VISVESVARAYA TECHNOLOGICAL UNIVERSITY

GNANASANGAMA, BELAGAVI-590018

TECHNICAL SEMINAR REPORT

"AUTOMATION TESTING FOR NYKAA"

Submitted in Partial fulfillment of the Requirements for the VI Semester of the Degree of

BACHELOR OF ENGINEERING

in

Information Science and Engineering

Submitted by

ADITI GARG, (1CR21IS007)
AKSHANSH RAO, (1CR21IS013)
AMISHA THAKUR, (1CR21IS017)

Under the Guidance of

Internal Guide
KANIKA AGRAWAL
Associate Professor
Dept. of ISE, CMRIT

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

CMR INSTITUTE OF TECHNOLOGY

#132, AECS LAYOUT, IT PARK ROAD, KUNDALAHALLI, BANGALORE-560037

2023-2024

DEPT. OF INFORMATION SCIENCE & ENGINEERING

Certificate

This is to certify that **ADITI GARG(1CR21IS007), AKSHANSH RAO(1CR21IS013), AMISHA THAKUR(1CR21IS017)** student of CMR Institute of Technology have undergone software testing project in partial fulfillment for the award of **Bachelor of Engineering** in **Information Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year **2022- 2023**. It is certified that all corrections/suggestions indicated for Initial Reviews have been incorporated in the Report. This **SOFTWARE TESTING PROJECT REPORT** has been approved as it satisfies the academic requirements in respect of project work prescribed for the said degree.

Sign. of Internal Guide
XXXXXXX
Assistant Professor
Department of ISE
CMRIT, Bengaluru

Sign. of HOD
Dr. Jagadishwari V
Associate Professor & HOD
Department of ISE
CMRIT, Bengaluru

Viva

Name of the examiners **Signature with date**

1.

2.

ACKNOWLEDGEMENT

Any work of significance requires a great deal of effort and time put into it. But a factor of even greater importance is efficient guidance and encouragement. In spite of all my dedicated work, this internship would not have been possible without continuous help and guidance provided by people who gave their unending support right from when this idea was conceived.

We would like to thank Dr**. Sanjay Jain,** Principal, CMRIT, Bangalore, for his constant co-operation and support throughout this Technical Seminar tenure.

We would like to thank **Dr. Jagadishwari V, Associate Professor & Head ,** Department of Information Science and Engineering, CMRIT for her constant guidance and support during this Technical Seminar period.

We would like to thank my guide, **Prof. ***, Designation,** Department of Information Science and Engineering, CMRIT for her constant guidance that helped me in completing the Technical Seminar work successfully.

Last but definitely not the least I would like to thank **My Family** and **Friends** who have always supported me in every path of the technical seminar work.

ADITI GARG (1CR21IS007)
AKSHANSH RAO (1CR21IS013)
AMISHA THAKUR(1CR21IS017)

DECLARATION

We, **ADITI GARG(1CR21IS007), AKSHANSH RAO(1CR21IS013), AMISHA THAKUR(1CR21IS017)** student of eight semesters B.E in Information Science and Engineering from CMR Institute of Technology, Bangalore, hereby declare that this ST project titled "**Automation testing of nykaa"** was carried out by us.

We have done the work assigned to me during the period and all the contents about work assigned are prepared and presented by me. The eight semester **SOFTWARE TESTING PROJECT REPORT** has been done by me under the supervision of **Prof. XXX (Guide Name)**, Department of ISE, Internal Guide, CMR Institute of Technology, Bangalore.

This work is submitted to Visvesvaraya Technological University in partial fulfillment of the requirement for the award of degree of Bachelor of Engineering of Technology in Information Science and Engineering during the academic year 2021-2022.

Place: Bangalore

ADITI GARG (1CR21IS007)
AKSHANSH RAO (1CR21IS013)
AMISHA THAKUR(1CR21IS01

Table of Contents

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 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.

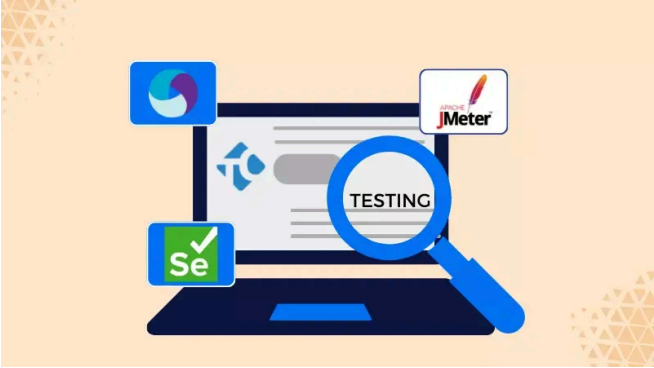

Fig 1 : Software testing

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.

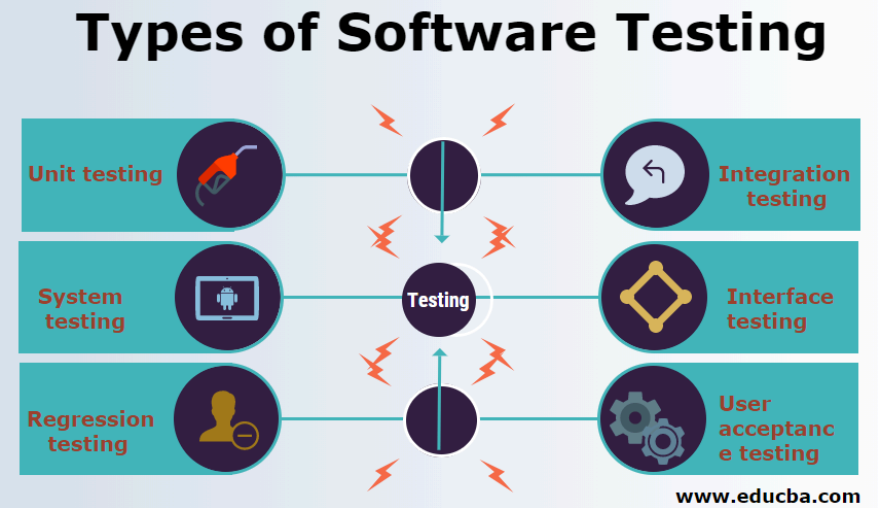

Fig 2 : Types of testing

- **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.

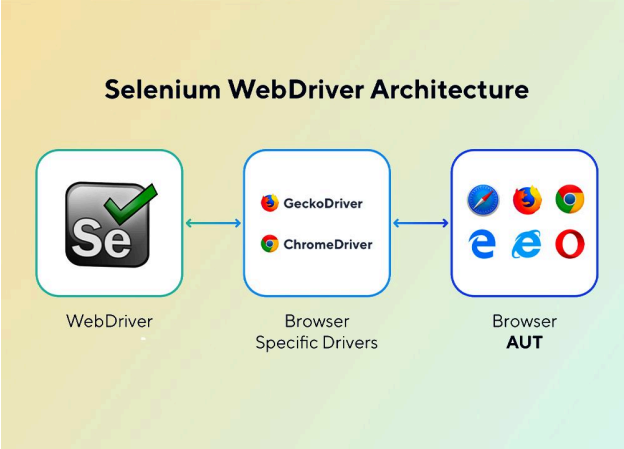

Fig 3 : Selenium web driver

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

Manual testing of complex web applications like Nykaa, characterized by intricate user flows, dynamic content, and frequent updates, poses significant challenges. The process can be time-consuming, prone to human error, and may not comprehensively cover all potential scenarios. Moreover, the repetitive nature of manual testing can lead to tester fatigue, further compromising the accuracy and thoroughness of the testing process. These limitations necessitate a more efficient and reliable approach to ensure the quality and functionality of the Nykaa platform.

Fig 4 : Nykaa website testing

1.5 Objective of the Project

This project addresses the challenges of manual testing by harnessing the power of Selenium WebDriver to automate a critical user flow on the Netflix website. The primary objectives of this project are as follows:

1. **Enhance Efficiency:** By automating the login, profile selection, search, content preview, and logout steps, we aim to significantly reduce the time and effort required for testing compared to manual execution. This will streamline the testing process and free up valuable resources for other testing activities.
2. **Improve Reliability:** Through automated test scripts, we seek to eliminate human errors that can occur during manual testing, ensuring consistent and reproducible test results. This will enhance the reliability and accuracy of the testing process, leading to greater confidence in the software's quality.
3. **Increase Test Coverage:** The automated test suite will be designed to cover a wide range of user scenarios, including valid and invalid logins, different profile selections, various search terms, and error handling for unexpected events. This increased coverage will help identify potential issues that might be missed in manual testing.

4. **Early Bug Detection:** By integrating the automated tests into the development workflow, we aim to detect and report defects early in the development cycle. This early feedback loop will allow for faster bug fixes, reducing the risk of critical issues reaching production and impacting the user experience.
5. **Continuous Integration:** The project aims to establish a foundation for continuous integration, where the automated test suite can be seamlessly integrated into the development pipeline. This will enable automated testing with every code commit, ensuring that new changes do not introduce regressions or break existing functionalities.

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.

Fig 5 : Manual testing

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.

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

FLOWCHART

3.1 Flowchart/Path Flow Diagram

1. **Initialize WebDriver**: Start a ChromeDriver instance.
2. **Navigate to Nykaa**: Open the Nykaa homepage (`https://www.nykaa.com/`).
3. **Click "Sign In"**: Locate and click the "Sign In" button.
4. **Enter Credentials**: Fill in the username and password fields.
5. **Click Login**: Submit the login form.
6. **Select Profile**: Choose the desired profile if prompted (e.g., "Ravi").
7. **Pause 10 seconds**: Allows manual inspection of the page.
8. **Search for "Lipstick"**: Click the search button, enter "Lipstick" in the search bar.
9. **Pause 5 seconds**: Allows manual inspection of the page.
10. **Click "Lipstick" Result**: Click on the first "Lipstick" search result.
11. **Pause 10 seconds**: Allows manual inspection of the page.
12. **Close Preview Popup**: If a preview appears, close it.
13. **Hover and Logout**: Hover over the profile icon and click "Sign Out."
14. **Pause 1 second**: Allows manual inspection of the page.
15. **Confirm Logout**: Click "Go Now" to confirm logout.
16. **Pause 3 seconds**: Allows manual inspection of the page.
17. **Verify Logout**: Check if the user is successfully logged out.
18. **Close Browser**: Terminate the WebDriver session.

CHAPTER 4

SYSTEM ARCHITECTURE AND DESIGN

The system architecture and design of Selenium are crucial to understanding how Selenium facilitates automated browser testing. Selenium is a popular open-source tool for automating web browsers, and its architecture is designed to provide a flexible and scalable way to control and interact with web applications.

1. Components of Selenium

- Selenium WebDriver: Provides a programming interface to interact with web browsers. It allows you to write tests in various programming languages (like Java, C#, Python, Ruby, JavaScript) to control the browser's behavior.

- Selenium IDE: A browser extension for Firefox and Chrome that provides a simple interface to record and playback tests. It's useful for creating and debugging simple test cases without needing to write code.

- Selenium Grid: Allows you to run tests on different machines and browsers in parallel. It is used for distributed test execution, enabling tests to be run across multiple environments.

- Selenium RC (Remote Control): An older component that has been replaced by WebDriver. It allowed for remote test execution but had limitations that WebDriver has since addressed.

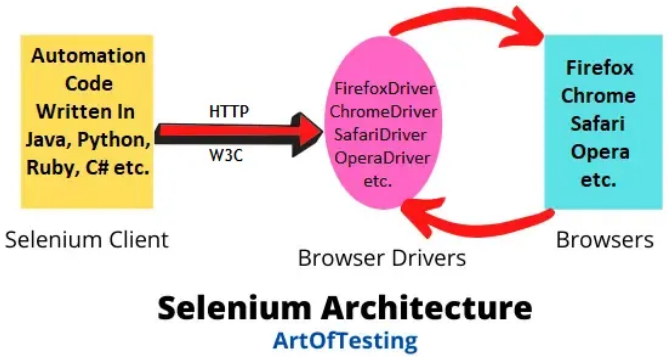

Fig 6 : Selenium architecture

2.Architecture Overview

- Test Scripts
  - Languages Supported: Selenium WebDriver supports various programming languages including Java, C#, Python, Ruby, and JavaScript.
  - Frameworks : Test scripts are often written using test frameworks like JUnit, TestNG (for Java), NUnit (for C#), or py test (for Python).

- WebDriver Interface
  - Browser Drivers: Selenium WebDriver interacts with different browsers through browser-specific drivers (e.g., ChromeDriver for Chrome, GeckoDriver for Firefox, EdgeDriver for Edge).
  - Browser-Specific Communication: WebDriver uses the WebDriver JSON Wire Protocol or the W3C WebDriver standard to communicate with browser drivers.

- Browser Drivers
  - ChromeDriver: Controls the Chrome browser.
  - GeckoDriver : Controls Firefox.
  - EdgeDriver: Controls Microsoft Edge.

3. Communication Protocols

- JSON Wire Protocol: An older protocol that Selenium WebDriver used to communicate with browser drivers. It has been largely replaced by the W3C WebDriver standard but is still supported for backward compatibility.

- W3C WebDriver Standard: A more recent standard that ensures consistency and compatibility across different browsers. It standardizes the WebDriver API to provide a uniform way to interact with web browsers.

4.Selenium Grid Architecture

- Hub: The central server that receives the test requests and routes them to the appropriate node. The hub is responsible for managing the test execution.
- Node: A machine (or server) that registers with the hub and runs the test scripts. Each node can be configured to run tests on different browsers and operating systems.

5. Integration with Testing Frameworks

- Selenium can be integrated with various testing frameworks and tools for better test management and reporting:

- TestNG/JUnit (Java): Provides advanced features for test configuration, grouping, and reporting.
- pytest (Python): Offers a powerful framework for writing and managing tests.

4.1 Test Cases Table

Test Case ID	Description	Expected Result	Actual Result
TC_01	Login with valid credentials	Successful login and redirect to profile selection page	Pass
TC_02	Select a specific profile	Profile is selected	Pass
TC_03	Search for brand lipstick	"lipstick" appears in the search results	Pass
TC_04	Click on the first "lipstick" search result	"lipstick" details page loads	Pass
TC_05	Close the preview popup	Preview popup is closed	Pass
TC_06	Hover over profile icon and click "Sign Out"	Logout confirmation page is displayed	Pass
TC_07	Click the "Go Now" button to confirm logout	User is successfully logged out	Pass

4.2 Code

```
print("Amisha you can do it")
from selenium import webdriver # type: ignore
from selenium.webdriver.common.by import By # type: ignore
from selenium.webdriver.common.keys import Keys # type: ignore
from selenium.webdriver.common.action_chains import ActionChains # type: ignore
from selenium.webdriver.support.ui import WebDriverWait # type: ignore
from selenium.webdriver.support import expected_conditions as EC # type: ignore
import time

# Initialize WebDriver
driver = webdriver.Chrome()
```

Fig 7: Code snippet

```
try:
    # Navigate to Nykaa
    driver.get("https://www.nykaa.com/")

    # Click "Sign In"
    sign_in_button = driver.find_element ((By.XPATH, "//button[normalize-space()='Sign in']"))
    sign_in_button.click()

    # Enter Credentials
    WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.NAME, "emailMobile"))).send_keys
    driver.find_element(By.NAME, "password").send_keys("ilovemymom")

    # Click Login
    login_button = driver.find_element(By.XPATH, "//a[normalize-space()='Log Out']")
    login_button.click()

    # Wait for profile selection (if any) and select profile # (This step is omitted here as Nykaa typi

    # Pause 10 seconds
    time.sleep(10)

    # Search for "Lipstick"
    search_box = driver.find_element((By.XPATH, "//input[@placeholder='Search on Nykaa']"))
```

Fig 8: Code snippet

CHAPTER 5

Results/Output

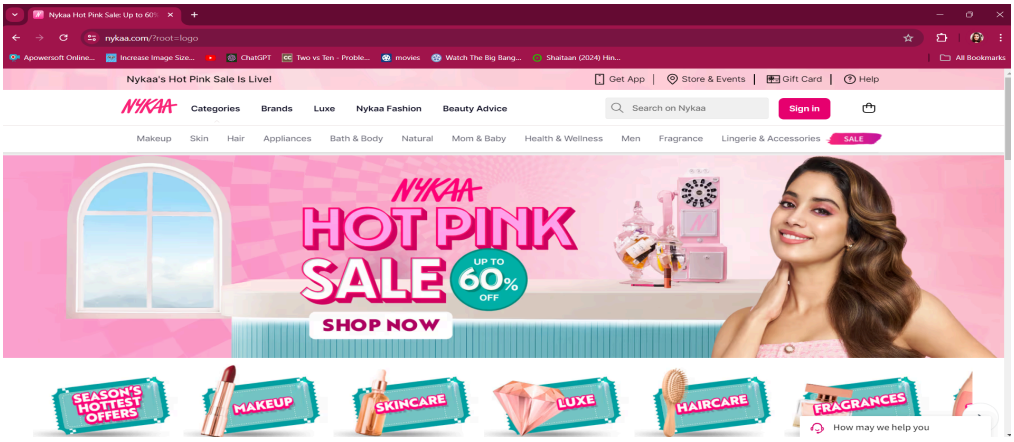

Fig 9 : Nykaa website opening

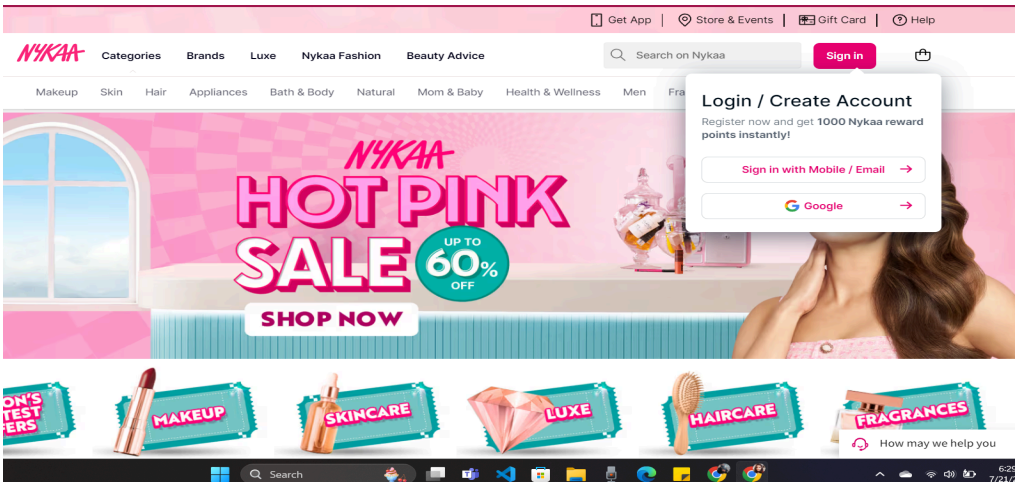

Fig 10 : Nykaa website login

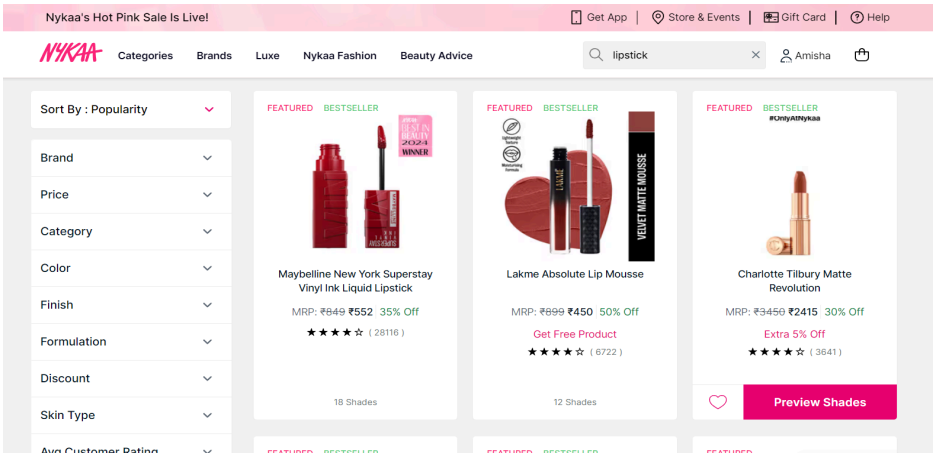

Fig 11 : Nykaa website after lipstick search

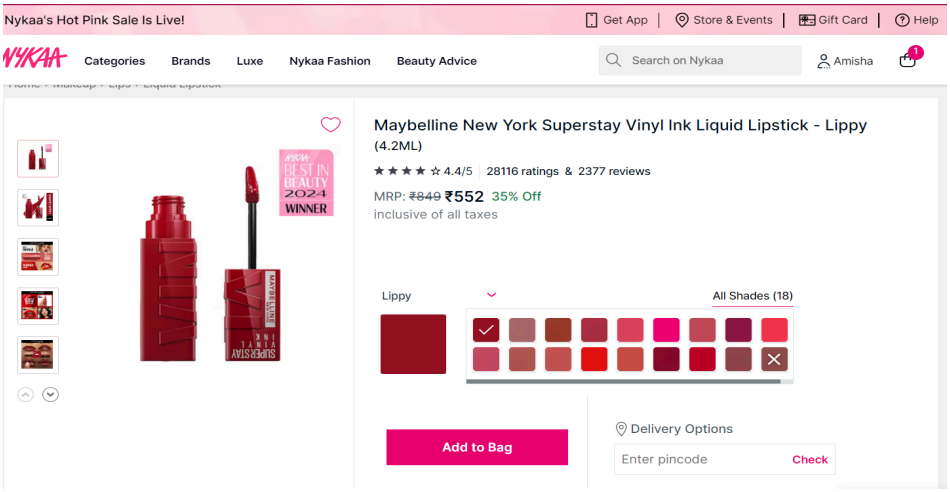

Fig 11 : Nykaa website opening after lipstick search

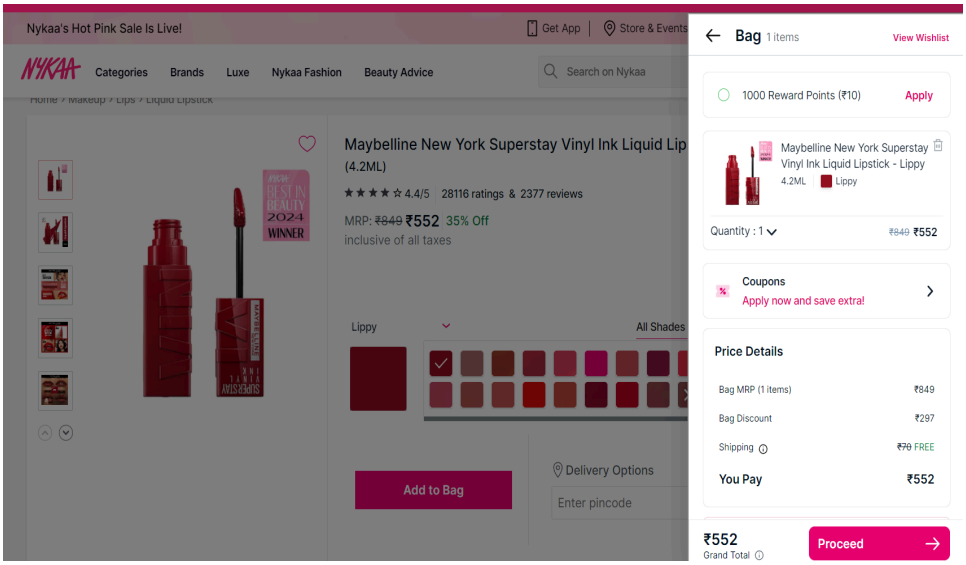

Fig 11 : Nykaa website opening after added to bag

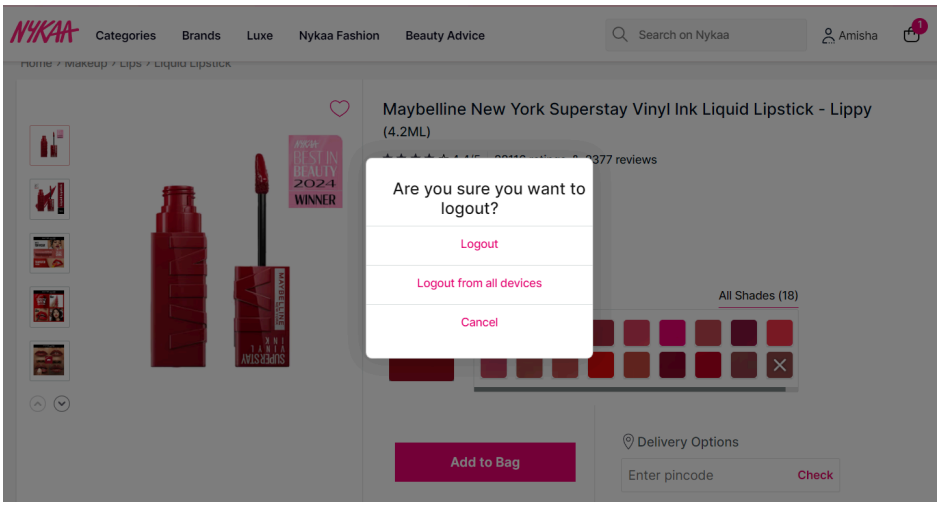

Fig 11 : Nykaa website log out page

CHAPTER 6

Conclusion/Future Scope

6.1 Conclusion

This project successfully showcases the power and efficiency of Selenium WebDriver in automating a crucial user journey on the Netflix platform. By simulating real user interactions, including login, profile selection, content search, and logout, the automated test suite has successfully verified the functionality and reliability of these core features. The integration of explicit waits and error handling mechanisms further ensures the stability and robustness of the test script, making it a valuable asset for continuous testing and regression testing efforts.

6.2 Future Scope

The current project provides a solid foundation for expanding the scope of automated testing on the Netflix 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 video playback, such as pausing, resuming, changing volume, and navigating between episodes.
- **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 Netflix'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 Netflix 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 continuously expanding and refining the automated test suite, Netflix can proactively detect and resolve issues, improve the user experience, and maintain a high level of quality and reliability for its vast global audience.

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: https://www.amazon.com/Agile-Testing-Practical-Guide-Testers/dp/0321534468
5. Nykaa Technology Blog: https://www.nykaa.com/beauty-blog/
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. Nykaa website : The official website for accessing and using Nykaa services: https://www.nykaa.com/?root=logo
10. Automation Step by Step (YouTube Channel): Offers step-by-step guides and tutorials for beginners in automation testing.