МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ ВЕБПРОЄКТУ З ВИКОРИСТАННЯМ SELENIUM ТА WEBDRIVER»**

Виконав: студент 2 курсу, групи 8.1213-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
Д.А. Іванов
(ініціали та прізвище)

Керівник професор кафедри комп'ютерних наук, професор, д.т.н. Чопоров С.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри фундаментальної та прикладної математики, доцент, к.ф.-м.н. Панасенко Є.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Іванову Данилу Артемовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Автоматизація тестування вебпроєкту з використанням Selenium та WebDriver

керівник роботи Чопоров Сергій Вікторович, д.т.н., професор
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » березня 2024 року № 385-с

2. Строк подання студентом роботи 25.11.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Автоматизація тестування вебпроєкту.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 07.03.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	22.03.2024	
2.	Збір вихідних даних.	25.04.2024	
3.	Обробка методичних та теоретичних	20.05.2024	
	джерел.		
4.	Розробка першого та другого розділу.	28.08.2024	
5.	Розробка третього розділу.	28.10.2024	
6.	Оформлення та нормоконтроль	20.11.2024	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	11.12.2024	

Студент _____ Д.А. Іванов
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.В. Чопоров
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Автоматизація тестування вебпроєкту з використанням Selenium та WebDriver»: 71 с., 13 рис., 22 джерела, 3 додатки.

АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ, ВЕБАВТОМАТИЗАЦІЯ, ІНТЕГРАЦІЯ, РОБОТА З ВЕБЕЛЕМЕНТАМИ, ТЕСТУВАННЯ, ЯКІСТЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, JAVA, SELENIUM, WEBDRIVER.

Об’єкт дослідження – процес автоматизованого тестування у вебпроєкті.

Мета роботи: дослідження та розробка методів автоматизації тестування вебпроєкту з використанням Selenium та WebDriver.

Метод дослідження – теоретико-методологічний аналіз, тестування стратегій, методи програмування, автоматизації та тестування програмного забезпечення.

У роботі проведено дослідження процесу автоматизації тестування вебпроєкту з використанням Selenium та WebDriver. Представлено аналіз і вибір інструментів для автоматизації тестування. Визначено вимоги до автоматизованих тестів, розроблено та реалізовано необхідні скрипти та тестові сценарії.

У роботі детально описано процес розробки та застосування автоматизованих тестів. Представлено реалізацію тестових сценаріїв для основних функціональних можливостей вебпроєкту, таких як реєстрація, вхід користувача, навігація по вебсторінках, заповнення форм та перевірка вмісту.

Результати дослідження можуть бути використані для вдосконалення процесу тестування вебпроєктів та розробки автоматизованих тестів з використанням Selenium та WebDriver.

SUMMARY

Master’s qualification paper “Automation of Web Project Testing Using Selenium and WebDriver”: 71 p., 13 figures, 22 references, 3 supplements.

INTEGRATION, JAVA, SELENIUM, SOFTWARE QUALITY, TEST AUTOMATION, TESTING, WEB AUTOMATION, WEBDRIVER, WORKING WITH WEB ELEMENTS.

The object of the study is the process of automated testing in a web project.

The aim of the study is research and develop methods for automating web project testing using Selenium and WebDriver.

The methods of research are: theoretical and methodological analysis, strategy testing, methods of programming, automation and software testing.

The paper studies the process of automating the testing of a web project using Selenium and WebDriver. The analysis and selection of tools for test automation are presented. The requirements for automated tests are defined, the necessary scripts and test scenarios are developed and implemented.

The process of developing and applying automated tests is described in detail. The paper presents the implementation of test scenarios for the main functionalities of a web project, such as registration, user login, web page navigation, form filling, and content validation.

The results of the study can be used to improve the process of testing web projects and develop automated tests using Selenium and WebDriver.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 8
1 Теоретична частина 10
1.1 Основи автоматизації тестування 10
1.2 Selenium та WebDriver: архітектура та можливості 15
1.3 Порівняння інших мов програмування 17
1.4 Використання фреймворків для автоматизованого тестування 21
1.5 Інтеграція автоматизованого тестування у процес розробки 26
1.6 Порівняння автоматизованого та ручного тестування 29
1.7 Випадки застосування автоматизованого тестування 32
1.8 Перспективи розвитку автоматизації тестування 38
1.9 Висновки до розділу 1 42
2 Практична частина 43
2.1 Об’єкт тестування 43
2.2 Структура бізнес-логіки сайту 44
2.3 Інтеграція автоматизованих тестів 48
2.4 Висновки до розділу 2 53
3 Аналіз тестування 54
3.1 Сценарії взаємодії 54
3.2 Опис виявлених помилок 55
3.3 Розподіл помилок та відсотки виявлення 57
3.4 Висновки до розділу 3 60
Висновки 61
Перелік посилань 62
Додаток А Реалізація тесту на перевірку функції реєстрації користувачів 64

Додаток Б Реалізація тесту на перевірку функції додавання товару..... 68
Додаток В Реалізація тесту на перевірку функції оформлення замовлення... 70

ВСТУП

У сучасному світі, де інформаційні технології займають центральне місце в багатьох сферах життя, ефективне тестування вебпроєктів набуває все більшої актуальності. З розвитком Інтернету та вебдодатків зростає необхідність у забезпеченні їхньої якості, надійності та функціональності. Автоматизація тестування є ключовим аспектом у цьому процесі, оскільки дозволяє швидко і точно виявляти помилки, знижуючи витрати часу та ресурсів.

Selenium і WebDriver стали невід'ємними інструментами в арсеналі тестувальників вебдодатків. Вони забезпечують можливість автоматизованого тестування, що дозволяє значно підвищити продуктивність і ефективність тестування. Завдяки цим інструментам можна автоматизувати рутинні завдання, які в іншому випадку вимагали б значних людських ресурсів.

Проблема, з якою стикаються багато компаній та окремі розробники, полягає в складності та трудомісткості ручного тестування. З кожним оновленням або зміною в коді вебпроєкту виникає потреба повторного тестування всіх функцій, що може займати багато часу і призводити до людських помилок. Автоматизація тестування вирішує цю проблему, дозволяючи швидко та ефективно перевіряти вебпроєкти на всіх етапах їхнього розвитку.

Використання Selenium і WebDriver дає можливість створювати скрипти для автоматизованого тестування, які можна легко масштабувати і адаптувати до різних проєктів. Це дозволяє розробникам зосередитися на більш творчих і складних завданнях, залишаючи рутинні перевірки автоматизованим системам. Крім того, автоматизоване тестування сприяє підвищенню якості кінцевого продукту, оскільки дозволяє виявляти і виправляти помилки на ранніх стадіях розробки.

Таким чином, автоматизація тестування вебпроєктів з використанням Selenium і WebDriver має безліч переваг. Вона знижує витрати на тестування, підвищує продуктивність і точність, а також сприяє поліпшенню якості програмного забезпечення. У рамках цієї роботи буде досліджено методи і підходи до автоматизованого тестування вебпроєктів з використанням цих інструментів, а також розглянуто практичні приклади їхнього застосування в реальних проєктах.

1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 Основи автоматизації тестування

Автоматизація тестування вебпроєктів є ключовою складовою сучасного розроблення програмного забезпечення. Це підхід, що дозволяє виконувати тестування додатків за допомогою спеціалізованих програмних інструментів, які автоматизують процес тестування. Такий підхід дозволяє значно підвищити продуктивність, зменшити ймовірність людських помилок та забезпечити вищу якість кінцевого продукту [1].

Одним з основних принципів автоматизації тестування є повторюваність. Завдяки автоматизації, тестові сценарії можуть бути виконані неодноразово з однаковими результатами. Це особливо важливо для регресійного тестування, яке має на меті перевірку, чи не вплинули нові зміни у коді на існуючий функціонал. Крім того, автоматизація тестування дозволяє зекономити час і ресурси, оскільки після початкової інвестиції в розробку тестів, їх виконання не потребує значних зусиль.

Проте, автоматизація тестування має і свої виклики. Наприклад, початкова розробка автоматизованих тестів може вимагати значних витрат часу і ресурсів. Також, підтримка автоматизованих тестів може бути складною, оскільки зміни у програмному забезпеченні можуть потребувати відповідних змін у тестах. Незважаючи на ці труднощі, переваги автоматизованого тестування зазвичай перевищують витрати, особливо у великих проєктах, де тестування має виконуватися регулярно.

Основні принципи автоматизації тестування включають планування, вибір інструментів, розробку тестів, їх виконання та аналіз результатів. Планування є першим кроком, і воно передбачає визначення, які саме тести будуть автоматизовані. Не всі тести підходять для автоматизації; доцільно автоматизувати ті тести, які виконуються часто і мають велику важливість для проєкту [3].

Вибір інструментів для автоматизації тестування є критичним етапом. Різні інструменти мають різні можливості і обмеження, і вибір неправильного інструменту може значно ускладнити процес автоматизації. Selenium є одним з найпопулярніших інструментів для автоматизації тестування вебпроєктів. Він підтримує різні мови програмування, такі як Java, Python, C#, і дозволяє автоматизувати взаємодію з вебсторінками [22].

Розробка тестів включає написання скриптів, які автоматизують виконання тестових сценаріїв. Ці тести повинні бути розроблені так, щоб вони були легко підтримувані і оновлювані. Це важливо, оскільки зміни в додатку можуть вимагати відповідних змін в тестах, і добре спроєктовані тести спрощують цей процес.

Виконання автоматизованих тестів може бути інтегроване в процес безперервної інтеграції (Continuous Integration, CI). Це дозволяє автоматично запускати тести кожного разу, коли в код додаються нові зміни, що допомагає виявляти помилки на ранніх стадіях розробки. Інструменти CI, такі як Jenkins, Travis CI або CircleCI, можуть бути налаштовані на автоматичне виконання тестів, що забезпечує постійний контроль за якістю програмного забезпечення.

Аналіз результатів тестування є завершальним етапом автоматизації. Він передбачає ретельне вивчення результатів виконаних тестів для виявлення і виправлення помилок. Автоматизовані тести можуть генерувати детальні звіти, що допомагає швидко ідентифікувати проблемні місця в коді і вжити необхідних заходів для їх виправлення.

Різниця між ручним та автоматизованим тестуванням полягає в тому, що ручне тестування виконується людиною, яка безпосередньо взаємодіє з додатком, слідуючи заздалегідь визначеним сценаріям. Це дозволяє перевірити зручність використання і інтуїтивність інтерфейсу, але є повільним і схильним до людських помилок. Автоматизоване тестування, навпаки, використовує програмні інструменти для виконання тестів, що забезпечує високу точність і швидкість, але може бути менш гнучким у випадках, коли потрібна оцінка зручності використання.

Методи автоматизації тестування включають Data-Driven Testing, коли тести використовують зовнішні джерела даних для виконання тестових сценаріїв з різними наборами даних, і Keyword-Driven Testing, де тести описуються за допомогою ключових слів, що представляють окремі дії. Також важливою методологією є Behavior-Driven Development (BDD), яка використовує зрозумілу мову для опису тестових сценаріїв, що полегшує створення і розуміння тестів для всіх учасників проєкту.

Автоматизоване тестування може включати різні види тестування:

– функціональне тестування: перевірка того, що всі функції додатка працюють відповідно до вимог;
– регресійне тестування: забезпечення того, що нові зміни в коді не вплинули негативно на існуючі функціональності;
– навантажувальне тестування: перевірка продуктивності додатка під великим навантаженням;
– тестування безпеки: оцінка безпеки додатка шляхом автоматизованого пошуку вразливостей;
– крос-браузерне тестування: перевірка роботи додатка в різних браузерах.

Переваги автоматизованого тестування:

– швидкість і ефективність: автоматизовані тести можуть виконуватися набагато швидше, ніж ручні тести, особливо коли йдеться про великі тестові набори;
– повторюваність: автоматизовані тести можуть бути запущені багато разів з однаковими результатами, що важливо для регресійного тестування;
– покращення точності: автоматизація знижує ймовірність людських помилок під час тестування;
– економія ресурсів: після початкової інвестиції в автоматизацію, витрати на виконання тестів значно знижуються.

Недоліки автоматизованого тестування:

– початкова інвестиція: автоматизація вимагає значних початкових витрат на розробку тестів і налаштування середовища;
– підтримка тестів: автоматизовані тести потребують регулярного оновлення і підтримки відповідно до змін в додатку;
– обмеження в деяких типах тестування: не всі типи тестування можуть бути легко автоматизовані, наприклад, тестування інтерфейсу користувача або досвід користувача.

Основні принципи автоматизації тестування:

– планування: важливо визначити, які тести будуть автоматизовані, на основі їх частоти виконання і важливості для проєкту;
– вибір інструментів: вибір правильних інструментів автоматизації (таких як Selenium для вебтестування) є критично важливим для успіху проєкту;
– розробка тестів: тести мають бути розроблені таким чином, щоб вони були легкими для підтримки та оновлення;
– виконання тестів: автоматизовані тести мають виконуватись регулярно, наприклад, під час кожного білду в процесі безперервної інтеграції;
– аналіз результатів: результати тестів мають бути ретельно аналізовані для виявлення і виправлення дефектів.

Різниця між ручним та автоматизованим тестуванням:

– ручне тестування: виконується вручну тестувальниками, що перевіряють додаток, слідуючи певним сценаріям – це підходить для початкового етапу розробки, для тестування UI та UX, а також для ад-хок тестування;
– автоматизоване тестування: використовує скрипти і програми для виконання тестів без участі людини – ідеально підходить для повторюваних тестів, регресійного тестування, тестування продуктивності і великих наборів тестів.

Методи автоматизації тестування:

– Data-Driven Testing: метод, який використовує зовнішні джерела даних (наприклад, файли, бази даних) для тестових сценаріїв – це дозволяє виконувати один і той самий тест з різними наборами даних;
– Keyword-Driven Testing: метод, при якому тести описуються за допомогою ключових слів, що представляють собою окремі дії тесту – це дозволяє не-технічним спеціалістам створювати автоматизовані тести;
– Behavior-Driven Development: метод, який використовує зрозумілу мову (зазвичай на основі Gherkin) для опису тестових сценаріїв, що полегшує розуміння і створення тестів для всіх учасників проєкту.

Інструменти автоматизації тестування:

– Selenium: один з найпопулярніших інструментів для автоматизації тестування вебдодатків – підтримує різні мови програмування, такі як Java, Python, C# та інші;
– JUnit і TestNG: фреймворки для автоматизації тестування на Java, які широко використовуються для написання і запуску тестів;
– Appium: інструмент для автоматизації мобільних додатків, що підтримує Android і iOS;
– Cucumber: інструмент для BDD, який інтегрується з Selenium і використовується для створення тестів, зрозумілих для бізнес-користувачів.

Автоматизація тестування є невід'ємною частиною сучасного процесу розробки програмного забезпечення. Вона дозволяє досягти високого рівня якості і ефективності, знижуючи витрати на тестування і підвищуючи надійність кінцевого продукту. Розуміння основних концепцій, методів та інструментів автоматизації тестування є ключовим для успішної реалізації будь-якого проєкту в цій області.

У розділі розглянуто основи автоматизації тестування, її переваги, виклики та ключові принципи. Автоматизація тестування дозволяє значно

підвищити ефективність та якість розробки програмного забезпечення, використовуючи спеціалізовані інструменти та методи. Правильний підхід до вибору тестів для автоматизації, інструментів та підтримки тестових сценаріїв є визначальним для успішного впровадження цієї практики.

1.2 Selenium та WebDriver: архітектура та можливості

Selenium і WebDriver є одними з найпопулярніших інструментів для автоматизації тестування вебдодатків. Їх гнучкість, можливості та широке прийняття в індустрії роблять їх незамінними для багатьох розробників і тестувальників. У цьому розділі розглянемо архітектуру Selenium і WebDriver, їх компоненти та основні можливості [6].

Архітектура Selenium побудована на принципах модульності та розширюваності. Вона складається з декількох ключових компонентів, кожен з яких виконує специфічну роль в процесі автоматизації тестування.

Selenium IDE (Integrated Development Environment) – це інструмент, призначений для створення простих тестових сценаріїв без необхідності написання коду. Він являє собою розширення для браузера (спочатку Firefox, а згодом і Chrome), яке дозволяє записувати взаємодію користувача з вебсторінкою і відтворювати ці дії автоматично. Це зручний інструмент для швидкого створення прототипів тестів і для початкового знайомства з автоматизацією тестування.

Selenium WebDriver є ядром автоматизації тестування в Selenium. Це інструмент, який надає програмний інтерфейс для взаємодії з браузерами. WebDriver підтримує різні браузери, включаючи Chrome, Firefox, Safari, Edge та Internet Explorer, що дозволяє проводити крос-браузерне тестування. WebDriver безпосередньо взаємодіє з браузером через його нативні функції, забезпечуючи більш швидке і точне виконання тестів порівняно з попередніми версіями Selenium [4].

WebDriver складається з кількох основних компонентів:

– драйвери браузера: для кожного підтримуваного браузера є свій драйвер, який відповідає за керування цим браузером (наприклад, для Chrome використовується ChromeDriver, для Firefox – GeckoDriver і так далі);
– клієнтські бібліотеки: WebDriver надає бібліотеки для різних мов програмування, включаючи Java, C#, Python, Ruby та інші – це дозволяє розробникам писати тести на знайомій їм мові;
– WebDriver API: набір команд і методів, що дозволяють здійснювати взаємодію з елементами вебсторінки, такими як кліки, введення тексту, навігація між сторінками, робота з фреймами та спливаючими вікнами.

Selenium Grid дозволяє розподіляти виконання тестів на кількох машинах одночасно, що значно скорочує час тестування великих проєктів. Grid складається з хаба та нодів. Хаб керує тестовими завданнями та розподіляє їх на ноди, які безпосередньо виконують тести на різних комбінаціях браузерів та операційних систем.

Selenium Server є частиною Selenium Grid і відповідає за управління тестовими завданнями та координацію нодів. Він також може використовуватись для віддаленого запуску тестів без необхідності встановлювати WebDriver безпосередньо на кожну тестову машину.

Selenium і WebDriver пропонують широкий спектр можливостей для автоматизації тестування:

– крос-браузерне тестування: підтримка різних браузерів та їх версій дозволяє перевіряти роботу вебдодатків у різних середовищах;
– підтримка різних мов програмування: можливість писати тести на Java, C#, Python, Ruby та інших мовах робить Selenium універсальним інструментом для різних команд розробників;
– гнучкість та розширюваність: завдяки модульній архітектурі, Selenium легко інтегрується з іншими інструментами та

фреймворками для тестування, такими як JUnit, TestNG, Cucumber та інші;

– підтримка складних операцій: WebDriver дозволяє виконувати складні дії з вебсторінками, включаючи роботу з динамічними елементами, AJAX-запитами, фреймами, спливаючими вікнами та файлами;
– паралельне виконання тестів: використання Selenium Grid дозволяє значно скоротити час тестування, виконуючи тести паралельно на різних комбінаціях браузерів і операційних систем.

Завдяки цим можливостям Selenium і WebDriver стали стандартом в індустрії автоматизації тестування вебдодатків, забезпечуючи розробників та тестувальників потужними інструментами для підвищення якості програмного забезпечення.

У цьому розділі розглянуто архітектуру та можливості Selenium і WebDriver як ключових інструментів для автоматизації тестування вебдодатків. Описано їх компоненти, такі як Selenium IDE, WebDriver, Selenium Grid та Selenium Server, які забезпечують потужну платформу для створення, виконання та масштабування тестів. Завдяки своїй універсальності, крос-браузерній підтримці та гнучкості, Selenium і WebDriver стали стандартом у галузі автоматизованого тестування.

1.3 Порівняння інших мов програмування

Автоматизація тестування вебпроєктів може здійснюватися за допомогою різних мов програмування, кожна з яких має свої унікальні особливості, переваги та недоліки. Вибір мови програмування залежить від багатьох факторів, таких як досвід команди, специфіка проєкту, а також вимоги до продуктивності та інтеграції з іншими інструментами [7].

Java є однією з найпопулярніших мов програмування для автоматизації тестування з Selenium WebDriver. Вона має багато переваг, що робить її

ідеальним вибором для багатьох тестувальників. Однією з головних причин популярності Java є її статус мови загального призначення з об’єктно-орієнтованою парадигмою. Завдяки цьому, Java дозволяє розробникам створювати складні та масштабовані тестові сценарії [11].

Переваги Java для автоматизації тестування:

– широка підтримка та документація: Java є однією з найпоширеніших мов програмування, тому існує велика кількість ресурсів, документації та спільнот, які можуть допомогти в розв’язанні будь-яких питань (багато великих компаній використовують Java, що сприяє активному розвитку екосистеми);
– висока продуктивність: Java є компільованою мовою, що забезпечує високу продуктивність виконання тестів – це дозволяє виконувати великі набори тестів швидше, що є важливим для забезпечення якості програмного забезпечення;
– сумісність з багатьма інструментами: Java легко інтегрується з багатьма іншими інструментами для автоматизації тестування, такими як TestNG, JUnit, Maven, Jenkins та іншими – це спрощує процес розробки, тестування та деплойменту, забезпечуючи високу ефективність роботи команд.

Недоліки Java:

– відносно висока складність: Java може бути складнішою для вивчення та використання порівняно з деякими іншими мовами програмування, такими як Python – висока складність синтаксису та велика кількість деталей можуть вимагати більше часу на навчання та адаптацію.

Python є популярною мовою програмування для автоматизації тестування завдяки своїй простоті та читабельності. Вона стає все більш поширеною серед тестувальників і розробників, оскільки дозволяє швидко створювати та підтримувати тестові сценарії [2].

Переваги Python для автоматизації тестування:

– простота та читабельність: Python має простий і зрозумілий синтаксис, що робить його легким у вивченні та використанні – це особливо

важливо для новачків, які тільки починають свій шлях у автоматизації тестування;

– широка підтримка бібліотек: Python має багатий набір бібліотек і фреймворків для автоматизації тестування, таких як PyTest, Robot Framework, та багато інших – це дозволяє швидко знайти та використовувати готові рішення для різних задач автоматизації;
– велика спільнота: як і Java, Python має велику і активну спільноту, що забезпечує підтримку та ресурси для розв'язання проблем – це включає в себе форуми, групи в соціальних мережах, конференції та багато інших ресурсів.

Недоліки Python:

– менша продуктивність: як інтерпретована мова, Python може бути повільнішим у виконанні тестів порівняно з компільованими мовами, такими як Java або C# – це може бути критичним фактором при виконанні великих наборів тестів;
– обмежена підтримка багатопоточності: через Global Interpreter Lock (GIL) в Python, підтримка багатопоточності є обмеженою, що може впливати на виконання складних тестових сценаріїв – це обмеження може бути важливим при тестуванні додатків, які активно використовують багатопоточність.

C# є популярною мовою програмування для автоматизації тестування, особливо серед розробників, які працюють з екосистемою Microsoft. Завдяки своїй потужності та гнучкості, C# часто використовується для створення складних і ефективних тестових сценаріїв [21].

Переваги C# для автоматизації тестування:

– інтеграція з Microsoft продуктами: C# добре інтегрується з іншими продуктами Microsoft, такими як Visual Studio, Azure DevOps та інші – це забезпечує безшовний робочий процес для команд, які використовують Microsoft технології;
– висока продуктивність: як компільована мова, C# забезпечує високу продуктивність виконання тестів – це дозволяє швидко виконувати

великі набори тестів, що є важливим для забезпечення якості програмного забезпечення;

– підтримка асинхронного програмування: C# має потужні можливості для асинхронного програмування, що дозволяє ефективніше управляти тестами – це важливо для тестування додатків, які активно використовують асинхронні операції.

Недоліки C#:

– відносно висока складність: як і Java, C# може бути складнішою для вивчення та використання порівняно з Python – складність синтаксису та велика кількість деталей можуть вимагати більше часу на навчання та адаптацію.

Ruby також використовується для автоматизації тестування, хоча і менш поширений, ніж Java або Python. В основному Ruby використовується разом з фреймворком Capybara для тестування вебдодатків. Ця мова відома своїм елегантним та лаконічним синтаксисом, що робить її популярною серед деяких розробників і тестувальників.

Переваги Ruby для автоматизації тестування:

– простота та елегантність: Ruby відомий своїм елегантним і простим синтаксисом, що робить його легким для вивчення та використання – це дозволяє швидко створювати та підтримувати тестові сценарії, що важливо для багатьох команд;

– швидкий розвиток: Ruby дозволяє швидко розробляти тестові сценарії завдяки своїй гнучкості та динамічній природі – це важливо для команд, які потребують швидкої адаптації до змін у вимогах проєкту.

Недоліки Ruby;

– менша продуктивність: як інтерпретована мова, Ruby може бути повільнішим у виконанні тестів порівняно з компільованими мовами – це може бути важливим фактором при виконанні великих наборів тестів;

– менша спільнота та підтримка: Ruby має меншу спільноту та підтримку порівняно з Java або Python, що може обмежувати доступні

ресурси та інструменти – це може створювати певні труднощі при розв’язанні специфічних проблем або пошуку готових рішень.

Кожна мова програмування має свої унікальні переваги та недоліки для автоматизації тестування з використанням Selenium WebDriver. Java забезпечує високу продуктивність і широку підтримку, Python вирізняється простотою та читабельністю, C# добре інтегрується з екосистемою Microsoft, а Ruby пропонує елегантний синтаксис для швидкого розвитку. Вибір мови залежить від конкретних вимог проєкту та досвіду команди. Важливо також враховувати можливість інтеграції з іншими інструментами, наявність ресурсів та підтримки, а також специфіку завдань, які необхідно вирішувати у процесі автоматизації тестування.

У цьому розділі було проведено порівняння популярних мов програмування, які використовуються для автоматизації тестування вебпроєктів із застосуванням Selenium WebDriver: Java, Python, C# та Ruby. Кожна з них має свої переваги та недоліки, що впливають на вибір мови залежно від особливостей проєкту та досвіду команди.

1.4 Використання фреймворків для автоматизованого тестування

Автоматизація тестування вебпроєктів є важливим аспектом забезпечення якості програмного забезпечення. Використання фреймворків допомагає спростити та прискорити процес створення, виконання і підтримки тестових сценаріїв. Окрім Selenium WebDriver, існує кілька інших популярних фреймворків, таких як Katalon Studio, Spock, Robot Framework, та інші. Кожен з цих фреймворків має свої унікальні можливості, переваги та недоліки, які роблять їх більш або менш придатними для конкретних завдань автоматизації тестування. У цьому розділі ми детально розглянемо деякі з цих фреймворків, порівняємо їх з Selenium WebDriver і проаналізуємо їхні можливості в контексті автоматизації тестування вебпроєктів [5].

Katalon Studio – це потужний інструмент для автоматизації тестування, який підтримує тестування вебдодатків, мобільних додатків та API. Він забезпечує зручний інтерфейс користувача і багатий набір функцій, що дозволяє швидко розробляти і виконувати тестові сценарії.

Переваги Katalon Studio:

– простота використання: Katalon Studio пропонує інтуїтивно зрозумілий інтерфейс, що дозволяє швидко створювати тестові сценарії без глибоких знань програмування – це робить його особливо корисним для команд, що складаються з не дуже технічних спеціалістів;
– інтеграція з CI/CD: інструмент легко інтегрується з популярними системами безперервної інтеграції та доставки, такими як Jenkins, Bamboo, та інші – це дозволяє автоматизувати процес тестування на різних етапах розробки програмного забезпечення;
– вбудовані шаблони та бібліотеки: Katalon Studio надає велику кількість вбудованих шаблонів та бібліотек, що дозволяє швидко налаштовувати тестові сценарії під конкретні потреби проєкту;
– підтримка різних типів тестування: інструмент підтримує функціональне, регресійне, паралельне та інші типи тестування, що робить його універсальним рішенням для різних видів автоматизації.

Недоліки Katalon Studio:

– обмежена гнучкість: хоча Katalon Studio пропонує багато вбудованих функцій, його можливості можуть бути обмеженими у порівнянні з більш гнучкими інструментами, такими як Selenium WebDriver, які дозволяють створювати кастомні рішення;
– вартість: преміум-версія Katalon Studio є платною, що може бути недоліком для малих команд або компаній з обмеженим бюджетом.

Spock – це фреймворк для тестування, розроблений для мов програмування Groovy та Java. Він використовує синтаксис BDD (Behavior-Driven Development), що дозволяє створювати тести, які легко читати і розуміти навіть нетехнічним фахівцям.

Переваги Spock:

– зручний синтаксис: синтаксис Spock дозволяє писати тестові сценарії у формі специфікацій, що робить їх більш зрозумілими та легкими для читання – це сприяє кращій співпраці між розробниками, тестувальниками та іншими членами команди;
– підтримка параметризованих тестів: Spock дозволяє легко створювати параметризовані тести, що забезпечує більш ефективне тестування різних варіантів вхідних даних;
– інтеграція з іншими інструментами: фреймворк добре інтегрується з іншими інструментами для розробки та тестування, такими як Gradle, Maven та інші, що полегшує його впровадження в існуючі процеси розробки.

Недоліки Spock:

– вимога знання Groovy: оскільки Spock базується на Groovy, його використання вимагає знання цієї мови програмування, що може стати бар'єром для команд, які звикли працювати з іншими мовами;
– обмежена підтримка мобільного тестування: Spock в основному орієнтований на тестування Java-програм та вебдодатків, тому може бути менш ефективним для тестування мобільних додатків у порівнянні з іншими інструментами.

Robot Framework – це ще один потужний інструмент для автоматизації тестування, який підтримує різні типи тестування, включаючи функціональне, регресійне та приймальне тестування. Він базується на використанні ключових слів, що дозволяє створювати тести у вигляді сценаріїв на природній мові.

Переваги Robot Framework:

– простота використання: завдяки використанню ключових слів, Robot Framework дозволяє створювати тестові сценарії, які легко читати і розуміти – це сприяє залученню нетехнічних фахівців до процесу тестування;

– широка підтримка: фреймворк підтримує різні типи тестування і має велику кількість вбудованих бібліотек, що забезпечує високу гнучкість і універсальність;
– інтеграція з іншими інструментами: Robot Framework добре інтегрується з іншими інструментами, такими як Selenium, Appium та інші, що дозволяє використовувати його для тестування різних типів додатків.

Недоліки Robot Framework:

– вимога знання специфічної синтаксису: хоча використання ключових слів спрощує створення тестів, воно також вимагає знання специфічного синтаксису Robot Framework, що може вимагати додаткового навчання;
– менша гнучкість у порівнянні з кодовими фреймворками: хоча Robot Framework забезпечує високу зручність використання, він може бути менш гнучким у порівнянні з кодовими фреймворками, такими як Selenium WebDriver, які дозволяють створювати кастомні рішення для складних сценаріїв.

Selenium WebDriver є одним з найпоширеніших і гнучких інструментів для автоматизації тестування вебдодатків. Він дозволяє взаємодіяти з браузерами на рівні API, що забезпечує високу гнучкість і контроль над тестовими сценаріями.

Переваги Selenium WebDriver:

– гнучкість: Selenium WebDriver дозволяє створювати кастомні рішення для складних сценаріїв тестування, що забезпечує високу гнучкість і адаптивність;
– широка підтримка браузерів: інструмент підтримує автоматизацію тестування на різних браузерах, таких як Chrome, Firefox, Safari, Edge та інших, що забезпечує крос-браузерну сумісність;
– велика спільнота: Selenium WebDriver має велику і активну спільноту, що забезпечує доступ до широкого спектру ресурсів, документації та підтримки.

Недоліки Selenium WebDriver:

– складність у використанні: використання Selenium WebDriver може вимагати глибоких знань програмування та розуміння архітектури вебдодатків, що може бути складним для новачків;
– відсутність вбудованих звітів: Selenium WebDriver не надає вбудованих інструментів для генерації звітів про виконання тестів, що вимагає використання додаткових інструментів та бібліотек для цього.

Кожен з розглянутих фреймворків має свої унікальні можливості та обмеження. Вибір фреймворка залежить від конкретних вимог проєкту, досвіду команди та наявних ресурсів.

Katalon Studio є чудовим вибором для команд, що потребують простого і зручного інструменту для швидкого створення тестових сценаріїв.

Spock підходить для проєктів на Groovy та Java, де важлива читабельність і зрозумілість тестів.

Robot Framework забезпечує простоту використання та підтримку різних типів тестування завдяки використанню ключових слів.

Selenium WebDriver, у свою чергу, пропонує високу гнучкість і контроль над тестуванням, що робить його ідеальним для складних і вимогливих проєктів.

Розділ аналізує фреймворки для автоматизації тестування, такі як Katalon Studio, Spock, Robot Framework та Selenium WebDriver. Katalon Studio вирізняється простотою використання, інтеграцією з CI/CD і вбудованими шаблонами, хоча обмежена гнучкість і вартість можуть бути недоліками. Spock підходить для Java і Groovy, завдяки зручному синтаксису та параметризованим тестам, але вимагає знань Groovy. Robot Framework зручний через використання ключових слів і сумісність із багатьма інструментами, однак має специфічний синтаксис і меншу гнучкість. Selenium WebDriver забезпечує високу гнучкість і контроль над тестами, але вимагає знань програмування і додаткових інструментів для звітності.

1.5 Інтеграція автоматизованого тестування у процес розробки

Автоматизоване тестування відіграє ключову роль у забезпеченні якості програмного забезпечення та його успішної доставки користувачам. Інтеграція цього виду тестування у процес безперервної інтеграції (CI), доставки (CD) та деплойменту дозволяє підвищити ефективність та швидкість розробки, а також зменшити кількість помилок та дефектів. Давайте розглянемо кожну з цих складових детальніше.

Безперервна інтеграція (Continuous Integration, CI) – це практика розробки програмного забезпечення, при якій розробники регулярно зливають свої зміни коду до спільного репозиторію, після чого запускаються автоматизовані процеси збирання, тестування та розгортання. Головною метою CI є підтримка постійної готовності коду до релізу, а також раннє виявлення та виправлення помилок.

Основні переваги CI:

- раннє виявлення помилок: завдяки автоматизованим тестам, помилки в коді виявляються на ранніх етапах розробки, що дозволяє їх виправити швидше та з меншими витратами;
- забезпечення якості: регулярна перевірка коду за допомогою автоматизованих тестів дозволяє підтримувати високу якість програмного забезпечення;
- зменшення ризиків: завдяки частим ітераціям і регулярній перевірці коду, ризик виникнення серйозних помилок у програмному забезпеченні значно зменшується;
- швидкий зворотний зв’язок: розробники отримують миттєвий зворотний зв’язок про стан проєкту після кожного злиття коду, що дозволяє їм швидко реагувати на проблеми та вносити необхідні зміни.

Безперервна інтеграція є основою для імплементації інших практик DevOps, таких як безперервна доставка (Continuous Delivery) та безперервний деплоймент (Continuous Deployment).

Безперервна доставка (Continuous Delivery, CD) – це практика, яка забезпечує можливість релізу програмного забезпечення в будь-який момент часу. Основною ідеєю CD є підтримка коду в такому стані, що він завжди готовий до релізу. Це досягається шляхом автоматизації всіх етапів розробки, включаючи збірку, тестування та деплоймент [19].

Основні переваги CD:

– швидкість релізу: завдяки автоматизації всіх етапів процесу розробки, нові версії програмного забезпечення можуть бути релізовані швидше та частіше;
– зниження ризиків: автоматизація тестування та деплойменту зменшує ризики, пов’язані з випуском нових версій програмного забезпечення;
– підвищення якості: постійне тестування та інтеграція дозволяють підтримувати високу якість програмного забезпечення.

Безперервний деплоймент – це розширення підходу безперервної доставки, що автоматизує процес розгортання програмного забезпечення на продуктивних середовищах. В цьому підході кожна зміна коду, що успішно проходить всі тестування, автоматично розгортається на продуктивне середовище без необхідності ручного втручання [18].

Переваги безперервного деплойменту:

– миттєва доставка: зміни коду, що успішно проходять всі тестування, автоматично розгортаються на продуктивному середовищі без затримок;
– автоматизація процесу: цей підхід забезпечує повну автоматизацію релізного процесу, що робить розгортання швидким і надійним.

Після кожного коміту коду до основної гілки, CI система автоматично запускає набір тестів для перевірки правильності роботи функцій додатку. Якщо тести успішно пройшли, нова версія додатку автоматично розгортається на продуктивному сервері, готова для використання покупцями.

Цей підхід дозволяє команді розробників швидко впроваджувати нові функції та виправлення без необхідності чекати на ручне розгортання, забезпечуючи при цьому високу якість і надійність програмного забезпечення.

Додавання автоматизованого тестування до процесу CI/CD стає стратегічним рішенням для будь-якої команди розробників. Це не лише поліпшує якість коду, а й сприяє швидкому впровадженню змін. Проте, важливо розуміти, що успішна імплементація цих практик вимагає від команди певного рівня експертизи та зосередженості на процесі.

Перш за все, потрібно визначити, які саме тести будуть автоматизовані. Це можуть бути модульні тести, що перевіряють окремі фрагменти коду на коректність роботи, або функціональні тести, які тестують програму як єдину систему. Вибір тестів залежить від потреб проєкту та його характеристик.

Після вибору тестів необхідно підготувати інфраструктуру для їх виконання. Це може включати в себе створення спеціальних середовищ для тестування, налаштування CI/CD системи, а також розробку інструментів для автоматизованого тестування. Важливо забезпечити, щоб ця інфраструктура була стабільною та надійною, оскільки будь-яка помилка може призвести до неправильної оцінки якості коду.

Крім того, необхідно встановити процеси для збору та аналізу результатів тестування. Це допоможе вчасно виявляти проблеми та вносити відповідні зміни в код. Також важливо вести історію змін у тестах та звітувати про їх стан, щоб мати можливість аналізувати тенденції та покращувати процес тестування з часом.

Нарешті, автоматизоване тестування вимагає постійного оновлення та підтримки. Код проєкту змінюється, і тести повинні пристосовуватися до цих змін. Також можуть змінюватися вимоги до програмного забезпечення або до самого процесу розробки, що також може вплинути на тести.

Отже, впровадження автоматизованого тестування у процес CI/CD – це складний, але дуже важливий крок для будь-якого проєкту. Це дозволяє забезпечити високу якість програмного забезпечення, ефективно впроваджувати зміни та забезпечувати надійність продукту.

Розділ розкриває значення автоматизованого тестування в забезпеченні якості програмного забезпечення, акцентуючи на його ролі в безперервній

інтеграції, доставці та деплойменті. CI (Continuous Integration) передбачає регулярне злиття змін до спільного репозиторію, запуск автоматизованих тестів і швидкий зворотний зв’язок, що дозволяє вчасно виявляти та виправляти помилки. CD (Continuous Delivery) автоматизує всі етапи розробки, забезпечуючи готовність до релізу в будь-який момент. Continuous Deployment дозволяє автоматично розгортати успішно протестовані зміни на продуктивному середовищі.

1.6 Порівняння автоматизованого та ручного тестування

Автоматизоване тестування та ручне тестування є двома основними підходами до перевірки якості програмного забезпечення, кожен з яких має свої переваги та недоліки. Вибір між ними залежить від конкретних умов та вимог проєкту. Часто найбільш ефективним є їх поєднання, що дозволяє використовувати сильні сторони кожного підходу.

Однією з головних переваг автоматизованого тестування є швидкість виконання тестів. Автоматизовані тести дозволяють значно скоротити час на виконання тестових сценаріїв, особливо коли йдеться про великі набори тестів, які потрібно виконувати регулярно. Це особливо важливо при регресійному тестуванні, коли необхідно перевіряти весь функціонал після кожної зміни в коді. Виконання таких тестів вручну може зайняти години або навіть дні, тоді як автоматизовані тести можуть бути виконані за лічені хвилини. Наприклад, компанія XYZ, впровадивши автоматизоване тестування, скоротила час на виконання регресійних тестів з двох днів до двох годин [20].

Точність і надійність – ще одна ключова перевага автоматизації. Автоматизовані тести виключають людський фактор, що зменшує ймовірність помилок через втому, неуважність або інші людські причини. Автоматизовані тести забезпечують стабільні й повторювані результати, що особливо важливо

для регресійного тестування та інших сценаріїв, де важлива точність і надійність.

Ручне тестування, з іншого боку, може бути менш точним через людські помилки, проте воно дозволяє тестувальнику використовувати свій досвід і інтуїцію для виявлення нетипових або складних дефектів. Наприклад, у проєкті ABC ручні тестувальники виявили кілька критичних помилок, які не були враховані в автоматизованих тестах через нестандартний сценарій використання.

Вартість та ресурси – важливий аспект при виборі між автоматизованим та ручним тестуванням. Початкові витрати на впровадження автоматизованого тестування можуть бути високими через потребу в інструментах, скриптах та навчанні. Проте, у довгостроковій перспективі автоматизація може знизити загальні витрати на тестування завдяки зменшенню потреби в ручній праці. Автоматизоване тестування дозволяє зменшити кількість необхідних людських ресурсів для виконання тестів, що може бути особливо важливим для великих проєктів з обмеженими ресурсами. Ручне тестування, навпаки, потребує менших початкових витрат, але вимагає більше часу і ресурсів у довгостроковій перспективі, оскільки кожен новий тест або зміна в існуючих тестах вимагає залучення тестувальників. Наприклад, у проєкті DEF автоматизація тестування дозволила скоротити витрати на 30% протягом двох років.

Гнучкість та адаптивність – ще один важливий фактор. Ручне тестування дозволяє швидко адаптуватися до змін у програмному забезпеченні, тоді як автоматизовані тести можуть потребувати значного часу на підтримку та оновлення у разі змін в інтерфейсі або логіці програми. Ручне тестування є більш гнучким, оскільки тестувальники можуть миттєво змінювати свої підходи в залежності від ситуації. Це особливо важливо на ранніх етапах розробки, коли продукт зазнає значних змін. Автоматизоване тестування, з іншого боку, менш гнучке, оскільки будь-яка зміна в коді може вимагати оновлення тестових скриптів, що може зайняти додатковий час і ресурси.

Охоплення тестування є критично важливим аспектом забезпечення якості програмного забезпечення. Автоматизоване тестування може забезпечити більше охоплення тестових випадків, особливо для регресійного тестування, де необхідно регулярно перевіряти великий обсяг функціональності. Автоматизовані тести можуть виконуватися паралельно, що дозволяє перевіряти велику кількість сценаріїв одночасно. Проте, ручне тестування може бути більш ефективним для дослідницького тестування, де важливіше гнучкість і творчий підхід.

Тестувальники можуть використовувати свій досвід та інтуїцію для виявлення складних дефектів, які можуть бути пропущені автоматизованими тестами. Наприклад, у проєкті GHI автоматизоване тестування забезпечило перевірку всіх основних функцій за одну ніч, тоді як ручні тестувальники знайшли критичний дефект у новій функції, яка ще не була включена в автоматизовані тести.

Людський фактор також грає важливу роль у тестуванні. Ручне тестування залежить від досвіду та знань тестувальника, що може бути як перевагою, так і недоліком. Досвідчені тестувальники можуть знайти дефекти, які автоматизовані тести можуть пропустити, але також можуть зробити помилки через втому або неуважність. Автоматизоване тестування виключає цей фактор, забезпечуючи стабільні результати незалежно від виконавця, але вимагає високої кваліфікації для розробки та підтримки тестів.

Застосування в різних типах проєктів може впливати на вибір між автоматизованим та ручним тестуванням. Автоматизоване тестування ідеально підходить для довготривалих проєктів з частими релізами, де важлива стабільність і повторюваність тестів. Воно особливо корисне для проєктів, де є необхідність у регулярному регресійному тестуванні або тестуванні на різних платформах. Ручне тестування, з іншого боку, може бути більш придатним для короткострокових проєктів або проєктів на ранніх стадіях розробки, де функціональність постійно змінюється і потрібна гнучкість.

Автоматизоване та ручне тестування мають свої унікальні переваги та обмеження. Автоматизоване тестування забезпечує швидкість, точність, надійність і широке охоплення, але потребує значних початкових витрат та може бути менш гнучким. Ручне тестування, хоча й повільніше і менш точне, забезпечує гнучкість та можливість використовувати досвід тестувальника для виявлення складних дефектів. Найбільш ефективним підходом часто є поєднання обох методів, що дозволяє максимально використовувати їх сильні сторони та мінімізувати недоліки.

В цьому розділі розкривається порівняння автоматизованого та ручного тестування як основних підходів до забезпечення якості програмного забезпечення. Обидва методи мають свої переваги та недоліки, і їх вибір залежить від специфіки проєкту. Автоматизоване тестування забезпечує високу швидкість виконання, точність і можливість повторюваності тестів, що особливо корисно для регулярного тестування великого обсягу функціоналу.

1.7 Випадки застосування автоматизованого тестування

Автоматизоване тестування вебпроєктів є невід’ємною частиною сучасного процесу розробки програмного забезпечення завдяки своїй ефективності та здатності зменшити людські помилки. У цьому розділі розглянемо різні випадки застосування автоматизованого тестування та їхні переваги. Поглиблене обговорення включатиме докладні приклади використання, а також висвітлення основних інструментів і методик, що використовуються для автоматизації тестування.

Регресійне тестування є важливим аспектом підтримки якості програмного забезпечення, особливо в умовах постійного оновлення та розвитку вебпроєктів. Основна мета регресійного тестування – гарантувати, що нові зміни або виправлення помилок не впливають негативно на існуючий функціонал системи.

Автоматизація регресійного тестування дозволяє значно зменшити час, необхідний для перевірки системи після кожного оновлення. Це досягається за допомогою створення автоматизованих тестових сценаріїв, які можна багаторазово виконувати без втручання людини. Ці сценарії перевіряють ключові функції програми, щоб переконатися, що вони працюють належним чином після внесення змін [9].

Приклад: інтернет-магазин, який регулярно додає нові функції та оновлює існуючі. Після додавання нової функції, наприклад, можливості відстеження замовлень в реальному часі, автоматизовані регресійні тести перевіряють основні функції вебсайту, такі як реєстрація, логін, пошук товарів та оформлення замовлень. Ці тести гарантують, що нові зміни не зламали основний функціонал, що дозволяє користувачам продовжувати користуватися вебсайтом без перешкод.

Автоматизоване регресійне тестування особливо корисне у великих проєктах, де внесення змін у код є частим явищем. За допомогою інструментів, таких як Selenium, можна автоматизувати складні сценарії взаємодії користувачів з вебсайтом, включаючи навігацію, заповнення форм та перевірку результатів. Це дозволяє розробникам швидко виявляти та виправляти помилки, зменшуючи ризик виникнення нових проблем після кожного релізу [8].

Тестування продуктивності є критично важливим для визначення, як система поводиться під різними навантаженнями, включаючи тестування на продуктивність, стрес-тестування та тестування на масштабованість. Автоматизоване тестування продуктивності дозволяє створювати сценарії, які імітують багатокористувацьке навантаження, що допомагає виявляти вузькі місця та покращувати загальну продуктивність системи.

Приклад: вебдодаток для інтернет-банкінгу, який обслуговує тисячі користувачів одночасно. Автоматизовані тести продуктивності можуть моделювати ситуації, коли тисячі користувачів намагаються увійти в систему, перевірити баланс, здійснити платежі або перевести кошти. За допомогою

інструментів, таких як Apache JMeter або Gatling, можна визначити, скільки одночасних користувачів може обробляти вебдодаток, перш ніж він почне працювати повільно або взагалі перестане відповідати.

Такі тести також допомагають визначити оптимальні конфігурації серверів, необхідні для забезпечення безперебійної роботи вебдодатка під високим навантаженням. Це дозволяє IT-відділам планувати ресурси та оптимізувати інфраструктуру для підтримки очікуваних навантажень, забезпечуючи при цьому позитивний користувацький досвід.

Безпека вебпроєктів є надзвичайно важливою, оскільки вразливості в системі можуть призвести до серйозних наслідків, таких як витік даних, несанкціонований доступ до системи та фінансові втрати. Автоматизоване тестування безпеки допомагає виявляти вразливості та забезпечувати захист вебпроєктів від потенційних загроз.

Приклад: для автоматизації тестування безпеки часто використовують інструменти, такі як OWASP ZAP або Burp Suite. Вони можуть бути інтегровані в процес розробки для постійного моніторингу безпеки вебдодатків. Наприклад, автоматизовані тести можуть виявляти вразливості, такі як SQL-ін’єкції, XSS-атаки, CSRF-атаки та інші. За допомогою цих інструментів можна виконувати сканування вебдодатків на предмет відомих вразливостей та автоматично генерувати звіти з результатами перевірок.

Тестування безпеки також включає перевірку автентифікації та авторизації користувачів, щоб переконатися, що тільки авторизовані користувачі мають доступ до певних функцій та даних. Автоматизовані тести можуть моделювати атаки на систему, щоб визначити, як вона реагує на спроби несанкціонованого доступу, та перевіряти захист конфіденційної інформації.

Сумісність вебпроєкту з різними браузерами та платформами є важливою для забезпечення позитивного користувацького досвіду. Автоматизоване тестування сумісності дозволяє перевірити, як вебпроєкт працює у різних браузерах (Chrome, Firefox, Safari, Edge) та на різних платформах (Windows, macOS, Linux).

Приклад: з використанням Selenium Grid можна паралельно запускати тести на різних браузерах та операційних системах. Це дозволяє виявляти проблеми, пов’язані з сумісністю, наприклад, некоректне відображення елементів інтерфейсу або помилки в функціоналі на певних платформах. Автоматизовані тести перевіряють, чи коректно відображаються всі елементи вебсайту, чи працюють інтерактивні елементи, такі як кнопки та форми, та чи відповідає функціональність очікуванням користувачів.

Автоматизація тестування сумісності допомагає забезпечити, що вебпроєкт буде функціонувати коректно для всіх користувачів незалежно від їхньої конфігурації системи. Це особливо важливо для комерційних вебсайтів та додатків, де позитивний користувацький досвід безпосередньо впливає на бізнес-результати.

Інтерфейс користувача (UI) є важливим аспектом вебпроєктів, оскільки він визначає, як користувачі взаємодіють з системою. Автоматизовані UI-тести перевіряють, чи всі елементи інтерфейсу працюють належним чином і чи відповідають вони очікуванням користувачів.

Приклад: автоматизовані тести можуть перевіряти, чи відображається повідомлення про успішне відправлення форми реєстрації, чи зникає модальне вікно після натискання кнопки закриття, чи зберігаються дані в базі даних після натискання кнопки «Зберегти». Вони також перевіряють коректність навігації по сторінках, роботу випадаючих меню, слайдерів та інших інтерактивних елементів.

Для автоматизації UI-тестування використовують інструменти, такі як Selenium WebDriver, TestCafe, або Cypress. Ці інструменти дозволяють створювати сценарії, які імітують дії користувачів на вебсайті, включаючи введення тексту в поля форм, натискання кнопок, переміщення між сторінками та перевірку результатів. Автоматизація UI-тестування дозволяє швидко виявляти проблеми в інтерфейсі та забезпечувати його зручність і функціональність для кінцевих користувачів.

API (Application Programming Interface) є важливою складовою багатьох сучасних вебпроєктів, оскільки вони забезпечують взаємодію між різними компонентами системи та з зовнішніми сервісами. Автоматизоване тестування API перевіряє коректність запитів та відповідей, обробку помилок та продуктивність API.

Приклад: тестування RESTful API для інтернет-магазину може включати перевірку створення, оновлення та видалення товарів у каталозі, а також перевірку відповідей на запити про наявність товарів на складі. За допомогою інструментів, таких як Postman або SoapUI, можна автоматизувати процес відправки API-запитів та перевірки отриманих відповідей.

Автоматизоване тестування API також включає перевірку обробки помилок, наприклад, як система реагує на некоректні запити або відсутність необхідних параметрів. Це дозволяє гарантувати стабільність та надійність API, що особливо важливо для інтеграцій з зовнішніми сервісами, такими як платіжні системи або сервіси доставки.

Інтернаціоналізація (I18N) та локалізація (L10N) є важливими для вебпроєктів, що мають глобальну аудиторію. Інтернаціоналізація забезпечує можливість адаптації вебпроєкту до різних мов та регіональних налаштувань, тоді як локалізація включає адаптацію контенту до конкретних мовних та культурних вимог.

Приклад: автоматизовані тести можуть перевіряти, чи коректно відображається текст на різних мовах, чи правильні формати дати і часу, валюти та інші локалізовані елементи. Для автоматизації тестування інтернаціоналізації та локалізації використовують інструменти, такі як Selenium, що дозволяють створювати сценарії для перевірки інтерфейсу на різних мовах.

Це включає перевірку коректності перекладу текстів, правильності відображення символів та спеціальних знаків, а також перевірку відповідності культурним нормам. Наприклад, тести можуть перевіряти, чи правильно відображаються цінники у валюті, що відповідає країні користувача, чи чи коректно працюють функції, пов’язані з форматами дати та часу.

Тестування потоку даних є важливим аспектом забезпечення коректної роботи системи, оскільки воно перевіряє, як дані переміщуються між різними компонентами системи, включаючи бази даних, кеші, черги повідомлень та інші зовнішні сервіси.

Приклад: автоматизовані тести можуть перевіряти коректність збереження замовлень у базі даних, відправку повідомлень користувачам та інтеграцію з платіжними системами після завершення покупки на вебсайті. Наприклад, після оформлення замовлення автоматизовані тести можуть перевірити, чи коректно збережені дані замовлення в базі даних, чи відправлено підтвердження на електронну пошту користувача та чи проведено платіж через зовнішню платіжну систему.

Для автоматизації тестування потоку даних часто використовують інструменти, такі як Apache Kafka для перевірки черг повідомлень або SQL-тести для перевірки баз даних. Це дозволяє гарантувати, що всі частини системи працюють узгоджено та дані коректно передаються між компонентами, забезпечуючи безперебійну роботу вебпроєкту.

Використання автоматизованого тестування у різних випадках дозволяє значно підвищити якість та надійність вебпроєктів, зменшуючи витрати на ручне тестування та пришвидшуючи процес розробки. Автоматизація дозволяє зекономити час і ресурси, забезпечуючи високий рівень перевірки функціональності, продуктивності, безпеки та сумісності вебпроєкту. Впровадження автоматизованого тестування сприяє стабільності систем, підвищенню задоволеності користувачів та успішному розвитку вебпроєктів у динамічному середовищі сучасного програмного забезпечення.

В цьому розділі розкривається використання автоматизованого тестування в різних аспектах вебпроєктів, включаючи регресійне тестування, тестування продуктивності, безпеки, сумісності, інтерфейсу користувача, API, а також тестування потоку даних та інтернаціоналізації. Автоматизоване тестування значно підвищує ефективність процесу розробки, зменшує ймовірність людських помилок та забезпечує стабільність і надійність

вебпроєктів. Наведені приклади застосування автоматизації дозволяють зекономити час і ресурси, підвищити якість продукту та забезпечити безперебійне функціонування в умовах високих навантажень і постійних оновлень.

1.8 Перспективи розвитку автоматизації тестування

Автоматизація тестування відіграє важливу роль у сучасній розробці програмного забезпечення, забезпечуючи високу якість, надійність та ефективність вебпроєктів. Однак, технології та методології автоматизації постійно розвиваються, і цей розділ розгляне перспективи розвитку автоматизації тестування, включаючи нові тенденції, інструменти та підходи, що можуть значно покращити процес тестування в майбутньому.

Однією з ключових тенденцій у розвитку автоматизації тестування є вдосконалення інструментів та платформ для автоматизованого тестування. Ці інструменти стають більш інтегрованими, потужними та простими у використанні, що дозволяє зменшити час на налаштування та виконання тестів.

Такі інструменти, як Selenium, Cypress та TestCafe, постійно оновлюються та вдосконалюються, щоб забезпечити кращу підтримку нових вебтехнологій, таких як React, Angular та Vue.js. Наприклад, Cypress забезпечує швидке виконання тестів та просту інтеграцію з CI/CD процесами, що робить його одним з найпопулярніших інструментів для автоматизованого тестування вебдодатків [10].

Крім того, нові інструменти, такі як Playwright від Microsoft, пропонують розширені можливості для автоматизації тестування, включаючи підтримку різних браузерів та платформ, а також можливість тестування складних сценаріїв взаємодії користувачів з вебдодатками.

Штучний інтелект та машинне навчання відкривають нові можливості для автоматизації тестування, дозволяючи створювати більш розумні та

адаптивні тестові сценарії. Інструменти на базі штучного інтелекту можуть автоматично генерувати тести, визначати найбільш критичні частини системи для тестування та навіть передбачати можливі помилки.

Приклад: інструменти, такі як Testim та Applitools, використовують штучний інтелект для автоматизації процесу створення та виконання тестів. Testim, наприклад, використовує штучний інтелект для автоматичного запису дій користувачів та створення тестових сценаріїв, які можуть адаптуватися до змін у вебдодатку. Applitools, з іншого боку, застосовує технології комп'ютерного зору для візуального тестування інтерфейсів, дозволяючи автоматично визначати зміни в інтерфейсі, які можуть вплинути на користувацький досвід [15].

Безперервна інтеграція та доставка (CI/CD) є важливими компонентами сучасного процесу розробки програмного забезпечення, і автоматизоване тестування відіграє ключову роль у цьому процесі. Впровадження CI/CD дозволяє автоматично виконувати тести після кожного внесення змін у код, забезпечуючи високу якість програмного забезпечення та швидке виявлення помилок.

Приклад: такі платформи, як Jenkins, GitLab CI/CD та CircleCI, забезпечують інтеграцію автоматизованого тестування в процес CI/CD. Це дозволяє автоматично запускати тести після кожного комміту в репозиторій, перевіряючи нові зміни та виявляючи можливі проблеми на ранніх етапах розробки. Крім того, ці платформи підтримують паралельне виконання тестів, що значно скорочує час на їх виконання [12].

Хмарні технології відкривають нові можливості для автоматизації тестування, дозволяючи виконувати тести в масштабованих і гнучких середовищах. Використання хмарних платформ для тестування забезпечує доступ до широкого спектру конфігурацій браузерів та операційних систем, що дозволяє забезпечити високу сумісність вебпроєктів.

Приклад: платформи, такі як BrowserStack, Sauce Labs та AWS Device Farm, надають можливість виконувати автоматизовані тести в хмарних середовищах. Це дозволяє запускати тести на різних браузерах та пристроях

без необхідності підтримувати власну інфраструктуру. Крім того, хмарні платформи забезпечують масштабованість, дозволяючи виконувати тести паралельно на багатьох конфігураціях, що значно прискорює процес тестування.

Зростаюча популярність архітектури мікросервісів та контейнеризації також впливає на автоматизацію тестування. Мікросервісна архітектура дозволяє розбивати додаток на невеликі, незалежні компоненти, що спрощує тестування та забезпечує високу гнучкість у розробці.

Приклад: інструменти, такі як Docker та Kubernetes, дозволяють створювати ізольовані середовища для тестування мікросервісів. Використання Docker для контейнеризації дозволяє запускати кожен мікросервіс у власному контейнері, що забезпечує незалежність та ізоляцію тестування. Kubernetes, з іншого боку, забезпечує оркестрацію контейнерів, дозволяючи автоматично масштабувати та керувати тестовими середовищами.

Тестування інтерфейсу користувача (UI) залишається важливим аспектом автоматизованого тестування, і нові інструменти та технології роблять цей процес більш ефективним та надійним. Використання інструментів для автоматизації UI-тестування дозволяє забезпечити високу якість інтерфейсу та позитивний користувацький досвід.

Приклад: інструменти, такі як Selenium, Cypress та Puppeteer, надають потужні можливості для автоматизації UI-тестування. Cypress, наприклад, забезпечує швидке виконання тестів та просту інтеграцію з CI/CD процесами, що робить його ідеальним вибором для тестування сучасних вебдодатків. Puppeteer, розроблений Google, дозволяє автоматизувати взаємодію з браузером Chrome, включаючи навігацію, заповнення форм та перевірку результатів [13].

Контейнеризація є важливою технологією для ізольованого тестування, оскільки вона дозволяє створювати окремі середовища для кожного тестового сценарію. Це забезпечує ізоляцію тестових середовищ та мінімізує ризик впливу одних тестів на інші.

Приклад: використання Docker для контейнеризації тестових середовищ дозволяє розробникам створювати ізольовані середовища для виконання тестів. Кожен контейнер може містити необхідні залежності та конфігурації для тестування конкретного компонента або функціоналу. Це дозволяє гарантувати, що тести виконуються в стабільних та передбачуваних умовах, незалежно від змін у основній системі.

Мобільні додатки стають все більш популярними, і автоматизація тестування мобільних додатків є важливою для забезпечення їхньої якості та надійності. Нові інструменти та платформи для автоматизації тестування мобільних додатків дозволяють перевіряти їхню функціональність, продуктивність та сумісність на різних пристроях і платформах.

Приклад: інструменти, такі як Appium та Espresso, забезпечують можливість автоматизованого тестування мобільних додатків на різних платформах, включаючи iOS та Android. Appium, наприклад, дозволяє створювати тести, які можуть виконуватися на різних пристроях та емуляторах, забезпечуючи високу сумісність та якість мобільних додатків. Espresso, розроблений Google, спеціалізується на автоматизації тестування додатків для Android, забезпечуючи швидке та надійне виконання тестів [14].

Інтеграція автоматизованого тестування з процесами DevOps є важливою для забезпечення безперервного розгортання та доставки програмного забезпечення. Використання автоматизованих тестів у DevOps дозволяє забезпечити якість на кожному етапі розробки та швидко виявляти проблеми [16].

Приклад: використання таких інструментів, як Jenkins, GitLab CI/CD та Azure DevOps, дозволяє інтегрувати автоматизоване тестування в процес DevOps. Це включає автоматичне виконання тестів після кожного комміту, перевірку нових змін та розгортання програмного забезпечення на виробничі сервери. Інтеграція тестування з DevOps забезпечує швидку ітерацію розробки, високу якість програмного забезпечення та швидке розгортання нових функцій.

Віртуальна (VR) та доповнена реальність (AR) стають все більш популярними, і автоматизація тестування для цих технологій є важливою для забезпечення якості та стабільності додатків. Нові інструменти та методи для автоматизації тестування VR/AR дозволяють перевіряти функціональність та продуктивність цих додатків.

Приклад: інструменти, такі як Unity Test Framework та Oculus Test Framework, забезпечують можливість автоматизованого тестування VR/AR додатків. Unity Test Framework, наприклад, дозволяє створювати автоматизовані тести для додатків, розроблених на платформі Unity, забезпечуючи перевірку функціональності та продуктивності. Oculus Test Framework спеціалізується на автоматизації тестування додатків для платформ Oculus, забезпечуючи високу якість та стабільність VR-додатків [17].

Перспективи розвитку автоматизації тестування включають вдосконалення інструментів та технологій, інтеграцію з процесами DevOps, використання штучного інтелекту та машинного навчання, а також нові підходи до тестування мобільних додатків, VR/AR та хмарних середовищ. Ці тенденції та інновації дозволяють забезпечити високу якість та ефективність вебпроєктів, зменшуючи час та витрати на тестування та забезпечуючи стабільність та надійність програмного забезпечення. Автоматизація тестування продовжує розвиватися, адаптуючись до нових викликів та вимог сучасного світу програмного забезпечення, що дозволяє розробникам створювати ще більш складні та інноваційні продукти.

1.9 Висновки до розділу 1

Цей розділ розкриває перспективи розвитку автоматизації тестування, зокрема нові тенденції, інструменти та підходи, які можуть значно покращити процес тестування в майбутньому. Автоматизація тестування відіграє важливу роль у забезпеченні високої якості та ефективності вебпроєктів, а також дозволяє скоротити час на виконання тестів і зменшити кількість помилок.

2 ПРАКТИЧНА ЧАСТИНА

2.1 Об’єкт тестування

Об’єктом тестування є вебдодаток, який являє собою повнофункціональний інтернет-магазин (див. рис. 2.1). Даний ресурс забезпечує користувачам можливість зручно переглядати каталог товарів, додавати їх до кошика, оформлювати замовлення та взаємодіяти з іншими функціями платформи.

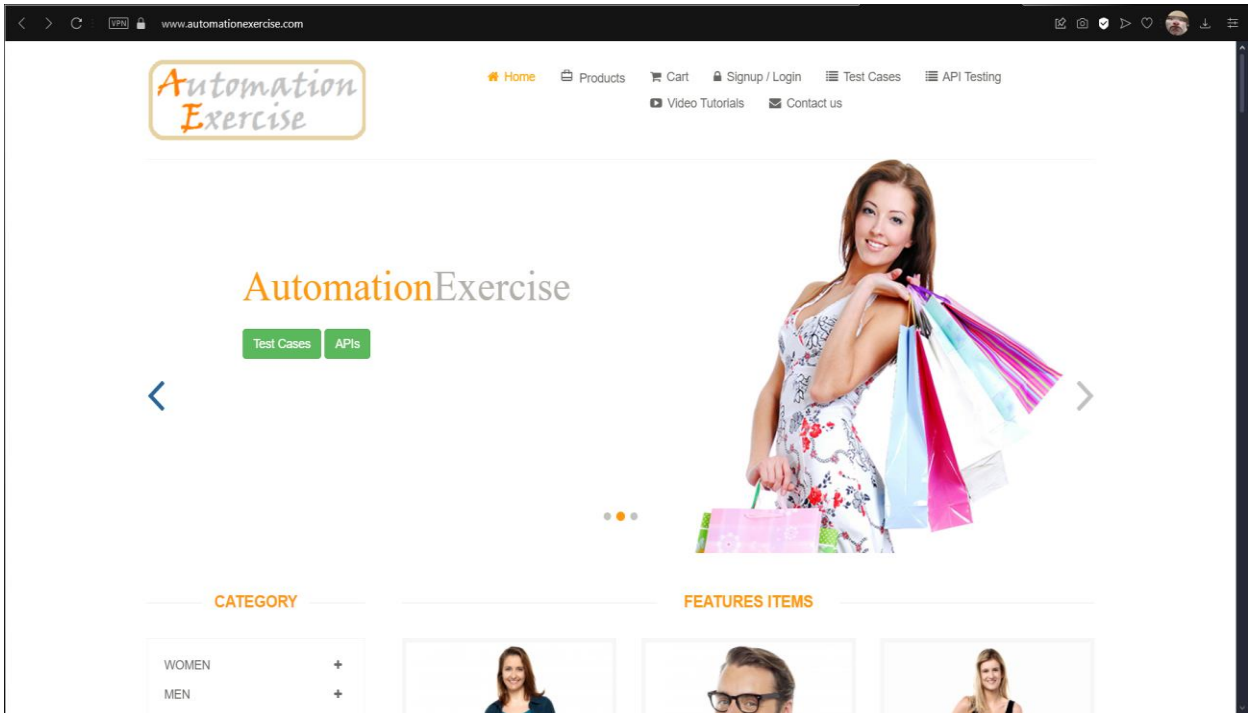


Рисунок 2.1 – Головна сторінка інтернет-магазину «Automation Exercise»

Сайт інтегрує в собі основні принципи електронної комерції, створюючи інтуїтивно зрозумілий інтерфейс і багатий функціонал для забезпечення потреб як кінцевих користувачів, так і адміністративного персоналу. Усі аспекти роботи вебдодатка спрямовані на спрощення процесу покупки, вдосконалення комунікації з клієнтами й забезпечення масштабованості для розширення бізнесу.

Вебдодаток орієнтований на широке коло користувачів: як звичайних покупців, що шукають зручний спосіб придбати товари онлайн, так і бізнес-клієнтів, які використовують сайт для просування свого бренду чи продукції. Його бізнес-логіка ґрунтується на інтерактивності, масштабованості та надійності, що робить платформу ефективним інструментом для реалізації сучасних стратегій електронної комерції.

У цьому розділі розглянуто вебдодаток як об’єкт тестування, який представляє собою повноцінний інтернет-магазин. Він забезпечує зручність для користувачів завдяки інтуїтивно зрозумілому інтерфейсу та функціям для перегляду товарів, оформлення замовлень і взаємодії з іншими можливостями платформи. Цей ресурс орієнтований як на звичайних покупців, так і на бізнес-клієнтів, надаючи ефективні інструменти для розвитку електронної комерції.

2.2 Структура бізнес-логіки сайту

Для аналізу об’єкта тестування важливо детально розглянути основні функціональні модулі, що забезпечують роботу вебдодатка.

Модуль управління користувачами є основою персоналізованого досвіду. Він охоплює функції реєстрації, авторизації, роботи з особистим кабінетом і забезпечення безпеки облікових записів.

Реєстрація нових користувачів реалізована через зручну вебформу, де користувач вводить свої дані, такі як ім’я, адреса електронної пошти, пароль і, за потреби, контактний номер телефону (див. рис. 2.2). У процесі реєстрації система автоматично перевіряє коректність введеної інформації. Наприклад, перевірка адреси електронної пошти гарантує, що вона має правильний формат, а введений пароль відповідає мінімальним вимогам безпеки, таким як наявність великих і малих літер, цифр і спеціальних символів. Крім цього, система автоматично блокує можливість повторної реєстрації на ту саму адресу електронної пошти, щоб уникнути дублювання облікових записів.

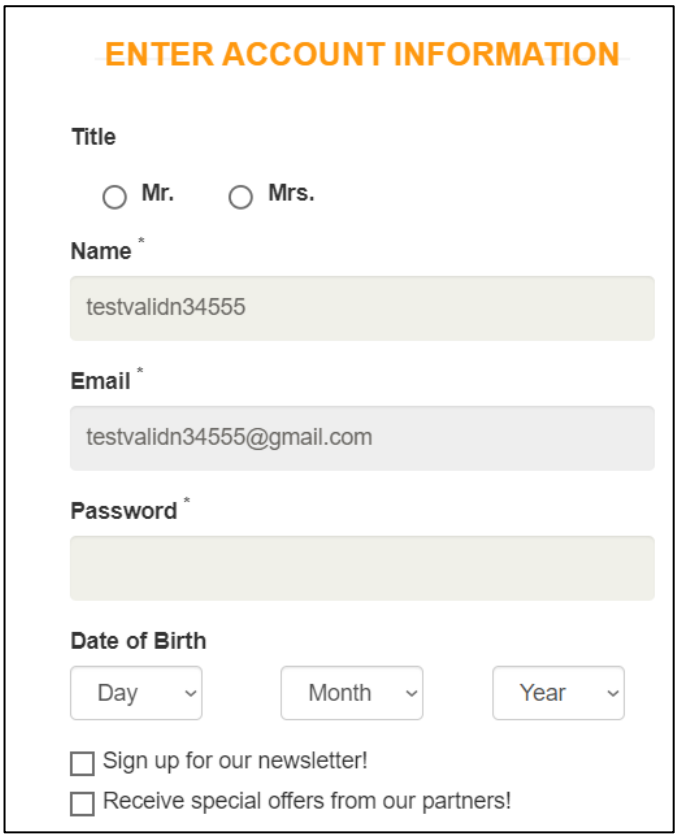


Рисунок 2.2 – Сторінка форми реєстрації

Каталог є ядром вебдодатка, що дозволяє користувачам швидко й ефективно знайти необхідні товари, ознайомитися з їх характеристиками та зробити усвідомлений вибір (див. рис. 2.3).

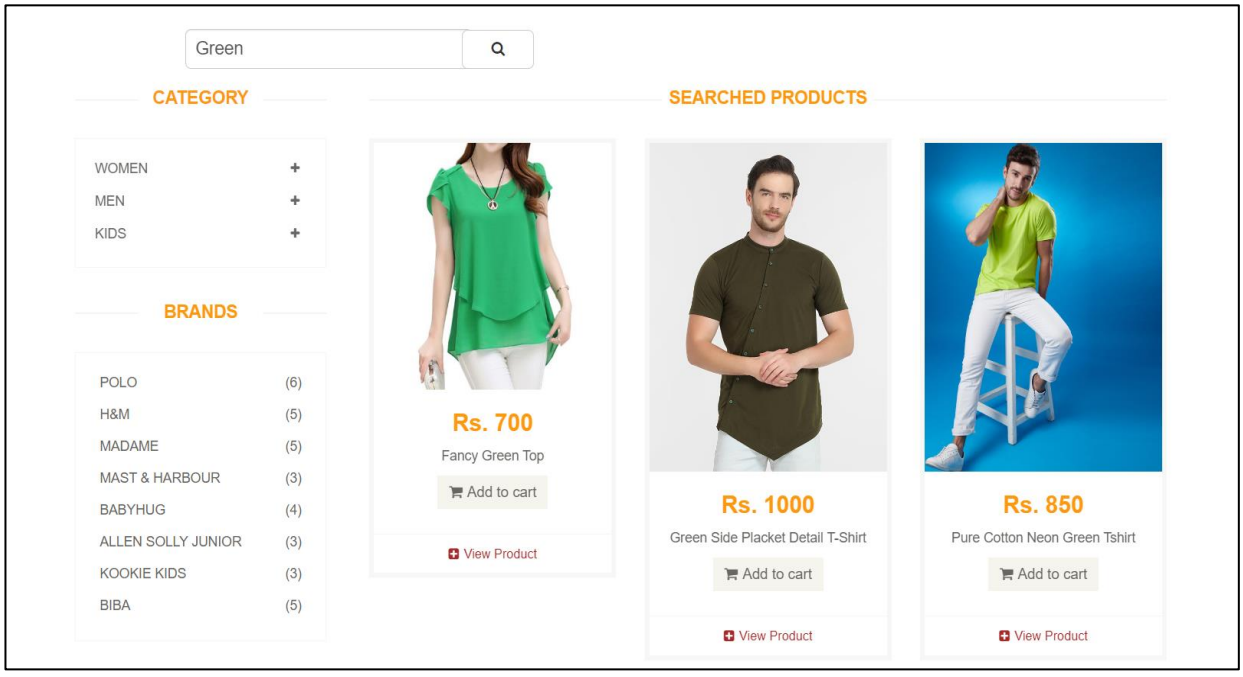


Рисунок 2.3 – Пошук у каталозі товарів

Структура каталогу побудована на принципі категоризації, де всі товари згруповані в основні категорії. Кожна категорія додатково поділена на підкатегорії для полегшення навігації. Наприклад, у розділі «Одяг» можна знайти підкатегорії «Чоловічий одяг», «Жіночий одяг», «Дитячий одяг». Така ієрархічна структура дає змогу користувачам з легкістю знаходити товари, які їх цікавлять.

Кожен товар у каталозі супроводжується зображеннями високої якості, що дозволяють розглянути продукт у деталях, коротким описом, який містить основні характеристики (наприклад, матеріал, виробник, ключові особливості), а також ціною. Відображення рейтингу продукту на основі відгуків користувачів сприяє формуванню довіри до товару.

Функція пошуку інтегрована в каталог і дозволяє вводити ключові слова, щоб знайти потрібний товар. Додатково система пропонує інструменти фільтрації, які допомагають звузити вибір за різними параметрами, такими як діапазон цін, рейтинг, популярність, категорія або навіть колір.

Детальна сторінка товару надає користувачеві повну інформацію про продукт (див. рис. 2.4). Крім основного опису й зображень, там розміщені відгуки інших покупців, які можуть містити коментарі та фото.

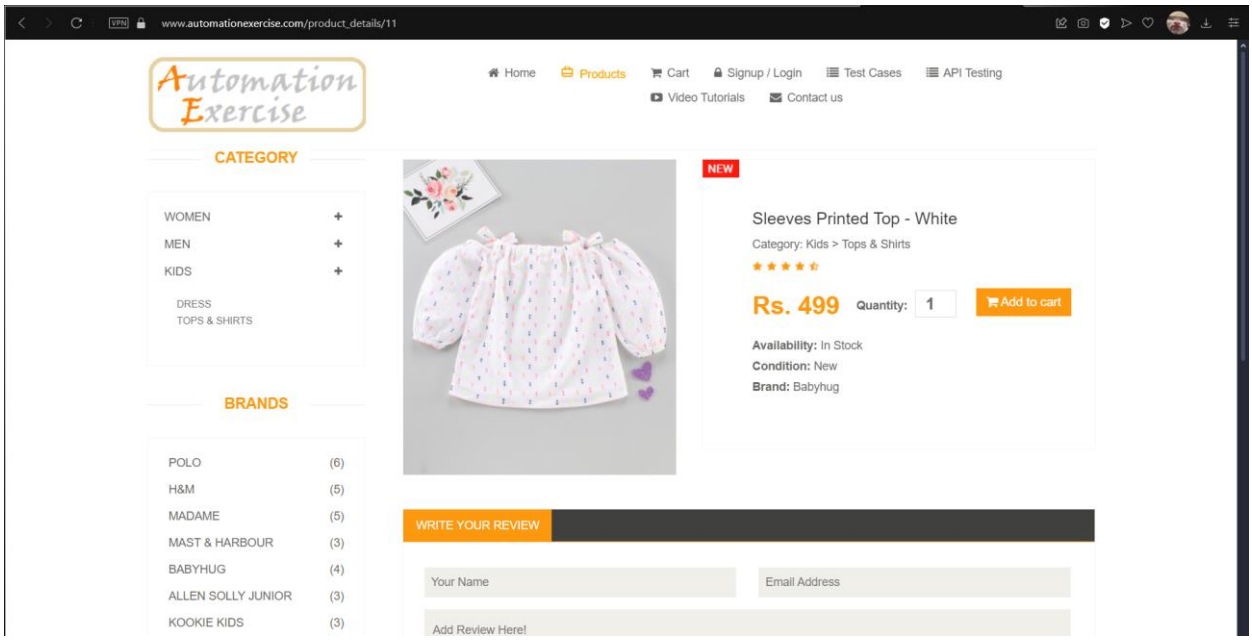


Рисунок 2.4 – Сторінка товару

Кошик дозволяє користувачеві зберігати обрані товари для подальшої покупки (див. рис. 2.5). У кошику відображаються всі обрані продукти, їх кількість, ціна, а також підсумкова вартість. Якщо на товари діють знижки або акції, система автоматично розраховує кінцеву ціну з урахуванням цих умов.

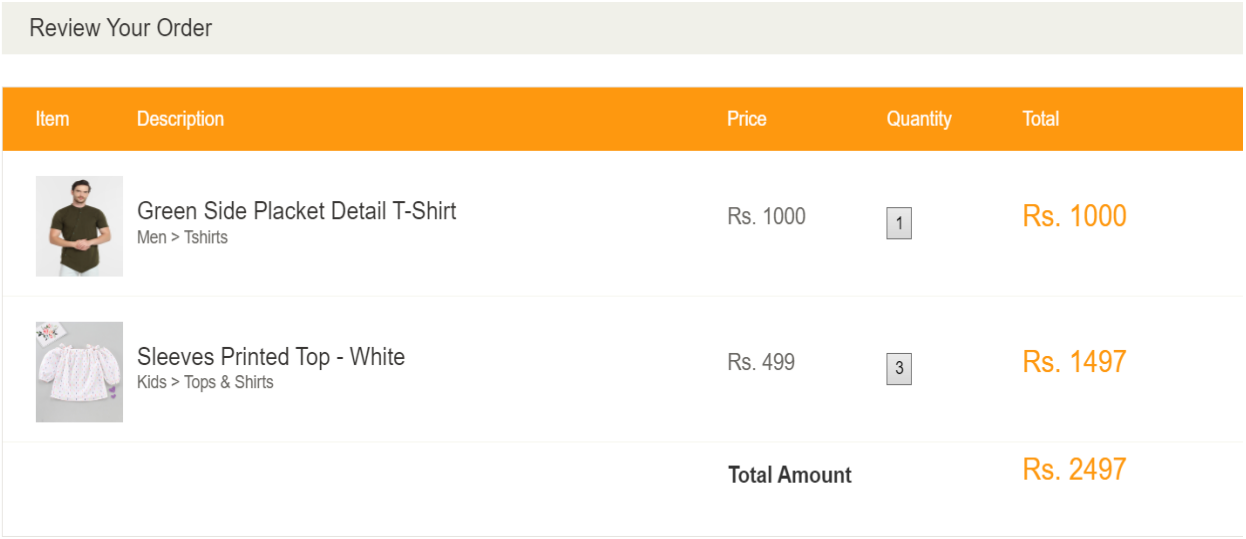


Рисунок 2.5 – Сторінка кошику товару

Користувач може змінювати кількість обраних товарів, видаляти непотрібні позиції або додавати нові. Інтерфейс кошика простий і зрозумілий, що знижує ризик помилок під час взаємодії.

Описана структура бізнес-логіки демонструє, як кожен модуль інтегрується в загальну систему. Вона побудована таким чином, щоб забезпечити максимальну зручність для користувачів, високу ефективність процесів і гнучкість у подальшому розширенні функціональності.

У цьому розділі розглянута структура бізнес-логіки вебдодатку, що включає основні функціональні модулі, які забезпечують ефективну роботу сайту. Описано процеси реєстрації користувачів, управління каталогом товарів, пошуку, а також роботу кошика для покупок. Кожен з цих модулів сприяє зручності користування та підвищує ефективність роботи інтернет-магазину, забезпечуючи швидке знаходження товарів, простоту оформлення покупок і надійність взаємодії з користувачами.

2.3 Інтеграція автоматизованих тестів

Для автоматизації тестування використовувався Selenium WebDriver, який дозволяє емулювати дії реального користувача у браузері, виконуючи різні сценарії роботи з сайтом, такі як реєстрація, авторизація, додавання товарів до кошика, оформлення замовлення тощо.

Першим етапом реалізації автоматизованого тестування стало визначення ключових функціональних модулів сайту. До них відносяться модулі реєстрації користувачів, авторизації, управління кошиком, пошуку товарів і оформлення замовлень.

Аналіз функціоналу показав, що ці області є найбільш важливими для забезпечення позитивного досвіду користувача і повинні бути ретельно перевірені. Основний акцент був зроблений на функціональні тести, які перевіряли основні сценарії використання сайту, регресійні тести для виявлення можливих помилок після внесення змін у код, а також тести інтерфейсу користувача для забезпечення коректної роботи елементів UI.

У процесі тестування було реалізовано тест на перевірку функції реєстрації користувачів (див. рис. 2.6). Написаний код тесту емулює дії, які виконує звичайний користувач (див. Додаток А).

```
// Введення даних користувача
WebElement nameField = driver.findElement(By.name("name"));
WebElement emailField = driver.findElement(By.cssSelector("[data-qa='signup-email']"));

// Добавляем замедление перед вводом данных
Thread.sleep( millis: 1000); // Задержка в 1 секунду
nameField.sendKeys( ...keysToSend: "Test User12");

Thread.sleep( millis: 1000); // Задержка в 1 секунду
emailField.sendKeys( ...keysToSend: "testuser@example86.com");

WebElement signupButton = driver.findElement(By.xpath( xpathExpression: "//button[text()='Signup']"));
```

Рисунок 2.6 – Код проходження першого етапу тестування

Тест перевіряє працездатність функції реєстрації. Процес починається з відкриття головної сторінки сайту. За допомогою Selenium тест знаходить

кнопку «Signup/Login», яка відповідає за перехід до форми реєстрації та авторизації, і виконує клік по ній.

Далі тест емулює введення даних у поля для імені та електронної пошти. Selenium ідентифікує ці поля за їхніми атрибутами. Коли дані введено, тест натискає кнопку «Signup». На цьому етапі система повинна відкрити сторінку з формою для введення додаткової інформації.

На новій сторінці тест заповнює форму реєстрації: вводить пароль, вибирає дату народження з відповідних випадаючих списків, обирає стать за допомогою радіо-кнопки. Далі заповнюються поля для введення імені, прізвища, адреси, міста, штату, поштового індексу, країни та номеру телефону (див. рис. 2.7).

```
WebElement cityField = driver.findElement(By.id("city"));
Thread.sleep( millis: 1000); // Задержка перед вводом города
cityField.sendKeys( ...keysToSend: "Test City");

WebElement stateDropdown = driver.findElement(By.id("state"));
Thread.sleep( millis: 1000); // Задержка перед выбором штата
stateDropdown.sendKeys( ...keysToSend: "California");

WebElement postalCodeField = driver.findElement(By.id("zipcode"));
Thread.sleep( millis: 1000); // Задержка перед вводом почтового индекса
postalCodeField.sendKeys( ...keysToSend: "12345");

WebElement countryDropdown = driver.findElement(By.id("country"));
Thread.sleep( millis: 1000); // Задержка перед выбором страны
countryDropdown.sendKeys( ...keysToSend: "United States");

WebElement phoneField = driver.findElement(By.id("mobile_number"));
Thread.sleep( millis: 1000); // Задержка перед вводом номера телефона
phoneField.sendKeys( ...keysToSend: "1234567890");
```

Рисунок 2.7 – Код проходження другого етапу тестування

Після заповнення всіх необхідних полів тест натискає кнопку «Create Account». Система повинна відобразити повідомлення «Account Created!», яке підтверджує успішну реєстрацію. Якщо повідомлення присутнє на сторінці, тест виводить у консоль інформацію про успішне створення облікового запису, у разі відсутності цього повідомлення тест вважається проваленим (див. рис. 2.8).

```
Test Passed: Account created successfully.

Process finished with exit code 0
```

Рисунок 2.8 – Повідомлення «Тест пройдений успішно»

Ще один важливий сценарій, який було автоматизовано, стосується додавання товару до кошика (див. Додаток Б).

Тест спрямований на перевірку функціональності додавання товарів до кошика (див. рис. 2.9). Він починається з відкриття сторінки каталогу товарів. Selenium знаходить кнопку «Add to cart» для вибраного товару за допомогою XPath-запиту та виконує клік, щоб додати товар до кошика.

```
// Відкриття сторінки каталогу
driver.get("https://www.automationexercise.com/products");

// Додавання першого товару до кошика
WebElement firstAddToCartButton = driver.findElement(By.xpath( xpathExpression: "(//a[text()='Add to cart'])[1]"));
firstAddToCartButton.click();

// Очікування та перевірка повідомлення про успіх
Thread.sleep( millis: 2000); // Затримка для завантаження елементу
WebElement firstSuccessMessage = driver.findElement(By.xpath( xpathExpression: "//div[@class='alert-success']"));
if (firstSuccessMessage.isDisplayed()) {
    System.out.println("Test Passed: First item successfully added to cart.");
} else {
    System.out.println("Test Failed: First item was not added to cart.");
}
```

Рисунок 2.9 – Код перевірки додавання першого товару

Після цього система повинна показати повідомлення про успішне додавання. Тест перевіряє наявність цього повідомлення, ідентифікуючи його за атрибутами, такими як клас або XPath. У випадку успішного відображення повідомлення сценарій вважається виконаним. Якщо повідомлення не з’являється, виводиться інформація про помилку.

Для розширеної перевірки тест додає кілька товарів до кошика (див. рис. 2.10). Потім здійснюється перехід на сторінку кошика, де перевіряється, чи всі додані товари коректно відображаються у списку. Цей сценарій

допомагає переконатися, що логіка додавання товарів працює правильно, система оновлює інформацію про кошик у реальному часі, а користувач отримує актуальні повідомлення про успішність дій (див. рис. 2.11).

```
// Повернення до каталогу і додавання другого товару
driver.navigate().back();
WebElement secondAddToCartButton = driver.findElement(By.xpath( xpathExpression: "(//a[text()='Add to cart'])[2]"));
secondAddToCartButton.click();

// Очікування та перевірка повідомлення про успіх
Thread.sleep( millis: 2000); // Затримка для завантаження елементу
WebElement secondSuccessMessage = driver.findElement(By.xpath( xpathExpression: "//div[@class='alert-success']"));
if (secondSuccessMessage.isDisplayed()) {
    System.out.println("Test Passed: Second item successfully added to cart.");
} else {
    System.out.println("Test Failed: Second item was not added to cart.");
}
```

Рисунок 2.10 – Код перевірки додавання другого товару

```
    // Перевірка, чи товари відображаються в кошику
    Thread.sleep( millis: 2000); // Затримка для завантаження сторінки кошика
    WebElement cartItem1 = driver.findElement(By.xpath( xpathExpression: "//tr[1]//td[@class='cart_description']"));
    WebElement cartItem2 = driver.findElement(By.xpath( xpathExpression: "//tr[2]//td[@class='cart_description']"));

    if (cartItem1.isDisplayed() && cartItem2.isDisplayed()) {
        System.out.println("Test Passed: Both items are displayed in the cart.");
    } else {
        System.out.println("Test Failed: Items are not displayed correctly in the cart.");
    }
} catch (InterruptedException e) {
    e.printStackTrace();
} finally {
```

Рисунок 2.11 – Код перевірки відображення товару у кошику

Окрім цього, було реалізовано автоматизований сценарій оформлення замовлення (див. Додаток В).

Цей сценарій є більш складним і перевіряє весь процес покупки від додавання товарів до кошика до підтвердження замовлення. Першим кроком тест відкриває сторінку каталогу, додає товар до кошика і переходить до сторінки перегляду кошика, натискаючи кнопку «View Cart» (див. рис. 2.12).

Далі тест переходить до етапу оформлення замовлення, натискаючи кнопку «Proceed To Checkout». Selenium вводить дані користувача для доставки, такі як адреса і місто, заповнюючи відповідні поля форми. Після

введення даних натискається кнопка «Place Order», яка завершує процес замовлення.

```
try {
    // Відкриття сторінки каталогу та додавання товару
    driver.get("https://www.automationexercise.com/products");
    WebElement addToCartButton = driver.findElement(By.xpath( xpathExpression: "//a[text()='Add to cart'][1]"));
    addToCartButton.click();

    // Перехід до кошика
    WebElement viewCartButton = driver.findElement(By.xpath( xpathExpression: "//a[text()='View Cart']"));
    viewCartButton.click();

    // Перехід до оформлення замовлення
    WebElement checkoutButton = driver.findElement(By.xpath( xpathExpression: "//a[text()='Proceed To Checkout']"));
    checkoutButton.click();

    // Введення даних доставки
    WebElement addressField = driver.findElement(By.name("address"));
    WebElement cityField = driver.findElement(By.name("city"));
    WebElement confirmOrderButton = driver.findElement(By.xpath( xpathExpression: "//button[text()='Place Order']"));

    addressField.sendKeys( ...keysToSend: "123 Test Street");
    cityField.sendKeys( ...keysToSend: "Test City");
    confirmOrderButton.click();
```

Рисунок 2.12 – Код проходження етапу оформлення замовлення

На заключному етапі тест перевіряє, чи відображається повідомлення про успішне оформлення замовлення з текстом «Order Placed!».

Якщо повідомлення є, тест вважається успішним. У разі його відсутності тест повідомляє про збій (див. рис. 2.13).

```
    // Перевірка успішного оформлення
    WebElement successMessage = driver.findElement(By.xpath( xpathExpression: "//h2[text()='Order Placed!']"));
    if (successMessage.isDisplayed()) {
        System.out.println("Test Passed: Order placed successfully.");
    } else {
        System.out.println("Test Failed: Order not placed.");
    }
} finally {
    driver.quit();
}
```

Рисунок 2.13 – Код перевірки успішного оформлення

Крім функціональних тестів, значну увагу було приділено регресійному тестуванню. Внесення змін у код, наприклад, оновлення дизайну чи додавання

нових функцій, могло впливати на існуючий функціонал. Автоматизовані скрипти допомагали швидко перевіряти всі основні сценарії, щоб переконатися у відсутності негативного впливу. Це включало перевірку роботи пошуку, фільтрів товарів, оновлення персональних даних у кабінеті користувача та інших ключових функцій.

Тестування інтерфейсу користувача забезпечило перевірку коректного відображення елементів на різних розширеннях екрану, включаючи мобільні пристрої. Автоматизовані тести емулювали дії користувачів, які переглядають сайт на смартфонах і планшетах, щоб забезпечити повну адаптивність дизайну та зручність користування.

Реалізація автоматизованого тестування значно підвищила ефективність перевірки якості сайту, зменшила час на тестування та забезпечила стабільність роботи навіть після регулярних оновлень коду. Це зробило процес тестування систематизованим і ефективним, дозволяючи швидко реагувати на будь-які проблеми.

2.4 Висновки до розділу 2

У цьому розділі розглянута інтеграція автоматизованих тестів за допомогою Selenium WebDriver для перевірки ключових функціональних модулів вебсайту. Описано процеси тестування реєстрації користувачів, додавання товарів до кошика, оформлення замовлень і регресійне тестування. Розроблені сценарії забезпечили ефективну перевірку функціональності сайту, виявлення помилок після змін у коді, а також перевірку адаптивності інтерфейсу.

3 АНАЛІЗ ТЕСТУВАННЯ

3.1 Сценарії взаємодії

В рамках автоматизованого тестування було перевірено найбільш критичні сценарії взаємодії користувачів з сайтом.

Реєстрація та авторизація: для тестування цього функціоналу було розроблено набір тестів, що перевіряють коректність заповнення форми реєстрації, валідацію введених даних (ім’я, електронна пошта, пароль тощо), а також можливість входу користувача за допомогою правильних та неправильних облікових даних.

Кошик і оформлення замовлення: важливою частиною тестування було перевірка коректності додавання товарів до кошика, зміни кількості товару, а також правильного підрахунку вартості замовлення з урахуванням знижок і податків. Також проводились тести для перевірки процесу оформлення замовлення, включаючи вибір способу оплати та доставки.

Перевірка адаптивності сайту: оскільки сайт повинен коректно працювати на різних пристроях, було створено тести для перевірки відображення сайту на мобільних пристроях та планшетах, що включає перевірку шрифтів, кнопок, зображень та інших елементів інтерфейсу.

Валідація введених даних: окремі тести були розроблені для перевірки коректності введення даних в поля форм (наприклад, адреса доставки, номер телефону, платіжні реквізити). Ці тести дозволили виявити помилки у валідації та виправити їх до запуску на реальному сайті.

Автоматизоване тестування сценаріїв взаємодії користувачів з сайтом продемонструвало ефективність та надійність основних функціональних процесів.

Таким чином, тестування критичних сценаріїв взаємодії забезпечило виявлення потенційних помилок та підтвердило готовність сайту до безперебійної роботи в реальних умовах.

У цьому розділі розглянуто автоматизовані сценарії тестування ключових взаємодій користувачів з вебсайтом. Зокрема, перевірено процеси реєстрації та авторизації, коректність роботи кошика, оформлення замовлення, а також адаптивність сайту на мобільних пристроях. Окремо були протестовані механізми валідації введених даних для забезпечення коректності інформації.

3.2 Опис виявлених помилок

Під час тестування реєстрації на сайті було виявлено, що система не здійснює належної перевірки валідності введеної електронної пошти. Користувач може ввести некоректну адресу електронної пошти. В результаті цього система дозволяє створювати облікові записи з недійсними або неповними адресами.

Баг у процесі реєстрації користувача:

– причина: відсутність належної валідації введених даних на стороні сервера – це може бути результатом неефективної або відсутньої логіки для перевірки правильності електронної пошти в процесі реєстрації;
– виправлення: для усунення цієї проблеми необхідно додати валідацію введеної адреси на сервері, крім того, варто реалізувати перевірку на синтаксичні помилки в адресі пошти ще на етапі введення, щоб користувач міг негайно побачити повідомлення про помилку.

Під час тестування функціональності додавання товарів до кошика виявлено помилку, коли товар додається до кошика, але насправді він не зберігається в базі даних. Користувач отримує повідомлення про успішне додавання товару до кошика, але після переходу до кошика товар відсутній. Ця проблема виникає при одночасному додаванні кількох товарів або при використанні певних категорій товарів. У деяких випадках сайту не вдається оновити список товарів у кошику, хоча на екрані це підтверджується.

Помилка в додаванні товару до кошика:

– причина: проблема може бути викликана неповною або некоректною обробкою сесій користувача або асинхронними запитами до серверу, коли система не встигає оновити кошик на сторінці після додавання товару;
– виправлення: для вирішення цієї проблеми необхідно провести оптимізацію обробки кошика, додавши механізм, який підтверджує успішне додавання товару до кошика перед відображенням повідомлення на сторінці; також слід перевірити правильність роботи асинхронних запитів і оновлення інформації в реальному часі.

Проблема була виявлена при оформленні замовлення. Система іноді неправильно розраховує вартість замовлення, особливо коли користувач змінює кількість товарів. В результаті користувач може отримати некоректну суму для оплати, що є великою проблемою для функціонування сайту.

Неправильний розрахунок вартості замовлення:

– причина: баг може бути пов’язаний з помилкою в обробці логіки знижок або неправильним оновленням даних в корзині під час зміни кількості товару;
– виправлення: для виправлення цієї проблеми потрібно ретельно перевірити код, що відповідає за перерахунок вартості при зміні кількості товару або застосуванні знижок; також важливо переконатися, що система правильно взаємодіє з базою даних для оновлення ціни в реальному часі.

Під час оформлення замовлення, коли користувач переходить до етапу підтвердження замовлення, іноді з’являється ситуація, коли на сторінці не з’являється повідомлення про успішне оформлення або підтвердження замовлення. У деяких випадках користувач також не отримує квитанцію про покупку на свою електронну пошту, хоча система повинна автоматично надсилати підтвердження після завершення процесу замовлення. Це викликає непорозуміння у користувачів, які можуть вважати, що замовлення не було успішно оформлено, або вони не отримують інформацію про свою покупку.

Збої на етапі підтвердження замовлення:

– причина: причиною цієї проблеми може бути неполадки в механізмі надсилання електронних листів або проблеми з сервером при обробці запиту на підтвердження замовлення; також проблема може бути пов’язана з помилками в роботі серверних скриптів, які відповідають за підтвердження та відправку листів;
– виправлення: для усунення цієї проблеми слід детально перевірити логіку надсилання електронних листів та підтвердження замовлення; рекомендується перевірити конфігурацію поштового сервера та систему обробки запитів, щоб упевнитися в їх коректній роботі.

Під час тестування було виявлено кілька критичних помилок, що потребують усунення для забезпечення коректної роботи вебсайту. Усі ці проблеми мають бути ретельно перевірені та виправлені, щоб покращити користувацький досвід і забезпечити безперебійну роботу сайту.

В цьому розділі було розглянуто виявлені критичні помилки під час тестування вебсайту, які потребують термінового усунення для забезпечення стабільної роботи системи. Основні проблеми включають недостатню валідацію електронних поштових адрес під час реєстрації, некоректну обробку сесій та асинхронних запитів при додаванні товарів до кошика, помилки в розрахунку вартості замовлення, а також збої на етапі підтвердження замовлення та надсилання квитанцій. Всі ці проблеми можуть негативно вплинути на користувацький досвід.

3.3 Розподіл помилок та відсотки виявлення

В результаті автоматизованого тестування сайту були виявлені кілька категорій помилок, які стосуються різних аспектів функціональності вебдодатка. Усі знайдені баги були класифіковані відповідно до типу їх прояву і важливості для користувачів та адміністраторів сайту. На основі проведених

тестів та зібраних результатів, можна визначити кількість виявлених помилок і їхній відсотковий розподіл серед усіх тестованих функцій.

Нижче наведено загальний огляд помилок, які були виявлені під час тестування.

Функціональні помилки: найбільшу частку серед виявлених помилок складають функціональні помилки, що виникали в процесі взаємодії користувачів з основними елементами сайту, такими як реєстрація, авторизація та оформлення замовлень. Це близько 40% від усіх виявлених помилок. Найбільше помилок було виявлено в процесах реєстрації та валідації даних, де деякі поля не проходили коректну перевірку. Також були виявлені помилки при обробці замовлень, зокрема підрахунок вартості замовлення іноді не враховував знижки або податки, що призводило до некоректних результатів.

Помилки інтерфейсу: інтерфейс сайту також зазнав кількох проблем, пов’язаних із адаптивністю та відображенням елементів на різних пристроях. Ці помилки склали 25% від загальної кількості помилок. Деякі елементи, такі як кнопки і текстові поля, не відображались належним чином на мобільних пристроях, що порушувало зручність використання сайту. Це могло призводити до негативного користувацького досвіду, особливо серед користувачів, які використовують смартфони або планшети.

Інтеграційні помилки: окрему групу складали інтеграційні помилки, що виникали при взаємодії сайту з зовнішніми системами, такими як платіжні шлюзи. Ці помилки становили 15% від загальної кількості. Проблеми були пов’язані з обробкою платежів через сторонні сервіси, де іноді відбувались збої під час підтвердження транзакцій або відображення статусу платежу.

Помилки валідації даних: виявлено ряд помилок, пов’язаних з неправильним введенням або відсутністю валідації деяких полів, таких як адреса доставки або номер телефону. Це становить 10% від усіх помилок. Наприклад, система не завжди коректно обробляла випадки, коли користувач вводив неправильний формат номера телефону або адресу електронної пошти.

Помилки продуктивності та стабільності: виявлені незначні проблеми, що стосуються продуктивності сайту, коли під час навантаження відбувався невеликий сплеск затримок при обробці великих обсягів запитів або при паралельному виконанні тестів на кількох пристроях. Ці проблеми становили 10% від усіх знайдених помилок.

Розподіл помилок можна подати у вигляді відсоткових значень, що демонструють, де найбільше зусиль потрібно спрямувати для покращення якості продукту:

– 40% функціональні помилки;
– 25% помилки інтерфейсу;
– 15% інтеграційні помилки;
– 10% помилки валідації даних;
– 10% проблеми продуктивності та стабільності.

Загалом, автоматизоване тестування виявило важливі проблеми, які можуть вплинути на якість користувацького досвіду. Виявлені помилки були класифіковані за типами і надають чітке розуміння тих аспектів, над якими необхідно працювати для подальшого покращення вебдодатку. Хоча деякі з помилок були не критичними, їх наявність все одно могла б призвести до негативних наслідків для кінцевих користувачів, зокрема до зниження зручності користування сайтом або проблем з виконанням основних функцій.

Особливу увагу необхідно звернути на функціональні помилки, оскільки саме ці проблеми можуть найбільше вплинути на ефективність та зручність використання платформи, таких як реєстрація, авторизація і оформлення замовлення. Також важливо усунути помилки, пов’язані з інтеграцією з платіжними системами, оскільки вони можуть серйозно вплинути на фінансові транзакції і довіру користувачів до сайту.

Помилки в інтерфейсі, зокрема адаптивність на мобільних пристроях, є важливими для забезпечення бездоганного користувацького досвіду, оскільки більшість користувачів здійснює покупки саме через мобільні телефони. Для цього необхідно впевнитися, що всі елементи інтерфейсу коректно відображаються на різних екранах та в різних роздільних здатностях.

Також варто зосередити увагу на проблемах продуктивності та стабільності. Важливо, щоб сайт працював швидко і стабільно навіть при високому навантаженні. Це забезпечить комфортну роботу для користувачів, незалежно від кількості одночасних запитів або обсягів даних.

Завдяки виявленим помилкам було сформовано чітке бачення тих аспектів, що потребують покращення.

3.4 Висновки до розділу 3

В цьому розділі були розглянуті результати тестування з акцентом на типи та розподіл помилок, виявлених під час автоматизованого тестування сайту. Аналіз показав, що найбільшу частину виявлених помилок складають функціональні помилки, пов’язані з процесами реєстрації, оформлення замовлень та валідації даних.

ВИСНОВКИ

У ході роботи було проведено автоматизацію тестування вебпроєкту з використанням Selenium та WebDriver. Ці інструменти забезпечили можливість ефективного створення та виконання тестових сценаріїв, що симулюють поведінку користувачів у вебдодатку.

Автоматизація тестування охопила ключові функціональні області вебпроєкту, включаючи перевірку коректності відображення елементів інтерфейсу, обробку форм, роботу з динамічним контентом та функції аутентифікації. Для цього було розроблено та виконано тестові сценарії, які реалізували перевірку функціоналу за допомогою мови програмування Java , з використанням можливостей WebDriver для управління браузером.

Для організації процесу автоматизованого тестування було розроблено специфікації, які дозволили структуровано описати тестову стратегію та підходи до перевірки функціональності вебпроєкту. Було визначено взаємозв’язок між тестовими сценаріями, браузерними драйверами та даними, які використовуються для тестування, що забезпечило чітку структуризацію процесу автоматизації.

Автоматизація тестування також продемонструвала можливість інтеграції з інструментами CI/CD, що дозволило забезпечити регулярне виконання тестів під час внесення змін у проєкт. Це сприяло підвищенню ефективності процесу розробки та тестування.

Загалом, використання Selenium та WebDriver підтвердило свою ефективність для автоматизації тестування вебпроєктів, забезпечуючи швидке виявлення помилок, зменшення ручної праці та покращення загальної якості продукту.

ПЕРЕЛІК ПОСИЛАНЬ

1. Avasarala S. Selenium WebDriver Practical Guide: Automated Testing for Web Applications. Packt Publishing, 2018. 340 p.

2. Buelta J. Python Automation Cookbook: Explore the world of automation using Python scripts with practical case studies and examples. Packt Publishing, 2018. 366 p.

3. Cocchiaro C. Selenium Framework Design in Data-Driven Testing: Build data-driven test frameworks using Selenium WebDriver. Apress, 2021. 385 p.

4. Collin M. Mastering Selenium WebDriver: Increase the performance, capability, and reliability of your automated checks by mastering Selenium WebDriver. Packt Publishing, 2020. 412 p.

5. Collin M. Mastering Selenium WebDriver 3.0: Boost the performance and reliability of your automated checks by mastering Selenium WebDriver, 2nd Edition. Packt Publishing, 2019. 401 p.

6. El-Far Z. Automated Software Testing with Selenium and WebDriver. Apress, 2019. 303 p.

7. Farcic V., Garcia A. Test-Driven Java Development: Invoke TDD principles for end-to-end application development with Java. Apress, 2020. 390 p.

8. Gundecha U. Selenium Testing Tools Cookbook: Over 90 recipes to help you build and run automated tests for your web applications with Selenium WebDriver. Packt Publishing, 2012. 300 p.

9. Gutmann M. Web Testing with Katalon Studio: A Step-by-Step Guide to Automated Web Testing. Apress, 2020. 311 p.

10. Humble J., Farley D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional, 2010. 512 p.

11. Johansen C. Test-Driven JavaScript Development. Addison-Wesley Professional, 2013. 496 p.

12. Kaczanowski T. Practical Unit Testing with JUnit and Mockito. Apress, 2019. 312 p.

13. Kapelonis K. Java Testing with Spock. Manning Publications, 2020. 360 p.

14. Percival H. Test-Driven Development with Python: Obey the Testing Goat: Using Django, Selenium, and JavaScript. O'Reilly Media, Inc., 2017. 614 p.

15. Prasad R. Learning Selenium Testing Tools – Third Edition: Explore automated testing with Selenium WebDriver. Packt Publishing, 2020. 434 p.

16. Rani D. Software Testing: Concepts and Operations. Apress, 2017. 468 p.

17. Rasmusson J. The Way of the Web Tester: A Beginner's Guide to Automating Tests. Test-Driven Technology, 2016. 244 p.

18. Rossel S. Continuous Integration, Delivery, and Deployment: Reliable and faster delivery of high-quality software from conception to production. Apress, 2017. 415 p.

19. Sharma A. Test Automation for Web Applications: Modern Techniques to Speed Up Web Application Testing. Packt Publishing, 2016. 284 p.

20. Szegedi L. Automated Testing in DevOps: A guide for developers and testers. Packt Publishing, 2019. 332 p.

21. Zhan Z. Selenium WebDriver Recipes in C#: Second Edition. Apress, 2021. 386 p.

22. Zhan Z. Selenium WebDriver Recipes in Java: The problem-solving guide to Selenium WebDriver in Java. Apress, 2015. 278 p.

ДОДАТОК А

Реалізація тесту на перевірку функції реєстрації користувачів

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

public class main {
   public static void main(String[] args) {

      System.setProperty("webdriver.chrome.driver",
"C:\\Users\\ROCK_Iknow\\Desktop\\chromedriver\\chromedriver.exe");
      WebDriver driver = new ChromeDriver();
      try {
         driver.get("https://www.automationexercise.com");

         WebElement signupLoginButton = driver.findElement(By.xpath("//a[text()=' Signup / 
Login']"));
         signupLoginButton.click();

         WebElement nameField = driver.findElement(By.name("name"));
         WebElement emailField = driver.findElement(By.cssSelector("[data-qa='signup-
email']"));

         Thread.sleep(1000);
         nameField.sendKeys("Test User12");

         Thread.sleep(1000);
         emailField.sendKeys("testuser@example12345.com");

         WebElement signupButton = driver.findElement(By.xpath("//button[text()='Signup']"));
```

```
        Thread.sleep(1000); // Задержка в 1 секунду
        signupButton.click();

        WebElement passwordField = driver.findElement(By.id("password"));
        Thread.sleep(1000);
        passwordField.sendKeys("TestPassword123");

        WebElement dayDropdown = driver.findElement(By.id("days"));
        Thread.sleep(1000);
        dayDropdown.sendKeys("1");

        WebElement monthDropdown = driver.findElement(By.id("months"));
        Thread.sleep(1000);
        monthDropdown.sendKeys("January");

        WebElement yearDropdown = driver.findElement(By.id("years"));
        Thread.sleep(1000);
        yearDropdown.sendKeys("1990");

        WebElement genderRadio = driver.findElement(By.id("id_gender1")); // Для примера выбираем мужской пол
        Thread.sleep(1000);
        genderRadio.click();

        WebElement firstNameField = driver.findElement(By.name("first_name"));
        Thread.sleep(1000);
        firstNameField.sendKeys("John");

        WebElement lastNameField = driver.findElement(By.name("last_name"));
        Thread.sleep(1000);
            lastNameField.sendKeys("Doe");

        WebElement addressField = driver.findElement(By.id("address1"));
        Thread.sleep(1000);
        addressField.sendKeys("123 Test St");
```

```
            WebElement cityField = driver.findElement(By.id("city"));
            Thread.sleep(1000); // Задержка перед вводом города
            cityField.sendKeys("Test City");

            WebElement stateDropdown = driver.findElement(By.id("state"));
            Thread.sleep(1000); // Задержка перед выбором штата
            stateDropdown.sendKeys("California");

            WebElement postalCodeField = driver.findElement(By.id("zipcode"));
            Thread.sleep(1000);
            postalCodeField.sendKeys("12345");

            WebElement countryDropdown = driver.findElement(By.id("country"));
            Thread.sleep(1000);
            countryDropdown.sendKeys("United States");

            WebElement phoneField = driver.findElement(By.id("mobile_number"));
            Thread.sleep(1000);
            phoneField.sendKeys("1234567890");

            WebElement submitButton = driver.findElement(By.cssSelector("[data-qa='create-
account']"));
            Thread.sleep(1000);
            submitButton.click();



            WebElement successMessage = driver.findElement(By.xpath("//b[text()='Account
Created!']"));
            if (successMessage.isDisplayed()) {
                System.out.println("Test Passed: Account created successfully.");
            } else {
                System.out.println("Test Failed: Account creation failed.");
            }

        } catch (InterruptedException e) {
```

```
            e.printStackTrace();
        } finally {
            driver.quit();
        }
    }
}
```

ДОДАТОК Б

Реалізація тесту на перевірку функції додавання товару

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

public class AddToCartTest {
    public static void main(String[] args) {
        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
        WebDriver driver = new ChromeDriver();

        try {
            // Відкриття сторінки каталогу
            driver.get("https://www.automationexercise.com/products");

            // Додавання першого товару до кошика
            WebElement firstAddToCartButton = driver.findElement(By.xpath("(//a[text()='Add to cart'])[1]"));
            firstAddToCartButton.click();

            // Очікування та перевірка повідомлення про успіх
            Thread.sleep(2000); // Затримка для завантаження елементу
            WebElement firstSuccessMessage = driver.findElement(By.xpath("//div[@class='alert-success']"));
            if (firstSuccessMessage.isDisplayed()) {
                System.out.println("Test Passed: First item successfully added to cart.");
            } else {
                System.out.println("Test Failed: First item was not added to cart.");
            }

            // Повернення до каталогу і додавання другого товару
            driver.navigate().back();
            WebElement secondAddToCartButton = driver.findElement(By.xpath("(//a[text()='Add to
```

```
cart'])[2]"));
        secondAddToCartButton.click();

        // Очікування та перевірка повідомлення про успіх
        Thread.sleep(2000); // Затримка для завантаження елементу
        WebElement secondSuccessMessage = driver.findElement(By.xpath("//div[@class='alert-
success']"));
        if (secondSuccessMessage.isDisplayed()) {
            System.out.println("Test Passed: Second item successfully added to cart.");
        } else {
            System.out.println("Test Failed: Second item was not added to cart.");
        }

        // Перехід до кошика
        WebElement viewCartButton = driver.findElement(By.xpath("//a[text()=' Cart']"));
        viewCartButton.click();

        // Перевірка, чи товари відображаються в кошику
        Thread.sleep(2000); // Затримка для завантаження сторінки кошика
        WebElement                                cartItem1                                =
driver.findElement(By.xpath("//tr[1]//td[@class='cart_description']"));
        WebElement                                cartItem2                                =
driver.findElement(By.xpath("//tr[2]//td[@class='cart_description']"));

        if (cartItem1.isDisplayed() && cartItem2.isDisplayed()) {
            System.out.println("Test Passed: Both items are displayed in the cart.");
        } else {
            System.out.println("Test Failed: Items are not displayed correctly in the cart.");
        }
    } catch (InterruptedException e) {
        e.printStackTrace();
    } finally {
        driver.quit();
    }
  }
}
```

ДОДАТОК В

Реалізація тесту на перевірку функції оформлення замовлення

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

public class CheckoutTest {
    public static void main(String[] args) {
        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
        WebDriver driver = new ChromeDriver();

        try {
            // Відкриття сторінки каталогу та додавання товару
            driver.get("https://www.automationexercise.com/products");
            WebElement addToCartButton = driver.findElement(By.xpath("//a[text()='Add to cart'][1]"));
            addToCartButton.click();

            // Перехід до кошика
            WebElement viewCartButton = driver.findElement(By.xpath("//a[text()='View Cart']"));
            viewCartButton.click();

            // Перехід до оформлення замовлення
            WebElement checkoutButton = driver.findElement(By.xpath("//a[text()='Proceed To Checkout']"));
            checkoutButton.click();

            // Введення даних доставки
            WebElement addressField = driver.findElement(By.name("address"));
            WebElement cityField = driver.findElement(By.name("city"));
            WebElement confirmOrderButton = driver.findElement(By.xpath("//button[text()='Place
```

```
Order']"));

            addressField.sendKeys("123 Test Street");
            cityField.sendKeys("Test City");
            confirmOrderButton.click();

            // Перевірка успішного оформлення
            WebElement successMessage = driver.findElement(By.xpath("//h2[text()='Order Placed!']"));
            if (successMessage.isDisplayed()) {
                System.out.println("Test Passed: Order placed successfully.");
            } else {
                System.out.println("Test Failed: Order not placed.");
            }
        } finally {
            driver.quit();
        }
    }
}
```