Chapter 1

Introduction

1.1 what is software testing?

Software testing is an indispensable process in the software development lifecycle (sdlc) that involves the systematic evaluation of a software product or service to ascertain its adherence to specified requirements, functionalities, and user expectations. This critical practice encompasses a range of techniques and methodologies aimed at identifying defects, errors, or inconsistencies within the software, thereby ensuring its reliability, usability, and overall quality. Testing acts as a safeguard against potential malfunctions, security vulnerabilities, and performance bottlenecks that could adversely impact the user experience. By subjecting the software to rigorous examination through various types of tests, such as functional, non-functional, and regression testing, organizations can significantly reduce the risk of releasing flawed software, enhance customer satisfaction, and build a strong reputation for delivering reliable products.

1.2 types of testing

Software testing is a multi-faceted domain, encompassing a wide array of techniques tailored to address specific quality attributes and ensure the overall robustness of software products. Let's delve into some of the most common types of testing employed in the industry:

- **manual testing:** this traditional approach relies on human testers to manually execute test cases by interacting with the software, emulating real-world user scenarios. While flexible and adept at uncovering usability issues, it can be time-consuming and prone to human error.
- **automated testing:** leveraging specialized tools and scripts, automated testing enables the execution of predefined test cases without human intervention. This methodology boasts increased efficiency, repeatability, and the ability to cover a wider range of test scenarios.
- **unit testing:** a fundamental building block of software testing, unit testing focuses on verifying the correctness of individual code units (functions, methods, or classes) in isolation. This helps pinpoint defects early in the development cycle.
- **integration testing:** as the name implies, integration testing checks the interaction between different software modules or components, ensuring seamless collaboration and data exchange.
- **system testing:** taking a holistic approach, system testing evaluates the entire integrated system against its requirements, verifying that it behaves as expected as a whole.
- **acceptance testing:** often performed by end-users or stakeholders, acceptance testing determines whether the developed system satisfies the criteria for acceptance and meets the needs and expectations of the intended audience.
- **performance testing:** this type of testing assesses the responsiveness, stability, and scalability of the software under different workloads and usage patterns, ensuring optimal performance under real-world conditions.
- **security testing:** a critical aspect of software testing, security testing involves identifying vulnerabilities, weaknesses, or potential entry points for unauthorized access, data breaches, or other security threats.
- **Usability Testing:** Focused on the user's perspective, usability testing evaluates how intuitive, user-friendly, and efficient the software is in achieving user goals.

- **Regression Testing:** After modifications or updates to the software, regression testing is conducted to ensure that existing functionalities have not been adversely affected and that new issues have not been introduced.
- **Exploratory Testing:** A less structured approach, exploratory testing involves testers learning about the software through experimentation and discovery, uncovering defects that might not be caught by scripted tests.

Each of these testing types plays a crucial role in the comprehensive evaluation of software, ensuring that it meets stringent quality standards and delivers a seamless user experience. By adopting a combination of manual and automated testing techniques, organizations can optimize their testing efforts, detect defects early, and ultimately release software that is both reliable and user-centric.

1.3 About Automation Tool Used – Selenium WebDriver

Selenium WebDriver is a powerful and versatile open-source framework widely adopted for automating web browser interactions. It empowers developers and testers to create scripts in various programming languages (such as Java, Python, C#, Ruby, etc.) that can simulate user actions on web pages, including clicking buttons, filling forms, navigating through links, and validating expected outcomes. With its cross-browser compatibility, Selenium WebDriver can interact with major browsers like Chrome, Firefox, Safari, Edge, and Internet Explorer, enabling comprehensive testing across different platforms and environments.

One of Selenium WebDriver's core strengths lies in its ability to locate and manipulate web elements through a variety of strategies, including IDs, class names, XPath expressions, and CSS selectors. This flexibility allows testers to interact with even the most complex web page structures. Furthermore, Selenium WebDriver offers implicit and explicit waits, allowing scripts to pause and synchronize with the application under test, ensuring that elements are loaded and ready for interaction before proceeding.

Beyond its core functionality, Selenium WebDriver integrates seamlessly with popular testing frameworks like TestNG and JUnit, providing features like test organization, reporting, and parallelization. This integration facilitates structured test development, enhances maintainability, and optimizes test execution time. Additionally, Selenium WebDriver's extensible architecture allows for the integration of third-party libraries and tools, expanding its capabilities and addressing specific testing needs.

1.4 Problem Statement

In the current digital era, e-commerce platforms are integral to the shopping experience, allowing users to purchase products conveniently from the comfort of their homes. However, the efficiency and reliability of these platforms are paramount to ensure a seamless user experience. Automated testing plays a crucial role in validating the functionality, performance, and security of these systems.

1.5 Objective of the Project

The primary objective of this project is to automate the testing of an e-commerce purchase flow to ensure its functionality, reliability, and performance. Specifically, the project aims to:

Automate E-Commerce Purchase Flow: Implement an automated test that simulates the complete purchase process on the "Tutorials Ninja"demo site, including product search, image viewing, cart management, and checkout.

Validate Product Search and Selection: Ensure that the product search functionality works correctly by verifying that the desired product ("HP LP3065") can be searched, selected, and viewed without issues.

Test Image Gallery Interaction: Validate the functionality of the product image gallery by automating interactions such as clicking through images and capturing screenshots.

Automate Cart and Checkout Process: Verify that the product can be added to the cart, and the checkout process can be completed smoothly, including date selection and form filling.

Ensure Accurate Billing and Shipping Information: Test the billing details entry for guest checkout, including filling out personal information, selecting country and region, and agreeing to terms and conditions.

Confirm Final Price and Order Success: Validate that the final price displayed is correct and that the order confirmation message accurately reflects a successful purchase.

Generate HTML Test Reports: Create detailed HTML reports of the test execution, including screenshots and results, to facilitate easy review and debugging of test outcomes.

Improve Test Coverage and Efficiency: Develop a maintainable and scalable test script that can be easily extended for additional test cases and scenarios, improving the overall efficiency of the testing process.

Chapter 2

Literature Survey

2.1. About Manual Testing

Manual testing is a time-tested approach in software quality assurance where human testers meticulously execute test cases, interact with the software, and assess its functionality based on predefined expectations. This hands-on approach involves meticulously following test scripts, simulating user scenarios, and carefully observing the software's behavior.

Advantages of Manual Testing:

1. Exploratory Testing: Manual testing allows testers to explore the software intuitively, uncovering unexpected issues or edge cases that might not be covered in scripted test cases. This is particularly valuable for uncovering usability problems or design flaws.

2. Flexibility and Adaptability: Manual testers can quickly adapt to changes in requirements, user scenarios, or software updates. They can readily modify their testing approach based on real-time observations.

3. Cost-Effective for Small Projects: For smaller projects with limited scope and budget, manual testing can be a more cost-effective option, as it doesn't require the initial investment in automation tools and infrastructure.

4. User Experience Focus: Manual testers can assess the software from a user's perspective, providing valuable insights into usability, intuitiveness, and overall user experience.

Disadvantages of Manual Testing:

1. Time-Consuming: Manual execution of test cases, especially for large and complex applications, can be extremely time-consuming and resource-intensive.

2. Prone to Human Error: Repetitive tasks can lead to fatigue and mistakes, potentially missing critical defects.

3. Limited Test Coverage: Due to time and resource constraints, manual testing may not be able to cover all possible combinations of inputs and scenarios, leading to potential gaps in test coverage.

4. Non-Repeatable: Manual tests are not easily repeatable, making it challenging to consistently reproduce and diagnose defects.

2.2. About Automation Testing

Automation testing represents a paradigm shift in software testing, where software tools and scripts take center stage in executing test cases, validating results, and generating comprehensive reports. This methodology has gained significant traction due to its ability to increase efficiency, accuracy, and test coverage.

Advantages of Automation Testing:

1. Efficiency and Speed: Automated tests can be executed rapidly, significantly reducing the time required for testing cycles. This accelerated feedback loop enables faster identification and resolution of defects.

2. Increased Accuracy: By eliminating human error, automated tests provide more consistent and reliable results, ensuring that the software behaves as expected under various conditions.

3. Expanded Test Coverage: Automation enables the execution of a vast number of test cases across different environments, platforms, and configurations, leading to broader test coverage and identifying issues that might be missed in manual testing.

4. Cost-Effective for Large Projects: While the initial setup of automated tests requires investment, it can be highly cost-effective in the long run, especially for large-scale projects with extensive test suites.

5. Reusability and Scalability: Automated test scripts can be reused for different versions of the software, and the test suite can be easily scaled to accommodate new functionalities.

6. 24/7 Availability: Automated tests can be scheduled to run at any time, even outside of business hours, enabling continuous testing and faster feedback.

7. Integration with CI/CD: Automated tests seamlessly integrate into CI/CD pipelines, providing immediate feedback on code changes and ensuring that each build meets quality standards.

Disadvantages of Automation Testing:

1. Initial Investment: Setting up and maintaining automated test scripts requires upfront effort, skilled resources, and investment in automation tools.

2. Limited Flexibility: Automated tests are less flexible than manual tests and may not be able to adapt to unexpected scenarios or changes in requirements as easily.

3. Maintenance Overhead: Automated test scripts need to be updated and maintained as the software evolves, adding to the overall maintenance effort.

4. False Positives/Negatives: Automated tests can sometimes produce inaccurate results due to scripting errors, environment issues, or unexpected application behavior.

5. Not a Replacement for Manual Testing: Automation testing cannot entirely replace manual testing. Manual testing is still essential for exploratory testing, usability testing, and verifying aspects that are difficult to automate.

Chapter 3

System Architecture and Design

3.1 Architecture/Methodology Used

1. Test Automation Framework: The project utilizes a test automation framework based on Selenium WebDriver and Pytest, which includes the following components:

- **Selenium WebDriver:** Selenium WebDriver is used to automate interactions with the web application. It provides the capability to simulate user actions such as clicking buttons, entering text, and navigating through pages. In this project, the Edge WebDriver is used to interact with the "Tutorials Ninja" demo site.
- **Pytest:** Pytest is employed as the testing framework to manage and execute the test cases. It provides powerful features such as fixtures, assertions, and reporting. Pytest is used to organize test cases, handle setup and teardown operations, and generate comprehensive HTML reports.

2. Architecture Overview:

The architecture of the automation framework is designed to ensure modularity, maintainability, and scalability. The key components include:

- **Test Scripts:** The test scripts contain the logic for automating the e-commerce purchase flow. They interact with the web elements using Selenium WebDriver and validate the functionalities of the application.
- **Fixtures:** Pytest fixtures are used to set up and tear down the WebDriver instance. Fixtures manage the lifecycle of the WebDriver, including initialization and cleanup, ensuring that each test runs in a clean environment.
- **Page Object Model (POM) (Optional):** Although not explicitly mentioned in the provided code, the Page Object Model is a recommended design pattern in test automation. It involves creating separate classes for different web pages, encapsulating the page-specific actions and elements. This pattern improves code readability and maintainability.

3. Methodology Used:

The methodology employed in this project includes:

- **Test Case Design:** Test cases are designed to cover the end-to-end purchase flow of the e-commerce platform. They include actions such as product search, image gallery interaction, cart management, checkout process, and order confirmation.

- **Test Execution:** The test scripts are executed using Pytest, which handles the execution of test cases, manages fixtures, and generates test reports. The tests simulate real user interactions with the application to validate its functionality.
- **Error Handling:** The scripts include basic error handling through assertions and logging. Assertions are used to verify expected outcomes, while screenshots and error messages are captured to aid in debugging.
- **Reporting:** Pytest generates HTML reports that include details of test execution, including passed/failed status, error messages, and screenshots. These reports provide valuable insights into test results and facilitate easy review.

4. Tools and Technologies:

- **Selenium WebDriver:** For browser automation and interaction with the web application.
- **Pytest:** For managing test execution and generating reports.
- **Edge WebDriver:** Specific WebDriver used for Microsoft Edge browser automation.
- **HTML Reporting:** Pytest's HTML report feature is used to generate comprehensive test reports.

5. Implementation Flow:

Setup:

- Initialize WebDriver and configure browser options.
- Open the target URL and maximize the browser window.

Test Execution:

- Perform actions according to the test case, such as searching for products, interacting with UI elements, and validating results.
- Capture screenshots and log relevant information.

Teardown:

- Close the browser and clean up resources.
- Generate and review test reports.

3.2 Flowchart:

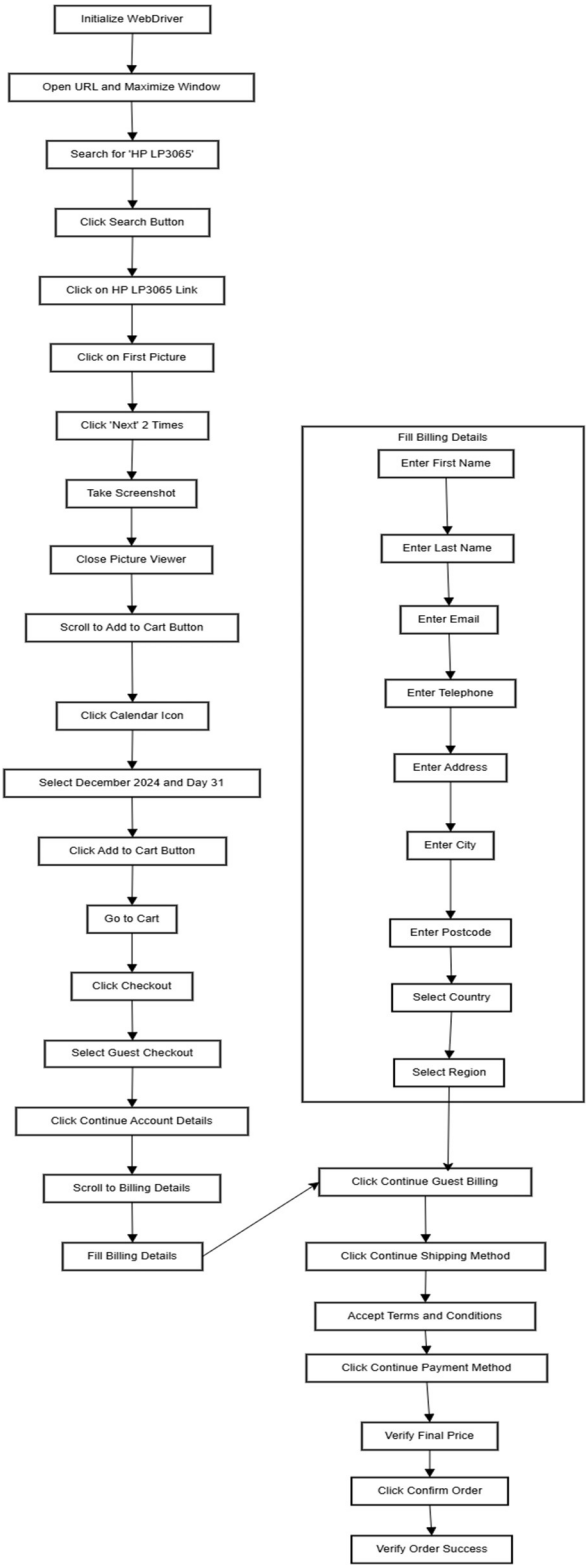

Fig 1. Flow Chart

Chapter 4

System Architecture and Design

4.1. Test Cases Table

Test Case ID	Description	Expected Result	Actual Result
TC001	Verify Product Search Functionality	Product search results display "HP LP3065".	PASS
TC002	Verify Product Selection and Image Gallery Interaction	Product details and images are displayed. Images can be navigated.	PASS
TC003	Verify Screenshot Capture	Screenshot is saved with a unique filename.	PASS
TC004	Verify Calendar Date Selection	Date 'December 31, 2024' is selected. Product is added to the cart.	PASS
TC005	Verify Cart and Checkout Process	Checkout page is displayed. Guest checkout option is selected.	PASS
TC006	Verify Billing Details Entry	Billing details are entered. Proceed to the next page	PASS
TC007	Verify Shipping Method and Payment Confirmation	Shipping method and payment details are accepted. Order confirmation is displayed.	PASS
TC008	Verify Order Confirmation	Confirmation message "Your order has been placed!" is displayed.	PASS

4.2. Black Box / White Box Testing

The implemented test is a black-box test, as it focuses on verifying the functionality of the e-commerce purchase flow automation without considering its internal code or implementation details.

4.3 Code:

mm.py

mm.py > test_tutorialsninja_purchase

```python
import random
import time
import pytest
from selenium.webdriver import ActionChains
from selenium import webdriver
from selenium.webdriver.support.select import Select
from selenium.webdriver.edge.options import Options
from selenium.webdriver.common.by import By

@pytest.fixture
def driver():
    edge_options = Options()
    driver = webdriver.Edge(options=edge_options)
    yield driver
    driver.quit()

def test_tutorialsninja_purchase(driver):
    driver.get('http://tutorialsninja.com/demo/')
    driver.maximize_window()

    search_box = driver.find_element(By.NAME, 'search')
    search_box.click()
    search_box.send_keys('HP LP3065')
    search_button = driver.find_element(By.XPATH, '//button[@class="btn btn-default btn-lg"]')
    search_button.click()
    time.sleep(2)
```

Fig 2 Code Snippet

mm.py

mm.py > test_tutorialsninja_purchase

```python

    screenshot_path = 'screenshot#' + str(random.randint(0, 101)) + '.png'
    driver.save_screenshot(screenshot_path)

    x_button = driver.find_element(By.XPATH, '//button[@title="Close (Esc)"]')
    x_button.click()
    time.sleep(1)

    add_to_button_2 = driver.find_element(By.XPATH, '//button[@id="button-cart"]')
    driver.execute_script("arguments[0].scrollIntoView(true);", add_to_button_2)
    time.sleep(1)

    calendar = driver.find_element(By.XPATH, '//i[@class="fa fa-calendar"]')
    calendar.click()
    time.sleep(1)

    next_click_calendar = driver.find_element(By.XPATH, '//th[@class="next"]')
    month_year = driver.find_element(By.XPATH, '//th[@class="picker-switch"]')

    while month_year.text != 'December 2024':
        next_click_calendar.click()
        time.sleep(0.1)

    calendar_date = driver.find_element(By.XPATH, '//td[text()="31"]')
    calendar_date.click()
    time.sleep(2)

```

Fig 3 Code Snippet

CHAPTER 5

Results/Output

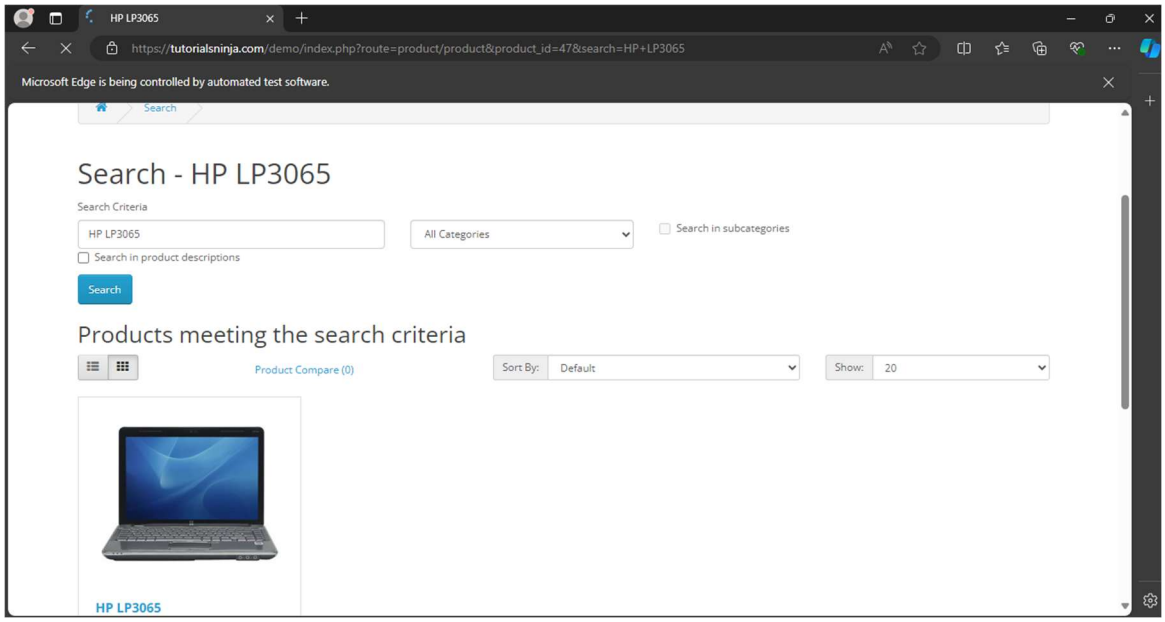

Fig 4 Search product is displayed on the screen

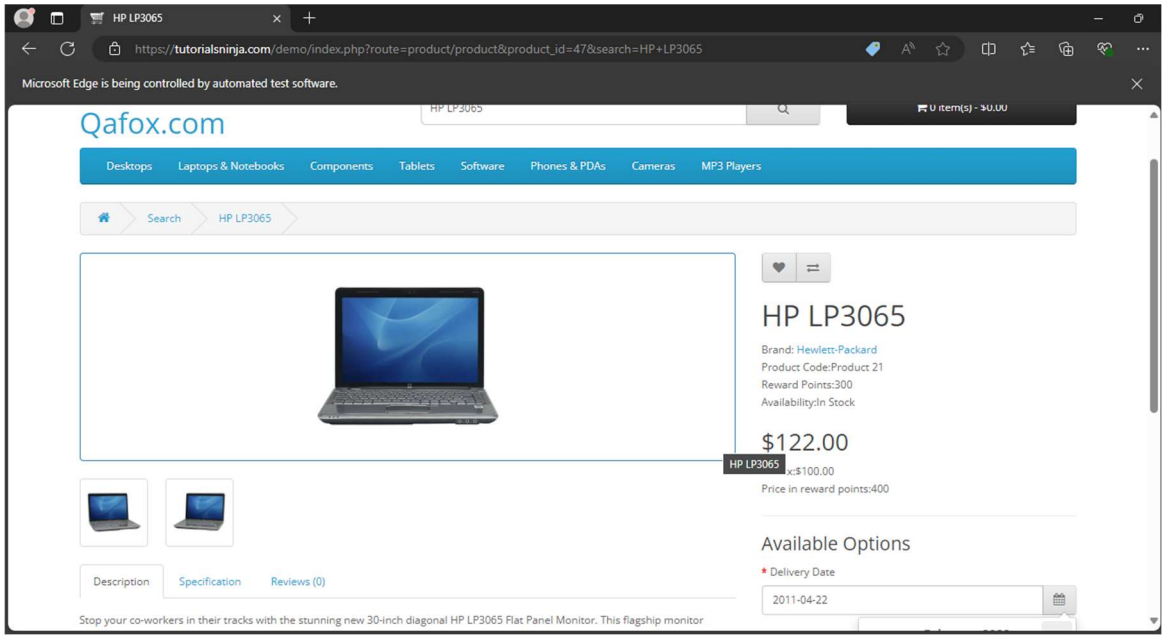

Fig 5 Products Description is visible

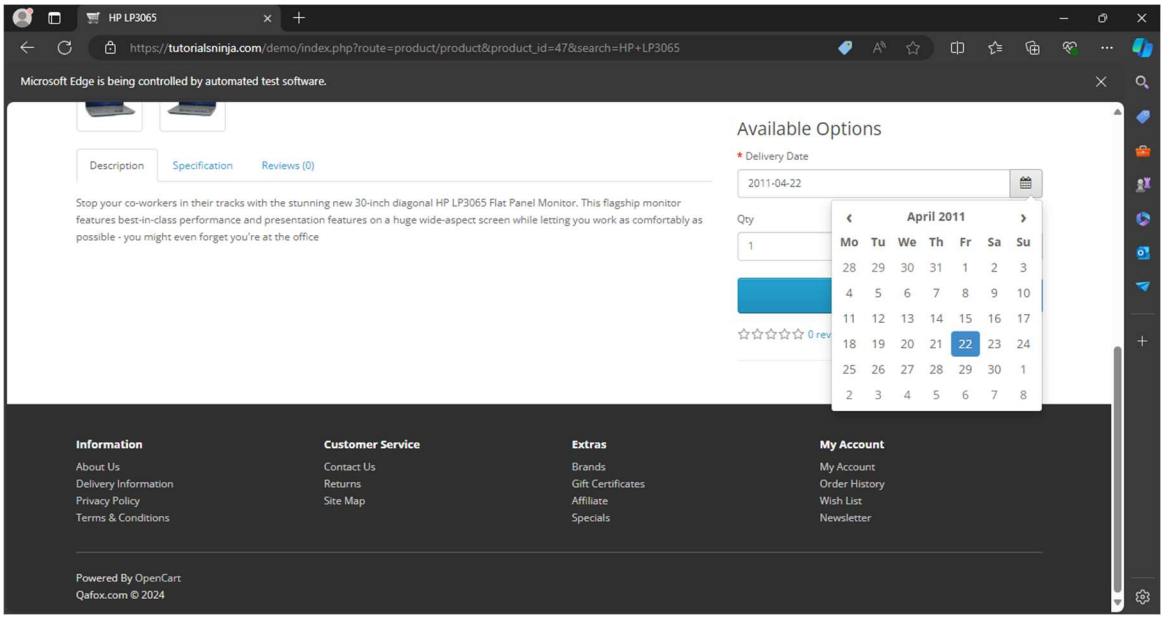

Fig 6 Setting the Products Delivery date

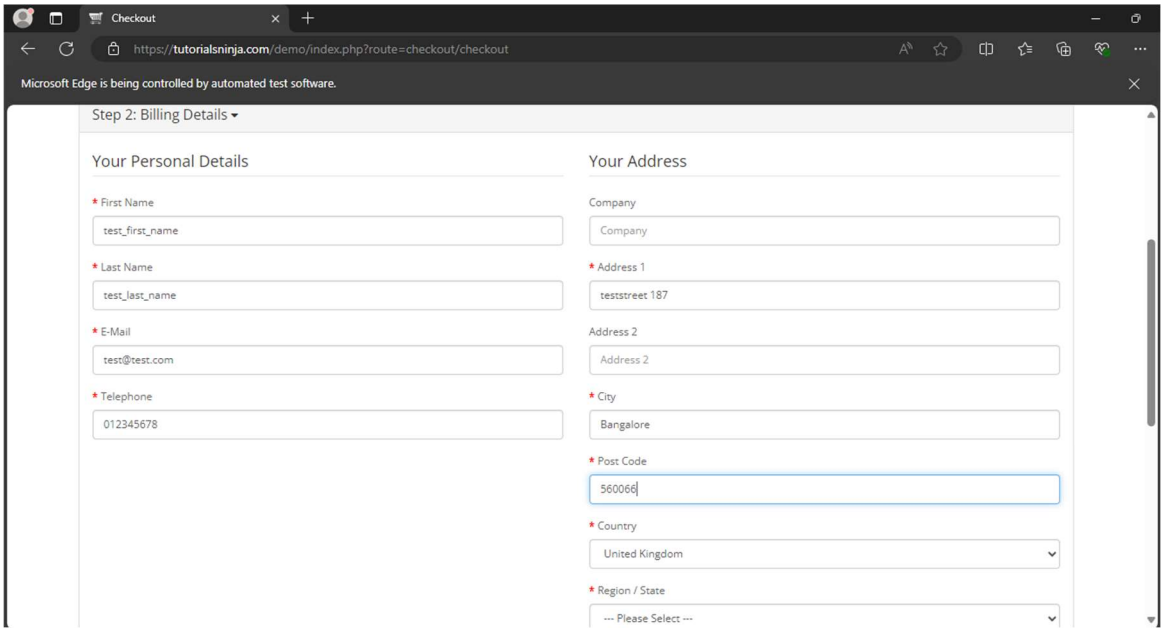

Fig 7 Filling the Billing details

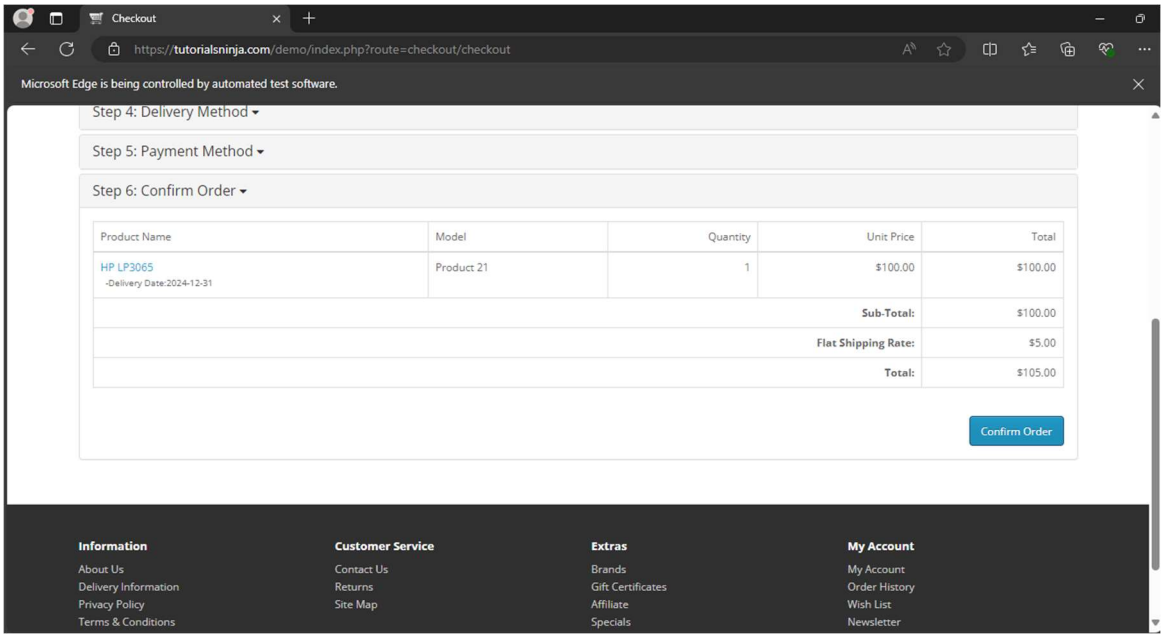

Fig 8 Confirming the Order

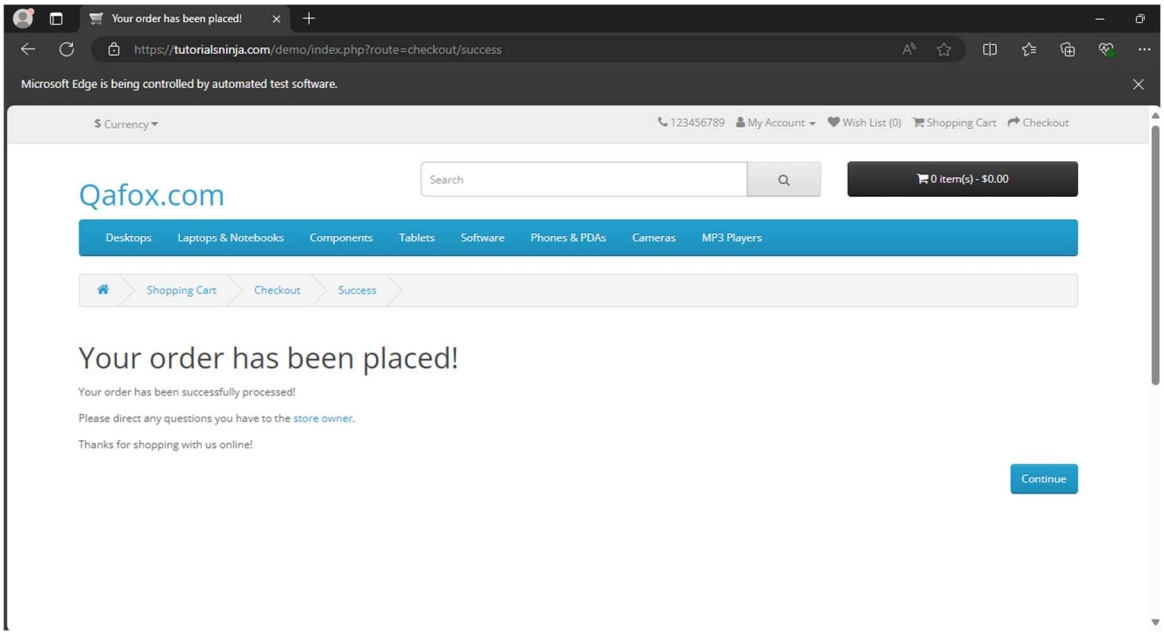

Fig 9 Order Placed

Chapter 6

Conclusion/Future Scope

6.1. Conclusion

The automated testing project for the e-commerce purchase flow using Selenium WebDriver and Pytest successfully demonstrates how automated tests can validate the critical functionalities of an online shopping application. Through a series of meticulously designed test cases, the project verifies key aspects such as product search, image gallery interaction, cart management, checkout process, and order confirmation.

6.2. Future Scope

The current project provides a solid foundation for expanding the scope of automated testing on the Tutorialsninja demo platform. Future enhancements could include:

Advanced Search Scenarios: Test various search filters, sorting options, and error handling for invalid search terms.
Content Interaction: Automate interactions with product reviews, such as liking and disliking reviews. Test product page navigation, including product description and specifications.
Profile Management: Test features related to profile creation, editing, and deletion.
Subscription Management: Automate tests for subscription plans, payment methods, and cancellation processes.
Recommendation Engine: Evaluate the accuracy and relevance of the platform's recommendation system by testing various user profiles and viewing histories.
Device Compatibility: Extend the test suite to cover different devices and screen sizes, ensuring a consistent user experience across platforms.
Localization Testing: Verify the correctness of content translations and UI elements for different regions.
Performance Testing: Assess the performance and responsiveness of the application under different load conditions.
Security Testing: Identify and mitigate potential security vulnerabilities in the login and profile management features.
Accessibility Testing: Ensure that the platform is accessible to users with disabilities, adhering to accessibility guidelines.

By focusing on these areas, the test suite can be significantly expanded to cover more functionalities and ensure a higher level of quality and reliability for the Tutorialsninja demo platform.

References

[1] Selenium Documentation: https://www.selenium.dev/documentation/
[2] XPath Tutorial: https://www.w3schools.com/xml/xpath_intro.asp
[3] JUnit Documentation: https://junit.org/junit5/
[4] Agile Testing by Lisa Crispin and Janet Gregory
[5] Selenium Python tutorial - Selenium Python Tutorial - GeeksforGeeks
[6] Test Automation in Practice by Bas Dijkstra and Dorothy Graham
[7] Complete Guide to Test Automation by Arnon Axelrod
[8] Software Testing: Principles and Practices by Srinivasan Desikan and Gopalaswamy Ramesh
[9] tutorialsninja : The demo website for accessing and using testing its services: http://tutorialsninja.com/demo/
[10] Automation Step by Step (YouTube Channel): Offers step-by-step guides and tutorials for beginners in automation testing.