**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ**
**ім. Ю.М. Потебні**
**ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ**

**КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

# Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему **Використання технології Page-Object Model для підвищення ефективності тестування Веб-застосунків**

Виконав: студент 2 курсу, групи 8.1213-іпз-2
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
М. О. Бойко
(ініціали та прізвище )
Керівник доцент, доцент, к.ф.-м.н. Г.П. Коломоєць
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ Дісітел
П.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2024

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ**
**ім. Ю.М. Потебні**
**ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ**

Кафедра електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти _____другий (магістерський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Програмне забезпечення систем
(код та назва)

**ЗАТВЕРДЖУЮ**

Завідувач кафедри_____Т. В. Критська
"_02_" _вересня__2024_року

**З А В Д А Н Н Я**
**НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ**

Бойку Миколі Олександровичу
(прізвище, ім'я, по батькові)

1.Тема роботи Використання технології Page-Object Model для підвищення ефективності тестування Веб-застосунків
керівник роботи Коломоєць Геннадій Павлович, к.ф.-м.н. доцент_
( прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від ___28.02.2024р._№_361-е

2. Строк подання студентом кваліфікаційної роботи_____
3. Вихідні дані бакалаврської роботи:
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз засобів та підходів до автоматизації тестування Веб-застосунків;
- Розробка тест-вимог для демонстраційного Веб-застосунку та реалізація проєкту з Selenium Web Driver;
- Реалізація проєкту з POM та Selenium Web Driver;
- Виконання порівняльного аналізу проєктів з автоматизованими тестами без та з використанням POM.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
слайдів презентації

6. Консультанти розділів бакалаврської роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата Завдання прийняв |
|---|---|---|
| | | |

7. Дата видачі завдання 02.09.2024 р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів магістерської роботи | Строк виконання етапів магістерської роботи | Примітка |
|---|---|---|---|
| 1 | Аналіз предметної області | 02.09-10.09.2024 | виконано |
| 2 | Аналіз існуючих методів автоматизованого тестування веб-застосунків | 11.09-12.09.2024 | виконано |
| 3 | Дослідження сучасних засобів автоматизованого тестування | 15.09-20.09.24 | виконано |
| 4 | Узгодження подальших дій з науковим керівником | 21.09-26.09.24 | виконано |
| 5 | Проєктування архітектури тестів | 27.09-28.09.24 | виконано |
| 6 | Розробка та впровадження тестів для веб-застосунку за допомогою Selenium WebDriver | 29.09-13.10.24 | виконано |
| 7 | Рефакторинг і оптимізація тестів для забезпечення масштабованості та підтримки | 14.10-16.10.24 | виконано |
| 8 | Представлення отриманих результатів науковому керівнику та узгодження плану подальшого дослідження | 17.10-19.10.24 | виконано |
| 9 | Перевірка роботоздатності автоматизованих тестів | 20.10-09.11.24 | виконано |
| 10 | Оформлення звіту | 10.11-17.11.23 | виконано |
| 11 | Оформлення презентації. | 18.11-19.11.23 | виконано |

Студент _____ М.О. Бойко
( підпис ) (прізвище та ініціали)

Керівник роботи _____ Г.П. Коломоєць
( підпис ) (прізвище та ініціали)

**Нормоконтроль пройдено**

Нормоконтролер _____ І.А. Скрипник
( підпис ) (прізвище та ініціали)

# АНОТАЦІЯ

Сторінок – 86

Рисунків – 16

Джерел – 23

Таблиць – 3

Бойко М.О. Використання технології Page-Object Model для підвищення ефективності тестування Веб-застосунків: кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник Коломоєць Г.П. Запоріжжя : ЗНУ, 2024. 86 с.

Мета полягає у дослідженні та оцінці використання технології Page-Object Model з метою підвищення ефективності процесу тестування Веб-застосунків. Робота спрямована на вивчення основних принципів і переваг Page-Object Model, розробку оптимального сценарію застосування цієї технології для автоматизації тестування, а також порівняння її ефективності з іншими підходами до автоматизації тестів веб-застосунків. Результати цього дослідження дозволять визначити потенціал технології Page-Object Model у покращенні якості та швидкості процесу тестування веб-застосунків і вказати на переваги її використання для оптимізації розробки програмного забезпечення.

У процесі дослідження було розглянуто загальну характеристику засобів та підходів до автоматизації тестування Веб-застосунків, були розроблені тест-вимоги для демонстраційного Веб-застосунку, розглянуто шаблон проєктування Page Object Model, його ідея та переваги використання. Виконано порівняльний аналіз проєктів з автоматизованими тестами без та з використанням Page Object Model.

Ключові слова: тестування, Веб-застосунок, Page Object Model, Selenium WebDriver, PageFactory.

# ABSTRACT

Pages – 86

Drawings – 16

Sources – 23

Tables – 3

Boiko M.O. Using Page-Object Model technology to improve the efficiency of testing Web applications: master's thesis in speciality 121 ‘Software Engineering’ / scientific adviser Kolomoyets H.P. Zaporizhzhia: ZNU, 2024. 86 p.

The aim is to study and evaluate the use of Page-Object Model technology to improve the efficiency of the Web application testing process. The work is aimed at studying the basic principles and advantages of the Page-Object Model, developing an optimal scenario for using this technology to automate testing, as well as comparing its effectiveness with other approaches to automating web application tests. The results of this study will allow us to determine the potential of Page-Object Model technology in improving the quality and speed of the web application testing process and point out the benefits of using it to optimise software development.

In the course of the study, the general characteristics of tools and approaches to automating web application testing were considered, test requirements for a demonstration web application were developed, and the Page Object Model design template, its idea, and the benefits of using it were considered. A comparative analysis of projects with automated tests without and with the use of the Page Object Model was performed.

Keywords: testing, Web application, Page Object Model, Selenium WebDriver, PageFactory.

# ЗМІСТ

ВСТУП..... 8

1 ТЕХНОЛОГІЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКІВ SELENIUM..... 13

1.1 Загальна характеристика засобів та підходів до автоматизації тестування веб-застосунків ..... 13

1.1.1 Основи автоматизації тестування: концепції, цілі та переваги13

1.1.2 Різновиди автоматизованого тестування..... 13

1.1.3 Популярні інструменти автоматизації тестування ..... 14

1.2 Характеристика фреймворку Selenium WebDriver ..... 15

1.2.1 Історія створення Selenium та його місце серед інструментів автоматизації 15

1.2.2 Архітектура Selenium WebDriver ..... 15

1.2.3 Підтримувані мови програмування та інтеграція з іншими інструментами..... 17

1.2.4 Переваги та обмеження Selenium WebDriver у порівнянні з альтернативами..... 17

1.3 Способи ідентифікації Веб-елементів..... 18

1.4 Методи WebDriver для пошуку елементів та для взаємодії з Веб-елементами 19

1.4.1 Методи для пошуку веб-елементів ..... 19

1.4.2 Методи для взаємодії з веб-елементами..... 20

1.5 Стратегії очікування Веб-елементів..... 21

1.5.1 Типи очікувань у Selenium..... 21

1.5.2 Основні умови для явного очікування..... 22

2 ПРОЄКТУВАННЯ АВТОМАТИЗОВАНИХ ТЕСТІВ ДЛЯ ВЕБ-ЗАСТОСУНКУ ..... 24

2.1 Опис функціоналу веб-застосунку ..... 24

2.2 Розробка тест-вимог для веб-застосунку ..... 29

2.3 Реалізація проєкту з Selenium Web Driver ..... 32

3 ОРГАНІЗАЦІЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКУ ІЗ ВИКОРИСТАННЯМ PAGE-OBJECT MODEL ..... 46

3.1 Шаблон проєктування Page Object Model (POM): ідея та переваги використання ..... 46

3.2 Основні принципи Page Object Model ..... 48

3.3 Проєктування архітектури тестів із використанням Page Object Model ... 49

3.4 Реалізація проєкту з POM та Selenium WebDriver автоматизованими тестами для демонстраційного Веб-застосунку ..... 50

4 ВИКОРИСТАННЯ PAGE FACTORY ПРИ ОРГАНІЗАЦІЇ ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКУ ..... 70

4.1 Використання класу PageFactory фреймворку Selenium WebDriver для підтримки автоматизованих тестів, побудованих із використанням POM ..... 70

4.2 Рефакторинг проєкту з POM та Selenium WebDriver за допомогою PageFactory ..... 71

4.3 Виконання порівняльного аналізу проєктів з автоматизованими тестами без та з використанням POM ..... 74

ВИСНОВКИ ..... 83

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ..... 84

# ВСТУП

**Актуальність теми**

У сучасному світі веб – застосунки відіграють важливу роль у багатьох аспектах життя людей. Вони використовуються для онлайн – банкінгу, онлайн – магазинів, спілкування, навчання та розваг. Тому актуальність тестування веб-застосунків неможливо переоцінити.

Ось деякі з причин, чому тестування веб-застосунків є таким важливим:

1. Забезпечення якості: Тестування допомагає виявити та виправити помилки в веб – застосунку, що робить його більш надійним і стійким до збоїв. Це важливо для забезпечення позитивного досвіду користувачів та запобігання втраті даних або фінансових втрат.
2. Підвищення безпеки: Тестування може допомогти виявити вразливості в веб – застосунку, які можуть бути використані зловмисниками для крадіжки даних або заподіяння шкоди. Це важливо для захисту конфіденційності користувачів та запобігання кібератак.
3. Збільшення зручності використання: Тестування може допомогти зробити веб-застосунок більш зручним для користувачів, гарантуючи, що він простий у навігації та використанні. Це дозволяє залучити більше користувачів.
4. Зниження витрат: Виявлення та виправлення помилок на ранніх стадіях розробки може значно заощадити час і гроші. Тестування також може допомогти запобігти виправленням помилок після запуску.

Окрім вищезазначеного, тестування веб-застосунків також може допомогти:

1. Поліпшити продуктивність веб-застосунку.
2. Забезпечити сумісність веб-застосунку з різними браузерами та пристроями.
3. Збільшити масштабованість веб-застосунку.
4. Зібрати відгуки користувачів про веб-застосунок.

Існує багато різних методів тестування веб-застосунків, і вибір відповідного методу буде залежати від конкретного веб-застосунку та його цілей.

Тестування веб-застосунків – це постійний процес, який слід проводити протягом усього життєвого циклу веб-застосунку. Це допоможе гарантувати, що веб-застосунок залишається надійним, безпечним, зручним для користувачів і відповідає всім вимогам.

**Мета дослідження**

Мета дослідження полягає в дослідженні і визначенні ефективності використання технології Page-Object Model (POM) для автоматизації тестування веб-застосунків.

**Завдання дослідження**

Аналіз існуючих засобів та підходів до автоматизації тестування Веб-застосунків. Огляд технології Page-Object Model, розробка тест-вимог для демонстраційного Веб-застосунку, реалізація проєкту з Selenium WebDriver та Page-Object Model. Виконання порівняльного аналізу проєктів з автоматизованими тестами без та з використанням Page-Object Model.

**Об’єкт дослідження**

Об'єктом дослідження є веб-застосунок Sauce Demo – демонстраційний сайт, призначений для тестування функціоналу автоматизованих систем та інструментів тестування. Sauce Demo надає можливість перевіряти різні аспекти веб-застосунків, зокрема авторизацію, навігацію, роботу з кошиком, оформлення замовлень та відображення даних.

Цей веб-застосунок доступний для використання на смартфонах, планшетах та десктопних пристроях, що дозволяє тестувальникам і розробникам працювати з ним незалежно від платформи. Sauce Demo слугує ідеальним

середовищем для вивчення та впровадження автоматизації тестування завдяки наявності простих і зрозумілих сценаріїв взаємодії.

**Предмет дослідження**

Предметом дослідження є технологія Page-Object Model, дослідження ефективності та переваг використання цієї технології для підвищення ефективності тестування Веб-застосунків.

**Методи дослідження**

Проведення огляду літератури щодо технологій автоматизованого тестування веб-застосунків, зокрема технології Page-Object Model. Налаштування середовища для тестування веб-застосунків з використанням технології Page-Object Model. Вибір веб-застосунку для проведення експериментального тестування. Розробка тестових сценаріїв на основі Page-Object Model для вибраного веб-застосунку. Виконання розроблених тестових сценаріїв з використанням технології Page-Object Model. Збір результатів тестування і оцінка ефективності та надійності автоматизованих тестів. Формулювання висновків щодо ефективності технології Page-Object Model для підвищення ефективності тестування веб-застосунків.

**Практичне значення одержаних результатів**

Практичне значення одержаних результатів цієї дослідницької роботи полягає в розробці та впровадженні ефективних стратегій автоматизованого тестування веб-застосунків з використанням технології Page-Object Model (POM). Отримані результати надають певні практичні переваги:

1. Покращення ефективності тестування: Розроблені методики та тестові сценарії на основі POM дозволяють підвищити швидкість та точність тестування веб-застосунків. Це сприяє вчасному виявленню помилок та забезпечує високу якість програмного забезпечення.

2. Зменшення трудомісткості тестування: Використання POM дозволяє створювати структуровані тестові скрипти, що спрощує підтримку і розширення тестової бази. Це зменшує витрати на тестування у майбутньому.
3. Підвищення якості веб-застосунків: Автоматизоване тестування з використанням POM дозволяє виявляти помилки та проблеми швидше і ефективніше, що призводить до покращення якості веб-застосунків.

Отримані результати цієї роботи є актуальними та практично цінними для професіоналів в галузі розробки програмного забезпечення та тестування, оскільки вони дозволять вдосконалити процеси тестування веб-застосунків і підвищити якість програмного забезпечення з урахуванням сучасних технологій автоматизації тестування.

**Апробація результатів**

Робота була апробована на конференції студентів, аспірантів, докторантів і молодих вчених «Молода наука-2023» / Запорізького національного університету [21].

Робота була апробована на конференції студентів, аспірантів, докторантів і молодих вчених «Молода наука-2024» / Запорізького національного університету [22].

**Глосарій**

*Page-Object Model* — паттерн проєктування у розробці автоматизованих тестів, який передбачає створення окремих класів або об'єктів для представлення сторінок або компонентів веб-застосунку. Його мета – ізолювати логіку взаємодії з інтерфейсом користувача (UI) від тестового сценарію, що підвищує читабельність, повторне використання коду і простоту підтримки.

*Веб-застосунок* — програмний продукт, який функціонує через веб – браузер і забезпечує інтерактивний доступ до інформаційних ресурсів, сервісів або функцій.

*Автоматизація тестування* — процес використання спеціальних програмних інструментів для виконання тестів, спрямований на зменшення ручної роботи тестувальника, підвищення швидкості й точності тестування.

*Тестовий сценарій* — послідовність дій, яку необхідно виконати для перевірки певної функціональності або характеристики програмного забезпечення.

*Локатор* — ідентифікатор веб-елемента, що використовується для його пошуку у DOM.

*DOM* — структура HTML-документа, яка представляє його як дерево об'єктів. Через DOM тестувальники взаємодіють з веб-елементами за допомогою локаторів.

*Selenium WebDriver* — бібліотека для автоматизації веб-застосунків. Вона дозволяє розробникам тестувати веб-застосунки, взаємодіючи з ними на рівні браузера.

*Веб-елемент* — це елемент на веб-сторінці, з яким взаємодіє користувач або автоматизований тест. Веб-елементи можуть бути різних типів, зокрема кнопки, текстові поля, посилання, випадаючі списки тощо.

*PageFactory* — це дизайн-патерн у автоматизації тестування, який використовується в рамках моделі Page Object Model для створення та управління об'єктами веб-сторінок. Page Factory полегшує ініціалізацію елементів сторінки, використовуючи анотації, які допомагають знаходити та взаємодіяти з елементами DOM.

# 1 ТЕХНОЛОГІЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКІВ SELENIUM

## 1.1 Загальна характеристика засобів та підходів до автоматизації тестування веб-застосунків

### 1.1.1 Основи автоматизації тестування: концепції, цілі та переваги

Автоматизація тестування веб-застосунків — це процес використання програмних інструментів для виконання тестів з метою перевірки функціональності, продуктивності, безпеки та інших характеристик системи. Її основними завданнями є зменшення витрат часу, підвищення точності тестів і забезпечення стабільності в умовах швидких змін у розробці [1].

Автоматизація дозволяє:

1. Збільшити ефективність тестування, особливо для складних систем з великою функціональністю.
2. Підвищити точність тестування, зменшуючи ризик людських помилок.
3. Забезпечити повторюваність тестів, що важливо для регресійного та інтеграційного тестування.
4. Зекономити ресурси при довготривалому використанні, особливо на етапах стабільної підтримки продукту.

Основна концепція автоматизації полягає у створенні тестів, які відтворюють дії користувачів або виконують тести заздалегідь визначених сценаріїв. Ефективна автоматизація потребує чіткої структури тестів, коректного вибору інструментів і ретельного аналізу обсягів автоматизації.

### 1.1.2 Різновиди автоматизованого тестування

Автоматизація охоплює кілька видів тестування, кожен з яких вирішує певні завдання [2]:

Функціональне тестування: зосереджується на перевірці, чи відповідає веб-застосунок вимогам. Тести автоматизують сценарії, які імітують дії користувача: заповнення форм, перевірка повідомлень про помилки, перехід між сторінками тощо.

Регресійне тестування: перевіряє, чи не порушена існуюча функціональність після внесення змін до коду. Автоматизація цього виду тестування особливо корисна, оскільки дає змогу швидко повторювати тестові сценарії.

Інтеграційне тестування: спрямоване на перевірку взаємодії між різними компонентами системи. Наприклад, тестування зв’язків між фронтендом і бекендом через API.

Навантажувальне тестування: оцінює продуктивність системи під високим навантаженням. Для цього використовуються спеціалізовані інструменти, однак фреймворки типу Selenium також можуть застосовуватися для окремих сценаріїв.

### 1.1.3 Популярні інструменти автоматизації тестування

Сучасний ринок пропонує широкий вибір інструментів для автоматизації веб-тестування, кожен з яких має свої переваги та недоліки:

1. Selenium [3]

– Відкритий інструмент, який підтримує тестування у більшості браузерів.
– Підтримує кілька мов програмування (Java, Python, C#).
– Гнучкий і масштабований, однак вимагає більшого налаштування порівняно з іншими інструментами.

2. Cypress [4]

– Орієнтований на JavaScript-застосунки.
– Інтеграція з сучасними фреймворками (React, Angular, Vue).
– Простіший у налаштуванні, але не підтримує всі браузери, як Selenium.

3. Playwright [5]

– Сучасний інструмент з підтримкою кількох мов програмування.
– Забезпечує кращу продуктивність для тестування багатобраузерних сценаріїв.
– Більш інтуїтивний у використанні порівняно з Selenium.

4. TestCafe [6]

– Простий інструмент для автоматизації, орієнтований на JavaScript.
– Забезпечує вбудоване управління браузерами.
– Не потребує зовнішніх драйверів, як Selenium.

## 1.2 Характеристика фреймворку Selenium WebDriver

### 1.2.1 Історія створення Selenium та його місце серед інструментів автоматизації

Історія створення Selenium та його місце серед інструментів автоматизації

Selenium – це один із найпопулярніших інструментів для автоматизації тестування веб-застосунків. Його розробка розпочалася у 2004 році, коли Джейсон Хаггінс створив першу версію для автоматизації тестування внутрішніх систем [7]. У 2006 році з’явився Selenium WebDriver, який став основою сучасного інструменту, що забезпечує гнучкість і масштабованість тестування.

Сьогодні Selenium є відкритим стандартом для автоматизації веб-тестування і використовується у багатьох компаніях, завдяки своїй підтримці кількох мов програмування, браузерів та платформ. Він часто порівнюється з такими інструментами, як Cypress, Playwright та TestCafe [23], але зберігає популярність завдяки широкому функціоналу та активній спільноті.

### 1.2.2 Архітектура Selenium WebDriver

Selenium WebDriver побудований за принципом клієнт-серверної архітектури, яка забезпечує взаємодію між тестовими скриптами та браузерами [8]. Основні компоненти:

1. Клієнтська бібліотека: надає інтерфейси для створення тестових скриптів. Підтримує кілька мов програмування, таких як Java, Python, C#, JavaScript, Ruby.
2. Драйвери браузерів: забезпечують зв’язок між WebDriver API і браузером. Наприклад, ChromeDriver, GeckoDriver, EdgeDriver.
3. Браузери: Selenium підтримує популярні браузери: Google Chrome, Mozilla Firefox, Microsoft Edge, Safari та Opera.
4. JSON Wire Protocol та W3C WebDriver Protocol: використовуються для передачі команд від клієнта до браузера. Нові версії Selenium (з 4.0) повністю перейшли на W3C WebDriver Protocol, що забезпечує стабільнішу взаємодію.

Приклад взаємодії: тестовий скрипт через клієнтську бібліотеку передає команду WebDriver API, яка надсилається до драйвера браузера. Драйвер виконує команду в браузері та повертає результат до скрипта.

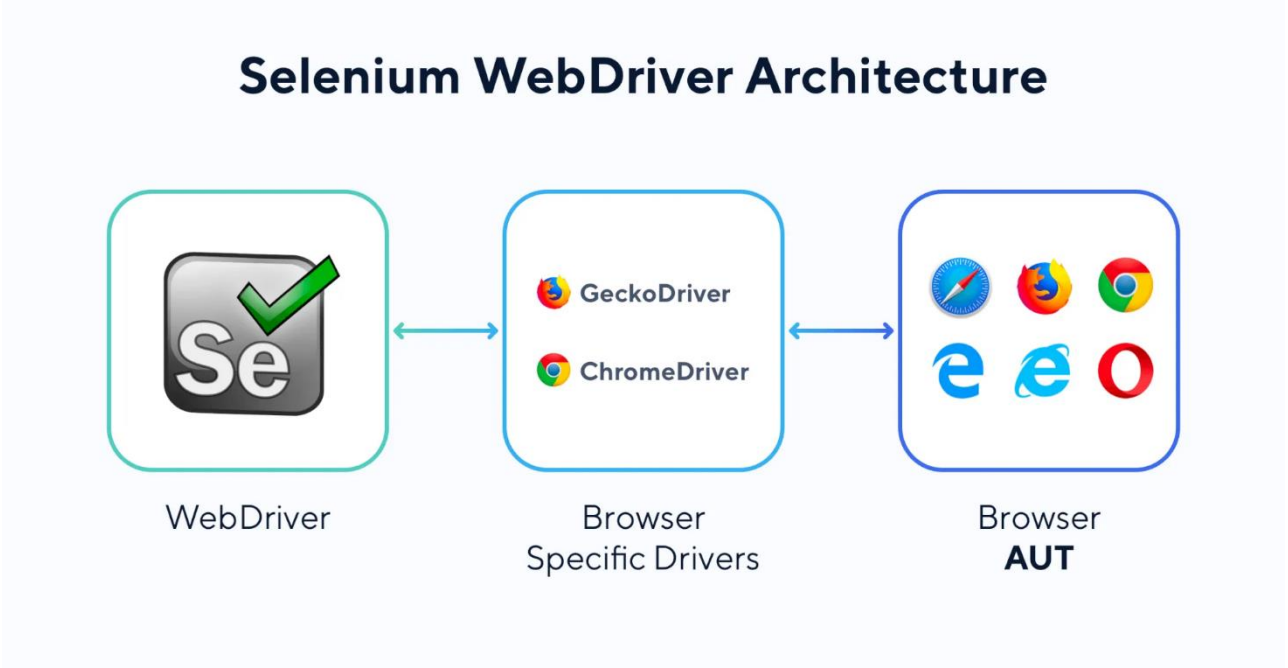



Рисунок 1 – Архітектура Selenium WebDriver

### 1.2.3 Підтримувані мови програмування та інтеграція з іншими інструментами

Selenium WebDriver підтримує найпоширеніші мови програмування, що дозволяє тестувальникам писати тести у знайомому середовищі [9]. Найпопулярніші мови:

1. Java – найпоширеніший вибір завдяки детальній документації та широкій спільноті.
2. Python – забезпечує простоту написання тестів, підходить для початківців.
3. C# – зручний для інтеграції у Windows-середовище.
4. JavaScript – актуальний для розробників на Node.js.

Selenium також легко інтегрується з іншими інструментами та фреймворками, що підвищує його ефективність:

1. Фреймворки для тестування: TestNG, JUnit, PyTest. Вони забезпечують організацію тестів, параметризацію та звітність.
2. Інструменти для управління залежностями: Maven, Gradle. Вони спрощують управління бібліотеками та драйверами.
3. CI/CD-системи: Jenkins, GitLab CI, Bamboo. Selenium часто використовують у пайплайнах для автоматизації регресійного тестування.
4. Хмарні сервіси для багатобраузерного тестування: BrowserStack, Sauce Labs. Це дозволяє тестувати у рідкісних браузерах і операційних системах.

### 1.2.4 Переваги та обмеження Selenium WebDriver у порівнянні з альтернативами

Переваги Selenium WebDriver:

1. Підтримка багатьох мов програмування – забезпечує гнучкість для команд з різним технічним бекграундом.

2. Велика спільнота та документація – доступність прикладів, порад і рішень для більшості проблем.
3. Підтримка різних браузерів – дозволяє проводити багатобраузерне тестування.
4. Можливість інтеграції – сумісність з інструментами CI/CD та хмарними сервісами.
5. Відкритий код – безкоштовність і можливість кастомізації.

Обмеження Selenium WebDriver

1. Складність налаштування – потребує значних зусиль для налаштування драйверів і тестового середовища.
2. Використання лише для веб-застосунків – не підходить для тестування настільних і мобільних застосунків без додаткових інструментів (наприклад, Appium для мобільних).
3. Швидкість виконання – програє в продуктивності сучасним інструментам, таким як Cypress чи Playwright.

## 1.3 Способи ідентифікації Веб-елементів

Один із ключових етапів автоматизації тестування веб-застосунків – це коректна ідентифікація веб-елементів. Selenium WebDriver надає кілька способів, які дозволяють знаходити елементи на сторінці та взаємодіяти з ними. Вибір методу залежить від структури веб-застосунку, доступних атрибутів елементів і сценарію тестування [10].

Основні способи ідентифікації:

1. By ID: ідентифікація за унікальним атрибутом id елемента.
2. By Name: використання атрибута name. Застосовується, коли id відсутній або не унікальний. Недолік: можливі конфлікти, якщо декілька елементів мають однакове ім’я.
3. By Class Name: використання атрибута class. Підходить для групування елементів за стилем або функціоналом. Недолік: часто

class використовується для оформлення, і його значення можуть змінюватися.

4. By Tag Name: ідентифікація за тегом HTML (наприклад, div, input, button).
5. By Link Text: ідентифікація за текстом посилання. Підходить для роботи з гіперпосиланнями.
6. By Partial Link Text: пошук посилання за частковим текстом. Використовується, коли текст посилання занадто довгий або змінюється.
7. By CSS Selector. використовує синтаксис CSS для доступу до елементів.
8. By XPath: найуніверсальніший спосіб, що дозволяє знаходити елементи за їхнім положенням у DOM. Підтримує як абсолютні, так і відносні шляхи. Недолік: складність у підтримці при зміні структури сторінки.

## 1.4 Методи WebDriver для пошуку елементів та для взаємодії з Веб-елементами

### 1.4.1 Методи для пошуку веб-елементів

Методи пошуку елементів базуються на класі By, який дозволяє задавати різні способи ідентифікації елементів на веб-сторінці [11].

1. findElement(By locator): використовується для пошуку одного елемента на сторінці. Якщо елемент не знайдено, викликається виняток NoSuchElementException.
2. findElements(By locator): повертає список усіх елементів, які відповідають заданому локатору. Якщо елементи не знайдено, повертається порожній список.

Обидва методи підтримують різні типи локаторів (By):

– By.id(String id)

- By.name(String name)
- By.className(String className)
- By.tagName(String tagName)
- By.linkText(String linkText)
- By.partialLinkText(String partialLinkText)
- By.cssSelector(String cssSelector)
- By.xpath(String xpathExpression)

### 1.4.2 Методи для взаємодії з веб-елементами

Методи класу WebElement дозволяють виконувати дії над знайденими елементами, такі як введення тексту, натискання кнопок, отримання інформації тощо [12].

1. click(): використовується для натискання на кнопки, посилання або інші інтерактивні елементи.
2. sendKeys(): використовується для введення тексту в текстові поля або емуляції натискання клавіш.
3. getText(): повертає текст, що знаходиться всередині веб-елемента.
4. getAttribute(String name): використовується для отримання значення атрибута веб-елемента.
5. getCssValue(String propertyName): отримує значення CSS-властивості елемента.
6. isDisplayed(): перевіряє, чи видимий елемент на сторінці.
7. isEnabled(): перевіряє, чи доступний елемент для взаємодії.
8. isSelected(): використовується для перевірки, чи обраний елемент, наприклад, чекбокс або радіобаттон.
9. clear(): очищує текстове поле.
10. submit(): використовується для відправки форми. Зазвичай викликається на елементі типу <input> або <button>.

## 1.5 Стратегії очікування Веб-елементів

В автоматизованому тестуванні веб-застосунків часто виникають ситуації, коли Selenium WebDriver намагається взаємодіяти з елементами, які ще не завантажилися або недоступні. Це призводить до помилок, таких як NoSuchElementException. Щоб вирішити цю проблему, використовуються стратегії очікування, які дозволяють тестам дочекатися необхідного стану веб-елементів перед взаємодією з ними [14].

### 1.5.1 Типи очікувань у Selenium

1. Неявне очікування (Implicit Wait): використовується для встановлення глобального часу очікування для пошуку всіх елементів. Selenium намагається знайти елемент протягом зазначеного часу.

Лістинг 1 – Приклад неявного очікування

```
driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(
10));
WebElement element = driver.findElement(By.id("username"));
```

Переваги: простота налаштування.

Недоліки: впливає на всі операції пошуку елементів, що може уповільнити тестування.

2. Явне очікування (Explicit Wait): дає змогу встановити умову для конкретного елемента, яку потрібно виконати до продовження тесту. Використовується разом із класом WebDriverWait.

Лістинг 2 – Приклад явного очікування

```
WebDriverWait wait = new WebDriverWait(driver,
Duration.ofSeconds(10));
```

```
WebElement element =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.i
d("username")));
```

Переваги: контроль над кожним очікуванням.

Недоліки: потребує більше коду для налаштування.

3. Fluent Wait дозволяє налаштувати гнучке очікування з інтервалами перевірки та можливістю обробки виключень. Це особливо корисно, коли потрібно більш точний контроль за очікуванням певних умов або стану елементів.

Лістинг 3 – Приклад Fluent Wait

```
Wait<WebDriver> wait =
    new FluentWait<>(driver)
        .withTimeout(Duration.ofSeconds(2))
        .pollingEvery(Duration.ofMillis(300))
        .ignoring(ElementNotInteractableException.class);

wait.until(
    d -> {
      revealed.sendKeys("Displayed");
      return true;
    });
```

### 1.5.2 Основні умови для явного очікування

Клас ExpectedConditions забезпечує низку готових умов, які часто використовуються:

1. visibilityOfElementLocated: очікує, поки елемент стане видимим.
2. elementToBeClickable: очікує, поки елемент стане клікабельним.
3. presenceOfElementLocated: очікує, поки елемент буде присутній у DOM, незалежно від його видимості.

4. textToBePresentInElement: очікує, поки заданий текст з'явиться в елементі.
5. alertIsPresent: очікує появу спливаючого вікна (alert).

# 2 ПРОЄКТУВАННЯ АВТОМАТИЗОВАНИХ ТЕСТІВ ДЛЯ ВЕБ-ЗАСТОСУНКУ

## 2.1 Опис функціоналу веб-застосунку

У якості веб-застосунку для тестування був обраний застосунок Sauce Demo [13]. Sauce Demo – це демонстраційний веб-застосунок, створений для тестування веб-інтерфейсів і перевірки функціоналу автоматизованих тестів. Його надає компанія Sauce Labs [15] як платформу для практичного використання інструментів автоматизації тестування, таких як Selenium, Cypress тощо.

Застосунок містить просту інтерактивну функціональність, що імітує типові процеси роботи інтернет-магазину. Завдяки зрозумілому дизайну та передбачуваній поведінці, Sauce Demo використовується як зручний інструмент для навчання, експериментів з автоматизацією тестів та оцінювання роботи тестових фреймворків.

Головним екраном застосунку є авторизація, нижче полів для введення логіну та паролю написани прийнятні логіни та пароль для користувачів.

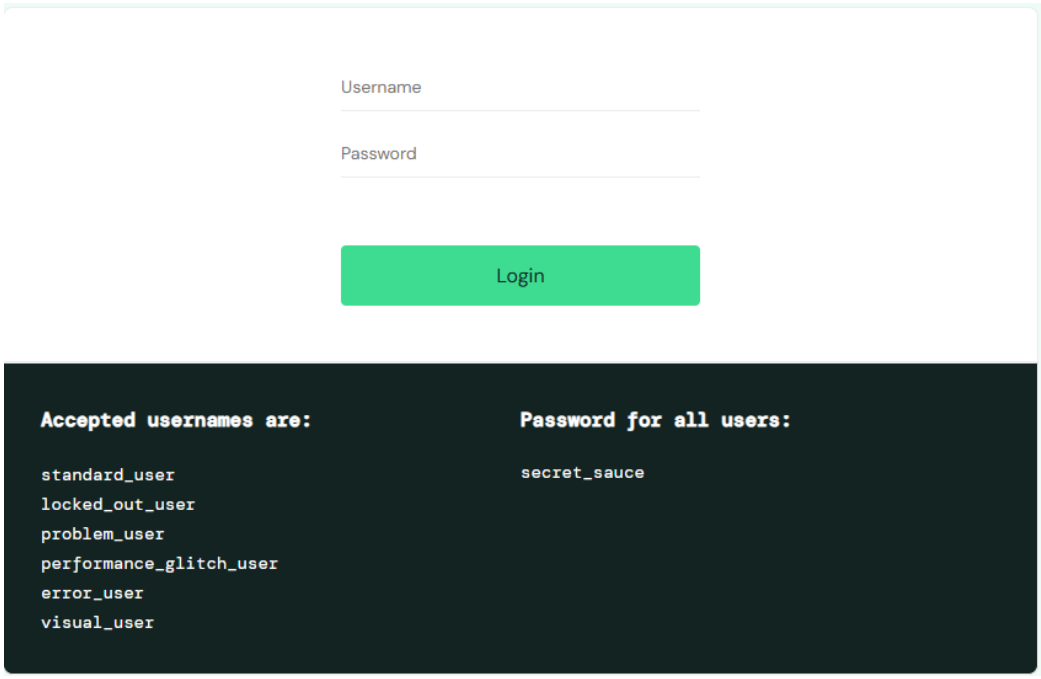



Рисунок 2 – Головний екран застосунку

Після успішної авторизації, веб-застосунок відображає список доступних товарів, кнопку сортування товарів та кнопку з кошиком, за допомогою якого можна оформити замовлення. Кожен товар містить назву, опис, зображення, ціну та кнопку додавання товару до кошику.

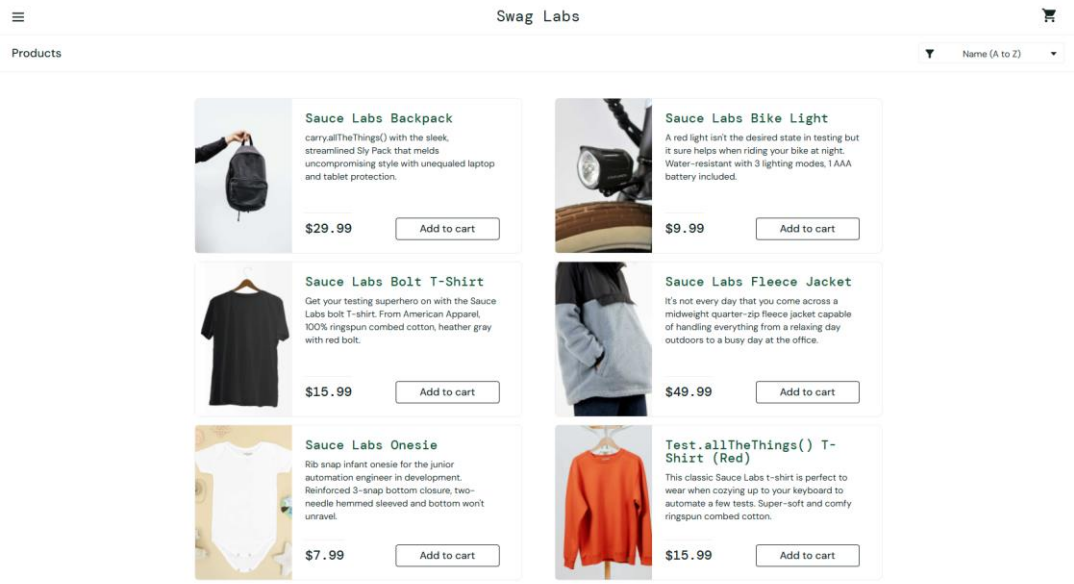



Рисунок 3 – Екран застосунку після авторизації

При взаємодії з товаром, натиснувши на нього, можна перейти до сторінки з більш детальною інформацію про певний товар.

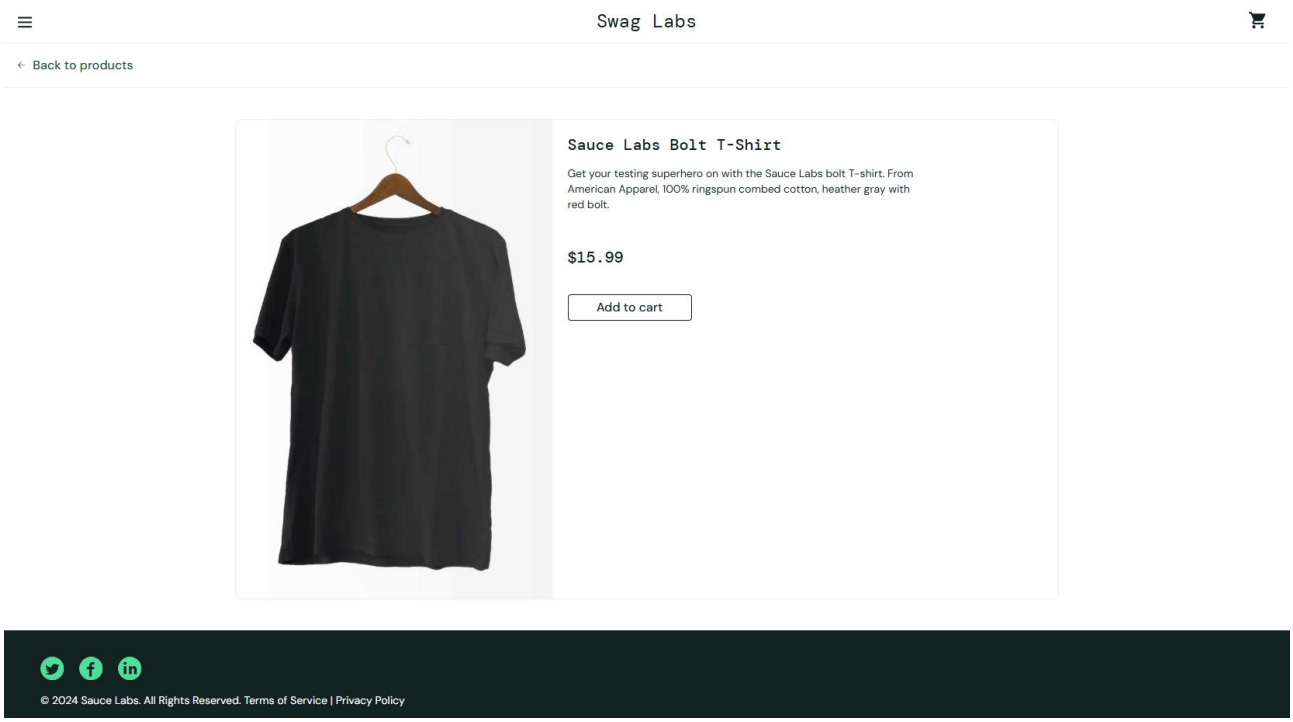



Рисунок 4 – Сторінка з деталями для обраного товару

Товари можна відсортовувати за ціною та за ім’ям.

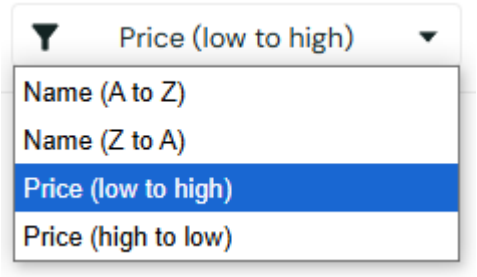



Рисунок 5 – Функціонал сортування товарів

Після додавання товарів до кошику, вони відображаються на сторінці кошику, де вказуються усі додані товари. Їх можна видалити, також є кнопки для повернення на головну сторінку та переходу для оформлення замовлення.

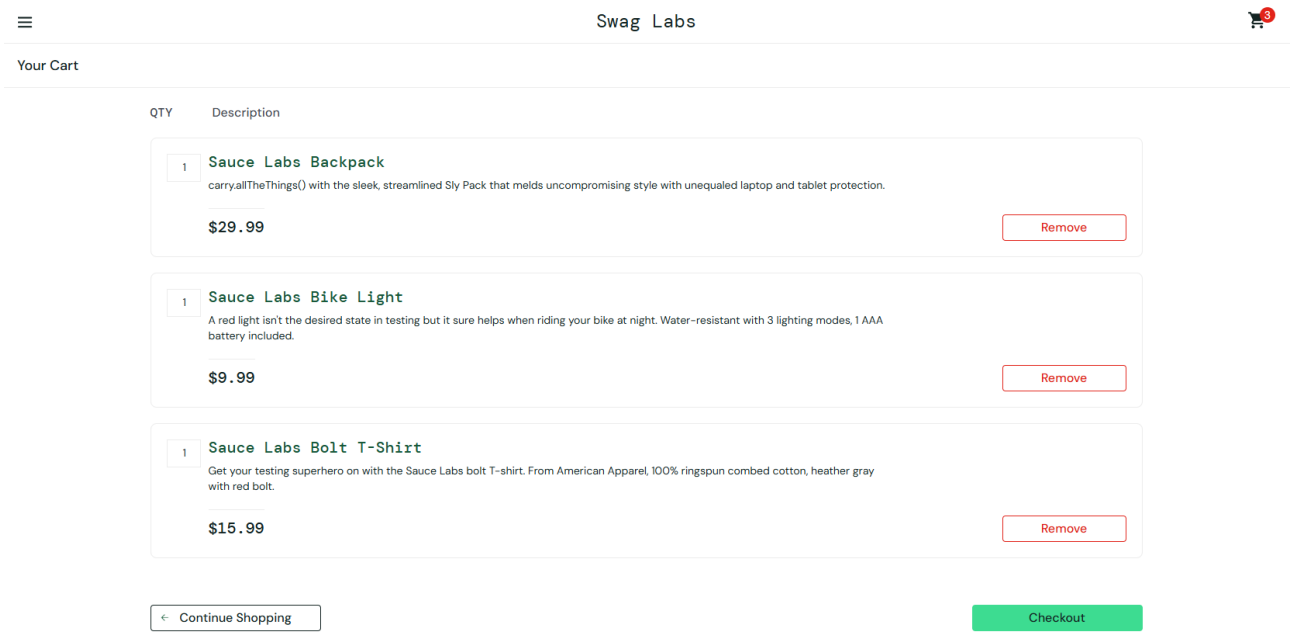



Рисунок 6 – Сторінка кошику

Після натискання на кнопку Checkout відкривається сторінка, де потрібно заповнити ім’я, призвіще та поштовий код для оформлення замовлення.

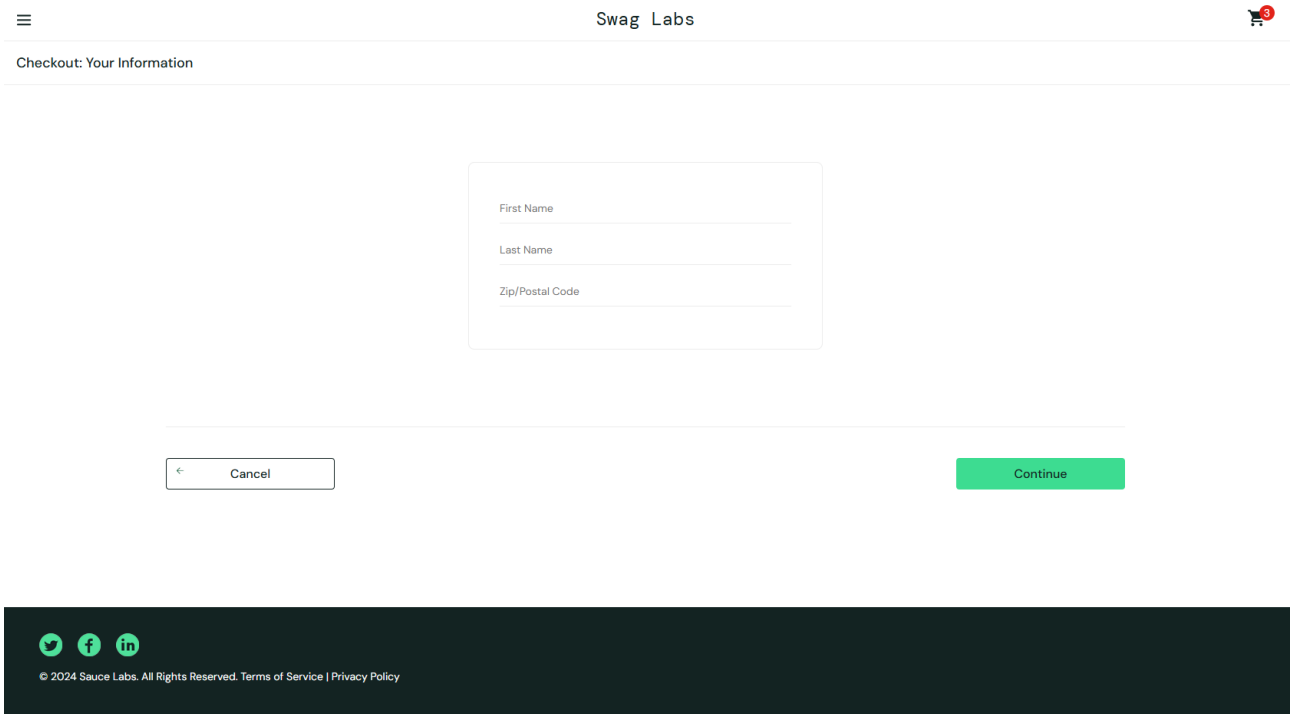



Рисунок 7 – Сторінка оформлення замовлення

Після заповнення інформації та натиснувши на кнопку Continue відкривається сторінка із завершенням замовлення, де зазначені замовлені товари, інформація про платіж, доставку та сума замовлення.

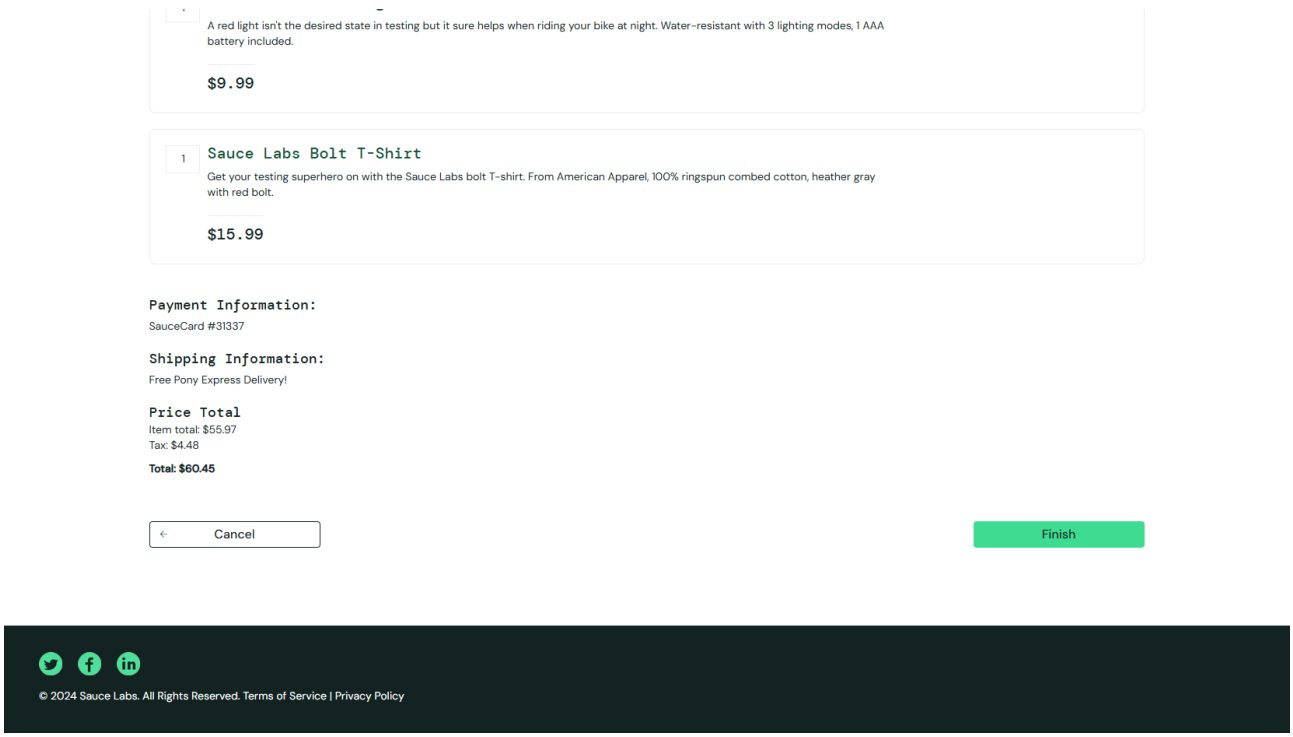



Рисунок 8 – Сторінка із підтвердженням замовлення

Після того, як буде натиснуто кнопку Finish, з’являється екран про успішне оформлення замовлення.

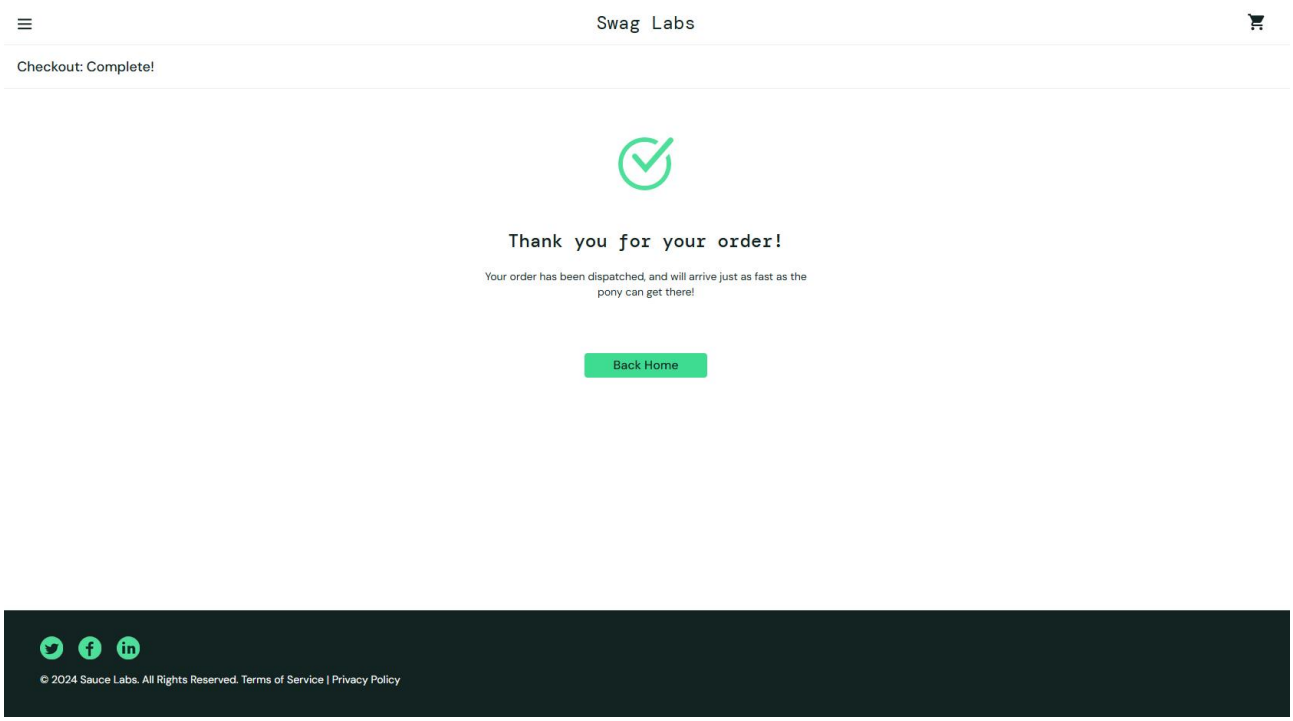



Рисунок 9 – Сторінка з повідомленням про успішне замовлення

## 2.2 Розробка тест-вимог для веб-застосунку

Для тестування функціоналу веб-застосунку Sauce Demo можна запропонувати тестування авторизації користувача, тестування оформлення замовлення, тестування функціональності перегляду товарів, тестування сортування за ціною, тестування сортування за алфавітом, тестування роботи кошику, видалення та додавання до нього товарів, тестування бургер-меню, тестування введення даних для замовлення. Сформовано відповідні тест-вимоги:

1. Тестування авторизації користувача:
   - Перейти на головну сторінку веб-застосунку;
   - Знайти поле для введення логіну;
   - Знайти поле для введеня пароля;
   - Знайти кнопку Login для авторизації;
   - Заповнити поле для введення логіну;
   - Заповнити поле для введення пароля;
   - Натиснути кнопку Login для завершення атворизації;

- Після натискання перевірити, чи відображається повідомлення про помилку, якщо повідомлення немає – тест пройдено, якщо повідомлення з’являється – тест не пройдено.

2. Тестування сортування за ціною:
   - Виконати аутентифікацію;
   - Знайти випадаючий список для сортування товарів;
   - Перевірити чи є на сторінці хоча б два товари для порівняння цін;
   - Обрати опцію сортування товарів від низької до високої ціни;
   - Порівняти ціну першого та наступного товару, якщо перша ціна менша або дорівнює ціні другого товару – тест пройдено, якщо ціна першого товару більша за ціну другого товару – тест не пройдено;
   - Обрати опцію сортування товарів від високої до низької ціни;
   - Порівняти ціну першого та наступного товару, якщо перша ціна більша або дорівнює ціні другого товару – тест пройдено, якщо ціна першого товару менша за ціну другого товару – тест не пройдено.
3. Тестування сортування за ім’ям:
   - Виконати аутентифікацію;
   - Знайти випадаючий список для сортування товарів;
   - Обрати опцію сортування товарів за алфавітом від A до Z;
   - Перевірити поточну назву товару з наступною назвою товару, якщо якщо всі назви товарів йдуть в алфавітному порядку – тест пройдено, якщо відсортовані неправильно – тест не пройдено;
   - Обрати опцію сортування товарів за алфавітом від Z до A;
   - Перевірити поточну назву товару з наступною назвою товару, якщо якщо всі назви товарів йдуть в протилежному алфавітному

порядку – тест пройдено, якщо відсортовані неправильно – тест не пройдено.

4. Тестування оформлення замовлення:
   - Виконати аутентифікацію;
   - Знайти кнопку додавання товарів до кошика;
   - Додати перші два товари до кошика;
   - Перейти до кошика;
   - Перевірити, що до кошика додано два товари;
   - Перейти до переходу до оформлення замовлення;
   - Знайти та заповнити поля ім’я, призвіще та поштовий код;
   - Перевірити, чи в замовленні присутні 2 товари;
   - Натиснути на кнопку відміни;
   - Перейти до кошика;
   - Заповнити поля для замовлення;
   - Завершення замовлення;
   - Перевірка наявності повідомлення про завершення замовлення.
5. Тестування функціональності перегляду товарів:
   - Виконати аутентифікацію;
   - Натиснути на перший товар;
   - Перевірити на сторінці товару що присутні ім’я, ціна та опис товару.
6. Тестування роботи кошику:
   - Виконати аутентифікацію;
   - Додати 3 товари до кошика;
   - Перевірити що у кошику знаходиться 3 товари;
   - Видалити 3 товари;
   - Перевірити, що у кошику не залишилось товарів.
7. Тестування бургер-меню:
   - Виконати аутентифікацію;

- Перейти до кошика;
- Відкрити бургер-меню;
- Натиснути на кнопку Logout;
- Перевірити що відкрита сторінка авторизації.

8. Тестування введення даних для замовлення:
   - Виконати аутентифікацію;
   - Додати товар до кошику;
   - Перевірити що товар є у кошику;
   - Натиснути на кнопку продовження та перевірити що виникає помилка, що поле імені не заповнено;
   - Заповнити поле імені та натиснути кнопку продовжити та перевірити що виникає помилка, що поле призвіща не заповнено;
   - Заповнити поле призвіща та натиснути кнопку продовжити та перевірити що виникає помилка, що поле індексу не заповнено.

## 2.3 Реалізація проєкту з Selenium Web Driver

Для створення тестів у файлі pom.xml зазначені основні залежності для автоматизації тестування веб-застосунків. Залежність TestNG [18] використовується як фреймворк для організації тестів і створення тестових сценаріїв з чіткою структурою. Бібліотека selenium-java забезпечує взаємодію з веб-елементами через Selenium WebDriver, дозволяючи автоматизувати роботу браузера. Залежність webdrivermanager автоматично завантажує і налаштовує відповідні драйвери для браузерів, спрощуючи конфігурацію тестового середовища. Файл pom.xml описаний у лістингу 4.

Лістинг 4 – Файл pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
```

```
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>SauceDemoTesting</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>21</maven.compiler.source>
        <maven.compiler.target>21</maven.compiler.target>
        <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>7.10.2</version>
        </dependency>
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>4.22.0</version>
        </dependency>
        <dependency>
            <groupId>io.github.bonigarcia</groupId>
            <artifactId>webdrivermanager</artifactId>
            <version>5.9.2</version>
        </dependency>
```

```
    </dependencies>
</project>
```

Для тестування функціональності авторизації було створено клас LoginTest та вміст тесту описаний у лістингу 5.

Лістинг 5 – Клас LoginTest для тестування авторизації

```
package test;

import io.github.bonigarcia.wdm.WebDriverManager;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class LoginTest {
    WebDriver driver;

    @BeforeTest
    public void setup() {
        WebDriverManager.chromedriver().setup();
        driver = new ChromeDriver();
        driver.get("https://www.saucedemo.com");
    }

    @Test
    public void testLogin() {
```

```
        WebElement username = driver.findElement(By.id("user-
name"));
        WebElement password =
driver.findElement(By.id("password"));
        WebElement loginButton =
driver.findElement(By.id("login-button"));

        username.sendKeys("standard_user");
        password.sendKeys("secret_sauce");
        loginButton.click();

        boolean isErrorDisplayed = isErrorMessageDisplayed();
        if (isErrorDisplayed) {
            System.out.println("Login failed. Incorrect
username or password.");
            Assert.fail("Login error message displayed.");
        } else {
            System.out.println("Login Success.");

Assert.assertTrue(driver.getCurrentUrl().contains("inventory.
html"), "User is not redirected to inventory page.");
        }
    }

    @AfterTest
    public void teardown() {
        if (driver != null) {
            driver.quit();
        }
    }

    private boolean isErrorMessageDisplayed() {
        try {
```

```
            WebElement errorMessage =
driver.findElement(By.cssSelector("[data-test='error']"));
            return errorMessage.isDisplayed();
        } catch (Exception e) {
            return false;
        }
    }
}
```

Результати тесту можна побачити на рисунку 10.

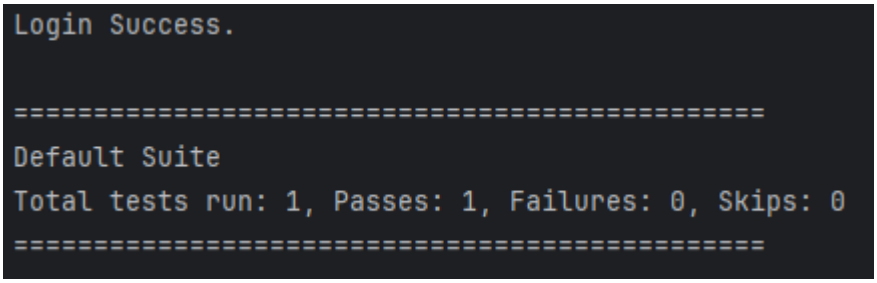



Рисунок 10 – Результати тесту авторизації

Якщо змінити вхідні дані для тесту, наприклад змінивши пароль, отримуємо результат на рисунку 11.

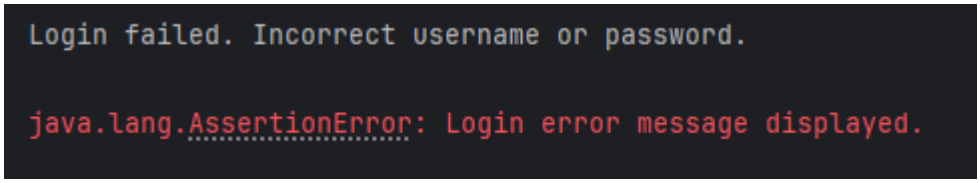



Рисунок 11 – Результати тесту після зміни вхідних даних

Для тестування функціональності сортування за ціною було створено клас SortingByPriceTest та вміст тесту описаний у лістингу 6.

Лістинг 6 – Клас SortingByPriceTest для тестування сортування за ціною

```
package test;
```

```
import io.github.bonigarcia.wdm.WebDriverManager;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.*;

import java.util.List;

public class SortingByPriceTest {

    WebDriver driver;

    @BeforeTest
    public void setup() {
        WebDriverManager.chromedriver().setup();
        driver = new ChromeDriver();
        driver.get("https://www.saucedemo.com");
        driver.manage().window().maximize();
    }

    @Test(priority = 1)
    public void testLogin() {
        WebElement username = driver.findElement(By.id("user-
name"));
        WebElement                password                =
driver.findElement(By.id("password"));
        WebElement               loginButton              =
driver.findElement(By.id("login-button"));

        username.sendKeys("standard_user");
```

```
        password.sendKeys("secret_sauce");
        loginButton.click();

        boolean isErrorDisplayed = isErrorMessageDisplayed();
        Assert.assertFalse(isErrorDisplayed, "Login failed. Incorrect username or password.");
    }

    @Test(priority = 2, dependsOnMethods = "testLogin")
    public void testSortProductsByPriceLowToHigh() {
        WebElement sortDropdown = driver.findElement(By.className("product_sort_container"));
        sortDropdown.click();
        WebElement lowToHighOption = driver.findElement(By.xpath("//option[@value='lohi']"));
        lowToHighOption.click();

        List<WebElement> productPrices = driver.findElements(By.className("inventory_item_price"));
        Assert.assertTrue(productPrices.size() >= 2, "Not enough products to compare.");

        for (int i = 0; i < productPrices.size() - 1; i++) {
            double currentPrice = extractPrice(productPrices.get(i).getText());
            double nextPrice = extractPrice(productPrices.get(i + 1).getText());
            Assert.assertTrue(currentPrice <= nextPrice, "Products are not sorted by price low to high.");
        }
    }

    @Test(priority = 3, dependsOnMethods = "testLogin")
```

```
    public void testSortProductsByPriceHighToLow() {
        WebElement sortDropdown =
driver.findElement(By.className("product_sort_container"));
        sortDropdown.click();
        WebElement highToLowOption =
driver.findElement(By.xpath("//option[@value='hilo']"));
        highToLowOption.click();

        List<WebElement> productPrices =
driver.findElements(By.className("inventory_item_price"));
        Assert.assertTrue(productPrices.size() >= 2, "Not
enough products to compare.");

        for (int i = 0; i < productPrices.size() - 1; i++) {
            double currentPrice =
extractPrice(productPrices.get(i).getText());
            double nextPrice =
extractPrice(productPrices.get(i + 1).getText());
            Assert.assertTrue(currentPrice >= nextPrice,
"Products are not sorted by price high to low.");
        }
    }

    @AfterTest
    public void teardown() {
        if (driver != null) {
            driver.quit();
        }
    }

    private double extractPrice(String priceText) {
        return Double.parseDouble(priceText.replace("$", ""));
    }
```

```
    private boolean isErrorMessageDisplayed() {
        try {
            WebElement errorMessage =
driver.findElement(By.cssSelector("[data-test='error']"));
            return errorMessage.isDisplayed();
        } catch (Exception e) {
            return false;
        }
    }
}
```

Результати тесту можна побачити на рисунку 12.

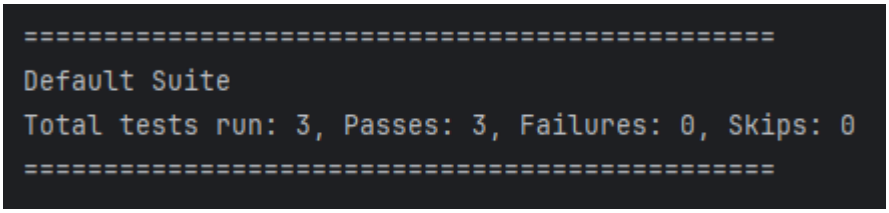



Рисунок 12 – Результатати тесту сортування за ціною

Для тестування функціональності оформлення замовлення було створено клас OrderTest та вміст тесту описаний у лістингу 7.

Лістинг 7 – Клас OrderTest для тестування оформлення замовлення

```
package test;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

import java.util.List;
```

```
public class OrderTest {
    public static void main(String[] args) {
        System.setProperty("webdriver.chrome.driver",
"C:\\Program Files\\chromedriver\\chromedriver-
win64\\chromedriver.exe");
        WebDriver driver = new ChromeDriver();

        driver.get("https://www.saucedemo.com");

        // Login elements
        WebElement username = driver.findElement(By.id("user-
name"));
        WebElement password =
driver.findElement(By.id("password"));
        WebElement loginButton =
driver.findElement(By.id("login-button"));

        // Send keys to elements
        username.sendKeys("standard_user");
        password.sendKeys("secret_sauce");
        loginButton.click();

        // Check for login error message
        boolean isErrorDisplayed =
isErrorMessageDisplayed(driver);

        if (isErrorDisplayed) {
            System.out.println("Login failed. Incorrect
username or password.");
            driver.quit();
            return;
        } else {
```

```
            System.out.println("Login Success.");
        }

        // Add first 3 items to Cart
        List<WebElement> addToCartButtons =
driver.findElements(By.className("btn_inventory"));
        for (int i = 0; i < 3; i++) {
            addToCartButtons.get(i).click();
        }

        // Enter cart screen
        WebElement cartButton =
driver.findElement(By.className("shopping_cart_link"));
        cartButton.click();

        // Check that cart has 3 items
        List<WebElement> cartItems =
driver.findElements(By.className("cart_item"));
        if (cartItems.size() != 3) {
            System.out.println("Test Failed: Expected 3 items
in the cart, but found " + cartItems.size());
            driver.quit();
            return;
        }

        WebElement checkoutButton =
driver.findElement(By.className("checkout_button"));
        checkoutButton.click();

        // Order elements
        WebElement firstNameField =
driver.findElement(By.id("first-name"));
```

```
        WebElement lastNameField =
driver.findElement(By.id("last-name"));
        WebElement zipCodeField =
driver.findElement(By.id("postal-code"));
        WebElement continueButton =
driver.findElement(By.className("btn_primary"));

        // Send keys to elements
        firstNameField.sendKeys("John");
        lastNameField.sendKeys("Doe");
        zipCodeField.sendKeys("12345");
        continueButton.click();

        WebElement checkoutOverviewTitle =
driver.findElement(By.className("title"));
        if (checkoutOverviewTitle.getText().equals("Checkout:
Overview")) {
            System.out.println("Checkout Overview page loaded
successfully.");
            WebElement finishButton =
driver.findElement(By.className("btn_action"));
            finishButton.click();

            WebElement confirmationMessage =
driver.findElement(By.className("complete-header"));
            if (confirmationMessage.getText().equals("Thank
you for your order!")) {
                System.out.println("Test Passed: Order was
placed successfully.");
            } else {
                System.out.println("Test Failed: Order was
not placed successfully.");
            }
```

```
        } else {
            System.out.println("Test Failed: Checkout
Overview page was not displayed.");
        }
        driver.quit();
    }

    private static boolean isErrorMessageDisplayed(WebDriver
driver) {
        try {
            WebElement errorMessage =
driver.findElement(By.cssSelector("[data-test='error']"));
            return errorMessage.isDisplayed();
        } catch (Exception e) {
            return false;  // If exception occurs (element
not found), error is not displayed
        }
    }
}
```

Результати тесту можна побачити на рисунку 13.

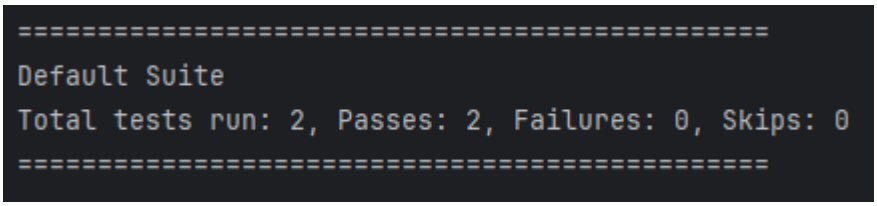



Рисунок 13 – Результати тесту оформлення замовлення

Тести, написані без використання підходу Page Object Model, мають кілька значних недоліків, які ускладнюють їх підтримке. Перш за все, код тестів стає громіздким через велику кількість дублювання, оскільки взаємодія з веб-елементами часто повторюється в різних місцях. Наприклад, пошук веб-

елементів, таких як кнопки чи поля вводу, виконується безпосередньо у кожному тесті, що не лише збільшує обсяг коду, але й створює потенційну проблему для внесення змін.

Ще одним недоліком є жорстка прив’язка до локаторів веб-елементів. Якщо ідентифікатори чи структура веб-сторінки зміняться, необхідно буде вручну оновлювати всі тести, в яких використовуються ці локатори. Це значно ускладнює рефакторинг і підвищує ризик помилок.

Крім того, через відсутність чіткої структури коду, його стає важко читати, розуміти та підтримувати. Тести стають залежними від реалізації конкретних сценаріїв, що знижує їхню гнучкість і повторне використання коду. Такий підхід створює значні труднощі при масштабуванні проекту, оскільки додавання нових тестів вимагає багато ручної роботи та може спричиняти конфлікти з уже існуючими.

# 3 ОРГАНІЗАЦІЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКУ ІЗ ВИКОРИСТАННЯМ PAGE-OBJECT MODEL

## 3.1 Шаблон проєктування Page Object Model (POM): ідея та переваги використання

Page Object Model – це широко використовуваний шаблон проєктування в автоматизованому тестуванні веб-застосунків, який спрямований на підвищення структурованості, читабельності та підтримуваності тестового коду. Основна концепція цього підходу полягає у створенні окремих класів, які відповідають конкретним веб-сторінкам або їхнім функціональним компонентам. У кожному з таких класів зберігається інформація про веб-елементи сторінки та методи для взаємодії з ними. Таким чином, код стає більш організованим, а тести набувають вигляду високорівневих сценаріїв, що легше сприймаються іншими розробниками чи тестувальниками [16].

Ідея Page Object Model базується на принципі інкапсуляції, де всі дії, пов’язані з веб-сторінкою, виконуються через методи відповідного класу. Наприклад, сторінка входу може мати методи для введення імені користувача, пароля та натискання кнопки входу. У тестових класах ці методи використовуються замість прямих викликів Selenium WebDriver для взаємодії з елементами сторінки, що дозволяє відокремити тестову логіку від деталей реалізації. Це забезпечує чітку ієрархію та сприяє зменшенню дублювання коду.

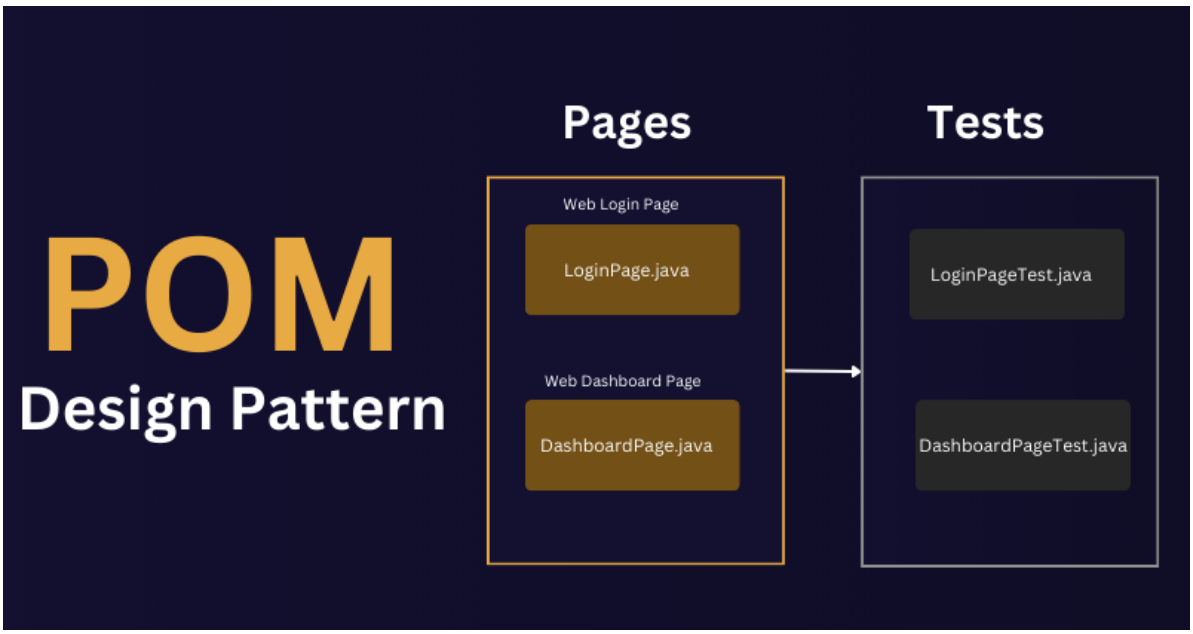



Рисунок 14 – Шаблон проєктування Page Object Model

Застосування Page Object Model надає низку переваг, які роблять цей підхід стандартом у сфері автоматизації тестування. Одна з ключових переваг – покращення підтримуваності тестового коду. У разі змін у структурі веб-застосунку, наприклад, якщо буде змінено локатор елемента на сторінці, це виправлення потрібно внести лише в одному місці – у відповідному класі сторінки. Таким чином, можна уникнути внесення змін у кожен тест, що суттєво скорочує час і зусилля, потрібні для підтримки тестів.

Ще однією важливою перевагою є підвищення читабельності тестів. Завдяки використанню Page Object Model, тестові сценарії виглядають як послідовність дій або бізнес-процесів, виконуваних у застосунку. Наприклад, замість того, щоб вручну викликати методи для пошуку елементів і взаємодії з ними, тест описує кроки на високому рівні: логін, додавання товарів до кошика, оформлення замовлення. Це сприяє кращому розумінню тестів навіть тими, хто не знайомий із технічними деталями реалізації.

Крім того, використання Page Object Model значно зменшує кількість дублювання коду. Загальні елементи сторінок, такі як меню навігації або панель кошика, можуть бути інкапсульовані в базовому класі, який успадковується

іншими сторінками. Це дозволяє використовувати спільну функціональність без необхідності повторного опису.

Останньою, але не менш важливою перевагою є полегшення інтеграції з інструментами для керування тестами, такими як TestNG чи JUnit. Оскільки POM сприяє створенню модульного та структурованого коду, його легко розширювати та інтегрувати з системами звітності або запускати на різних середовищах. Це дозволяє масштабувати тестовий проєкт і підвищити ефективність автоматизації в довгостроковій перспективі.

## 3.2 Основні принципи Page Object Model

При використанні Page Object Model (POM) важливо дотримуватися кількох основних принципів, які сприяють ефективності тестування веб-застосунків [17].

Розділення на сторінки: кожна сторінка веб-застосунку повинна мати свій окремий клас, що відповідає за взаємодію з цією сторінкою. Наприклад, LoginPage, HomePage, ProductPage.

Елементи сторінки як об'єкти: кожен елемент на сторінці (наприклад, кнопка, поле введення) представлений в класі сторінки як об'єкт. Це дозволяє зосередитися на реалізації взаємодії з елементами.

Методи для взаємодії з елементами: клас сторінки має методи, що описують дії з елементами сторінки, такі як введення тексту, клікання на кнопку, перевірка видимості елементу. Ці методи ізолюють логіку тесту від локаторів елементів.

Використання цих принципів дозволяє створити чітко структуровані тестові скрипти, які легко розуміти і підтримувати, що робить тестування веб-застосунків більш ефективним.

## 3.3 Проєктування архітектури тестів із використанням Page Object Model

Проєктування архітектури тестів із використанням шаблону Page Object Model є ключовим етапом створення стійкої, модульної та легко підтримуваної системи автоматизованого тестування. Основний принцип полягає у поділі тестової логіки від функціональних деталей веб-застосунку. Це дозволяє зосередитись на створенні зрозумілих і легко масштабованих тестів.

На першому етапі проєктування слід визначити основні функціональні частини веб-застосунку, які будуть відповідати окремим класам у структурі Page Object. Кожен клас представляє певну веб-сторінку або її логічну частину. Архітектура тестів повинна бути розроблена таким чином, щоб кожна сторінка мала чітко визначені методи для виконання конкретних дій, наприклад, заповнення форми чи переходу між розділами. Це дозволяє ізолювати змінні компоненти, такі як локатори елементів, від тестового сценарію, що робить систему більш стабільною до змін у застосунку.

У процесі проєктування необхідно враховувати загальні компоненти, які використовуються на кількох сторінках, такі як меню навігації, кошик чи панель входу. Для цього створюються базові класи, які містять методи та елементи, спільні для всіх сторінок. Успадкування цих базових класів іншими сторінками дозволяє зменшити дублювання коду і полегшує підтримку в майбутньому. Наприклад, якщо змінюється структура меню, то оновлення базового класу автоматично адаптує всі сторінки, що його використовують.

Після визначення структури Page Object важливо забезпечити чітке розмежування між тестовими сценаріями та функціональністю класів сторінок. Тести повинні бути зосереджені на перевірці бізнес-логіки, а класи сторінок – на реалізації взаємодії з інтерфейсом. Такий підхід дозволяє розробникам тестів зосередитись на створенні сценаріїв, не турбуючись про зміну локаторів чи механізмів взаємодії.

### 3.4 Реалізація проєкту з POM та Selenium WebDriver автоматизованими тестами для демонстраційного Веб-застосунку

Архітектура проєкту базується на чітко структурованій організації класів, що забезпечує зручність розробки, тестування та підтримки автоматизованих тестів для веб-застосунку. Проєкт включає три основні пакети: components, pages та test. Кожен з цих пакетів відповідає за певний аспект роботи з веб-інтерфейсом і тестами.

У пакеті pages містяться класи, що відповідають за сторінки веб-застосунку. Кожен клас у цьому пакеті представляє одну зі сторінок і містить методи для взаємодії з елементами цієї сторінки. Наприклад, CartPage дозволяє взаємодіяти з кошиком, а HomePage містить методи для роботи з головною сторінкою застосунку, що відображає всі доступні товари після авторизації. Серед важливих класів у цьому пакеті є TopPart та LoginPage.

Клас TopPart є абстрактним класом, який визначає загальні елементи верхньої частини інтерфейсу, такі як кнопка меню-гамбургер і кнопка кошика. У цьому класі використовуються аннотації @FindBy для ініціалізації елементів сторінки за допомогою PageFactory. Він забезпечує методи для кліку на ці кнопки, зокрема clickCartButton(), що перекидає користувача на сторінку кошика, і clickHamburgerButton(), що відкриває меню. Цей клас може бути використаний як базовий для всіх сторінок, що містять подібні елементи верхньої частини інтерфейсу.

Другим абстрактним класом є TestRunnerFirst. Це базовий клас для тестів, який ініціалізує веб-драйвер та запускає тестові сценарії. Клас містить методи для налаштування та завершення тестів, такі як beforeClass() і tearDownClass(), що забезпечують налаштування середовища перед тестами та очищення після виконання тестів. Клас також містить метод loadApplication(), який відкриває основну сторінку веб-застосунку та повертає об'єкт LoginPage, що дозволяє ініціалізувати процес авторизації. Цей клас є основою для всіх тестів і забезпечує централізоване керування тестовим середовищем.

У пакеті test знаходяться класи, що реалізують тестові сценарії. Вони використовують методи класів сторінок для взаємодії з веб-інтерфейсом. Тести організовано за принципом Page Object Model, де кожен тестовий клас взаємодіє з відповідними класами сторінок, що забезпечує високий рівень абстракції та зручність при підтримці тестів. Наприклад, тести для перевірки додавання товарів до кошика чи сортування товарів за ціною чи назвою використовують методи класів сторінок для виконання операцій.

Уся структура архітектури дозволяє зручно підтримувати проект, додаючи нові функціональності або змінюючи існуючі. Завдяки використанню POM логіка взаємодії з елементами сторінок чітко відокремлена від тестових сценаріїв, що полегшує рефакторинг та масштабування тестів, а також зменшує дублювання коду та підвищує його повторне використання.

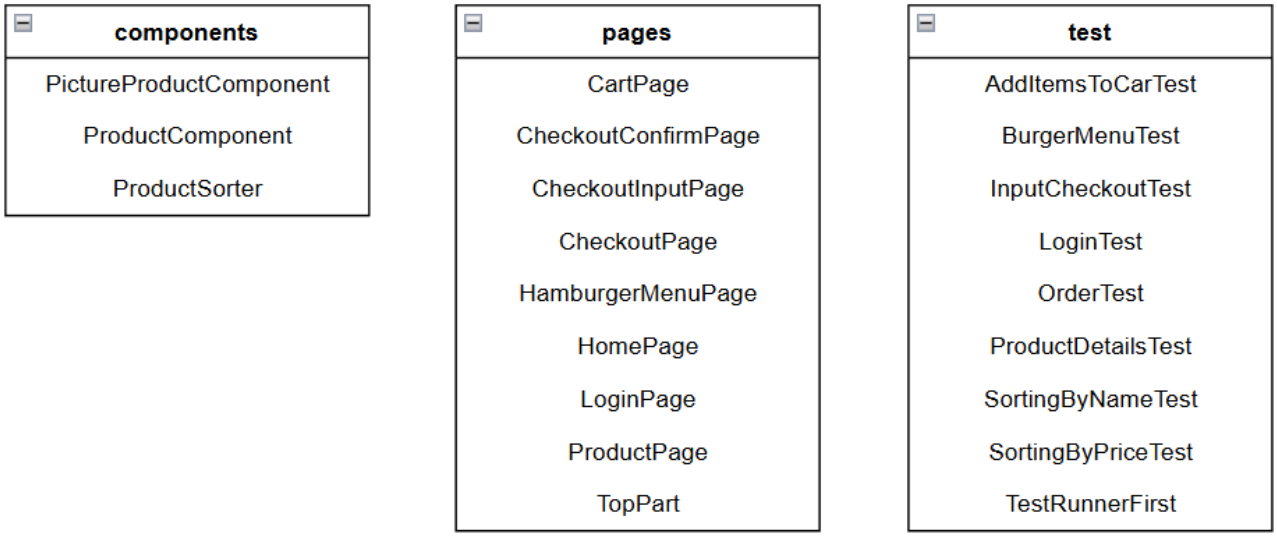



Рисунок 15 – Архітектура проєкту

Як приклад реалізації тесту з Page Object Model можна розглянути OrderTestPOM, це сценарій оформлення замовлення. Кроки для створення тесту оформлення замовлення:

1. Аналіз сторінок веб-застосунку: було визначено сторінки, які беруть участь у процесі оформлення замовлення: сторінка авторизації, головна

сторінка зі списком товарів, кошик, сторінка введення даних для оформлення замовлення та сторінка підтвердження замовлення.

2. Створення класів Page Object для кожної сторінки: для кожної сторінки було створено окремий клас, у якому описані елементи сторінки та методи взаємодії з ними.

3. Реалізація тестового сценарію: було розроблено тест, який використовує методи класів Page Object для перевірки повного циклу оформлення замовлення, від авторизації до підтвердження.

4. Тестування та оптимізація: тест був виконаний кілька разів для перевірки коректності реалізації та стабільності роботи. Було проведено оптимізацію локаторів і методів для покращення читабельності та продуктивності.

Для виконання тесту були створені такі класи сторінок: LoginPage, HomePage, CartPage, CheckoutInputPage, CheckoutPage, CheckoutConfirmPage, TopPart.

Результатом проєктування тесту для оформлення замовлення є діаграма класів зображена на рисунку 16.

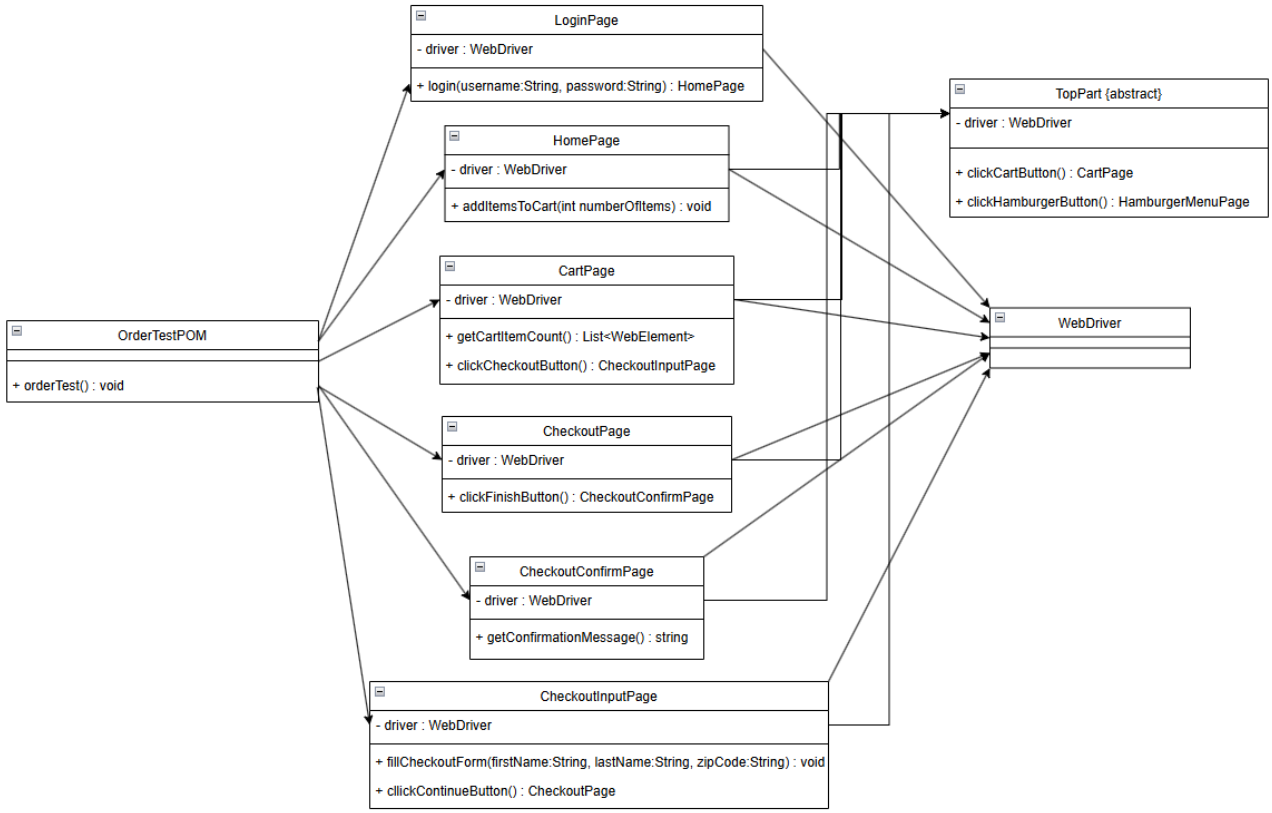



Рисунок 16 – Діаграма класів для тесту оформлення замовлення

Лістинг 8 – Абстрактний клас TopPart

```
package pages;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;

public abstract class TopPart {
    protected WebDriver driver;

    @FindBy(id = "react-burger-menu-btn")
    private WebElement hamburgerButton;

    @FindBy(className = "app_logo")
```

```
    private WebElement siteTitle;

    @FindBy(className = "shopping_cart_link")
    private WebElement cartButton;

    public TopPart(WebDriver driver) {
        this.driver = driver;
        PageFactory.initElements(driver, this);
    }

    public CartPage clickCartButton() {
        cartButton.click();
        return new CartPage(driver);
    }

    public HamburgerMenuPage clickHamburgerButton() {
        hamburgerButton.click();
        return new HamburgerMenuPage(driver);
    }
}
```

Лістинг 9 – Клас LoginPage для авторизації

```
package pages;

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

public class LoginPage {
    private WebDriver driver;

    private By usernameField = By.id("user-name");
    private By passwordField = By.id("password");
```

```
    private By loginButton = By.id("login-button");
    //private By errorMessage = By.cssSelector("[data-
test='error']");

    public LoginPage(WebDriver driver) {
        this.driver = driver;
        if (!driver.getTitle().contains("Swag Labs")) {
            throw new IllegalStateException("This is not the
Login Page. Current page: " + driver.getCurrentUrl());
        }
    }

    public HomePage login(String username, String password) {
        WebElement usernameElement =
driver.findElement(usernameField);
        WebElement passwordElement =
driver.findElement(passwordField);
        WebElement loginButtonElement =
driver.findElement(loginButton);

        usernameElement.sendKeys(username);
        passwordElement.sendKeys(password);
        loginButtonElement.click();

        return new HomePage(driver);
    }

//    public boolean isErrorMessageDisplayed() {
//        try {
//            return
driver.findElement(errorMessage).isDisplayed();
//        } catch (Exception e) {
//            return false;
```

```
//         }
//     }
}
```

Лістинг 10 – Клас HomePage для головної сторінки

```
package pages;

import components.ProductSorter;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;

import java.util.List;

public class HomePage extends TopPart {

    @FindBy(className = "btn_inventory")
    private List<WebElement> addToCartButtons;

    @FindBy(xpath = "//*[@id=\"item_4_img_link\"]")
    private WebElement firstProduct;

    private ProductSorter productSorter;

    public HomePage(WebDriver driver) {
        super(driver);
        productSorter = new ProductSorter(driver);
        PageFactory.initElements(driver, this);
    }
```

```
    public void addItemsToCart(int numberOfItems) {
        for (int i = 0; i < numberOfItems; i++) {
            addToCartButtons.get(i).click();
        }
    }

    public ProductPage clickFirstProduct() {
        firstProduct.click();
        return new ProductPage(driver);
    }

    public void sortProductsAlphabeticallyAZ() {
        productSorter.sortAlphabeticallyAZ();
    }

    public void sortProductsAlphabeticallyZA() {
        productSorter.sortAlphabeticallyZA();
    }

    public List<WebElement> getProductNames() {
        return
driver.findElements(By.className("inventory_item_name"));
    }

    public boolean isSortedAlphabeticallyAZ() {
        return
productSorter.areProductsSortedAlphabeticallyAZ();
    }

    public boolean isSortedAlphabeticallyZA() {
        return
productSorter.areProductsSortedAlphabeticallyZA();
    }
```

```
    public void sortProductsByLowToHighPrice() {
        productSorter.sortByPriceLowToHigh();
    }

    public void sortProductsByHighToLowPrice() {
        productSorter.sortByPriceHighToLow();
    }

    public boolean arePricesSortedLowToHighPrice() {
        return productSorter.arePricesSortedLowToHigh();
    }

    public boolean arePricesSortedHighToLowPrice() {
        return productSorter.arePricesSortedHighToLow();
    }

    public boolean areAtLeastTwoProductsForPriceComparison()
{
        List<WebElement> products = getProductNames();
        return products.size() >= 2;
    }

    public String getCurrentUrl() {
        return driver.getCurrentUrl();
    }
}
```

Лістинг 11 – Клас CartPage для сторінки кошика

```
package pages;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
```

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

import java.util.List;

public class CartPage extends TopPart {

    // Локатор для елементів у кошику (тепер використовуємо
FindBy для кнопок)
    @FindBy(className = "cart_item")
    private List<WebElement> cartItems;

    @FindBy(className = "checkout_button")
    private WebElement checkoutButton;

    @FindBy(css = ".cart_button")
    private List<WebElement> removeButtons;

    public CartPage(WebDriver driver) {
        super(driver);
        PageFactory.initElements(driver, this);
    }

    public List<WebElement> getCartItems() {
        return cartItems;
    }

    public void removeAllItems() {
        for (WebElement removeButton : removeButtons) {
            removeButton.click();
        }
    }
```

```
    public CheckoutInputPage clickCheckoutButton() {
        checkoutButton.click();
        return new CheckoutInputPage(driver);
    }
}
```

Лістинг 12 – Клас CheckoutInputPage для сторінки із заповненням даних для замовлення

```
package pages;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;

public class CheckoutInputPage extends TopPart {

    @FindBy(id = "first-name")
    private WebElement firstNameField;

    @FindBy(id = "last-name")
    private WebElement lastNameField;

    @FindBy(id = "postal-code")
    private WebElement zipCodeField;

    @FindBy(className = "btn_primary")
    private WebElement continueButton;

    @FindBy(css = "[data-test='error']")
    private WebElement errorMessage;

    public CheckoutInputPage(WebDriver driver) {
```

```
        super(driver);
        PageFactory.initElements(driver, this);
    }

    public void fillCheckoutForm(String firstName, String
lastName, String zipCode) {
        firstNameField.sendKeys(firstName);
        lastNameField.sendKeys(lastName);
        zipCodeField.sendKeys(zipCode);
    }

    public CheckoutPage clickContinueButton() {
        continueButton.click();
        return new CheckoutPage(driver);
    }

    public String getErrorMessage() {
        return errorMessage.getText();
    }
}
```

Лістинг 13 – Клас CheckoutPage для сторінки з деталями замовлення

```
package pages;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;

import java.util.List;

public class CheckoutPage extends TopPart {
```

```
    @FindBy(className = "btn_action")
    private WebElement finishButton;

    @FindBy(className = "cart_item")
    private List<WebElement> checkoutItems;

    @FindBy(className = "back")
    private WebElement canselButton;
    public CheckoutPage(WebDriver driver) {
        super(driver);
        PageFactory.initElements(driver, this);
    }
    public CheckoutConfirmPage clickFinishButton() {
        finishButton.click();
        return new CheckoutConfirmPage(driver);
    }
    public List<WebElement> getCheckoutPageItems() {
        return checkoutItems;
    }
    public void clickCanselButton() {
        canselButton.click();
    }
}
```

Лістинг 14 – Клас CheckoutConfirmPage для сторінки з підтвердженням замовлення

```
package pages;

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

```
public class CheckoutConfirmPage extends TopPart {

    @FindBy(className = "complete-header")
    private WebElement confirmationMessage;

    @FindBy(id = "back-to-products")
    private WebElement backHomeButton;

    public CheckoutConfirmPage(WebDriver driver) {
        super(driver);
        PageFactory.initElements(driver, this);
    }

    public String getConfirmationMessage() {
        return confirmationMessage.getText();
    }

    public void clickBackHomeButton() {
        backHomeButton.click();
    }
}
```

Опис реалізації вище зазначених класів:

TopPart: абстрактний клас, який представляє верхню частину веб-сторінки з основними елементами інтерфейсу, такими як кнопка бургер-меню, назва сайту та кнопка кошика. Клас використовує анотацію @FindBy для знаходження елементів на сторінці: кнопка гамбургера, логотип сайту та кнопка кошика. Конструктор класу ініціалізує ці елементи за допомогою PageFactory.initElements, що дозволяє автоматично прив’язати елементи до відповідних полів класу. Клас надає методи clickCartButton і clickHamburgerButton, які виконують кліки на відповідні кнопки і повертають відповідні сторінки (CartPage і HamburgerMenuPage), що сприяє навігації між

різними частинами застосунку. Цей клас слугує для абстрагування і взаємодії з постійними елементами на всіх сторінках, що мають верхню частину інтерфейсу.

LoginPage: представляє сторінку авторизації веб-застосунку. Він містить елементи для введення даних користувача, такі як поля для введення імені користувача та пароля, а також кнопку для входу. За допомогою анотацій @FindBy та PageFactory.initElements ці елементи ініціалізуються при створенні об'єкта класу. Конструктор класу перевіряє, чи знаходиться веб-драйвер на правильній сторінці, перевіряючи її заголовок, і якщо це не сторінка входу, викидається виключення IllegalStateException.

Метод login дозволяє здійснити процес авторизації, передаючи параметри для імені користувача та пароля. Якщо при спробі входу з'являється повідомлення про помилку (неправильний логін або пароль), метод виводить відповідне повідомлення і повертає null. У разі успішного входу метод повертає новий об'єкт сторінки HomePage. Метод isErrorMessageDisplayed використовується для перевірки наявності повідомлення про помилку на сторінці, яке може виникнути при невірних даних для входу.

HomePage: розширює клас TopPart і представляє головну сторінку веб-застосунку. Він містить кілька елементів сторінки, таких як кнопки для додавання товарів до кошика, а також перший товар, на який можна клікнути для перегляду детальної інформації. Клас також інтегрує компонент ProductSorter, який надає можливість сортувати продукти за різними критеріями, такими як алфавітний порядок або ціна.

Метод addItemsToCart дозволяє додавати кілька товарів до кошика, передаючи кількість товарів, які потрібно додати, і клікаючи по відповідним кнопкам для кожного товару. Метод clickFirstProduct переходить на сторінку продукту, клікаючи на перший товар у списку.

Клас містить кілька методів для сортування товарів за алфавітним порядком: sortProductsAlphabeticallyAZ, sortProductsAlphabeticallyZA та за ціною: sortProductsByLowToHighPrice, sortProductsByHighToLowPrice,

використовуючи методи компонента ProductSorter. Також є методи для перевірки, чи правильно відсортовані продукти, такі як isSortedAlphabeticallyAZ та isSortedAlphabeticallyZA для алфавітного сортування, а також arePricesSortedLowToHighPrice та arePricesSortedHighToLowPrice для перевірки сортування за ціною.

Метод areAtLeastTwoProductsForPriceComparison перевіряє, чи є на сторінці хоча б два продукти для порівняння ціни. Цей клас інтегрує важливі функціональні можливості для роботи з товарами, кошиком і сортуванням на головній сторінці.

CartPage: розширює клас TopPart і представляє сторінку кошика. Він містить елементи сторінки, які дозволяють користувачеві взаємодіяти з товарами в кошику, такими як кнопки для видалення товарів та кнопка для переходу до оформлення замовлення.

Метод getCartItems повертає список елементів кошика, що дозволяє перевіряти, які товари є в кошику. Метод removeAllItems дозволяє видалити всі товари з кошика, натискаючи на кожну кнопку видалення. Це забезпечує можливість очищення кошика перед оформленням нового замовлення чи іншими операціями.

Метод clickCheckoutButton відповідає за перехід на сторінку введення даних для оформлення замовлення, натискаючи на кнопку "Checkout". Після кліку цей метод створює і повертає новий об'єкт класу CheckoutInputPage, що дозволяє продовжити процес оформлення замовлення.

CheckoutInputPage: розширює TopPart і представляє сторінку введення даних для оформлення замовлення. Він містить поля введення для імені, прізвища та поштового індексу, а також кнопку для продовження оформлення замовлення.

Метод fillCheckoutForm дозволяє заповнити форму введення даних, приймаючи три параметри: ім'я, прізвище та поштовий індекс, і автоматично заповнює відповідні поля форми. Це забезпечує можливість введення даних для замовлення без необхідності ручного введення на сторінці.

Метод clickContinueButton натискає кнопку "Continue" і після цього переходить на сторінку CheckoutPage, де користувач може завершити оформлення замовлення. Повертає новий об'єкт класу CheckoutPage, що дозволяє продовжити процес замовлення.

Метод getErrorMessage повертає текст помилки, який може відображатися на сторінці, якщо введені дані некоректні або відсутні. Цей метод дає змогу перевірити, чи була помилка при введенні даних на сторінці.

CheckoutPage: розширює TopPart і відповідає за взаємодію з елементами на сторінці підтвердження замовлення. Він містить кілька елементів, таких як кнопка для завершення покупки, список товарів на сторінці та кнопка для скасування.

Метод clickFinishButton натискає на кнопку "Finish", яка підтверджує завершення процесу покупки. Після натискання кнопки, користувача перенаправляє на сторінку підтвердження замовлення, де він може переглянути деталі покупки та отримати підтвердження. Цей метод повертає об'єкт класу CheckoutConfirmPage, що дозволяє переходити до наступного етапу після завершення покупки.

Метод getCheckoutPageItems повертає список всіх товарів, які додаються в кошик на стадії оформлення замовлення, даючи можливість тестувати чи всі товари, які повинні бути на сторінці, дійсно відображаються.

Метод clickCanselButton натискає на кнопку "Cancel", що дозволяє скасувати процес оформлення замовлення та повернутися на попередню сторінку, наприклад, на сторінку з кошиком.

CheckoutConfirmPage: розширює TopPart та містить два основні елементи: повідомлення про успішну покупку та кнопку для повернення на головну сторінку. Повідомлення про успішну покупку зберігається в елементі confirmationMessage, і метод getConfirmationMessage дозволяє отримати текст цього повідомлення для перевірки, чи було завершення покупки успішним. Крім того, на сторінці є кнопка backHomeButton, яка дає змогу користувачу повернутися на головну сторінку з товарами. Метод clickBackHomeButton

натискає цю кнопку, що дозволяє тестувати сценарії, де після підтвердження покупки користувач може повернутися до основної сторінки для подальших покупок.

Для виконання тесту був створений клас OrderTestPOM. У класі OrderTestPOM реалізовано тест для перевірки процесу оформлення замовлення на веб-сайті. Тест починається з того, що завантажується сторінка входу за допомогою методу loadApplication(), після чого виконується логін з використанням стандартних даних користувача: ім'я: standard_user, пароль: secret_sauce. Після успішного входу перевіряється, чи не є сторінка головною сторінкою HomePage, що підтверджується через Assert.assertNotNull().

Далі додаються два товари до кошика. Після цього переходимо до сторінки кошика CartPage за допомогою кнопки кошика і перевіряємо кількість товарів в кошику за допомогою getCartItems().size(), порівнюючи її з очікуваним значенням (2). Далі відбувається перехід до сторінки CheckoutInputPage, де заповнюється форма з персональними даними користувача, після чого натискається кнопка continueButton для переходу до сторінки огляду замовлення CheckoutPage. На цій сторінці також перевіряється, чи кількість товарів на сторінці огляду замовлення відповідає кількості товарів у кошику.

Тест також перевіряє можливість скасування процесу оформлення, натискаючи на кнопку cancelButton. Після цього тест продовжує оформлення замовлення і переходить до CheckoutConfirmPage, де перевіряється наявність повідомлення про успішне замовлення за допомогою getConfirmationMessage(). Після підтвердження успішного замовлення натискається кнопка повернення на головну сторінку clickBackHomeButton().

Цей тест перевіряє весь процес покупки від входу на сайт до підтвердження замовлення, забезпечуючи впевненість у правильності функціонування основних етапів оформлення замовлення на сайті.

Лістинг 15 – Тест оформлення замовлення

```
package test;
```

```
import org.testng.Assert;
import org.testng.annotations.Test;
import pages.*;

public class OrderTestPOM extends TestRunnerFirst {

    @Test
    public void orderTest() {
        LoginPage loginPage = loadApplication();
        HomePage homePage = loginPage.login("standard_user",
"secret_sauce");
        Assert.assertNotNull(homePage, "Test failed: Wrong
username or password.");
        homePage.addItemsToCart(2);
        CartPage cartPage = homePage.clickCartButton();
        int cartItemCount = cartPage.getCartItems().size();
        Assert.assertEquals(cartItemCount, 2, "Expected 2
items in the cart.");

        CheckoutInputPage checkoutInputPage =
cartPage.clickCheckoutButton();
        checkoutInputPage.fillCheckoutForm("John", "Doe",
"12345");
        CheckoutPage checkoutPage =
checkoutInputPage.clickContinueButton();
        int checkoutItemCount =
checkoutPage.getCheckoutPageItems().size();
        Assert.assertEquals(checkoutItemCount, cartItemCount,
"Expected 2 items on the checkout overview page.");
        checkoutPage.clickCanselButton();
        homePage.clickCartButton();
        cartPage.clickCheckoutButton();
```

```
        checkoutInputPage.fillCheckoutForm("John", "Doe",
"12345");
        checkoutInputPage.clickContinueButton();
        CheckoutConfirmPage checkoutConfirmPage =
checkoutPage.clickFinishButton();
        String confirmationMessage =
checkoutConfirmPage.getConfirmationMessage();
        Assert.assertEquals(confirmationMessage, "Thank you
for your order!", "Order confirmation message not displayed
correctly.");
        checkoutConfirmPage.clickBackHomeButton();
    }
}
```

Архітектура класу OrderTestPOM демонструє переваги підходу POM, де кожен етап взаємодії із застосунком ізольований у відповідних класах Page Object. Це не лише підвищує зручність читання та модифікації тестів, але й забезпечує можливість масштабування та повторного використання коду.

Результатом реалізації став набір автоматизованих тестів, які перевіряють основний функціонал Sauce Demo з використанням POM. Це дозволило забезпечити зручність тестів, а також можливість легко адаптувати проєкт до змін у веб-застосунку.

# 4 ВИКОРИСТАННЯ PAGE FACTORY ПРИ ОРГАНІЗАЦІЇ ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКУ

## 4.1 Використання класу PageFactory фреймворку Selenium WebDriver для підтримки автоматизованих тестів, побудованих із використанням POM

Клас PageFactory у фреймворку Selenium WebDriver є ключовим інструментом для ефективної реалізації шаблону Page Object Model. Його основна мета полягає в автоматизації ініціалізації веб-елементів на сторінках веб-застосунків, що значно підвищує зручність та надійність роботи з цими елементами в автоматизованих тестах. Використання PageFactory дозволяє спростити процес створення класів сторінок, уникнути дублювання коду та зробити архітектуру проєкту більш організованою.

Основним принципом роботи PageFactory є використання анотацій, таких як @FindBy, для визначення локаторів веб-елементів. Ці анотації надають гнучкість у способах знаходження елементів, дозволяючи використовувати різні механізми пошуку, такі як ідентифікатори, класи, CSS-селектори чи XPath [19]. При цьому самі веб-елементи автоматично ініціалізуються під час створення об’єкта класу сторінки. Це означає, що розробникам тестів більше не потрібно вручну викликати методи для знаходження елементів у DOM-структурі, що економить час та зменшує ризик помилок, пов’язаних із некоректною роботою локаторів.

Використання PageFactory є невід'ємною частиною структурованого тестування з використанням POM. Шаблон POM передбачає розділення логіки взаємодії з веб-сторінками від самої тестової логіки, а PageFactory ідеально доповнює цей підхід. Завдяки автоматичній ініціалізації веб-елементів тестова логіка стає менш громіздкою та більш зосередженою на перевірці бізнес-логіки застосунку. Це значно полегшує рефакторинг і підтримку проєктів, особливо

якщо тестова система повинна обробляти часті зміни у DOM-структурі веб-застосунку.

Окрім спрощення роботи з локаторами, PageFactory підтримує механізми відкладеної ініціалізації (lazy loading) [20]. Це означає, що веб-елементи ініціалізуються лише тоді, коли до них здійснюється перше звернення. Такий підхід дозволяє уникнути зайвого навантаження на систему та забезпечує кращу продуктивність тестів. Наприклад, якщо у класі сторінки оголошено десятки веб-елементів, але в певному тестовому сценарії використовується лише декілька з них, то лише ці елементи будуть знайдені та завантажені у пам’ять.

У довгостроковій перспективі використання PageFactory сприяє кращій масштабованості та підтримуваності проєктів. Впровадження цього підходу спрощує впровадження нових тестових сценаріїв. Завдяки структурованій організації коду, де кожна сторінка веб-застосунку має окремий клас із чітко визначеними методами та елементами, тестова архітектура стає більш прозорою і зрозумілою навіть для нових членів команди.

Таким чином, PageFactory є потужним інструментом у наборі засобів автоматизатора тестування, який дозволяє поєднати простоту реалізації з високою ефективністю. Це робить його оптимальним вибором для створення масштабованих та надійних тестових проєктів, побудованих із використанням шаблону POM.

## 4.2 Рефакторинг проєкту з POM та Selenium WebDriver за допомогою PageFactory

Рефакторинг проєкту, побудованого на основі шаблону Page Object Model, за допомогою класу PageFactory дозволяє значно покращити структуру тестів та зменшити обсяг ручної роботи при визначенні і взаємодії з веб-елементами. Використання PageFactory забезпечує автоматичну ініціалізацію елементів, що сприяє підвищенню зручності, ефективності та читабельності коду.

Як приклад розглянемо рефакторинг класу CheckoutInputPage, який є частиною архітектури тестового проєкту. У початковій реалізації методи для взаємодії з полями вводу та кнопками спираються на явне використання локаторів через метод findElement(). Це призводить до необхідності часто повторювати цей виклик у коді, що не тільки збільшує обсяг коду, але й підвищує ризик помилок під час роботи з локаторами. Після впровадження PageFactory в клас CheckoutInputPage структура змінилася, забезпечивши більшу компактність. Завдяки анотаціям @FindBy кожен веб-елемент оголошується і ініціалізується безпосередньо в полі класу. Наприклад, замість викликів findElement() для кожного поля форми, таких як ім’я, прізвище або поштовий код, ці елементи були оголошені через відповідні анотації. Це дозволяє одночасно забезпечити декларативний підхід до визначення локаторів і усунути дублювання коду.

У конструкції класу CheckoutInputPage додано виклик методу PageFactory.initElements(driver, this), що забезпечує автоматичну ініціалізацію всіх веб-елементів, позначених анотацією @FindBy. Такий підхід значно спрощує розширення функціональності класу. Наприклад, додавання нових елементів для взаємодії на сторінці більше не вимагає змін в інших частинах коду, що підвищує модульність і гнучкість проєкту.

Методи класу також стали більш чистими і зосередженими на бізнес-логіці. Наприклад, метод fillCheckoutForm дозволяє легко заповнити форму, використовуючи поля firstNameField, lastNameField і zipCodeField. Усі ці поля попередньо ініціалізовані, що усуває необхідність викликати findElement у кожному з них. Завдяки цьому зменшується кількість рядків коду, підвищується його читабельність і знижується ймовірність помилок у тестах, пов’язаних із некоректними локаторами.

Крім того, впровадження PageFactory підвищує продуктивність у сценаріях із багатьма елементами. Наприклад, у класі CheckoutInputPage, навіть якщо сторінка має кілька десятків елементів, вони будуть ініціалізовані лише тоді,

коли це дійсно необхідно. Це забезпечує кращу ефективність роботи тестів, оскільки ресурси використовуються тільки за потребою.

Лістинг 16 – Клас CheckoutInputPage після рефакторингу із використанням PageFactory

```
package pages;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;

public class CheckoutInputPage extends TopPart {

    @FindBy(id = "first-name")
    private WebElement firstNameField;

    @FindBy(id = "last-name")
    private WebElement lastNameField;

    @FindBy(id = "postal-code")
    private WebElement zipCodeField;

    @FindBy(className = "btn_primary")
    private WebElement continueButton;

    @FindBy(css = "[data-test='error']")
    private WebElement errorMessage;

    public CheckoutInputPage(WebDriver driver) {
        super(driver);
        PageFactory.initElements(driver, this);
```

```
    }

    public void fillCheckoutForm(String firstName, String
lastName, String zipCode) {
        firstNameField.sendKeys(firstName);
        lastNameField.sendKeys(lastName);
        zipCodeField.sendKeys(zipCode);
    }

    public CheckoutPage clickContinueButton() {
        continueButton.click();
        return new CheckoutPage(driver);
    }

    public String getErrorMessage() {
        return errorMessage.getText();
    }
}
```

Таким чином, рефакторинг із використанням PageFactory сприяє створенню кращої архітектури тестів. Завдяки такому підходу автоматизовані тести стають більш модульними, простими для розуміння та легкими у підтримці. Це дозволяє підвищити продуктивність роботи тестувальників, скоротити час розробки нових тестових сценаріїв і полегшити впровадження змін у веб-застосунку.

## 4.3 Виконання порівняльного аналізу проєктів з автоматизованими тестами без та з використанням POM

Автоматизоване тестування веб-застосунків є важливим етапом забезпечення якості, проте його ефективність і зручність підтримки значно залежать від обраного підходу до побудови тестової архітектури. Розглянемо та

проаналізуємо два підходи: тестування без використання шаблону Page Object Model та тестування, побудованого на основі Page Object Model.

У випадку тестування без використання POM, тестові сценарії часто мають монолітну структуру, де всі дії, локатори елементів, перевірки й бізнес-логіка тісно переплетені між собою. Це створює проблему надмірної залежності тестів від змін у веб-застосунку, оскільки будь-яка модифікація інтерфейсу призводить до необхідності оновлення всіх тестів, що залежать від відповідного елемента. Наприклад, якщо змінюється ідентифікатор кнопки, доведеться вручну змінити локатор у кожному тесті, що використовує цю кнопку. Такий підхід обмежує можливості масштабування та уповільнює процес розробки. Для прикладу приведений тест SortingByNameTest без використання Page Object Model.

Лістинг 17 – Тест SortingByNameTest без використання POM

```
package test;

import io.github.bonigarcia.wdm.WebDriverManager;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.*;

import java.util.List;

public class SortingByNameTest {
    private WebDriver driver;

    @BeforeTest
    public void setup() {
        WebDriverManager.chromedriver().setup();
        driver = new ChromeDriver();
```

```
        driver.get("https://www.saucedemo.com");
        driver.manage().window().maximize();
    }

    @Test(priority = 1)
    public void testLogin() {
        WebElement username = driver.findElement(By.id("user-
name"));
        WebElement password =
driver.findElement(By.id("password"));
        WebElement loginButton =
driver.findElement(By.id("login-button"));

        username.sendKeys("standard_user");
        password.sendKeys("secret_sauce");
        loginButton.click();

        boolean isErrorDisplayed = isErrorMessageDisplayed();
        Assert.assertFalse(isErrorDisplayed, "Login failed.
Incorrect username or password.");
    }

    @Test(priority = 2, dependsOnMethods = "testLogin")
    public void testSortProductsAlphabetically() {
        sortProducts("az");

        List<WebElement> productNames =
driver.findElements(By.className("inventory_item_name"));

        boolean isSortedAlphabetically = true;
        for (int i = 0; i < productNames.size() - 1; i++) {
            String currentProductName =
productNames.get(i).getText();
```

```
            String nextProductName = productNames.get(i +
1).getText();

            if (currentProductName.compareTo(nextProductName)
> 0) {
                isSortedAlphabetically = false;
                break;
            }
        }

        Assert.assertTrue(isSortedAlphabetically, "Test
Failed: Products are not sorted alphabetically (A to Z).");
        System.out.println("Test Passed: Products are sorted
alphabetically (A to Z).");
    }

    @Test(priority = 3, dependsOnMethods = "testLogin")
    public void testSortProductsReverseAlphabetically() {
        sortProducts("za");

        List<WebElement> productNames =
driver.findElements(By.className("inventory_item_name"));

        boolean isSortedReverseAlphabetically = true;
        for (int i = 0; i < productNames.size() - 1; i++) {
            String currentProductName =
productNames.get(i).getText();
            String nextProductName = productNames.get(i +
1).getText();

            if (currentProductName.compareTo(nextProductName)
< 0) {
                isSortedReverseAlphabetically = false;
```

```
                break;
            }
        }

        Assert.assertTrue(isSortedReverseAlphabetically,
"Test Failed: Products are not sorted reverse alphabetically
(Z to A).");
        System.out.println("Test Passed: Products are sorted
reverse alphabetically (Z to A).");
    }

    @AfterTest
    public void teardown() {
        if (driver != null) {
            driver.quit();
        }
    }

    private boolean isErrorMessageDisplayed() {
        try {
            WebElement errorMessage =
driver.findElement(By.cssSelector("[data-test='error']"));
            return errorMessage.isDisplayed();
        } catch (Exception e) {
            return false;
        }
    }

    private void sortProducts(String sortOrder) {
        WebElement sortDropdown =
driver.findElement(By.className("product_sort_container"));
        sortDropdown.click();
```

```
        WebElement sortOption =
driver.findElement(By.xpath("//option[@value='" + sortOrder +
"']"));
        sortOption.click();
    }
}
```

Шаблон POM пропонує вирішення цих проблем завдяки створенню окремих класів для кожної сторінки чи її логічних частин. Ці класи відповідають за управління елементами сторінки та надають методи, що відображають бізнес-логіку. Завдяки цьому тести стають більш модульними: будь-яка зміна на сторінці вимагає лише оновлення відповідного класу, тоді як тести, що викликають ці методи, залишаються незмінними. Такий підхід значно скорочує обсяг роботи при рефакторингу. Нижче наведений тест SortingByNameTestPOM, але вже з використанням Page Object Model.

Лістинг 18 – Клас Тест SortingByNameTestPOM із використанням Page Object Model

```
package test;

import org.testng.Assert;
import org.testng.annotations.Test;
import pages.*;

public class SortingByNameTestPOM extends TestRunnerFirst {

    @Test
    public void sortingByNameTest() {
        LoginPage loginPage = loadApplication();
        HomePage homePage = loginPage.login("standard_user",
"secret_sauce");
```

```
        Assert.assertNotNull(homePage, "Test failed: Wrong
username or password.");

        homePage.sortProductsAlphabeticallyAZ();
        boolean isSortedAlphabetically =
homePage.isSortedAlphabeticallyAZ();
        Assert.assertTrue(isSortedAlphabetically, "Test
Failed: Products are not sorted alphabetically.");
        System.out.println("Test Passed: Products are sorted
alphabetically from A to Z.");

        homePage.sortProductsAlphabeticallyZA();
        isSortedAlphabetically =
homePage.isSortedAlphabeticallyZA();
        Assert.assertTrue(isSortedAlphabetically, "Test
Failed: Products are not sorted alphabetically.");
        System.out.println("Test Passed: Products are sorted
alphabetically from Z to A.");
    }
}
```

Щодо швидкості виконання тестів (Таблиця 1), відмінності між підходами можуть бути мінімальними, проте реальний виграш від POM проявляється у прискоренні процесу розробки, підтримки та масштабування тестів. Витрати часу на розробку нового тесту без POM зростають експоненціально зі збільшенням кількості функцій, тоді як у проєкті з POM ці витрати залишаються лінійними завдяки можливості повторного використання вже існуючих компонентів.

Таблиця 1 – Порівняння швидкості виконання тестів з POM та без POM

| Назва тесту | Середня швидкість виконання тесту без POM | Середня швидкість виконання тесту з POM |
|---|---|---|
| `LoginTest` | 2 sec 600 ms | 2 sec 150 ms |
| `OrderTest` | 2 sec 600 ms | 2 sec 900 ms |
| `ProductDetailsTest` | 2 sec 200 ms | 2 sec 300 ms |
| `SortingByNameTest` | 2 sec 400 ms | 2 sec 520 ms |
| `SortingByPriceTest` | 2 sec 500 ms | 2 sec 600 ms |
| `AddItemsToCartTest` | 2 sec 600 ms | 2 sec 500 ms |
| `InputCheckoutTest` | 2 sec 500 ms | 2 sec 700 ms |

Окремо слід зазначити можливості рефакторингу, які надає POM. У системах, де застосовується цей шаблон, локатори та бізнес-логіка ізольовані в окремих класах. Це означає, що при зміні, наприклад, дизайну сторінки, необхідно оновити лише відповідний клас сторінки, а тестові сценарії залишаться незмінними. У проєктах без POM рефакторинг є складнішим, оскільки локатори знаходяться по всьому коду тестів, і кожен із них потребує ручного оновлення.

Таблиця 2 – Часу на рефакторинг при зміні елементів веб-інтерфейсу

| Тпи змін на сторінці | Без POM | З POM |
|---|---|---|
| Зміна локаторів на одній сторінці | Необхідно змінювати всі тести, що використовують ці елементи | Зміни торкаються тільки відповідного класу сторінки |
| Зміна бізнес-логіки | Потрібно змінювати всі тести, пов'язані з логікою | Зміни впливають лише на логіку відповідного класу |
| Додавання нового тесту | Потрібно створювати нові локатори та перевірки в кожному тесті | Можна використовувати існуючі сторінки та компоненти |

Модульність тестів, створених із використанням POM, також значно перевищує модульність тестів, побудованих без цього підходу. Завдяки використанню окремих класів для кожної сторінки забезпечується чітке розмежування між бізнес-логікою тесту і технічними деталями роботи з веб-інтерфейсом. Це підвищує зрозумілість тестів і дозволяє іншим членам команди легко адаптуватися до структури проєкту.

Таблиця 3 – Порівняння модульності тестів

| Параметр | Без POM | З POM |
|---|---|---|
| Ізоляція змін | Зміни потребують коригування в багатьох тестах | Зміни обмежуються лише відповідним класом сторінки |
| Можливість повторного використання | Низька, через дублювання коду | Висока, компоненти можуть бути повторно використані в інших тестах |
| Гнучкість при масштабуванні | Низька, важко додавати нові сторінки/тести | Висока, можна швидко додавати нові сторінки і компоненти |

Таким чином, аналіз показує, що хоча проєкти з тестами, побудованими без POM, можуть бути доцільними для простих застосунків із невеликою кількістю тестових сценаріїв, вони швидко стають неефективними в умовах масштабування. Використання шаблону POM забезпечує низку переваг: зменшення часу на рефакторинг, підвищення модульності та зручності підтримки, а також оптимізацію процесу розробки. У результаті, навіть із врахуванням початкових витрат часу на проєктування архітектури, POM є незамінним інструментом для створення ефективного й гнучкого процесу автоматизованого тестування.

# ВИСНОВКИ

1. В процесі виконання роботи було розглянуто та досліджено основи автоматизованого тестування веб-застосунків за допомогою фреймворку Selenium WebDriver, а також застосування шаблону проектування Page Object Model для організації тестів.
2. В рамках дослідження було розглянуто використання класу PageFactory для ініціалізації елементів веб-сторінок, що дозволяє значно зменшити дублювання коду, покращити читабельність і зручність підтримки тестів. Було продемонстровано, як застосування цього підходу сприяє скороченню часу на розробку і підтримку тестових сценаріїв.
3. Проведено рефакторинг тестів для демонстраційного веб-застосунку, зокрема з використанням класу PageFactory, що забезпечує більш чисту та зручну структуру тестового коду.
4. У результаті виконання порівняльного аналізу між тестами, побудованими без використання POM, та тестами, реалізованими із застосуванням цього шаблону, були виявлені значні переваги в плані швидкості розробки, модульності, зручності рефакторингу та підтримки тестів. Використання POM дозволяє скоротити час на внесення змін, підвищує ефективність процесу тестування і дає можливість легше масштабувати проєкт.
5. Зважаючи на результати дослідження, можна стверджувати, що застосування Page Object Model є критично важливим для організації ефективного процесу автоматизованого тестування веб-застосунків, зокрема в умовах великого масштабу проєктів, де важлива підтримка та швидкість розробки нових тестів.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Java testing patterns / J. Thomas et al. Wiley & Sons, Incorporated, John, 2008 P. 320.

2. Parsons D. Unit Testing with JUnit. *Foundational Java*. London, 2012. P. 219–244.

3. WebDriver. *Selenium*. URL: https://www.selenium.dev/documentation/webdriver/ (дата звернення: 05.09.2024).

4. Testing frameworks for javascript | write, run, debug | cypress. *Testing Frameworks for Javascript | Write, Run, Debug | Cypress*. URL: https://www.cypress.io/ (дата звернення: 05.09.2024).

5. Installation | Playwright. *Fast and reliable end-to-end testing for modern web apps | Playwright*. URL: https://playwright.dev/docs/intro (дата звернення: 05.09.2024).

6. Getting Started | Overview | Guides. *Cross-Browser End-to-End Testing Framework | TestCafe*. URL: https://testcafe.io/documentation/402635/guides/overview/getting-started (дата звернення: 14.09.2024).

7. A Brief History of the Selenium Testing Framework › TESTINGMIND. *TESTINGMIND › Connecting QA Professionals Globally*. URL: https://www.testingmind.com/a-brief-history-of-the-selenium-testing-framework/ (дата звернення: 14.09.2024).

8. Selenium webdriver в автоматизації тестування. *Онлайн-курси від компанії QATestLab | Головна сторінка*. URL: https://training.qatestlab.com/blog/technical-articles/selenium-webdriver/ (дата звернення: 15.09.2024).

9. Mastering Selenium WebDriver. Packt Publishing, 2015. 280 p.

10. Ramya P., Sindhura V., Sagar P. V. Testing using selenium web driver. *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, Coimbatore, 22–24 February 2017. 2017.

11. findElement and findElements in Selenium: Examples | BrowserStack. *BrowserStack*.
URL: https://www.browserstack.com/guide/findelement-in-selenium (дата звернення: 05.10.2024).
12. The Complete Guide to Handling Web Elements in Selenium. *#1 Unified, Gen-AI Cloud-Based Test Automation Platform*. URL: https://testsigma.com/blog/web-element-in-selenium/ (дата звернення: 05.10.2024).
13. Swag Labs. *Swag Labs*. URL: https://www.saucedemo.com/ (дата звернення: 14.10.2024).
14. Bruns A., Kornstadt A., Wichmann D. Web Application Tests with Selenium. *IEEE Software*. 2009. Vol. 26, no. 5. P. 88–91.
15. About Sauce Labs. URL: https://saucelabs.com/company/about-us (дата звернення: 14.10.2024).
16. Raghavendra S. Page Object Model (POM). *Java Testing with Selenium*. Berkeley, CA, 2024. P. 261–284.
17. Chaubal P. Page Object Model Using Selenium WebDriver and Java. Independently Published, 2018.
18. TestNG Documentation. *TestNG Documentation*. URL: https://testng.org/ (дата звернення: 20.10.2024).
19. Page Object Model and Page Factory in Selenium | BrowserStack. *BrowserStack*. URL: https://www.browserstack.com/guide/page-object-model-in-selenium (дата звернення: 05.11.2024).
20. Mamadaliev D. Selenium WebDriver PageFactory: Simplifying Test Automation (Lazy Proxying and Annotations). *Medium*. URL: https://diyorbekmamadaliev.medium.com/selenium-pagefactory-simplifying-test-automation-lazy-proxying-and-annotations-410880985643 (дата звернення: 05.11.2024).
21. Бойко М.О., Коломоєць Г.П. доцент, канд. фіз.-мат. наук – науковий керівник. Використання технології Appium для тестування мобільного застосунку. Збірник наукових праць студентів, аспірантів, докторантів і молодих

вчених «Молода наука-2023» / Запорізький національний університет. – Запоріжжя : ЗНУ, 2023. – Т.5. – С. 76-78.

22. Бойко М.О., Коломоєць Г.П. доцент, канд. фіз.-мат. наук – науковий керівник. Використання технології Page-Object Model для підвищення ефективності тестування веб-застосунків. Збірник наукових праць студентів, аспірантів, докторантів і молодих вчених «Молода наука-2024» / Запорізький національний університет. – Запоріжжя : ЗНУ, 2024. – Т.5. – С. 167-169.

23. Cypress vs Selenium vs Playwright vs Puppeteer: Core Differences | BrowserStack. *BrowserStack*. URL: https://www.browserstack.com/guide/cypress-vs-selenium-vs-playwright-vs-puppeteer (дата звернення: 06.11.2024).

**Декларація**

**академічної доброчесності**

**здобувача ступеня вищої освіти ЗНУ**

Я, Бойко Микола Олександрович, студент 2 курсу, форми навчання денної, Інженерного навчально-наукового інституту, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти ipz19bd-03@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «**Використання технології Page-Object Model для підвищення ефективності тестування Веб-застосунків**» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

згоден/згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 29.12.2024 Підпис _____ Бойко Микола Олександрович
(студент)

Дата 29.12.2024 Підпис _____ Коломоєць Геннадій Павлович
(науковий керівник)