windows, Linux, MacOS.

За архітектурою (рис. 1.7) складається з чотирьох компонентів:

– драйвер контролю браузера;
– клієнтська бібліотека Selenium;
– браузер;
– протокол JSON Wire.

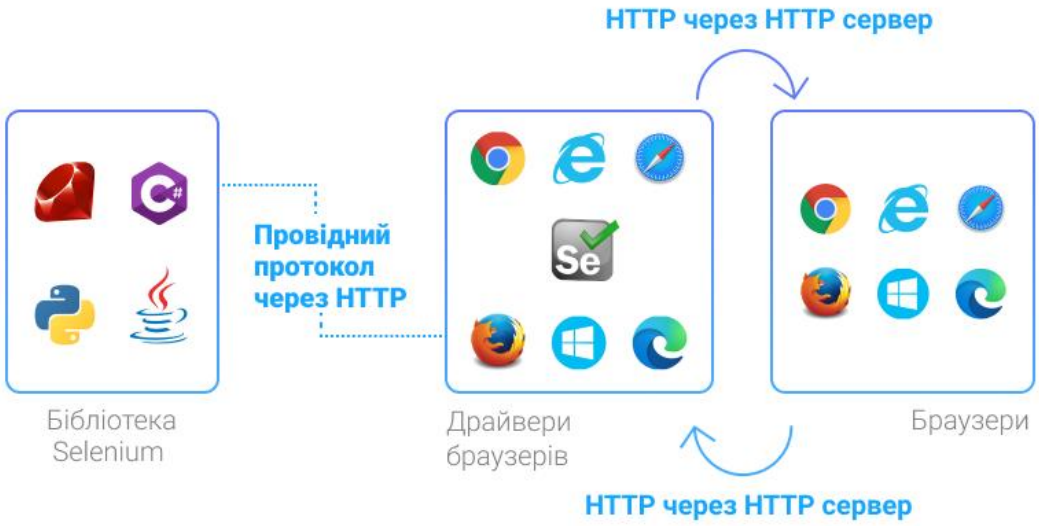

Рисунок 1.7 – Архітектура Selenium WebDriver

Behave – це Python фреймворк для Behavior-Driven Development (BDD), який дозволяє описувати автоматизовані тестові сценарії на мові Gherkin.

Приклад сценарію (Gherkin):

```
Feature: Авторизація користувача
    Scenario: Успішна авторизація
    Given користувач з ім’ям "example" та паролем "password"
    When користувач вводить коректні дані
    And натискає кнопку "Увійти"
    Then користувач повинен бути авторизований
```

В цьому прикладі «Given», «When», «Then» – це ключові слова Gherkin, які описують кроки тесту, а «Feature» та «Scenario» – використовуються для організації сценаріїв [11].

При мануальному тестуванні зазвичай використовують Jira для формування баг-репортів, а при автоматизованому – Allure.

Allure – це інструмент для формування звітів при виконанні тестів. Після виконання сценаріїв BDD формується звітність, яка містить: загальну статистику (кількість успішних, провалених, пропущених тестів, відсоток співвідношень, середній час виконання), структуру кейсів (розподіл за сценаріями, тестовими наборами), хронологію запусків, логи виконання (зберігає повідомлення з терміналу середовища розробки), скріншоти, діаграми. Звіти можна переглянути за допомогою вебзастосунку на локальному середовищі (рис. 1.8) [12].

Рисунок 1.8 – Приклад звіту Allure

З альтернативних інструментів у галузі автоматизованого тестування можна виділити наступні: Cypress, TestCafe, JUnit у зв'язці із Selenium, Cucumber.

Cypress – сучасний інструмент, орієнтований на фронтенд-тестування JavaScript. Тести виконуються безпосередньо в браузері, забезпечуючи швидке виконання та інтуїтивний інтерфейс. Серед переваг можна відзначити швидкість та простоту для початківців, але з недоліків обмежена підтримка браузерів і робота лише з JavaScript.

TestCafe – не потребує драйверів чи додаткових плагінів – працює одразу після встановлення. Підтримує всі популярні браузери. Переваги інструменту: простота та зручність, підтримка мобільних та десктопних платформ. Серед недоліків – менша екосистема плагінів у порівнянні з Selenium або Cypress.

JUnit застосовується для Java-проєктів, забезпечує структуру тестування.

Cucumber – BDD фреймворк, орієнтований на співпрацю між технічними та бізнес-учасниками процесу, має схожість з Behave, але підтримує інші мови (Java, JavaScript тощо).

1.5 Аналіз та порівняння підходів BDD та традиційного Unit/Functional тестування

Традиційне тестування функціональності ПЗ зазвичай поділяється на кілька рівнів, ключовими з яких є модульне (Unit) та функціональне (Functional) тестування.

Модульне тестування зосереджується на перевірці найменших ізольованих компонентів або функцій ПЗ, щоб переконатися, що кожен із них працює правильно. Це тестування виконується переважно розробниками і належить до методу «Білої скриньки», оскільки вимагає знання внутрішньої структури коду.

Функціональне тестування спрямоване на перевірку функцій системи відповідно до вимог та специфікації. Воно розглядає систему з точки зору користувача і належить до методу «Чорної скриньки».

Основні характеристики традиційного функціонального тестування:

– орієнтація на вимоги;

– технічна мова;

– проблеми з комунікацією.

Behavior-Driven Development (BDD) – це гнучка методологія розробки ПЗ, яка розширює концепцію TDD (Test-Driven Development) та спрямована на покращення комунікації між усіма учасниками проєкту: розробниками, тестувальниками та замовниками.

BDD дозволяє не лише автоматизувати перевірку функціоналу (як функціональне тестування), але й одночасно підвищити якість бізнес-документації та забезпечити чітке узгодження очікувань між замовником та командою, що є важливою складовою наукового дослідження в галузі забезпечення якості ПЗ.

Переваги BDD-підходу у контексті дослідження:

– покращення комунікації та прозорість;

– бізнес-орієнтація;

– ефективність для E2E-тестування;

– спрощення регресійного тестування.

У таблиці 1.2 представлено порівняння BDD-підходу із традиційними методами тестування за ключовими критеріями.

Таблиця 1.2 – Порівняння BDD та традиційних методів тестування

Підхід / Критерій	BDD (Behavior-Driven Development)	Традиційне функціональне тестування
Основна мета	Створення спільної, зрозумілої документації та визначення поведінки системи.	Перевірка відповідності функцій технічним вимогам (специфікації).
Основа сценаріїв	Сценарії користувача (Given-When-Then), орієнтовані на бізнес-логіку.	Тест-кейси на основі вимог.

Кінець таблиці 1.2

Підхід / Критерій	BDD (Behavior-Driven Development)	Традиційне функціональне тестування
Мова опису	Природна мова (українська/англійська), зрозуміла для всіх учасників.	Технічна, професійна мова.
Рівень абстракції	Високий – фокус на зовнішній поведінці користувача (E2E).	Середній/Низький – фокус на окремих функціях або API.
Головна перевага	Покращення комунікації та прозорості між бізнесом та розробкою.	Глибина покриття окремих функцій.

2 АНАЛІЗ ОБ’ЄКТУ, СЕРЕДОВИЩА ТА ЗАСОБІВ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ

2.1 Загальна характеристика об’єкту тестування

Вебзастосунок реалізовано на базі WordPress з використанням плагіну WoCommerce.

WordPress – це популярна безкоштовна система для створення вебсайтів, яка належить до класу CMS (Content Management System). Вона схожа на конструктор сайтів (рис. 2.1), але має значно ширші можливості для налаштування. WordPress підтримує тисячі плагінів – спеціальних розширень, які дають змогу додавати нові функції та робити сайт більш складним і гнучким у використанні. Завдяки цьому платформу використовують як для простих блогів, так і для повноцінних інтернет-магазинів або корпоративних ресурсів.

WordPress є універсальним інструментом, що підходить для створення найрізноманітніших вебресурсів. З його допомогою можна реалізувати інтернет-магазин, персональний блог, сайт-візитку компанії, інформаційний портал, портфоліо тощо [13, 14].

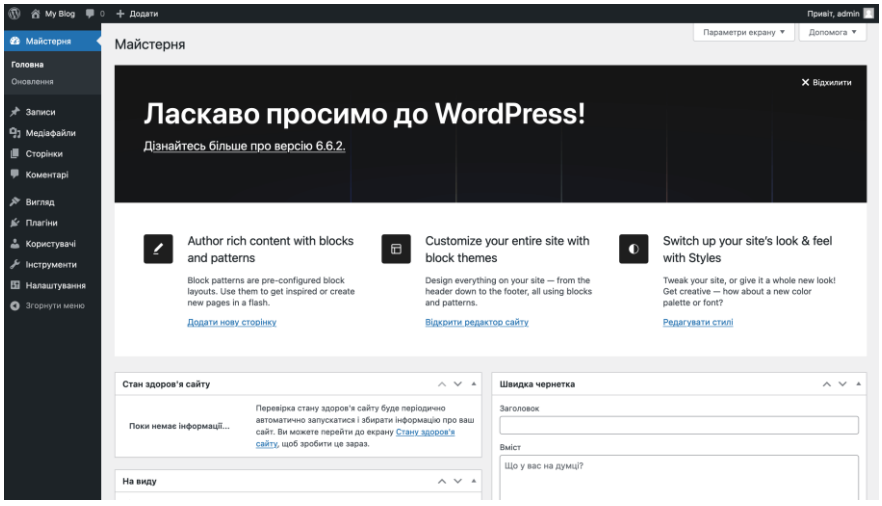

Рисунок 2.1 – Інтерфейс панелі WordPress

WooCommerce – це функціональний модуль електронної комерції для платформи WordPress, що дозволяє створювати інтернет-магазини різного рівня складності без необхідності володіння знаннями у сфері веброзробки (такими як PHP, HTML чи CSS). Плагін забезпечує повний набір інструментів для керування асортиментом продукції, обробки замовлень, організації приймання оплати та реалізації додаткових функцій, необхідних для ефективного ведення онлайн-бізнесу.

WordPress підтримує імпорт готових шаблонів сайтів, тому було прийнято рішення використовувати тему існуючого інтернет-магазину «Botiga» як об’єкт дослідження (рис. 2.2).

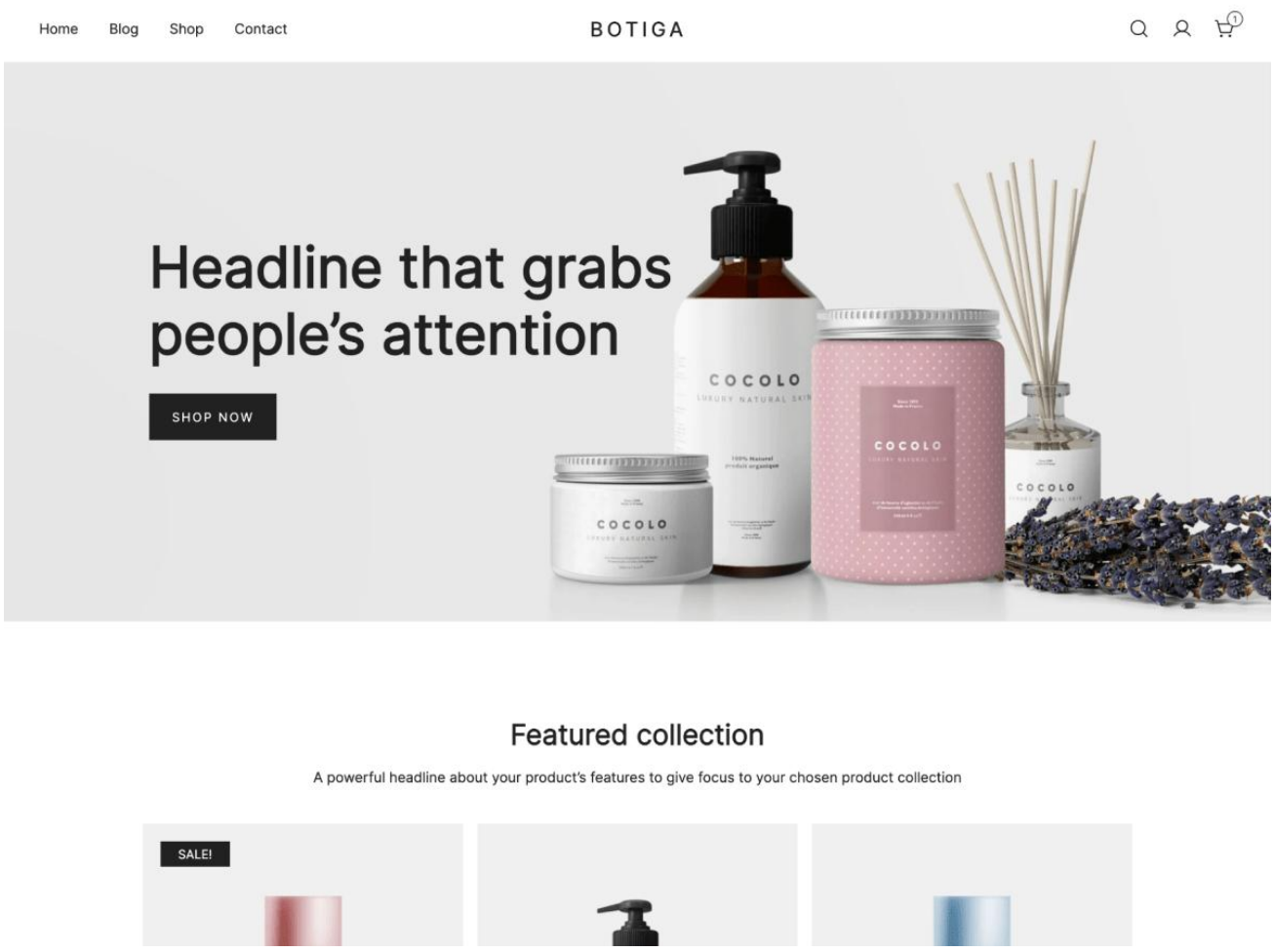

Рисунок 2.2 – Сторінка інтернет-магазину «Botiga»

Шаблон «Botiga» містить у собі такі елементи:

– головна сторінка;
– сторінка авторизації;
– каталог;
– картка товару;
– кошик;
– чек-аут;
– особистий кабінет.

2.2 Архітектура середовища тестування

Практичну складову кваліфікаційної роботи реалізовано на прикладі інтернет-магазину на базі WordPress з використанням плагіну WoCommerce. Сайт працює на локальному сервері XAMPP, який включає Apache (вебсервер) та MySQL (базу даних). Усі дані: користувачі, товари, замовлення тощо зберігаються у базі даних MySQL (доступ реалізовано через адміністративну панель phpMyAdmin). Для демонстрації взаємодії інструментів автоматизованого тестування з об’єктом побудовано архітектурну схему проєкту (рис. 2.3).

XAMPP – безкоштовний багатоплатформовий пакет із відкритим вихідним кодом, до складу якого входять HTTP-сервер Apache, системи керування базами даних MariaDB і MySQL, інтерпретатори мов PHP та Perl, а також додаткові бібліотеки, що забезпечують роботу повноцінного вебсервера [15].

Apache HTTP Server – це вільне програмне забезпечення, яке є найпопулярнішим вебсервером у світі. Він використовується для обслуговування вебсайтів і вебдодатків, обробляючи HTTP-запити та передаючи відповідні дані користувачам.

MySQL – це вільна система керування реляційними базами даних (СКБД) з відкритим вихідним кодом, яка використовується для створення та керування базами даних, особливо в контексті веброзробки.

phpMyAdmin – це безкоштовна вебпрограма з відкритим вихідним кодом, написана на PHP, яка використовується для управління базами даних MySQL і MariaDB через вебінтерфейс.

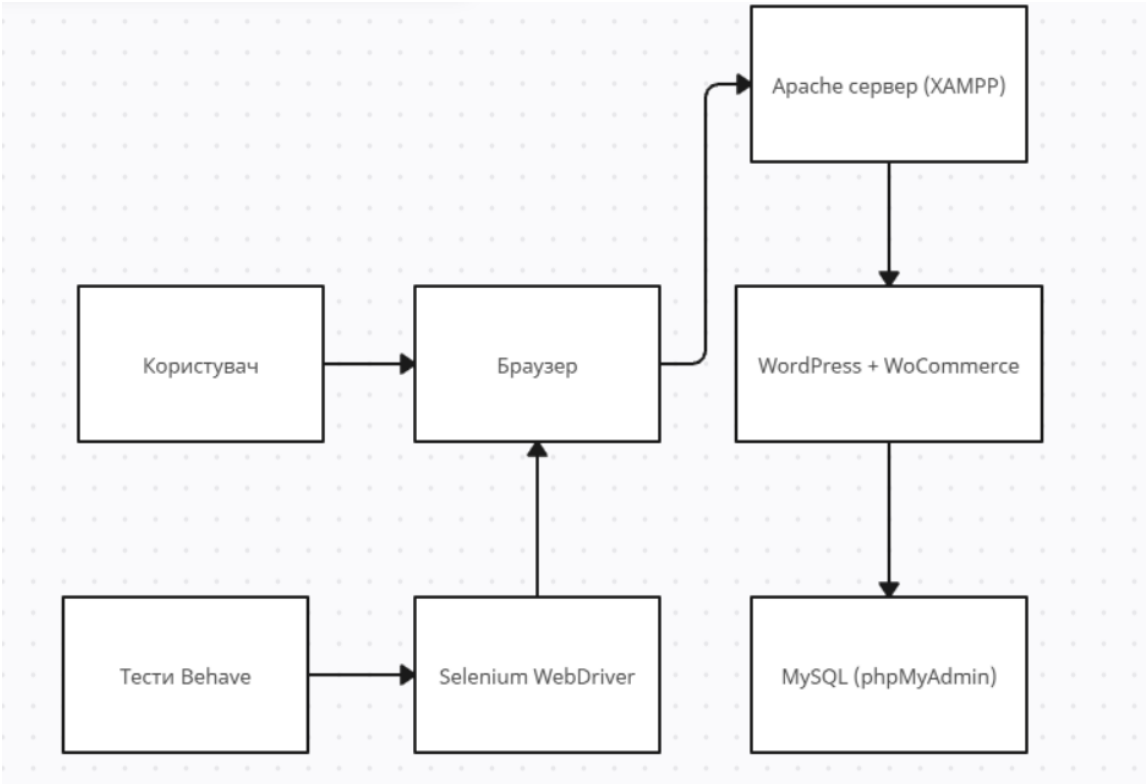

Рисунок 2.3 – Схема архітектури середовища тестування

Опис кожного компоненту окремо:

- Apache обробляє запити до сайту;
- WordPress + WooCommerce – CMS для керування контентом і плагін електронної комерції;
- phpMyAdmin – графічний інтерфейс для взаємодії з базою даних;
- MySQL – база даних, де зберігається вся інформація сайту;
- фронт-енд (HTML/CSS/JavaScript) – інтерфейс користувача, генерується WordPress-ом;
- панель адміністратора WordPress – використовується для керування товарами, користувачами, замовленнями.

Особливості архітектури:

- усі компоненти запущені на одному сервері (localhost);
- вебзастосунок розгорнуто локально, що спрощує налагодження та тестування;
- немає розділення на мікросервіси чи окремі API – усе працює всередині WordPress;
- стандартна структура бази даних WooCommerce (таблиці wp_users, wp_posts, wp_woocommerce_order_items тощо).

2.3 Огляд функціональних можливостей

Основні функції користувача:

- авторизація в акаунті;
- перегляд каталогу товарів;
- додавання товарів у кошик;
- зміна кількості товарів у кошику;
- видалення товарів із кошика;
- використання купонів на знижку;
- перехід до оформлення замовлення (checkout);
- заповнення персональних даних (піб, адреса, email, телефон);
- підтвердження замовлення;
- перегляд історії замовлень в особистому кабінеті;
- редагування особистих даних.

Функції адміністратора (у межах WordPress):

- додавання/редагування/видалення товарів;
- управління замовленнями;
- керування купонами на знижку;
- перегляд звітів і аналітики;
- управління користувачами (адміністрування облікових записів).

2.4 Інтерфейс користувача та UI-елементи

Об’єкт тестування на базі шаблону для WordPress «Botiga» має структуру: header (логотип, меню навігації, кошик, кабінет користувача), main (каталог товарів, картки продуктів, перелік категорій продукції), footer (контакти, інформація про доставку/оплату, посилання на політику конфіденційності тощо).

До основних UI-елементів відносяться:

– форма авторизації користувача;
– картка товару;
– форма редагування кошику;
– форма оформлення замовлення;
– особистий кабінет.

UI-елементи, які важливі для тестування:

– кнопки;
– інтерактивні елементи;
– повідомлення про помилку або успішні дії;
– таблиці.

2.5 Поточний підхід до тестування

2.5.1 BDD (Behavior-Driven Development)

Тести формалізовані у вигляді Gherkin-сценаріїв (.feature файли). Це дозволяє описати поведінку системи зрозумілою мовою («Given-When-Then»), що підвищує прозорість тестування. Використано фреймворк Behave.

2.5.2 Функціональне тестування

Перевіряється працездатність ключових бізнес-процесів інтернет-магазину: авторизація, додавання товарів, робота з кошиком, оформлення замовлень. Тести орієнтовані на відповідність очікуваній поведінці.

2.5.3 UI-тестування (E2E на рівні користувача)

Взаємодія з вебзастосунком відбувається через браузер (Selenium WebDriver). Емуляція дій реального користувача: кліки по кнопках, введення даних у форми, перевірка повідомлень і таблиць.

2.5.4 Негативне тестування

Перевірка роботи системи при введенні некоректних даних (порожні поля, хибні email/телефон/індекс). Дає змогу переконатися, що система правильно обробляє помилки.

2.5.5 Регресійне тестування (частково)

Автотести можуть запускатися повторно після змін у коді чи конфігурації сайту, перевіряючи, що існуючий функціонал не зламався.

2.5.6 Візуалізація результатів (репортинг)

Інтеграція з Allure дає змогу аналізувати результати тестів, бачити статус проходження сценаріїв і прикріплені скріншоти. Це відповідає підходу Test Reporting & Monitoring.

2.6 Обґрунтування вибору стеку

Обрання стеку технологій та методології для автоматизованого тестування є критичним етапом дослідницької роботи, після чого саме вони отримують ефективність, гнучкість та можливість кількісної оцінки результатів. У цьому дослідженні було обрано комбінацію BDD, Python (як мова програмування), Selenium WebDriver, Behave та Allure Report, яка оптимально відповідає меті дослідження: оцінки ефективності BDD-підходу при тестуванні веб-застосунків.

У контексті BDD тести є не лише перевіркою, але й формальною специфікацією очікуваної поведінки системи. Це дозволяє перейти від простого тестування реалізації до дослідження якості специфікації через автоматизацію.

Чітка структури BDD-сценаріїв (Given-When-Then) спрощує трасування вимог та створює об'єктивну базу для порівняльного аналізу ефективності з традиційними методами.

Selenium є фактичним індустріальним стандартом для автоматизації наскрізного (E2E) тестування веб-застосунків. Він надає необхідний API для взаємодії з елементами ведення браузера, що є критичним місцем для імітації реальної поведінки користувача.

Python – це інтерпретована, високопродуктивна мова, що має простий та зрозумілий синтаксис. Завдяки потужній екосистемі та широкій підтримці спільноти, Python є єдиним із найпопулярніших мов для автоматизації тестування (разом із Java та JavaScript). Його лаконічність дозволяє швидко створювати та підтримувати тестовий код, що є фактором для оцінки швидкості розробки автотестів у дослідженні.

Фреймворк Behave пропонує хуки та механізми контексту, які сприяють організації тестового оточення та його очищення, підвищуючи надійність та відтворюваність тестів. Це критично для отримання точних кількісних даних.

Allure генерує деталізовані інтерактивні звіти, які чітко відображають: загальний час виконання, кількість успішних/невдалих/пропущених тестів, історію прогонів, скріншоти помилок та логування.

3 АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

3.1 Ініціація проєкту

Для зберігання чистоти написання коду та структуризації проєкту необхідно дотримуватись вимог, які зазначені у документаціях до фреймфорків.

При використанні Behave слід заздалегідь створити директорії: «features», «features/steps», «pages». Директорія «features» містить у собі файли «.feature», де прописані сценарії на мові Gherkin, «features/steps» – файли «.py», де реалізована взаємодія кроків сценаріїв з функціями Python, які використовують екземпляри класів POM, а «pages» – також файли «.py», де відбувається саме логіка процесів, які виконуються у методах класу.

Для контролю версій Git у корені проєкту потрібно створити файл «.gitignore», де списком занести файли чи категорії, які не будуть зберігатися у комітках гілки розробки.

Git-коміт – це «знімок» змін у проєкті, який зберігає поточний стан. Кожен коміт містить інформацію про внесені зміни, автора, час їх створення та зв’язок з попередніми комітами, що дозволяє відстежувати версії коду та повертатись до потрібного стану проєкту [16].

Заздалегідь у базі даних, а саме у таблиці «wp_users» через інтерфейс PhpMyAdmin було створено користувача «user1», який не мав прав адміністратора, на відміну від користувача, який автоматично створюється WordPress-ом. Також через панель адміністратора WordPress створено купон «DISCOUNT25», який надає знижку 25% на чек.

3.2 Складання сценаріїв тестування

Проаналізувавши об’єкт тестування, було виділено перелік сценаріїв.

Авторизація:

– успішна авторизація;
– авторизація з хибним паролем;
– авторизація з хибним логіном;
– авторизація з порожніми полями.

Робота з кошиком:

– додавання товару в кошик;
– збільшення та зменшення одиниць певного товару;
– видалення товару з кошика;
– активація купона на знижку у кошику;
– пустий кошик(не можна перейти до чек-ауту).

Оформлення замовлень:

– оформлення з валідними даними;
– оформлення з хибним форматом даних (zip, phone, email);
– спроба оформлення без заповнення обов'язкових полів.

Особливості профілю:

– зміна особистих даних;
– перевірка наявності заказів в особистому кабінеті.

Посилаючись на розділ 1.4, на мові Gherkin було створено такі сценарії:

У файлі «login.feature» реалізовано тестування авторизації користувача.

Feature: Авторизація користувача

Scenario: Успішна авторизація

Given Користувач відкриває головну сторінку сайту

When Переходить до авторизації

And Вводить логін "user1" і пароль "user1_password"

And Натискає кнопку Login

Then Відображається сторінка особистого профілю

Scenario: Авторизація з хибним паролем

Given Користувач відкриває головну сторінку сайту

When Переходить до авторизації

And Вводить логін "user1" і пароль "wrong_password"

And Натискає кнопку Login

Then Користувач отримав помилку при авторизації

Scenario: Авторизація з хибним логіном

Given Користувач відкриває головну сторінку сайту

When Переходить до авторизації

And Вводить логін "not_registered_user" і пароль "not_registered_user_password"

And Натискає кнопку Login

Then Користувач отримав помилку при авторизації

Scenario: Авторизація з порожніми полями

Given Користувач відкриває головну сторінку сайту

When Переходить до авторизації

And Не вводить логін та пароль

And Натискає кнопку Login

Then Користувач отримав помилку при авторизації

У файлі «cart.feature» реалізовано тестування кошика користувача.

Feature: Кошик користувача

Scenario: Порожній кошик

Given Користувач авторизувався

When Перейшов до кошика

Then Отривав повідомлення про порожній кошик

Scenario: Додавання товару в кошик

Given Користувач авторизувався

When Перейшов до сторінки Shop

And Обрав товар: "“Eternal Sunset Collection Lip and Cheek”"

And Перейшов до кошика

Then Кошик містить товар

Scenario: Збільшення кількості одиниць товару

Given Користувач авторизувався

When Перейшов до сторінки Shop

And Обрав товар: "“Eternal Sunset Collection Lip and Cheek”"

And Перейшов до кошика

And Збільшив кількість одиниць для товару: "Eternal Sunset Collection Lip and Cheek"

Then Кількість одиниць товару збільшилась

Scenario: Зменшення кількості одиниць товару

Given Користувач авторизувався

When Перейшов до сторінки Shop

And Обрав товар: "“Eternal Sunset Collection Lip and Cheek”"

And Перейшов до кошика

And Зменшив кількість одиниць для товару: "Eternal Sunset Collection Lip and Cheek"

Then Кількість одиниць товару зменшилась

Scenario: Активація купона на знижку у кошику

Given Користувач авторизувався

When Перейшов до кошика

And Увів купон: "DISCOUNT25"

Then Отримав знижку 25%

Scenario: Видалення товару з кошика

Given Користувач авторизувався

When Перейшов до кошика

And Видалив товар: "Eternal Sunset Collection Lip and Cheek"

Then Кошик не містить товар: "Eternal Sunset Collection Lip and Cheek"

У файлі «checkout.feature» реалізовано тестування оформлення замовлення.

Feature: Оформлення замовлень

Scenario: Оформлення замовлення з коректними даними

Given Користувач авторизувався

When Заповнив кошик

And Перейшов до сторінки оформлення замовлення

And Вказав коректні дані (61000, 0991234567, user1@email.com)

Then Успішно зробив замовлення

Scenario: Оформлення замовлення з хибними даними

Given Користувач авторизувався

When Заповнив кошик

And Перейшов до сторінки оформлення замовлення

And Вказав коректні дані (unformatted postcode, unformatted phone, unformatted email)

Then Користувач отримав помилку

Scenario: Оформлення замовлення з порожніми полями

Given Користувач авторизувався

When Заповнив кошик

And Перейшов до сторінки оформлення замовлення

And Взагалі не вказав дані

Then Користувач отримав помилку

У файлі «profile.feature» реалізовано тестування змін у особистому профілі.

Feature: Особистий профіль

Scenario: Успішна зміна особистих даних

Given Користувач авторизувався

When Перейшов до особистих даних

And Вказав дані: Petr, Petrov, PETROFF, petr@gmail.com

And Вказав дані: Ivan, Ivanov, IVANOFF, ivan@gmail.com

Then Особисті дані змінено

Scenario: Хибна зміна особистих даних

Given Користувач авторизувався

When Перейшов до особистих даних

And Залишив порожні поля

Then Отримав помилку при редагуванні

Scenario: Перевірка наявності замовлень

Given Користувач авторизувався

When Перейшов до своїх замовлень

Then Побачив свої замовлення

3.3 Створення сторінок POM

У файлах сторінок директорії «pages» створено класи та методи, які використовують вхідні данні та імітують взаємодію користувача з інтерфейсом.

Код файлу «login_page.py»:

```
from selenium.webdriver.common.by import By
from selenium.webdriver.remote.webdriver import WebDriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.support import expected_conditions as EC


class LoginPage:
    def __init__(self, driver: WebDriver):
        self.driver = driver
        self.url = "http://localhost/mysite/"
        self.profile_link = (By.CLASS_NAME, "wc-account-link")
        self.username_input = (By.ID, "username")
        self.password_input = (By.ID, "password")
```

```
        self.login_button = (By.NAME, "login")
        self.error_message = (By.CLASS_NAME, "woocommerce-error")
        self.navigation = (By.CLASS_NAME, "woocommerce-MyAccount-navigation")

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

    def auth_page(self):
        self.driver.find_element(*self.profile_link).click()

    def auth_fields(self, username, password):
        self.driver.find_element(*self.username_input).send_keys(username)
        self.driver.find_element(*self.password_input).send_keys(password)

    def login(self):
        wait = WebDriverWait(self.driver, 10)
        button = wait.until(EC.element_to_be_clickable(self.login_button))
        ActionChains(self.driver).move_to_element(button).pause(1).click().perform()

    def is_logged_in(self):
        WebDriverWait(self.driver, 10)
        self.driver.find_element(*self.navigation)
```

```
def is_error_displayed(self):
    wait = WebDriverWait(self.driver, 10)
    error = wait.until(EC.visibility_of_element_located(self.error_message))
    return error.text
```

У файлі «login_page.py» створено клас LoginPage з методами: open, auth_page, auth_fields, login, is_logged_in, is_error_displayed. Селектори полів вводу даних, кнопок тощо ініціалізовані у конструкторі класу «__init__». Опис методів класу наведено у таблиці 3.1.

Таблиця 3.1 – Опис методів класу LoginPage

Назва методу	Опис
open	Відкриття головної сторінки сайту
auth_page	Клік на кнопку авторизації
auth_fields	Заповнення полів на сторінці авторизації
login	Клік на кнопку «Login»
is_logged_in	Перевірка успішної авторизації
is_error_displayed	Отримання помилки

Код файл «cart_page.py»:

```
from selenium.webdriver.remote.webdriver import WebDriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.common.action_chains import ActionChains
```

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


class CartPage:
    def __init__(self,driver: WebDriver, product_name, coupon=None):
        self.driver = driver
        self.product_name = product_name
        self.coupon = coupon
        self.shop_page = (By.XPATH, "//ul[@id='primary-menu']//a[normalize-space(text())='Shop']")
        self.cart_page = (By.ID, "site-header-cart")
        self.empty_cart_message = (By.CLASS_NAME, "cart-empty")
        self.add_to_cart_button = (By.CSS_SELECTOR, f'a[aria-label*="{product_name}"]')
        self.product_title = (By.XPATH, f"//tr[contains(@class, 'cart_item')]//td[@class='product-name']//a[text()='{product_name}']")
        self.items_quantity = (By.XPATH, f'//tr[contains(@class, "cart_item") and .//td[@class="product-name"]//a[text()="{product_name}"]]//input[@type="number"]')
        self.quantity_plus = (By.XPATH, f'//tr[contains(@class, "cart_item") and .//td[@class="product-name"]//a[text()="{product_name}"]]//a[contains(@class, "botiga-quantity-plus")]')
        self.quantity_minus = (By.XPATH, f'//tr[contains(@class, "cart_item") and .//td[@class="product-name"]//a[text()="{product_name}"]]//a[contains(@class, "botiga-quantity-minus")]')
```

```
        self.product_remove = (By.XPATH, f"//a[@class='remove' and contains(@aria-label, 'Remove {product_name}')]")
        self.coupon_code_field = (By.ID, "coupon_code")
        self.coupon_code_apply = (By.CSS_SELECTOR, 'button.button[name="apply_coupon"][value="Apply coupon"]')
        self.coupon_code_message = (By.CLASS_NAME, "woocommerce-message")
        self.cart_items = (By.CLASS_NAME, "count-number")
        self.update_cart_button = (By.CLASS_NAME, "update_cart")

    def cart_is_empty(self):
        wait = WebDriverWait(self.driver, 10)
        message = wait.until(EC.visibility_of_element_located(self.empty_cart_message))
        return message.text

    def user_go_to_shop_page(self):
        wait = WebDriverWait(self.driver, 60)
        button = wait.until(EC.visibility_of_element_located(self.shop_page))
        ActionChains(self.driver).move_to_element(button).pause(2).click().perform()

    def user_add_product_to_cart(self):
        wait = WebDriverWait(self.driver, 10)
        button = wait.until(EC.element_to_be_clickable(self.add_to_cart_button))
```

```
    ActionChains(self.driver).move_to_element(button).pause(0.5).click().perform()

def user_go_to_cart(self, context):
    wait = WebDriverWait(self.driver, 60)
    button = wait.until(EC.element_to_be_clickable(self.cart_page))
    ActionChains(self.driver).move_to_element(button).pause(2).click().perform()

def get_cart_items(self):
    return self.driver.find_element(*self.cart_items).text

def user_change_items_quantity_pluse(self):
    wait = WebDriverWait(self.driver, 10)
    button = wait.until(EC.element_to_be_clickable(self.quantity_plus))
    ActionChains(self.driver).move_to_element(button).pause(0.5).click().perform()

def user_change_items_quantity_minus(self):
    wait = WebDriverWait(self.driver, 10)
    button = wait.until(EC.element_to_be_clickable(self.quantity_minus))
    ActionChains(self.driver).move_to_element(button).pause(0.5).click().perform()

def user_update_cart(self):
    wait = WebDriverWait(self.driver, 10)
```

```
        button = wait.until(EC.element_to_be_clickable(self.update_cart_button))
        ActionChains(self.driver).move_to_element(button).pause(0.5).click().perform()

    def get_items_quantity(self):
        wait = WebDriverWait(self.driver, 10)
        quantity_input =
wait.until(EC.presence_of_element_located(self.items_quantity))
        return int(quantity_input.get_attribute("value"))

    def user_remove_product_from_cart(self):
        wait = WebDriverWait(self.driver, 10)
        button = wait.until(EC.element_to_be_clickable(self.product_remove))
        ActionChains(self.driver).move_to_element(button).pause(0.5).click().perform()

    def is_product_removed(self):
        WebDriverWait(self.driver,
20).until(EC.invisibility_of_element_located(self.product_title))
        products = self.driver.find_elements(*self.product_title)
        return products

    def user_add_coupon(self):
        self.driver.find_element(*self.coupon_code_field).send_keys(*self.coupon)
        wait = WebDriverWait(self.driver, 10)
```

```
        button = wait.until(EC.element_to_be_clickable(self.coupon_code_apply))
        ActionChains(self.driver).move_to_element(button).pause(0.5).click().perform()

    def get_coupon_code_message(self):
        wait = WebDriverWait(self.driver, 20)
        message =
wait.until(EC.visibility_of_element_located(self.coupon_code_message))
        return message.text
```

У файлі «cart_page.py» створено клас CartPage з методами: cart_is_empty, user_go_to_shop_page, user_add_product_to_cart, user_go_to_cart, get_cart_items, user_change_items_quantity_pluse, user_change_items_quantity_minus, user_update_cart, get_items_quantity, user_remove_product_from_cart, is_product_removed, user_add_coupon, get_coupon_code_message. Селектори полів вводу даних, кнопок тощо ініціалізовані у конструкторі класу «__init__». Опис методів класу наведено у таблиці 3.2.

Таблиця 3.2 – Опис методів класу CartPage

Назва методу	Опис
cart_is_empty	Перевірка на порожній кошик
user_go_to_shop_page	Користувач відкриває каталог
user_add_product_to_cart	Користувач додає продукт у кошик
user_go_to_cart	Користувач переходить до кошику

Кінець таблиці 3.2

Назва методу	Опис
get_cart_items	Кількість товарів у кошику
user_change_items_quantity_pluse	Збільшення кількості одиниць певного товару
user_change_items_quantity_minus	Зменшення кількості одиниць певного товару
user_update_cart	Оновлення кошику
get_items_quantity	Кількість одиниць певного товару
user_remove_product_from_cart	Видалення товару з кошика
is_product_removed	Перевірка видалення товару
user_add_coupon	Додавання купону
get_coupon_code_message	Отримання помилки при застосуванні купону

Код файлу «checkout_page.py»:

```
from selenium.webdriver.remote.webdriver import WebDriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC


class CheckoutPage:
    def __init__(self, driver: WebDriver, postcode = None, phone_number = None, email = None):
        self.driver = driver
```

```
        self.postcode = postcode
        self.phone_number = phone_number
        self.email = email
        self.checkout_button = (By.CLASS_NAME, "checkout-button")
        self.first_name_field = (By.ID, 'billing_first_name')
        self.last_name_field = (By.ID, 'billing_last_name')
        self.street_address_field = (By.ID, 'billing_address_1')
        self.city_field = (By.ID, 'billing_city')
        self.postcode_field = (By.ID, 'billing_postcode')
        self.phone_number_field = (By.ID, 'billing_phone')
        self.email_field = (By.ID, 'billing_email')
        self.terms_check_box = (By.ID, 'terms')
        self.place_order_button = (By.ID, 'place_order')
        self.error_message = (By.CLASS_NAME, "woocommerce-error")
        self.checkout_message_success = (By.CLASS_NAME, "woocommerce-notice--
success")

    def user_go_to_checkout(self):
        wait = WebDriverWait(self.driver, 10)
        button = wait.until(EC.element_to_be_clickable(self.checkout_button))
        ActionChains(self.driver).move_to_element(button).pause(0.5).click().perform()

    def fill_input(self, locator, value):
```

```
        element = self.driver.find_element(*locator)
        element.clear()
        element.send_keys(value)

    def user_entered_data(self):
        self.fill_input(self.first_name_field, "Ivan")
        self.fill_input(self.last_name_field, "Ivanov")
        self.fill_input(self.street_address_field, "st. Nauki, 22")
        self.fill_input(self.city_field, "Kharkiv")
        self.fill_input(self.postcode_field, self.postcode)
        self.fill_input(self.phone_number_field, self.phone_number)
        self.fill_input(self.email_field, self.email)

        self.driver.execute_script("arguments[0].checked = true;",
self.driver.find_element(*self.terms_check_box))
        wait = WebDriverWait(self.driver, 10)
        button_place_order =
wait.until(EC.element_to_be_clickable(self.place_order_button))

ActionChains(self.driver).move_to_element(button_place_order).pause(5).click().per
form()

    def user_skipped_data(self):
```

```
        self.driver.find_element(*self.first_name_field).clear()
        self.driver.find_element(*self.last_name_field).clear()
        self.driver.find_element(*self.street_address_field).clear()
        self.driver.find_element(*self.city_field).clear()
        self.driver.find_element(*self.postcode_field).clear()
        self.driver.find_element(*self.phone_number_field).clear()
        self.driver.find_element(*self.email_field).clear()

        self.driver.execute_script("arguments[0].checked = true;",
self.driver.find_element(*self.terms_check_box))
        wait = WebDriverWait(self.driver, 10)
        button_place_order =
wait.until(EC.element_to_be_clickable(self.place_order_button))

ActionChains(self.driver).move_to_element(button_place_order).pause(5).click().per
form()

    def checkout_success(self):
        wait = WebDriverWait(self.driver, 100)
        message =
wait.until(EC.presence_of_element_located(self.checkout_message_success))
        return message.text

    def checkout_error(self):
```

```
wait = WebDriverWait(self.driver, 100)
error = wait.until(EC.presence_of_element_located(self.error_message))
self.driver.execute_script("arguments[0].scrollIntoView({behavior: 'smooth', block: 'center'});", error)
return error.text
```

У файлі «checkout_page.py» створено клас CartPage з методами: user_go_to_checkout, fill_input, user_entered_data, user_skipped_data, checkout_success, checkout_error. Селектори полів вводу даних, кнопок тощо ініціалізовані у конструкторі класу «__init__». Опис методів класу наведено у таблиці 3.3.

Таблиця 3.3 – Опис методів класу CheckoutPage

Назва методу	Опис
user_go_to_checkout	Користувач переходить до сторінки оформлення замовлення
fill_input	Допоміжний метод для спрощення коду при заповненні полів даними
user_entered_data	Користувач заповнює данні
user_skipped_data	Користувач залишає поля порожніми
checkout_success	Успішне створення замовлення
checkout_error	Отримання помилки при створенні замовлення

Код файлу «profile_page.py»:

```
from selenium.common import TimeoutException
```

```
from selenium.webdriver.remote.webdriver import WebDriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC


class ProfilePage:
    def __init__(self, driver: WebDriver, first_name = None, last_name = None, display_name = None, email = None):
        self.driver = driver
        self.first_name = first_name
        self.last_name = last_name
        self.display_name = display_name
        self.email = email
        self.account_details = (By.XPATH, "//a[contains(@href, 'edit-account') and normalize-space(text())='Account details']")
        self.first_name_field = (By.ID, 'account_first_name')
        self.last_name_field = (By.ID, 'account_last_name')
        self.display_name_field = (By.ID, 'account_display_name')
        self.email_field = (By.ID, 'account_email')
        self.save_changes_button = (By.XPATH, "//button[@name='save_account_details' and @value='Save changes' and text()='Save changes']")
```

```
        self.error_message = (By.CLASS_NAME, "woocommerce-error")


    def user_go_to_account_details_page(self):
        wait = WebDriverWait(self.driver, 10)
        account_details = wait.until(EC.element_to_be_clickable(self.account_details))

ActionChains(self.driver).move_to_element(account_details).pause(2).click().perfor
m()


    def previous_account_details(self):
        previous_first_name =
self.driver.find_element(*self.first_name_field).get_attribute('value')
        previous_last_name =
self.driver.find_element(*self.last_name_field).get_attribute('value')
        previous_display_name =
self.driver.find_element(*self.display_name_field).get_attribute('value')
        previous_email =
self.driver.find_element(*self.email_field).get_attribute('value')
        return previous_first_name, previous_last_name, previous_display_name,
previous_email


    def user_skipped_data(self):
        self.driver.find_element(*self.first_name_field).clear()
        self.driver.find_element(*self.last_name_field).clear()
```

```
        self.driver.find_element(*self.display_name_field).clear()
        self.driver.find_element(*self.email_field).clear()

        wait = WebDriverWait(self.driver, 10)
        save_changes_button =
wait.until(EC.element_to_be_clickable(self.save_changes_button))

ActionChains(self.driver).move_to_element(save_changes_button).pause(2).click().p
erform()

    def clear_and_fill_fields(self, locator, value):
        element = self.driver.find_element(*locator)
        element.clear()
        element.send_keys(value)

    def user_enter_data(self):
        self.clear_and_fill_fields(self.first_name_field, self.first_name)
        self.clear_and_fill_fields(self.last_name_field, self.last_name)
        self.clear_and_fill_fields(self.display_name_field, self.display_name)
        self.clear_and_fill_fields(self.email_field, self.email)

        wait = WebDriverWait(self.driver, 10)
```

```
        save_changes_button =
wait.until(EC.element_to_be_clickable(self.save_changes_button))

ActionChains(self.driver).move_to_element(save_changes_button).pause(2).click().p
erform()


    def save_changes_error(self):
        wait = WebDriverWait(self.driver, 100)
        error = wait.until(EC.presence_of_element_located(self.error_message))
        return error.text


class OrdersPage:
    def __init__(self, driver: WebDriver):
        self.driver = driver
        self.orders_page = (By.XPATH, "//a[contains(@href, '/my-account/orders/') and
normalize-space(text())='Orders']")
        self.orders_list = (By.XPATH, "//table[contains(@class, 'woocommerce-orders-
table')]")
        self.empty_orders_list = (By.CLASS_NAME, "woocommerce-info")


    def user_go_to_order_page(self):
        wait = WebDriverWait(self.driver, 10)
        orders_page = wait.until(EC.element_to_be_clickable(self.orders_page))
```

```
ActionChains(self.driver).move_to_element(orders_page).pause(2).click().perform()


    def user_get_orders_list(self) -> bool:
        try:
            WebDriverWait(self.driver,
10).until(EC.presence_of_element_located(self.orders_list))
            return True
        except TimeoutException:
            return False


    def orders_list_is_empty(self):
        wait = WebDriverWait(self.driver, 10)
        empty_orders_list =
wait.until(EC.presence_of_element_located(self.empty_orders_list))
        return empty_orders_list.text
```

У файлі «profile_page.py» створено клас ProfilePage з методами: user_go_to_account_details_page, previous_account_details, user_skipped_data, clear_and_fill_fields, user_enter_data, save_changes_error, та класс OrdersPage з методами: user_go_to_order_page, user_get_orders_list, orders_list_is_empty. Селектори полів вводу даних, кнопок тощо ініціалізовані у конструкторах класів «__init__». Опис методів класів наведено у таблицях 3.4 та 3.5.

Таблиця 3.4 – Опис методів класу ProfilePage

Назва методу	Опис
user_go_to_account_details_page	Користувач переходить до особистих своїх особистих даних
previous_account_details	Зберігає попередні данні для порівняння
user_skipped_data	Користувач залишає поля порожніми
clear_and_fill_fields	Допоміжний метод для заповнення полів
user_enter_data	Користувач вводить нові данні
save_changes_error	Отримання помилки при зміні даних користувача

Таблиця 3.5 – Опис методів класу OrdersPage

Назва методу	Опис
user_go_to_order_page	Користувач переходить до своїх замовлень
user_get_orders_list	Користувач отримує список своїх замовлень
orders_list_is_empty	Список замовлень порожній

3.4 Реалізація кроків тестування на базі сторінок POM

Кроки тестування прописуються таким чином, щоб для проведення тестів з різноманітними даними не потрібно було змінювати або створювати новий фрагмент коду. Приклад структури коду для кроку тестування:

```
@given ("Користувач відкриває головну сторінку сайту")
def step_open_site(context):
```

```
    options = Options()
    options.add_argument("--start-maximized")
    context.driver = webdriver.Chrome(options=options)
    context.page = LoginPage(context.driver)
    context.page.open()


@when('Вводить логін "{username}" і пароль "{password}"')
def step_login(context, username, password):
    context.page.auth_fields(username, password)
```

«@given» – декоратор (у випадку з Behave може бути when чи then), який дозволяє модифікувати роботу функції, обернувши її в іншу функцію. Функція «step_open_site» відкриває браузер Chrome у розгорнутому вигляді, створює екземпляр сторінки авторизації «LoginPage» та переходить на неї. Функція «step_login» отримує дані «username»та «password» з кроку сценарія і передає їх до екземпляру «context.page» класу «LoginPage». Таким чином перед запуском тесту можна редагувати вхідні данні тестування у сценарії, не змінюючи при цьому код [17].

Код кроків тестування файлу «login_steps.py»:

```
from behave import given, when, then
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from pages.login_page import LoginPage
from tools import attach_screenshot
```

```
@given ("Користувач відкриває головну сторінку сайту")
def step_open_site(context):
    options = Options()
    options.add_argument("--start-maximized")
    context.driver = webdriver.Chrome(options=options)
    context.page = LoginPage(context.driver)
    context.page.open()

@when("Переходить до авторизації")
def step_open_auth_window(context):
    context.page.auth_page()

@when('Вводить логін "{username}" і пароль "{password}"')
def step_login(context, username, password):
    context.page.auth_fields(username, password)

@when("Не вводить логін та пароль")
def step_login_with_empty_fields(context):
    pass

@when("Натискає кнопку Login")
```

```
def step_click_login(context):
    context.page.login()


@then("Відображається сторінка особистого профілю")
def step_check_login(context):
    context.page.is_logged_in()
    attach_screenshot(context, "Користувач потрапив до особистого профілю")
    context.driver.close()


@then ("Користувач отримав помилку при авторизації")
def step_user_get_error(context):
    error_message = context.page.is_error_displayed()
    attach_screenshot(context, error_message)
    context.driver.close()
```

Код кроків тестування файлу «cart_steps.py»:

```
from behave import given, when, then
from tools import attach_screenshot
from pages.cart_page import CartPage
from features.steps import login_steps


@given("Користувач авторизувався")
```

```
def step_user_is_logged_in(context):
    login_steps.step_open_site(context)
    login_steps.step_open_auth_window(context)
    login_steps.step_login(context, "user1", "user1_password")
    login_steps.step_click_login(context)
    context.page = CartPage(context.driver, product_name="", coupon="")


@when("Перейшов до сторінки Shop")
def step_user_go_to_shop_page(context):
    context.page.user_go_to_shop_page()


@when('Обрав товар: "{product_name}"')
def step_user_add_product_product_to_cart(context, product_name):
    context.page = CartPage(context.driver, product_name)
    context.page.user_add_product_to_cart()


@when("Перейшов до кошика")
def step_user_go_to_cart_page(context):
    context.page.user_go_to_cart(context)


@when('Збільшив кількість одиниць для товару: "{product_name}"')
def step_user_change_items_quantity_pluse(context, product_name):
```

```
    context.page = CartPage(context.driver, product_name)
    context.previous_quantity = context.page.get_items_quantity()
    context.page.user_change_items_quantity_pluse()

@when('Зменшив кількість одиниць для товару: "{product_name}"')
def step_user_change_items_quantity_minus(context, product_name):
    context.page = CartPage(context.driver, product_name)
    context.previous_quantity = context.page.get_items_quantity()
    context.page.user_change_items_quantity_minus()

@when('Видалив товар: "{product_name}"')
def step_user_remove_product_from_cart(context, product_name):
    context.page = CartPage(context.driver, product_name)
    context.page.user_remove_product_from_cart()

@when('Увів купон: "{coupon}"')
def step_user_add_coupon(context, coupon):
    context.page = CartPage(context.driver, product_name="", coupon=coupon)
    context.page.user_add_coupon()

@then("Отримав знижку 25%")
def step_user_get_discount(context):
```

```
    try:
        message = context.page.get_coupon_code_message()
        attach_screenshot(context, "Купон успішно використано")
        context.driver.close()
        assert message == "Coupon code applied successfully."
    except AssertionError:
        attach_screenshot(context, "Купон не використано")
        context.driver.close()


@then('Кошик не містить товар: "{product_name}"')
def step_product_removed(context, product_name):
    try:
        elements = context.page.is_product_removed()
        attach_screenshot(context, "Товар видалено з кошика")
        context.driver.close()
        assert len(elements) == 0
    except AssertionError:
        attach_screenshot(context, "Товар не видалився")
        context.driver.close()


@then("Кількість одиниць товару збільшилась")
def step_items_quantity_is_increase(context):
```

```
    try:
        new_quantity = context.page.get_items_quantity()
        attach_screenshot(context, "Кількість одиниць товару збільшилась")
        context.driver.close()
        assert new_quantity > context.previous_quantity
    except AssertionError:
        attach_screenshot(context, "Кількість одиниць товару не змінилась")
        context.driver.close()


@then("Кількість одиниць товару зменшилась")
def step_items_quantity_is_decrease(context):
    try:
        new_quantity = context.page.get_items_quantity()
        attach_screenshot(context, "Кількість одиниць товару зменшилась")
        context.driver.close()
        assert new_quantity < context.previous_quantity
    except AssertionError:
        attach_screenshot(context, "Кількість одиниць товару не змінилась")
        context.driver.close()


@then ("Кошик містить товар")
def step_cart_has_items(context):
```

```
    try:
        items = context.page.get_cart_items()
        attach_screenshot(context, "Кошик містить товар")
        context.driver.close()
        assert len(items) >= 1
    except AssertionError:
        attach_screenshot(context, "Помилка: Порожній кошик")
        context.driver.close()


@then ("Отривав повідомлення про порожній кошик")
def step_user_has_empty_cart(context):
    try:
        message = context.page.cart_is_empty()
        attach_screenshot(context, "Порожній кошик")
        context.driver.close()
        assert message == "Your cart is currently empty."
    except AssertionError:
        attach_screenshot(context, "У кошику є товар")
        context.driver.close()
```

Код кроків тестування файлу «checkout_steps.py»:

```
from behave import when, then
```

```
from pages.checkout_page import CheckoutPage
from features.steps import cart_steps
from tools import attach_screenshot


@when("Заповнив кошик")
def step_user_fill_cart(context):
    cart_steps.step_user_go_to_shop_page(context)
    cart_steps.step_user_add_product_product_to_cart(context, '"“Deep Sweep 2% BHA Pore Cleaning Toner”"')
    cart_steps.step_user_add_product_product_to_cart(context, '"“Eternal Sunset Collection Lip and Cheek”"')
    cart_steps.step_user_add_product_product_to_cart(context, '"“Facial Treatment Essence (Pitera Essence)”"')


@when("Перейшов до сторінки оформлення замовлення")
def step_user_go_to_checkout(context):
    cart_steps.step_user_go_to_cart_page(context)
    context.page = CheckoutPage(context.driver)
    context.page.user_go_to_checkout()


@when('Вказав коректні дані ({postcode}, {phone_number}, {email})')
def step_user_entered_valid_data(context, postcode, phone_number, email):
```

```
    context.page = CheckoutPage(context.driver, postcode, phone_number,
email)
    context.page.user_entered_data()


@when("Взагалі не вказав дані")
def step_user_skipped_data(context):
    context.page = CheckoutPage(context.driver)
    context.page.user_skipped_data()


@then("Успішно зробив замовлення")
def step_user_send_order_successfully(context):
    try:
        message = context.page.checkout_success()
        attach_screenshot(context, "Користувач успішно зробив замовлення")
        context.driver.close()
        assert message == "Thank you. Your order has been received."
    except AssertionError:
        error_message = context.page.checkout_error()
        attach_screenshot(context,  error_message)
        context.driver.close()


@then("Користувач отримав помилку")
def step_user_send_order_successfully(context):
```

```
        try:
            error_message = context.page.checkout_error()
            attach_screenshot(context,  error_message)
            context.driver.close()
        except AssertionError:
            attach_screenshot(context, "Невідома помилка")
            context.driver.close()
```

Код кроків тестування файлу «profile_steps.py»:

```
from behave import when, then
from pages.profile_page import ProfilePage, OrdersPage
from tools import attach_screenshot


@when("Перейшов до особистих даних")
def step_user_go_to_personal_info(context):
    context.page = ProfilePage(context.driver)
    context.page.user_go_to_account_details_page()


@when("Вказав дані: {first_name}, {last_name}, {display_name}, {email}")
def step_user_edit_personal_info(context, first_name, last_name,
display_name, email):
    context.page = ProfilePage(context.driver, first_name, last_name,
display_name, email)
```

```
        context.page.new_account_details = first_name, last_name, display_name,
email
        context.page.user_enter_data()
        context.previous_account_details = context.page.previous_account_details()


    @when("Залишив порожні поля")
    def step_user_skipped_data(context):
        context.page.user_skipped_data()



    @when("Перейшов до своїх замовлень")
    def step_user_go_to_orders(context):
        context.page = OrdersPage(context.driver)
        context.page.user_go_to_order_page()


    @then("Особисті дані змінено")
    def step_user_updated_personal_info(context):
        try:
            attach_screenshot(context, "Дані користувача оновлено")
            context.driver.close()
            assert context.page.previous_account_details !=
context.page.new_account_details
        except AssertionError:
```

```
        attach_screenshot(context, "Дані не змінилися")
        context.driver.close()


@then("Отримав помилку при редагуванні")
def step_user_get_error(context):
    try:
        error_message = context.page.save_changes_error()
        attach_screenshot(context, error_message)
        context.driver.close()
    except AssertionError:
        attach_screenshot(context, "Дані відредаговано")
        context.driver.close()


@then("Побачив свої замовлення")
def step_user_get_orders(context):
    try:
        message = context.page.user_get_orders_list()
        attach_screenshot(context, "Замовленян користувача")
        context.driver.close()
        assert message == True
    except AssertionError:
        error_message = context.page.orders_list_is_empty()
```

```
attach_screenshot(context, error_message)
context.driver.close()
```

3.5 Формування звітності тестування

Генерація звітності відбувається завдяки системі Allure. Запуск тестів викликається консольними командами (можуть містити певні аргументи) наведені в таблиці 3.6.

Таблиця 3.6 – Консольні команди для виклику тестів

Консольна команда	Опис
behave -f pretty	Виконує запуск усіх сценаріїв проєкту
behave features/«назва файлу із сценарієм»	Виконує запуск певного сценарію
behave -f allure_behave.formatter:AllureFormatter -o allure-results	Виконує запуск усіх сценаріїв з використанням Allure
allure generate allure-results -o allure-report --clean	Створення звітності Allure після виконання тестів
allure open allure-report	Перегляд звіту Allure

3.6 Результати тестування

Після проведення тестування результати можна переглянути запустивши веб версію Allure. У звітності можна побачити які тести пройшли перевірку, а які ні, на якому етапі тестування сталася критична помилка, додаткові матеріали (скріншоти), тривалість тестування тощо. Результати тестування вебзастосунку наведено на рисунках 3.1 – 3.16

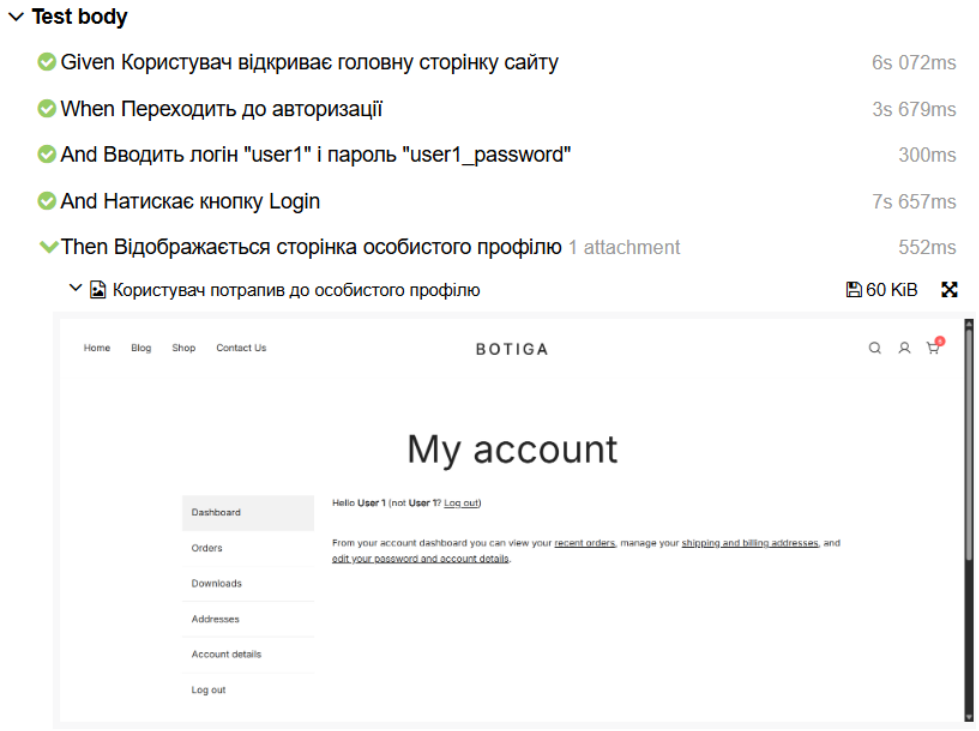

Рисунок 3.1 – Успішна авторизація

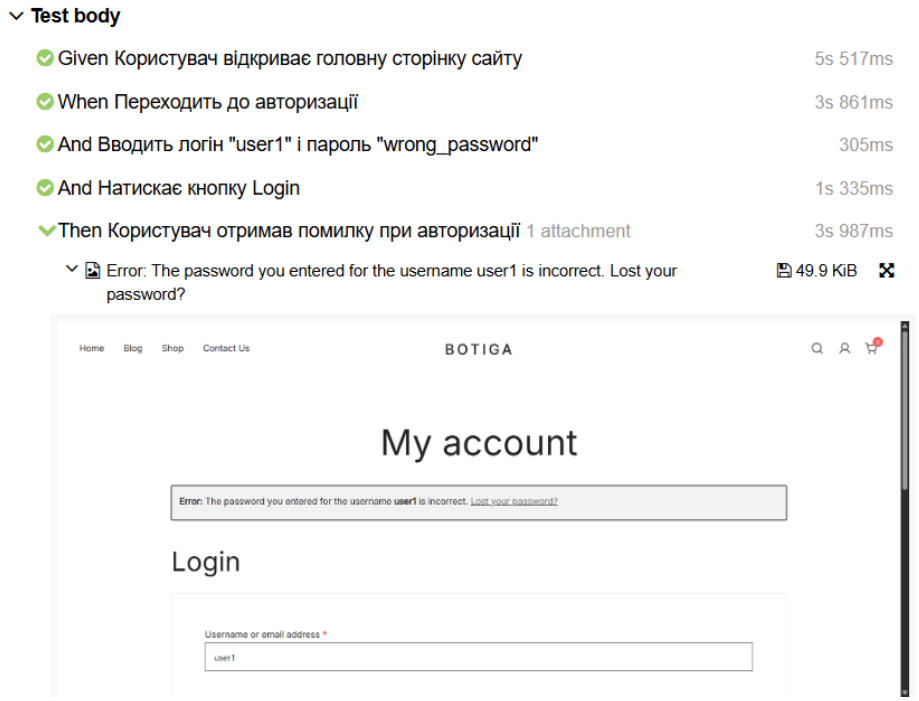

Рисунок 3.2 – Авторизація з хибним паролем

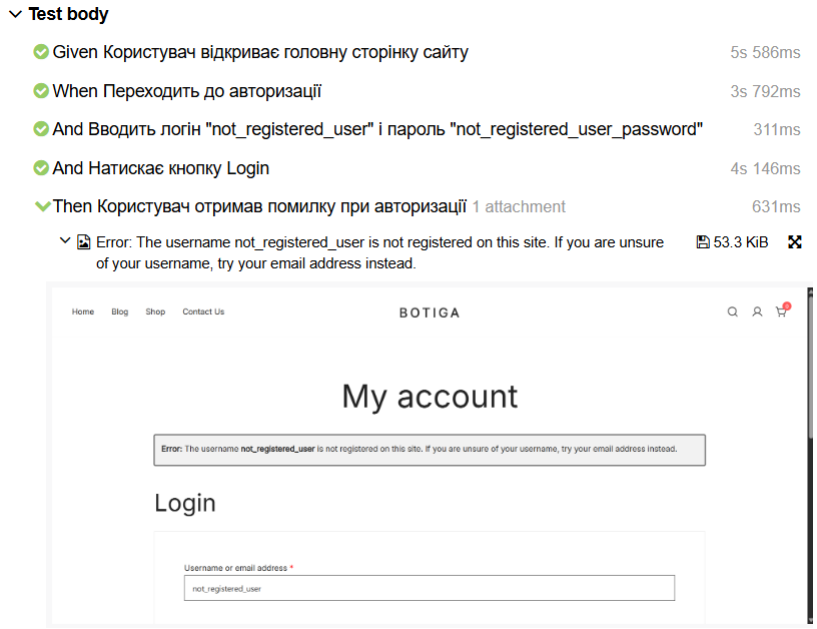

Рисунок 3.3 – Авторизація з хибним логіном

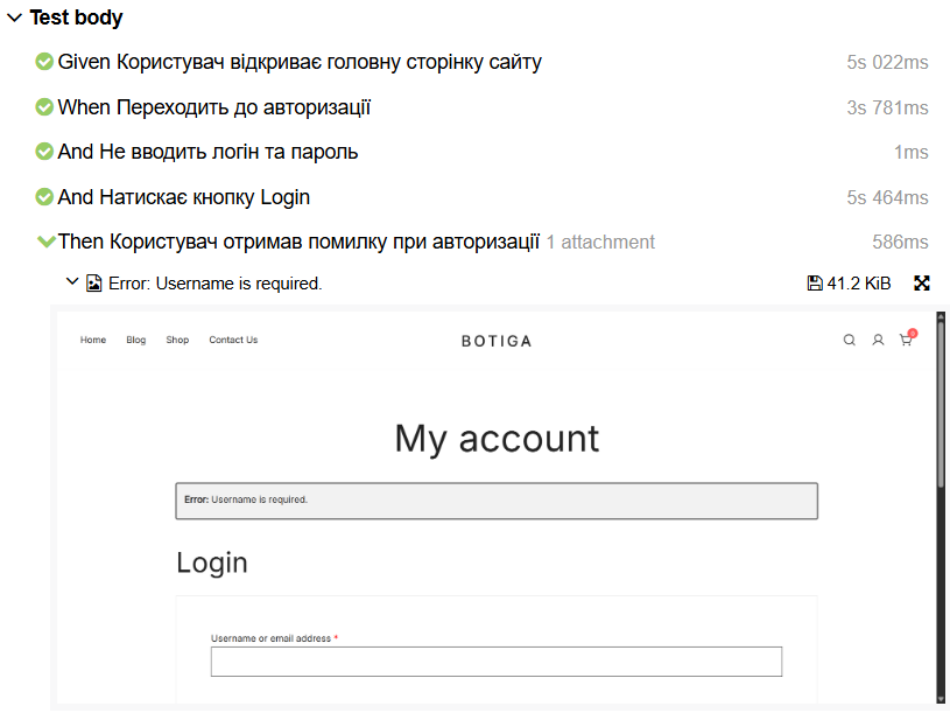

Рисунок 3.4 – Авторизація з порожніми полями

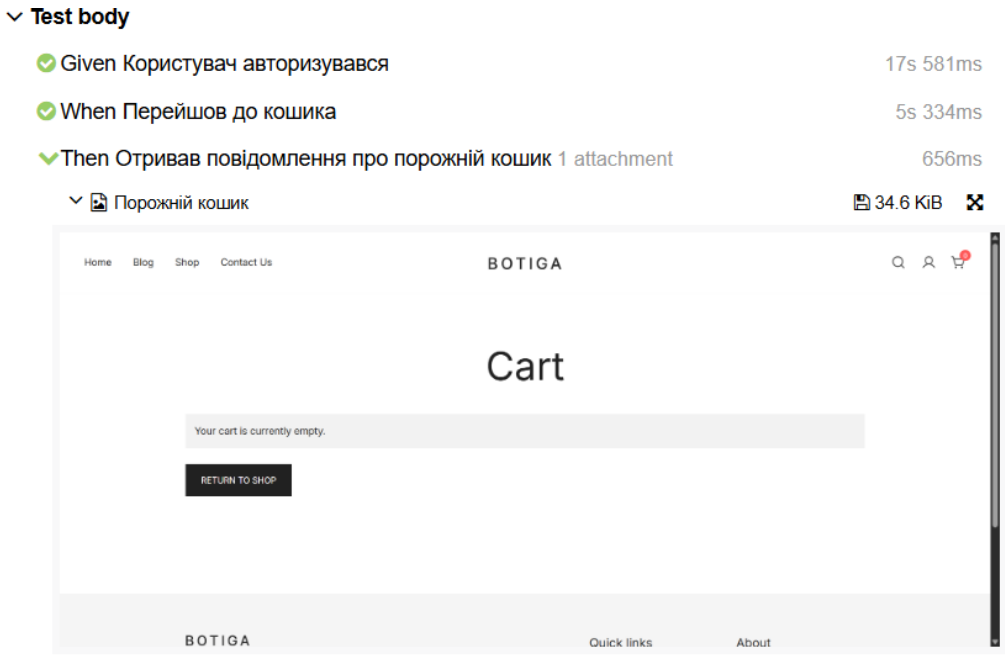

Рисунок 3.5 – Порожній кошик

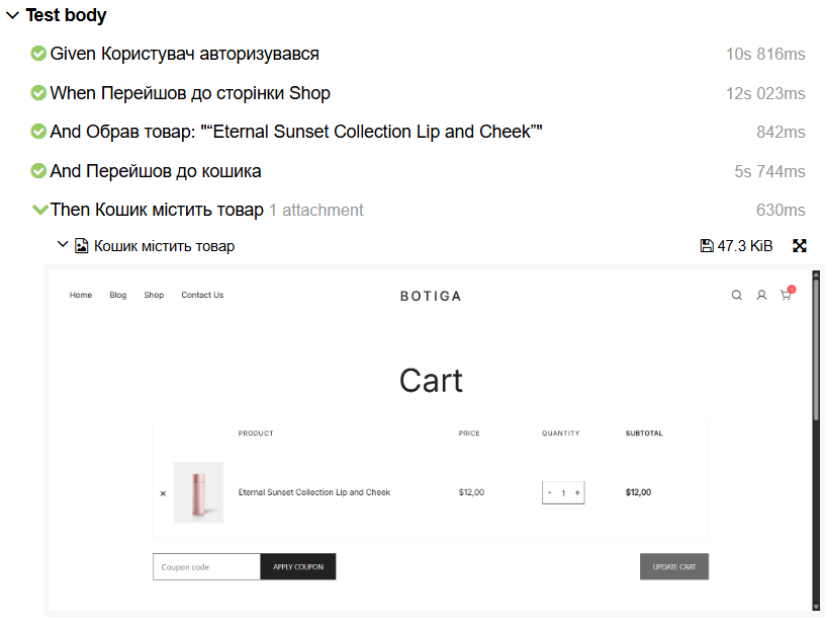

Рисунок 3.6 – Додавання товару в кошик

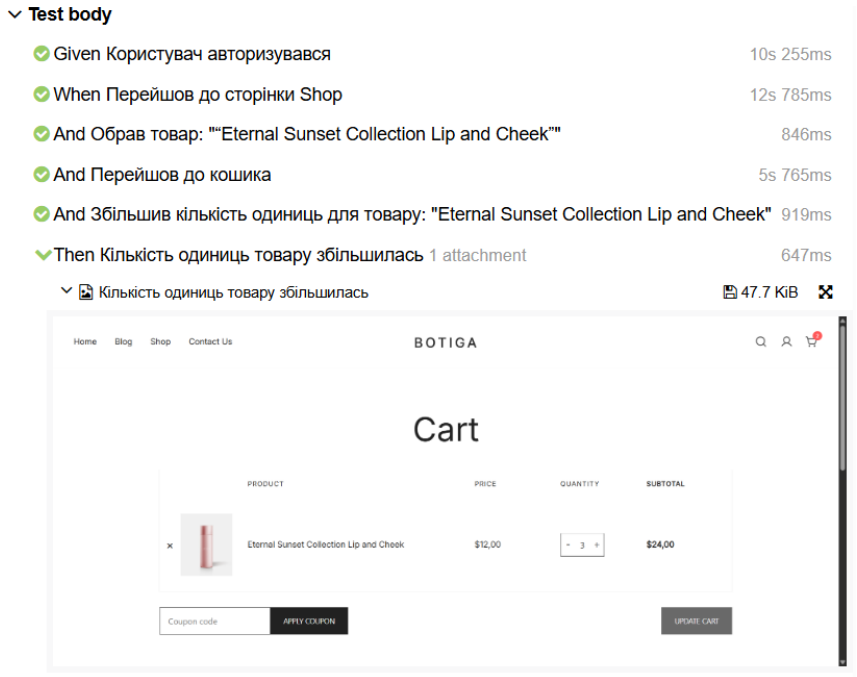

Рисунок 3.7 – Збільшення кількості одиниць товару

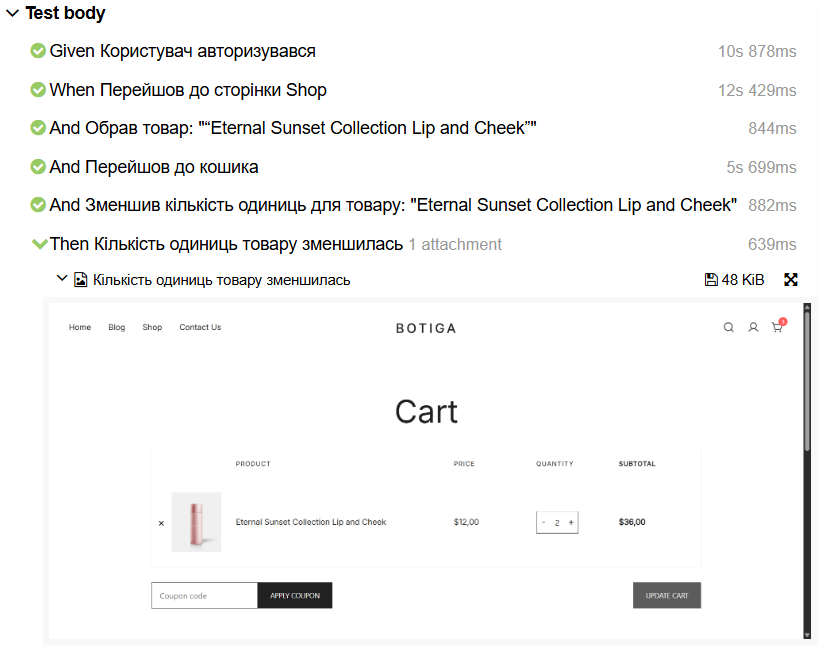

Рисунок 3.8 – Зменшення кількості одиниць товару

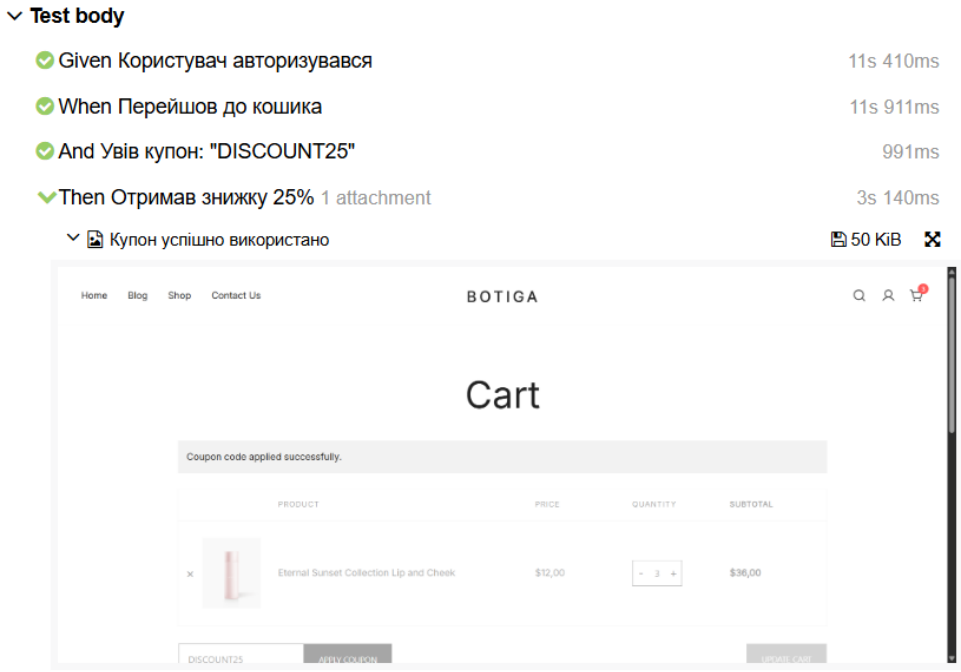

Рисунок 3.9 – Активація купону на знижку

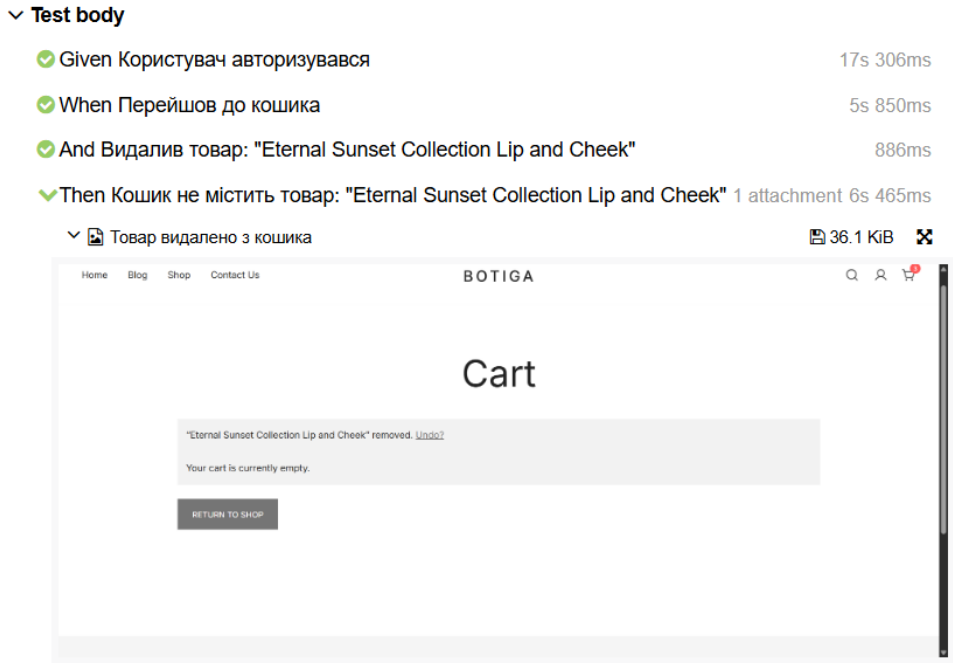

Рисунок 3.10 – Видалення товару з кошику

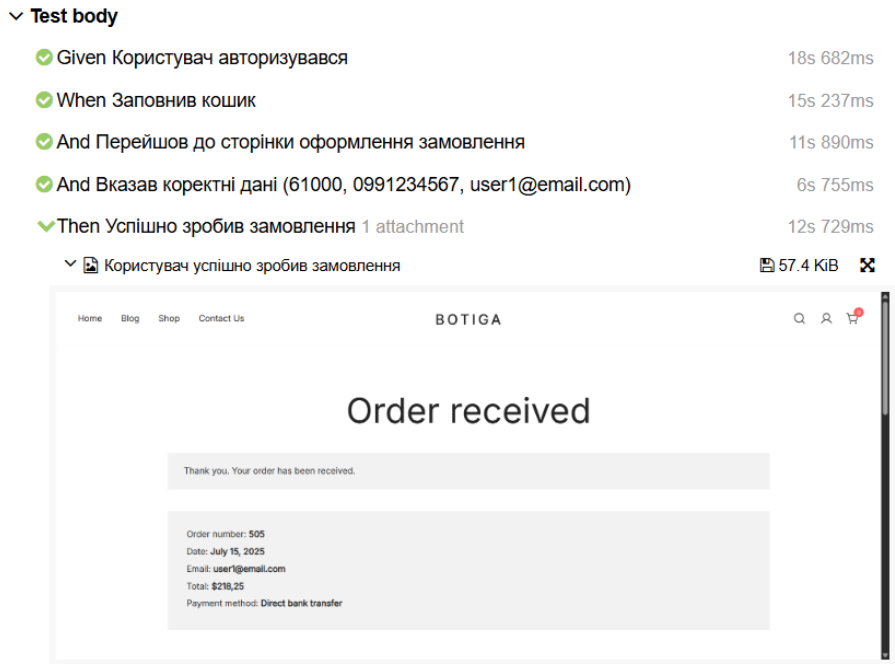

Рисунок 3.11 – Оформлення замовлення з коректними даними

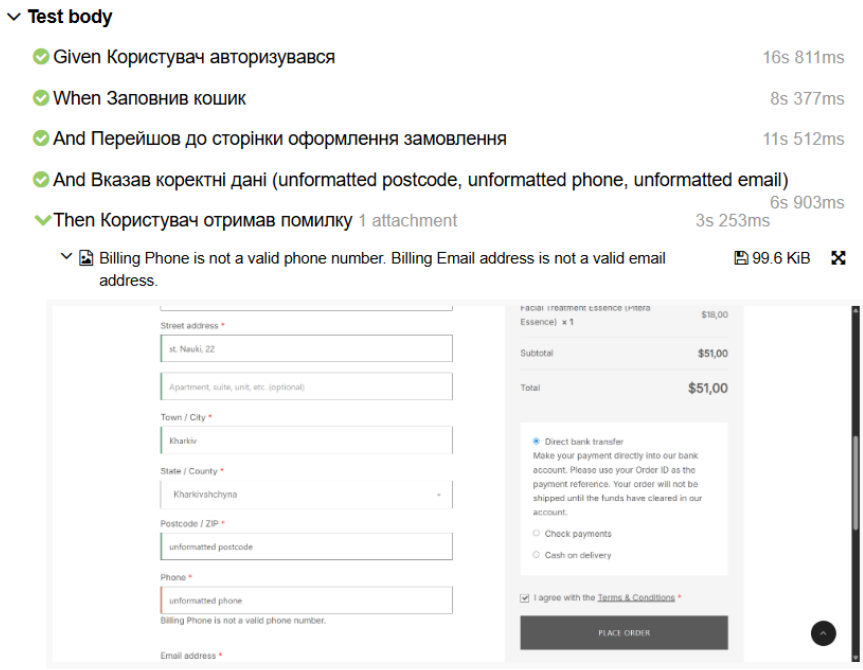

Рисунок 3.12 – Оформлення замовлення з хибними даними

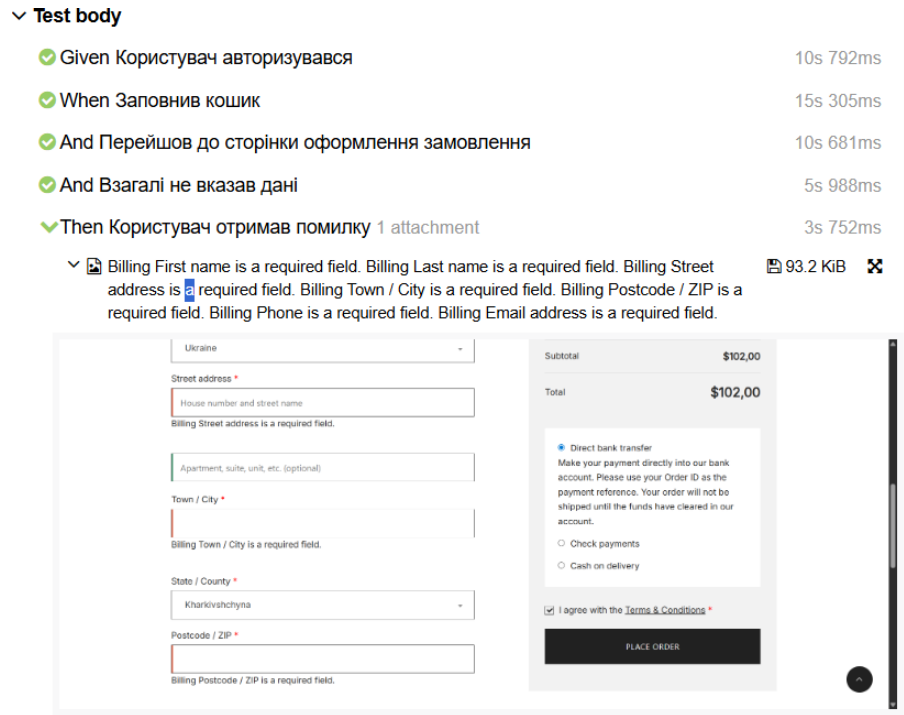

Рисунок 3.13 – Оформлення замовлення з порожніми полями

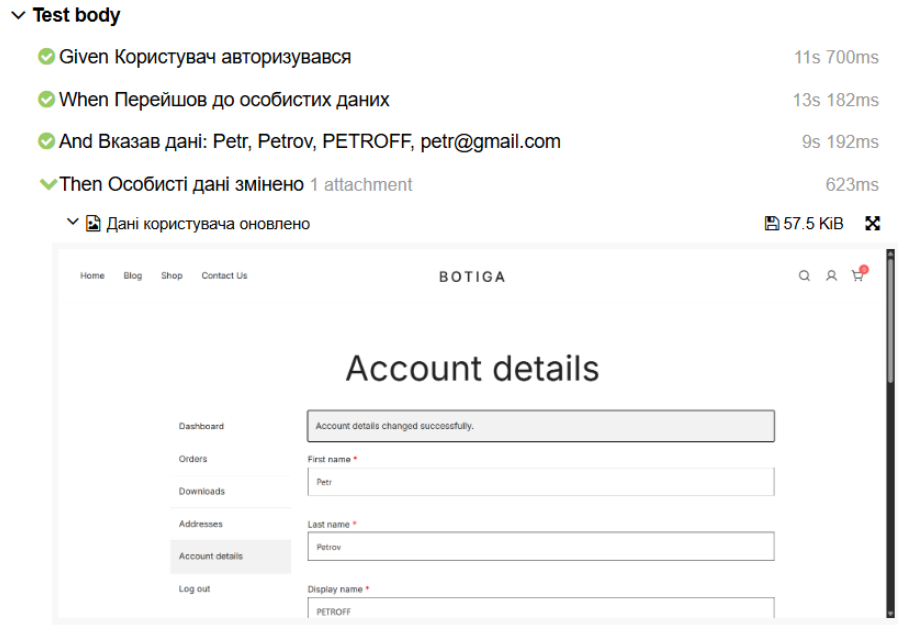

Рисунок 3.14 – Успішна зміна особистих даних

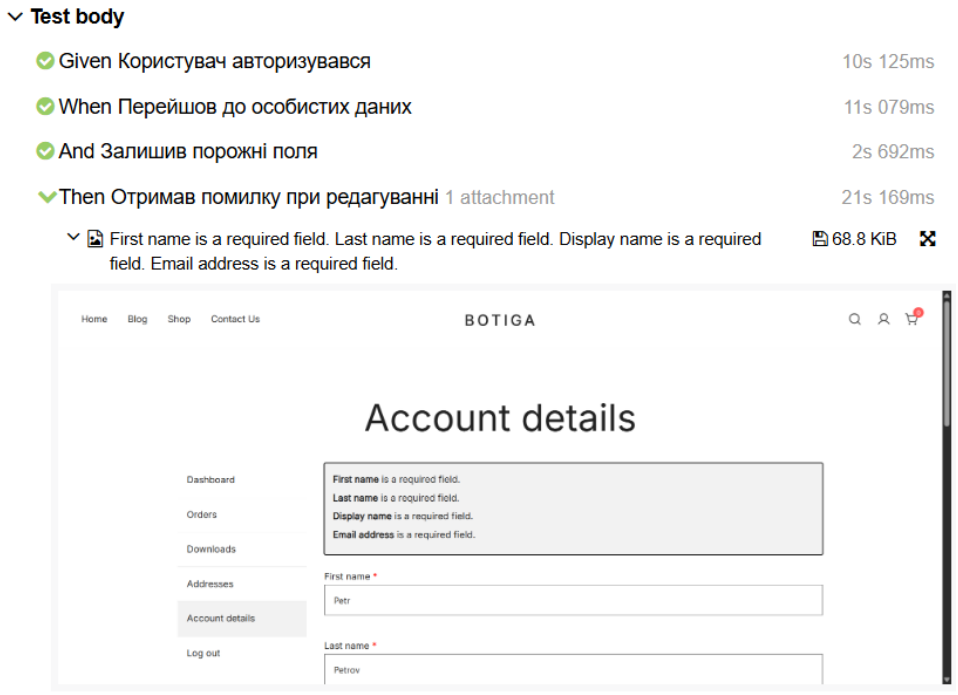

Рисунок 3.15 – Хибна зміна особистих даних

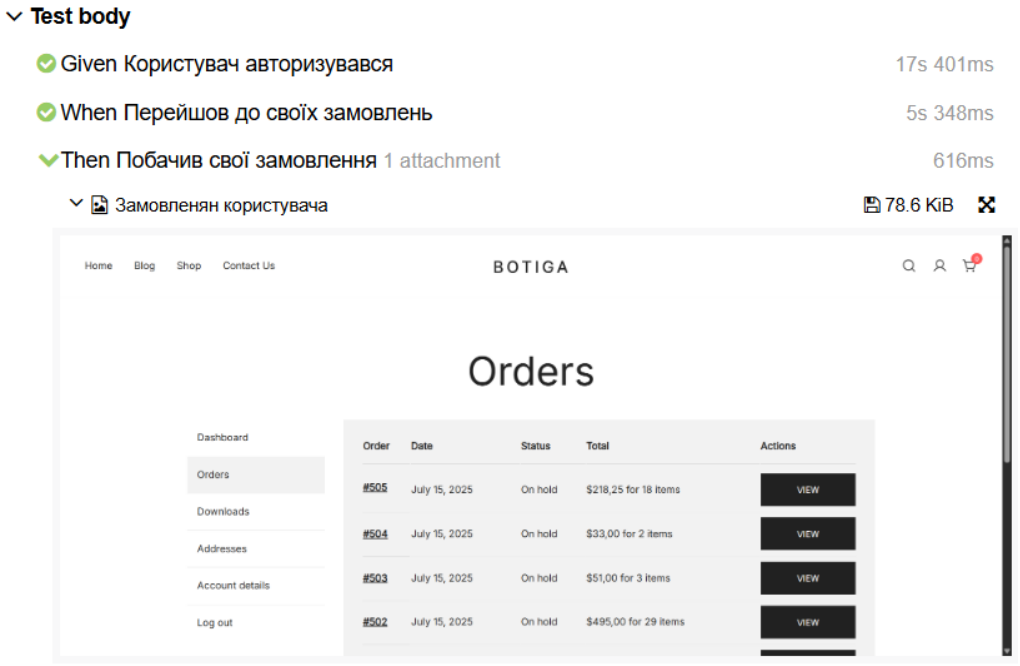

Рисунок 3.16 – Перевірка наявності замовлень

3.7 Аналіз ефективності (ручне та автоматизоване тестування)

Об’єктом аналізу ефективності є повний набір із 16 сценаріїв, які покривають ключовий функціонал вебзастосунку: авторизацію, роботу з кошиком, оформлення замовлення та редагування профілю. Протокол результатів ручного тестування подано у вигляді таблиці 3.11, а перелік тест-кейсів – у таблицях 3.7–3.10 Аналіз результатів ручного та автоматизованого тестування наведено у таблиці 3.12. Застосування автоматизованого тестування дозволило скоротити час на виконання повного регресійного тестового набору на 87,5% у порівнянні з ручним виконанням. Це є прямим кількісним доказом підвищення оперативної ефективності (Performance Efficiency) процесу тестування, особливо для цілої регресії.

Показник ефективності напряму залежить від технічних характеристик клієнтської та серверної складової. У випадку з тестуванням вебдодатку встановленому на власному локальному середовищі, слід ураховувати зменшення ефективності пристрою (ноутбуку), а значить збільшення часу на виконання автотестів.

Таблиця 3.7 – Авторизація користувача

№ ТК	Назва тест-кейсу	Передумови	Кроки	Очікуваний результат
СНК-001	Оформлення замовлення з коректними даними	Користувач авторизувався.	Заповнив кошик, перейшов до сторінки оформлення замовлення, вказав коректні дані (61000, 0991234567, user1@email.com).	Успішно зробив замовлення.

Кінець таблиці 3.7

№ ТК	Назва тест-кейсу	Передумови	Кроки	Очікуваний результат
CHK-002	Оформлення замовлення з хибними даними	Користувач авторизувався.	Заповнив кошик, перейшов до сторінки оформлення замовлення, вказав некоректні дані (unformatted postcode, unformatted phone, unformatted email).	Користувач отримав помилку.
CHK-003	Оформлення замовлення з порожніми полями	Користувач авторизувався.	Заповнив кошик, перейшов до сторінки оформлення замовлення, взагалі не вказав дані.	Користувач отримав помилку.

Таблиця 3.8 – Кошик користувача

№ ТК	Назва тест-кейсу	Передумови	Кроки	Очікуваний результат
CART-001	Порожній кошик	Користувач авторизувався.	Перейшов до кошика.	Отримав повідомлення про порожній кошик.
CART-002	Додавання товару в кошик	Користувач авторизувався.	Перейшов до сторінки Shop, обрав товар: ""“Eternal Sunset Collection Lip and Cheek”"", перейшов до кошика.	Кошик містить товар.

Кінець таблиці 3.8

№ ТК	Назва тест-кейсу	Передумови	Кроки	Очікуваний результат
CART-003	Збільшення кількості одиниць товару	Користувач авторизувався.	Перейшов до сторінки Shop, обрав товар: "“Eternal Sunset Collection Lip and Cheek”", перейшов до кошика, збільшив кількість одиниць для товару: "Eternal Sunset Collection Lip and Cheek".	Кількість одиниць товару збільшилась.
CART-004	Зменшення кількості одиниць товару	Користувач авторизувався.	Перейшов до сторінки Shop, обрав товар: "“Eternal Sunset Collection Lip and Cheek”", перейшов до кошика, зменшив кількість одиниць для товару: "Eternal Sunset Collection Lip and Cheek".	Кількість одиниць товару зменшилась.
CART-005	Активація купона на знижку у кошику	Користувач авторизувався.	Перейшов до кошика, увів купон: "DISCOUNT25".	Отримав знижку 25%.
CART-006	Видалення товару з кошика	Користувач авторизувався.	Перейшов до кошика, видалив товар: "Eternal Sunset Collection Lip and Cheek".	Кошик не містить товар: "Eternal Sunset Collection Lip and Cheek".

Таблиця 3.9 – Оформлення замовлень

№ ТК	Назва тест-кейсу	Передумови	Кроки	Очікуваний результат
CHK-001	Оформлення замовлення з коректними даними	Користувач авторизувався.	Заповнив кошик, перейшов до сторінки оформлення замовлення, вказав коректні дані (61000, 0991234567, user1@email.com).	Успішно зробив замовлення.
CHK-002	Оформлення замовлення з хибними даними	Користувач авторизувався.	Заповнив кошик, перейшов до сторінки оформлення замовлення, вказав некоректні дані (unformatted postcode, unformatted phone, unformatted email).	Користувач отримав помилку.
CHK-003	Оформлення замовлення з порожніми полями	Користувач авторизувався.	Заповнив кошик, перейшов до сторінки оформлення замовлення, взагалі не вказав дані.	Користувач отримав помилку.

Таблиця 3.10 – Особистий профіль

№ ТК	Назва тест-кейсу	Передумови	Кроки	Очікуваний результат
PROF-001	Успішна зміна особистих даних	Користувач авторизувався.	Перейшов до особистих даних, вказав дані: Petr, Petrov, PETROFF, petr@gmail.com.	Особисті дані змінено.

Кінець таблиці 3.10

№ ТК	Назва тест-кейсу	Передумови	Кроки	Очікуваний результат
PROF-002	Хибна зміна особистих даних	Користувач авторизувався.	Перейшов до особистих даних, залишив порожні поля.	Отримав помилку при редагуванні.
PROF-003	Перевірка наявності замовлень	Користувач авторизувався.	Перейшов до своїх замовлень.	Побачив свої замовлення.

Таблиця 3.11 – Протокол результатів ручного тестування

№ ТК	Назва тест-кейсу	Фактичний результат	Статус
1	Успішна авторизація	Відображається сторінка особистого профілю.	Passed
2	Авторизація з хибним паролем	Користувач отримав помилку при авторизації.	Passed
3	Авторизація з хибним логіном	Користувач отримав помилку при авторизації.	Passed
4	Авторизація з порожніми полями	Користувач отримав помилку при авторизації.	Passed
5	Порожній кошик	Отримав повідомлення про порожній кошик.	Passed
6	Додавання товару в кошик	Кошик містить товар.	Passed
7	Збільшення кількості одиниць товару	Кількість одиниць товару збільшилась.	Passed
8	Зменшення кількості одиниць товару	Кількість одиниць товару зменшилась.	Passed

Кінець таблиці 3.11

№ ТК	Назва тест-кейсу	Фактичний результат	Статус
9	Активація купона на знижку у кошику	Отримав знижку 25%.	Passed
10	Видалення товару з кошика	Кошик не містить товар: "Eternal Sunset Collection Lip and Cheek".	Passed
11	Оформлення замовлення з коректними даними	Успішно зробив замовлення.	Passed
12	Оформлення замовлення з хибними даними	Користувач отримав помилку.	Passed
13	Оформлення замовлення з порожніми полями	Користувач отримав помилку.	Passed
14	Успішна зміна особистих даних	Особисті дані змінено.	Passed
15	Хибна зміна особистих даних	Отримав помилку при редагуванні.	Passed
16	Перевірка наявності замовлень	Побачив свої замовлення.	Passed

Таблиця 3.12 – Порівняння часу виконання повного тестового набору

Параметр	Оцінка ручного виконання	Фактичне автоматизоване виконання (Allure)	Скорочення, хв. сек.	Скорочення, %
Середній час на 1 сценарій	4 хв.	~15 сек.	3 хв. 45сек.	83,75%
Загальний час на 1 прогін (16 сценаріїв)	64 хв. (1 год. 4 хв.)	~8 хв.	56 хв.	87,5%

Кінець таблиці 3.12

Параметр	Оцінка ручного виконання	Фактичне автоматизоване виконання (Allure)	Скорочення, хв. сек.	Скорочення, %
Загальний час на 10 прогонів (Регресія)	640 хв. (10 год. 40 хв.)	~80 хв.	9 год. 20 хв.	87,5%
Вартість, USD	$65	$8,45	-	86,89%

Ефективність тестування вимірюється не лише часом, але й надійністю та точністю результатів. Порівняння ручного та автоматизованого тестування подано у таблиці 3.13.

Таблиця 3.13 – Порівняння ручного та автоматизованого тестування

Параметр	Ручне тестування	Автоматизоване тестування (BDD)	Перевага автоматизації
Точність виконання	Схильність до людських помилок, втоми, пропусків кроків.	Висока (0% помилок виконання за умови коректного коду).	Виняткова відтворюваність та точність.
Об’єктивність звітності	Суб’єктивні записи, залежність від фахівця.	Об’єктивні звіти Allure з хронометражем, логами та скріншотами (Рисунки 3.1–3.16).	Стандартизація та візуалізація звіту.

Кінець таблиці 3.13

Параметр	Ручне тестування	Автоматизоване тестування (BDD)	Перевага автоматизації
Покриття (Code Coverage)	Неможливо гарантувати повне покриття без додаткових інструментів.	Забезпечується фокус на покритті бізнес-поведінки (BDD).	Чітке покриття критичної функціональності.
ROI (Return on Investment)	Низький для часто повторюваних тестів.	Високий, особливо на етапі регресії та CI/CD.	Економія ресурсів на тривалій перспективі.

Автоматизоване тестування на базі BDD-сценаріїв мінімізує вплив людського фактора, забезпечуючи 100% відтворюваність та об’єктивну звітність Allure, що критично важливо для забезпечення якості (Quality Assurance) на етапі підтримки програмного забезпечення.

Дослідження показало, що використання BDD-підходу Behave значно підвищило ефективність. Сценарії Gherkin, які використовувалися в дослідженні, виступали як єдина точка істини щодо очікуваної поведінки системи.

Завдяки розділенню тестового коду на логічний (BDD-сценарій) та технічний (POM-сторінки та кроки) рівні, при зміні UI вебзастосунку не виникає необхідності переписувати бізнес-логіку сценарію. Це підвищує підтримуваність та гнучкість системи автоматизації.

ВИСНОВКИ

У магістерській кваліфікаційній роботі було здійснено всебічний аналіз предметної області тестування програмного забезпечення та існуючих інструментів, що складають основу для розробки сучасної системи забезпечення якості. У ході аналізу було виокремлено основні функції вебзастосунку «Інтернет-магазин», що потребують детального автоматизованого тестування для гарантування його стабільності та надійності. Одними з таких функцій є механізми авторизації, роботи з кошиком та оформлення замовлення, які відіграють важливу роль у забезпеченні коректної взаємодії користувача з системою.

В рамках тестування системи було використано підхід Behavior-Driven Development (BDD), що дозволив створити зрозумілі та читабельні тестові сценарії. Використання автоматизації тестування на основі Python, Selenium WebDriver та фреймворку Behave дало змогу зменшити час виконання великої кількості повторюваних перевірок на 87,5%, підвищити точність отриманих результатів до 100% (виключивши людський фактор) і здешевити витрати на 86,89%. Розроблені автоматизовані сценарії для перевірки функціональності системи в цілому сприяли зниженню кількості людських помилок і підвищенню ефективності тестування на етапах розробки. Для функціональних модулів, які часто змінюються, були створені модульні тести, що дають змогу оперативно виявляти помилки під час інтеграції нових можливостей.

Тестування ключових функцій системи – авторизації та оформлення замовлення – було проведено з метою оцінки коректності обробки даних, дотримання бізнес-логіки та стабільності роботи застосунку. Тестування показало, що система здатна ефективно обробляти запити та забезпечувати правильне функціонування.

Крім того, для візуалізації та аналізу результатів тестування було застосовано систему звітів Allure, яка дозволяє наочно відображати успішність виконання тестів та деталі помилок. Це сприяло швидкому виявленню і виправленню дефектів, що є невід’ємною частиною процесу забезпечення якості.

На основі результатів тестування можна дійти висновку, що використання підходу BDD та обраних інструментів продемонструвало високу ефективність і забезпечило надійність, стабільність та продуктивність вебзастосунку.

Таким чином, підсумком є те, що автоматизоване тестування виступає ключовим елементом процесу розробки, покликаним гарантувати не лише коректну функціональність, а й ефективність та стабільність під час реальної експлуатації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. FoxmindEd: Тестування програмного забезпечення. URL: https://foxminded.ua/testuvannia-prohramnoho-zabezpechennia/ (дата звернення: 11.11.2025)
2. Manifesto for Agile Software Development: Маніфест розробки гнучкого програмного забезпечення. URL: https://agilemanifesto.org/iso/uk/manifesto.html (дата звернення 11.11.2025)
3. QATestLab: Популярні життєві цикли розробки ПЗ. URL: https://training.qatestlab.com/blog/technical-articles/popular-software-development-life-cycles/ (дата звернення: 11.11.2025)
4. QATestLab: Огляд видів тестування. URL: https://training.qatestlab.com/blog/technical-articles/review-the-types-of-testing/ (дата звернення 12.11.2025)
5. QaLight: White/black/grey box-тестування. URL: https://qalight.ua/baza-znaniy/white-black-grey-box-testuvannya/ (дата звернення 12.11.2025)
6. QATestLab: Огляд видів тестування. URL: https://training.qatestlab.com/blog/technical-articles/review-the-types-of-testing/ (дата звернення 12.11.2025)
7. Wikipedia: Автоматизоване тестування. URL: https://uk.wikipedia.org/wiki/Автоматизоване_тестування (дата звернення 12.11.2025)
8. W3C. Document Object Model. URL: https://www.w3.org/DOM/Overview (дата звернення 13.11.2025)
9. QaLight: Автоматизоване тестування. URL: https://qalight.ua/baza-znaniy/avtomatizovane-testuvannya/ (дата звернення 13.11.2025)
10. Selenium: Selenium Documentation. URL: https://www.selenium.dev/documentation/ (дата звернення 13.11.2025)

11. Behave: behave 1.4.0.dev0 documentation. URL: https://behave.readthedocs.io/en/latest/ (дата звернення 13.11.2025)
12. Allure: Introduction. URL: https://allurereport.org/docs/ (дата звернення 14.11.2025)
13. HOSTiQ: Що таке WordPress та як їм користуватись. URL: https://hostiq.ua/wiki/ukr/wordpress-review/ (дата звернення 14.11.2025)
14. WordPress.org: About. URL: https://wordpress.org/about/ (дата звернення 14.11.2025)
15. Wikipedia: XAMPP. URL: https://en.wikipedia.org/wiki/XAMPP (дата звернення 15.11.2025)
16. Wikipedia: Коміт (керування версіями). URL: https://uk.wikipedia.org/wiki/Коміт_(керування_версіями) (дата звернення 15.11.2025)
17. aCode: Декоратори Python. URL: https://acode.com.ua/decorators-python/ (дата звернення 15.11.2025)
18. Білоус Д. Ю., Дегтярьов О. В. Автоматизоване тестування вебзастосунків на основі підходу Behavior-Driven Development // Розвиток сучасної науки: актуальні питання теорії та практики : матеріали IX Всеукр. студент. наук. конф. (Київ, 21 листоп. 2025 р.). – Київ : Ін-т науково-технічної інтеграції та співпраці, 2025. – С. 479–480. – DOI: 10.62732/liga-ukr-21.11.2025. – ISBN 978-617-8582-04-3.