VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belgaum-590018

A Mini PROJECT REPORT ON

"Automation Testing on Amazon Webpage"

Submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF ENGINEERING IN
INFORMATION SCIENCE AND ENGINEERING

Submitted by

Mohammed Shaamil AG 1CR21IS097

Under the Guidance of

Prof. Saba Tahseen,
Dr. Sushelamma
Asst. Professor, Department of ISE, CMRIT

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

CMR INSTITUTE OF TECHNOLOGY

AECS LAYOUT, ITPL PARK ROAD, BENGALURU - 560037 2023-2024

CMR Institute of Technology
AECS Layout, Bengaluru-560037
Department of Information Science and Engineering

CERTIFICATE

Certified that the Mini Project Work in Software Testing Laboratory entitled **"Amazon Website Testing"** is a bonafide work carried out by **Mohammed Shaamil AG (1CR21IS097)** in partial fulfillment for the award of Bachelor of Engineering of the Visvesvaraya Technological University, Belgaum during the year 2023-2024. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report deposited in the department library. The mini-project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said Degree.

Prof. Saba Tahseen
Project Guide

Dr. Sushelamma
Project Guide

NAME:

USN:

Examiners Details:

External Examiner (Name & Signature with date) ----------

Internal Examiner (Name & Signature with date) ----------

ABSTRACT

This project focuses on developing an automated testing suite to validate the core functionalities of the Amazon India website using Selenium WebDriver and Python with PyTest. The automation script simulates key user interactions such as navigating to the Amazon homepage, selecting product categories, sorting products by price, and searching for specific items. By utilizing Selenium methods and JavaScript Executor, the script dynamically interacts with web elements, ensuring comprehensive coverage and accurate validation of Amazon India's essential features. This approach enhances the efficiency and reliability of the testing process while reducing manual effort and minimizing human error.

The project follows a structured methodology, starting with the creation of a WebDriver session and navigating to the Amazon India homepage. It locates necessary web elements, performs actions on them, and asserts the expected outcomes. Test results are logged in an Excel file, providing a clear record of test cases, statuses, and timestamps, which promotes better maintainability and traceability. The detailed test cases and their results demonstrate that the primary user interactions on the Amazon India website function correctly, thereby supporting a seamless user experience. This automation testing project contributes to the operational excellence of the Amazon platform by ensuring that critical user flows are rigorously tested and validated.

Additionally, the testing suite integrates smoothly with CI/CD pipelines for continuous testing and deployment. Detailed logging and reporting mechanisms provide clear visibility into test execution and outcomes, allowing for quick issue identification and resolution. The use of data-driven testing ensures a wide range of scenarios are covered, improving the overall quality and reliability of the Amazon India website. This automated approach significantly reduces manual testing time and effort, accelerating the development process and ensuring a superior user experience.

ACKNOWLEDGMENT

The satisfaction and euphoria that accompany a successful completion of any task would be incomplete without the mention of people who made it possible. Success is the epitome of hard work and perseverance, but steadfast of all is encouraging guidance.

So, it is with gratitude that I acknowledge all those whose guidance and encouragement served as beacon of light and crowned our effort with success. I would like to thank Dr. Sanjay Jain, Principal, CMRIT, Bangalore, for providing an excellent academic environment in the college and his never-ending support for the B.E program.

I would like to express my gratitude towards Dr. Jagadishwari V, Associate Professor and HOD, Department of Information Science and Engineering CMRIT, Bangalore, who provided guidance and gave valuable suggestions regarding the project.

I consider it a privilege and honor to express my sincere gratitude to our internal guide Prof. Jayashree M and Prof. Kanika Agrawal, Assistant Professor, Department of Information Science and Engineering, CMRIT, Bangalore, for their valuable guidance throughout the tenure of this project work.

I would like to thank all the faculty members who have always been very cooper- active and generous. Conclusively, I also thank all the non- teaching staff and all others who have done immense help directly or in- directly during our project.

Mohammed Shaamil AG
1CR21IS097

Contents

List of Figures

List of Tables

1.INTRODUCTION

1.1 What is Software Testing

Software testing is a crucial process in the software development life cycle that involves evaluating and verifying that a software application meets the specified requirements and functions as intended. The primary objective of software testing is to identify and rectify defects, ensure the software's quality, and enhance its reliability and performance. By executing the software under controlled conditions, testers can uncover errors, inconsistencies, and gaps in the software's behavior compared to the expected outcomes. This process not only improves the software's functionality but also ensures that it provides a seamless user experience, thereby increasing user satisfaction and trust in the product. There are various levels and types of software testing, each serving a unique purpose and focusing on different aspects of the software. Unit testing involves examining individual components or modules of the software to ensure they work correctly. Integration testing evaluates the interaction between different modules, ensuring they function together as intended.

System testing tests the complete system to verify that it meets the specified requirements, while acceptance testing involves real-world testing by end-users to ensure the software meets their needs and expectations. These levels of testing collectively ensure that the software is robust, reliable, and also ready for the deployment. Testing methodologies can be broadly categorized into manual and automated testing. Software testing encompasses a broad range of methodologies, each tailored to address specific aspects of software quality and functionality. These methodologies can be broadly classified into manual and automated testing, each with various subtypes. The following sections detail the primary types of software testing, their objectives, methodologies, and applications.

Manual testing involves human testers executing test cases without the aid of automated tools, making it ideal for detecting user interface issues and evaluating the overall user experience. Automated testing, on the other hand, uses specialized software tools to execute predefined test cases, making it highly efficient for repetitive tasks, regression testing, and performance testing. Tools like Selenium, QTP, and JUnit facilitate automated testing, ensuring consistency, speed, and accuracy in test execution. Automated testing is particularly beneficial in continuous integration and continuous deployment (CI/CD) environments, where quick feedback and frequent releases are most essential. The importance of software testing cannot be overstated.

1.2 Types of Testing

Software testing encompasses a broad range of methodologies, each designed to address specific aspects of software quality and functionality. The types of testing can be broadly classified into manual and automated testing, with various subtypes under each category. This section explores the key types of software testing, detailing their objectives, methodologies, and applications.

1.2.1 Manual Testing

Manual testing involves human testers executing test cases without the aid of automated tools. This type of testing relies on the tester's intuition, experience, and knowledge to identify defects. Manual testing is essential for understanding the user interface and user experience, as it allows testers to observe and interact with the application as end-users would. Common types of manual testing include:

- **Exploratory Testing:** Testers explore the application without predefined test cases, using their creativity and intuition to identify defects. This type is particularly useful in early development stages or when requirements are not well-defined.
- **Ad-Hoc Testing:** Similar to exploratory testing, ad-hoc testing is informal and unstructured. Testers randomly test the application without any plan or documentation, aiming to find defects through random use cases.
- **Usability Testing:** Focuses on the user interface and overall user experience. Testers evaluate the application's ease of use, navigation, and overall design, ensuring that it meets user expectations and provides a satisfactory experience.
- **Regression Testing:** Ensures that recent changes or additions to the code have not adversely affected existing functionality. Testers re-execute previously conducted test cases to confirm that the software still performs as expected.

1.2.2 Automated Testing

Automated testing uses specialized tools to execute test cases automatically. This approach is efficient for repetitive tasks, large-scale regression testing, and performance testing, ensuring consistency and speed in test execution. Automated testing is particularly valuable in continuous integration and delivery (CI/CD) environments. Key types of automated testing include:

- **Unit Testing:** Focuses on individual components or modules of the software. Automated unit tests are written by developers to validate that each unit performs as designed. Tools like JUnit for Java and NUnit for .NET are commonly used for unit testing.
- **Integration Testing:** Validates the interaction between different modules or services within the application. Automated integration tests ensure that combined components work together correctly. Tools like Selenium WebDriver can automate integration tests.

- **System Testing:** Involves testing the complete and integrated software system. Automated system tests verify that the software meets the specified requirements. This type of testing covers end-to-end scenarios and validates the overall functionality.
- **Performance Testing:** Evaluates the application's performance under various conditions, including load testing, stress testing, and scalability testing. Automated performance tests simulate user load and measure response times, throughput, and resource utilization. Tools like Apache JMeter and LoadRunner are widely used for performance testing.

1.2.3 Black Box Testing

Black box testing focuses on testing the functionality of the software without knowledge of its internal code structure. Testers provide input and observe the output to verify that the application meets the specified requirements. This type of testing includes:

- **Functional Testing:** Validates that the software performs its intended functions correctly. Testers create test cases based on the software's requirements and specifications, ensuring that all functionalities work as expected.
- **Non-Functional Testing:** Assesses non-functional aspects of the software, such as performance, usability, reliability, and security. Non-functional testing ensures that the software meets the required standards for quality attributes.
- **User Acceptance Testing (UAT):** Conducted by end-users to verify that the software meets their needs and requirements. UAT is the final validation before the software goes live, ensuring that it delivers the expected value to users.

1.2.4 White Box Testing

White box testing, also known as clear box testing, involves testing the internal structures or workings of an application. Testers require knowledge of the code and design to create test cases that cover specific code paths, logic branches, and conditions. This type of testing includes:

- **Unit Testing:** As described under automated testing, unit testing involves testing individual components or modules at the code level to ensure they function correctly.
- **Integration Testing:** In white box integration testing, testers focus on the internal interactions between integrated units, ensuring that data flow and control logic are correct.
- **Security Testing:** Involves testing the application for security vulnerabilities and weaknesses. White box security testing requires knowledge of the application's code and architecture to identify potential threats and ensure robust security measures.

1.2.5 Grey Box Testing

Grey box testing is a hybrid approach that combines elements of both black box and white box testing. Testers have partial knowledge of the internal workings of the application, allowing them

to design better-informed test cases. This approach is useful for identifying issues that may not be evident through purely black box or white box testing methods. Key applications:

- **Penetration Testing:** Testers simulate attacks on the application to identify security vulnerabilities. Grey box penetration testing involves knowledge of the application's code and architecture, enabling more targeted and effective testing.
- **End-to-End Testing:** Grey box end-to-end testing validates the entire application flow, from start to finish, covering all integrated components and interactions. This approach ensures comprehensive coverage and identifies issues across the entire system.
- **Regression Testing:** In gray box regression testing, testers use knowledge of the code changes to focus their testing efforts on affected areas, ensuring that recent modifications do not introduce new defects.

1.2.6 Performance Testing

Performance testing evaluates the speed, responsiveness, and stability of a software application under various load conditions. This type of testing ensures that the application can handle real-world usage scenarios effectively. Key types of performance testing include:

- **Load Testing:** Measures the application's performance under expected user loads. Load testing identifies bottlenecks and ensures that the application can handle the anticipated number of users without performance degradation.
- **Stress Testing:** Evaluates the application's behavior under extreme conditions, such as high traffic or limited resources. Stress testing helps identify the application's breaking point and its ability to recover from failures.
- **Scalability Testing:** Assesses the application's ability to scale up or down in response to changing user loads. Scalability testing ensures that the application can maintain performance and stability as the user base grows or shrinks.
- **Volume Testing:** Involves testing the application with a large volume of data to evaluate its performance and behavior. Volume testing ensures that the application can handle large datasets without performance issues.

1.2.7 Usability Testing

Usability testing focuses on the user interface and overall user experience. Testers evaluate the application's ease of use, navigation, and design to ensure that it meets user expectations and provides a satisfactory experience. Key aspects of usability testing include:

- **User Interface (UI) Testing:** Evaluates the application's graphical interface, ensuring that all elements are accessible, intuitive, and visually appealing. UI testing ensures that the application is user-friendly and aesthetically pleasing.

- **Accessibility Testing:** Ensures that the application is accessible to all users, including those with disabilities. Accessibility testing evaluates the application against accessibility standards and guidelines, ensuring compliance and inclusivity.
- **Navigation Testing:** Assesses the ease of navigation within the application. Navigation testing ensures that users can easily find and access features and information, providing a smooth and intuitive user experience.
- **User Feedback:** Involves gathering feedback from real users to identify usability issues and areas for improvement. User feedback provides valuable insights into the application's user experience and helps prioritize enhancements.

1.2.8 Security Testing

Security testing identifies vulnerabilities, threats, and risks in a software application to ensure that it is protected against unauthorized access, data breaches, and other security threats. This type of testing includes:

- **Penetration Testing:** Simulates attacks on the application to identify security vulnerabilities. Penetration testing helps identify weaknesses in the application's defenses and provides recommendations for improving security.
- **Vulnerability Scanning:** Uses automated tools to scan the application for known vulnerabilities. Vulnerability scanning helps identify and remediate security issues before they can be exploited by attackers.
- **Risk Assessment:** Evaluates the application's risk profile, identifying potential threats and their impact. Risk assessment helps prioritize security efforts and ensure that critical vulnerabilities are addressed promptly.
- **Security Auditing:** Involves reviewing the application's security policies, procedures, and controls. Security auditing ensures that the application complies with security standards and regulations, providing a robust security posture.
- **Network Compatibility:** Assesses the application's performance under different network conditions, such as varying bandwidths and latencies. Network compatibility testing ensures that the application can handle different network environments and provides a consistent experience for all users.

In conclusion, the various types of software testing play a crucial role in ensuring the quality, reliability, and security of software applications. By employing a combination of manual and automated testing methods, organizations can identify and address defects early in the development process

1.3 About Automation Tool Used

Selenium is one of the most widely used automation tools in the software testing industry, known for its ability to automate web applications for testing purposes. It provides a suite of tools that support automation across different browsers and platforms, making it a versatile choice for many testers and developers. This section will provide an in-depth look at Selenium, its components, and its application in automating tests for web applications.

Overview of Selenium

Selenium is an open-source tool that automates web browsers. It provides a playback tool for authoring functional tests without needing to learn a test scripting language (Selenium IDE). Additionally, it offers a test domain-specific language (Selenium WebDriver) to write tests in a number of popular programming languages, including Java, C#, Groovy, Perl, PHP, Python, Ruby, and Scala.

Key Features of Selenium:

- **Cross-Browser Compatibility:** Selenium supports multiple browsers like Chrome, Firefox, Internet Explorer, Edge, and Safari. This allows testers to ensure their web applications work across different browsers.
- **Cross-Platform Support:** Selenium runs on various operating systems, including Windows, macOS, and Linux.
- **Support for Multiple Languages:** Selenium supports multiple programming languages, enabling testers to write scripts in their preferred language.
- **Integration with CI/CD Tools:** Selenium can be easily integrated with continuous integration and continuous deployment tools like Jenkins, allowing for automated build and test processes.

Components of Selenium

Selenium is composed of several tools, each serving a specific purpose in the automation process. These include:

1. **Selenium IDE (Integrated Development Environment):**
    - A Firefox and Chrome plugin that allows testers to record their actions as they follow the workflow that needs to be tested.
    - It is useful for creating quick bug reproduction scripts and for automation-aided exploratory testing.
2. **Selenium WebDriver:**
    - A programming interface to create and execute test cases. WebDriver can control the browser from the OS level.
    - Provides bindings for many programming languages.

3. **Selenium Grid:**
    - A tool used to run parallel tests across different machines and different browsers simultaneously, which significantly speeds up the testing process.
    - It is especially useful for cross-browser testing and cross-platform testing.

Using Selenium for Test Automation

Setting Up Selenium:

- To begin using Selenium, you need to download the Selenium WebDriver for your preferred browser and language bindings.
- You also need to set up the development environment. For Java, this typically involves setting up an Integrated Development Environment (IDE) like Eclipse or IntelliJ IDEA, and managing dependencies using a build tool like Maven or Gradle.

Integration with Other Tools

Selenium can be integrated with a variety of tools to enhance its capabilities:

- **TestNG/JUnit:** Frameworks for structuring and running tests, and for generating detailed test reports.
- **Jenkins:** For continuous integration and continuous deployment, allowing automated tests to run as part of the build process.
- **Maven/Gradle:** Build automation tools that manage project dependencies and build configurations.
- **Allure:** For creating detailed and visually appealing test reports.

Conclusion

Selenium is a powerful tool for automating web application testing, offering flexibility, cross-platform support, and integration capabilities that make it a preferred choice among testers. By leveraging Selenium, organizations can ensure their web applications are robust, reliable, and perform well across different environments, ultimately delivering a high-quality user experience.

1.4 Problem Statement

The primary problem addressed by this project is the inefficiency and limitations of manual testing in ensuring the robust functionality and user experience of the Amazon India website. Specifically, the project seeks to automate the testing of key functionalities such as navigating the homepage, selecting product categories, sorting products, and performing product searches. By implementing an automated testing solution using Selenium WebDriver and Python with PyTest, the project aims to streamline the testing process, increase test coverage, and improve the overall quality and reliability of the Amazon India website. This automated approach is designed to reduce the time and effort required for manual testing, minimize human errors, and ensure a consistent and seamless user experience across various scenarios.

1.5 Objective of the Project

The objective of this project is to develop a comprehensive automated testing framework for the Amazon India website using Selenium WebDriver and Python with PyTest. This framework will be designed to:

1. **Enhance Test Efficiency**: By automating repetitive and time-consuming test cases, the project aims to significantly reduce the time and effort required for testing. This allows for faster release cycles and more efficient use of testing resources.
2. **Increase Test Coverage**: Automated tests will cover a wide range of functionalities and scenarios, including navigating the homepage, selecting product categories, sorting products, and performing product searches. This ensures that all critical aspects of the application are thoroughly tested.
3. **Ensure Consistency and Reliability**: Automated tests provide consistent and repeatable results, reducing the risk of human error and ensuring that defects are identified and addressed promptly. This leads to a more reliable and stable application.
4. **Support Cross-Browser and Cross-Platform Testing**: The project will ensure that the automated tests can be executed across different browsers (Chrome, Firefox, Safari, Edge) and operating systems (Windows, macOS, Linux), ensuring compatibility and performance across various environments.
5. **Facilitate Continuous Integration and Continuous Deployment (CI/CD)**: By integrating the automated testing framework with CI/CD tools like Jenkins, the project will enable continuous testing as part of the build and deployment pipeline. This ensures that any changes to the codebase are automatically tested, promoting early detection of issues and maintaining high software quality.
6. **Provide Detailed Reporting and Analysis**: The automated testing framework will generate comprehensive reports and logs, providing detailed insights into test execution, coverage, and results.

By achieving these objectives, the project will deliver a robust and scalable automated testing solution that enhances the quality assurance process for the Amazon India website, ensuring a high-quality user experience and supporting the platform's ongoing growth.

1.6 Schedule (Gantt Chart)

Project Phases and Tasks with Dates

Task	Start Date	End Date	Duration (weeks)
Project Planning and Setup	May 16, 2024	May 21, 2024	1
Requirement Analysis	May 21, 2024	May 28, 2024	1
Environment Setup	June 7, 2024	June 10, 2024	1
Test Case Design	June 11, 2024	July 4, 2024	3
Script Development	July 5, 2024	July 18, 2024	2
Test Execution	July 19, 2024	July 22, 2024	1
Reporting and Analysis	July 19, 2024	July 22, 2024	1

Fig 1.6.1: Project phases and Task along with dates

Gantt Chart with Dates

Ask	May 17	May 24	May 31	June 7	June 14	June 21	June 28	July 5	July 12	July 19
Project Planning and Setup	X	X								
Requirement Analysis			X							
Environment Setup				X						
Test Case Design					X	X	X	X	X	
Script Development										
Test Execution										X
Reporting and Analysis										X
Project Review and Closure										X

Fig 1.6.2: Gantt Chart

2. Literature Survey

2.1 About Manual Testing with Advantages and Disadvantages

Manual testing is a process where testers manually execute test cases without the help of any automation tools. The tester plays the role of an end user and verifies that all features of the application are working correctly by following a set of predefined test cases. This method is essential for ensuring the software's usability and identifying issues from a human perspective.

Advantages of Manual Testing:

1. **Human Insight:**
   - Manual testing allows testers to find issues based on human intuition and experience, particularly those related to user interface and user experience that automated tests might miss.
2. **Flexibility:**
   - It is adaptable to changes in real-time, allowing testers to quickly adjust test cases as needed without waiting for scripts to be modified.
3. **Cost-Effective for Small Projects:**
   - For small projects or applications with limited test cases, manual testing can be more cost-effective than investing time and resources into automation.
4. **Better Understanding of the Application:**
   - Through manual testing, testers gain a deeper understanding of the application, which can be beneficial for identifying potential areas of improvement.

Disadvantages of Manual Testing:

1. **Time-Consuming:**
   - Manual testing is often slower compared to automated testing, as each test case needs to be executed individually by a human tester.
2. **Prone to Human Error:**
   - Testers can make mistakes, miss defects, or inconsistently perform test cases, leading to less reliable results.
3. **Not Suitable for Repetitive Tasks:**
   - Manual testing becomes inefficient for tasks that need to be repeated frequently, such as regression testing after every new build.
4. **Limited Test Coverage:**
   - Due to time and resource constraints, manual testing might not cover all possible test scenarios, potentially leaving some defects undetected.
5. **Difficulty in Executing Complex Test Scenarios:**
   - Complex test scenarios that involve multiple variables can be challenging to execute manually.

Overall, while manual testing is crucial for certain aspects of the testing process, especially those requiring human judgment, it has limitations that can be addressed through automation.

2. About Automation Testing with Advantages and Disadvantages

Automation testing involves using specialized tools to execute pre-scripted test cases on a software application before it is released into production. This method is designed to reduce the effort and time required for repetitive testing tasks and to improve the accuracy and reliability of test results.

Advantages of Automation Testing:

1. **Speed and Efficiency:**
   - Automated tests can be executed much faster than manual tests, significantly reducing the time needed for testing cycles and enabling more frequent testing.
2. **Consistency and Accuracy:**
   - Automation eliminates human error, ensuring that tests are performed consistently and accurately each time they are run.
3. **Reusability of Test Scripts:**
   - Once automated test scripts are created, they can be reused across multiple testing cycles without additional cost or effort, enhancing the efficiency of the testing process.
4. **Increased Test Coverage:**
   - Automated testing allows for a broader test coverage, as more test cases can be executed in less time, including complex and data-driven tests.
5. **Supports Continuous Integration/Continuous Deployment (CI/CD):**
   - Automation integrates seamlessly with CI/CD pipelines, enabling continuous testing and ensuring that code changes are automatically tested, leading to quicker and safer releases.
6. **Cost-Effective in the Long Run:**
   - Although the initial investment in automation tools and script development can be high, the long-term benefits of reduced manual effort and faster testing cycles lead to overall cost savings.

Disadvantages of Automation Testing:

1. **High Initial Investment:**
   - Setting up an automation testing environment requires significant upfront investment in tools, infrastructure, and skilled personnel.
2. **Script Development and Maintenance:**

   - Writing and maintaining automated test scripts require specialized skills and can be time-consuming. Changes in the application may necessitate frequent updates to the scripts.
3. **Not Suitable for All Types of Testing:**
   - Certain types of testing, such as exploratory testing, usability testing, and ad-hoc testing, rely on human intuition and cannot be effectively automated.
4. **Limited to Predefined Test Cases:**
   - Automated testing is confined to the test cases that have been pre-scripted, potentially missing defects in areas not covered by these scripts.
5. **Complex Setup:**
   - The setup of an automation framework can be complex and may require integration with various tools and systems, adding to the complexity of the testing process.
6. **Requires Continuous Monitoring:**
   - Automated tests need to be continuously monitored and maintained to ensure they are running correctly and adapting to changes in the application.

Overall, automation testing is a powerful method for improving the efficiency, coverage, and reliability of the testing process, though it comes with challenges that must be managed to maximize its benefits.

Conclusion

Automation testing, using Selenium WebDriver and Python with PyTest, involves executing pre-scripted test cases on the Amazon India website to streamline the testing process and enhance accuracy. Manual testing, where testers verify application features without automation tools, is crucial for assessing usability and identifying user experience issues from a human perspective. While manual testing is valuable for tasks requiring human insight, its limitations can be addressed with automation, which offers speed, efficiency, and consistency for repetitive and extensive test cases. A robust testing strategy for Amazon India combines both manual and automated approaches to achieve comprehensive coverage and ensure high-quality software delivery. This hybrid approach leverages the strengths of both methods, with automation handling repetitive tasks and large-scale testing, while manual testing focuses on more nuanced aspects such as user interface and experience.

3. System Architecture and Design

3.1 Block Diagram/Architecture/Methodology Used with a Brief Explanation

The architecture of the automated testing framework for the amazon web application using Selenium and Java is designed to ensure modularity, scalability, and ease of maintenance. The architecture is divided into several key components, each responsible for a specific part of the testing process. Below is a high-level block diagram of the system architecture:

Block Diagram:

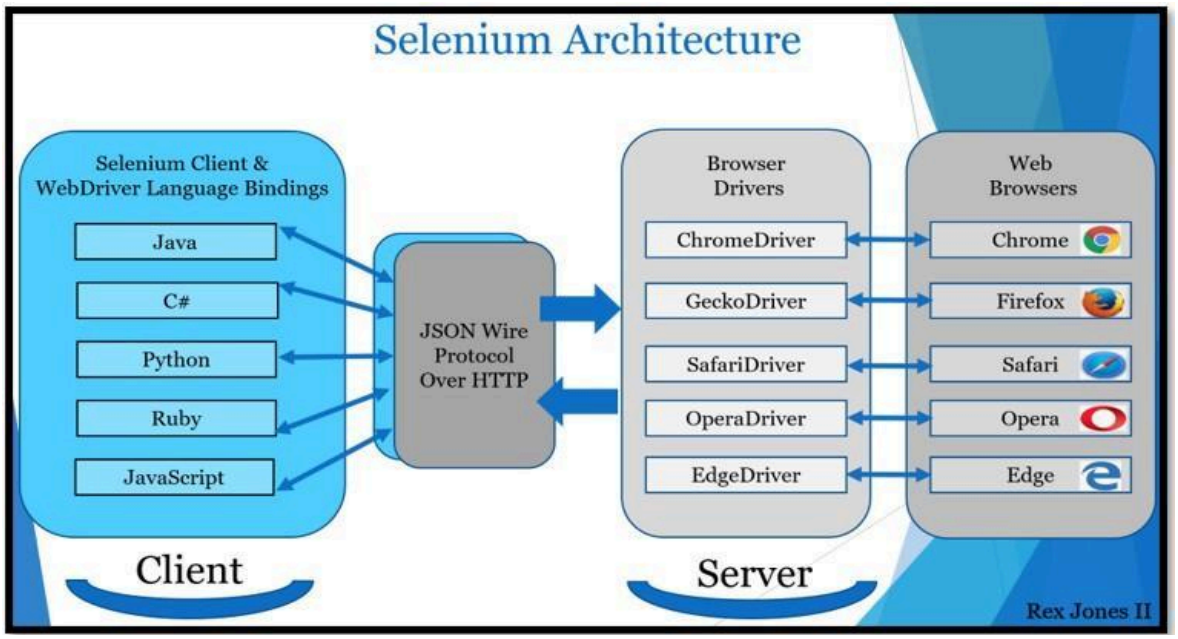


Fig 3.1.1: Selenium architecture

The Above Figure 3.1 illustrates the flow of commands in Selenium WebDriver and the data flow

Selenium Language Bindings

These are the client libraries in different programming languages that users can utilize to write their test scripts. The supported languages shown are:

- Java
- Ruby
- C#
- Python
- JavaScript

JSON Wire Protocol

This is a transport mechanism used to send commands from the Selenium client to the browser driver. The client libraries send HTTP requests to the browser driver using this protocol, which translates these commands into actions within the browser.

Browser Drivers

These drivers serve as intermediaries between the Selenium commands and the actual browser. Each browser has a specific driver:

- Firefox Driver
- Chrome Driver
- Safari Driver
- Opera Driver
- Edge Driver

These drivers receive the commands via the JSON Wire Protocol and execute them in the respective browsers.

Real Browsers

These are the actual web browsers that will be automated. The drivers interact with these browsers to perform the necessary actions, such as clicking buttons, entering text, navigating between pages, etc. The browsers listed are:

- Firefox Browser
- Chrome Browser
- Safari Browser
- Opera Browser
- Edge Browser

Communication Flow

1. **Client Libraries to JSON Wire Protocol**: The user writes a test script using one of the Selenium language bindings (e.g., Java, Python). The script sends commands through the JSON Wire Protocol.
2. **Browser Driver to Real Browser**: The browser driver executes these commands in the actual browser, performing the actions specified in the test script.

Summary

- **Selenium Language Bindings**: Allow writing test scripts in multiple languages.
- **Browser Drivers**: Translate commands from JSON Wire Protocol to actions in the browser.
- **Real Browsers**: Where the actual test execution happens.

This architecture allows Selenium to be a versatile and powerful tool for automating web browsers across different platforms and programming languages.

3.2 Flow Chart/Path Flow Diagram with Brief

Explanation Flow Chart:

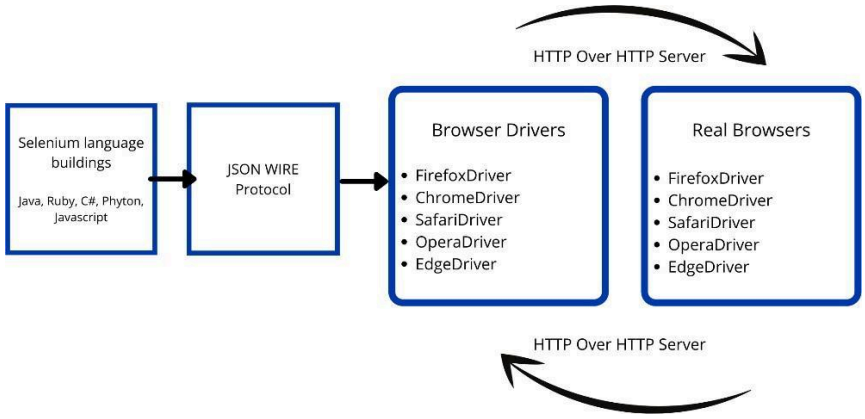


Fig 3.2.1: Flow chart

The above Figure 3.2 illustrates the command flow in Selenium WebDriver, detailing how test scripts written in various languages (Java, C#, PHP, Ruby, Python) are processed. The scripts use Selenium client libraries to send commands via the JSON Wire Protocol over HTTP to browser drivers such as Firefox Driver, InternetExplorerDriver, and Opera Driver. These drivers act as intermediaries, translating the commands into actions that the actual browsers (Firefox, Internet Explorer, Google Chrome, etc.) can execute. JSON protocol serves as the intermediary that translates the commands written in the test script into a format that the browser driver can understand. It operates over HTTP, sending the commands as HTTP requests. The real browsers then perform the specified actions and send back HTTP responses through the same chain, ensuring effective automation of web applications across different browsers and platforms. This flow ensures that Selenium WebDriver can automate web browsers across different platforms and programming languages by utilizing a standardized protocol and specific drivers for each browser

4. Implementation

4.1 Test Cases Table

Test Case ID	Description	Steps to Execute	Expected Result	Actual Result	Status
TC001	Login to amazon	1. Open amazon 2. Click login button 3. Enter Phone number and otp	User should be logged in successfully	User is logged in successfully	Successful
TC002	Search item	1. Go to search bar 2. Enter an item 3. Click on search	Item should be searched successfully	Item is searched successfully	Successful
TC003	Increment item count	1. Click on '+' option 2. View the item	Item should be incremented successfully	Item is incremented successfully	Successful

4.2 Black Box / White Box

Testing Black Box Testing:

- **Objective**: Verify the functionality of the amazon automation tasks without knowing the internal code structure.
- **Test Cases**: The same test cases as listed in the table above.
- **Method**: Input various data into the UI, observe the outputs, and compare them with expected results.

White Box Testing:

- **Objective**: Verify the internal workings of the amazon automation script.
- **Focus Areas**:
    - Validate the proper handling of web elements.
    - Ensure correct usage of Selenium WebDriver methods.
    - Check for appropriate waits and exception handling.
- **Code Coverage**: Ensure that all paths in the methods like
    - login(),
    - parsing_phone_number(),
    - enter_otp(),
    - product_search(),
    - add_to_cart(),
    - increment_item(),
    - Decrement_item(),
    - checkout(),
    - payment()

    These are executed during testing.

4.3 Code

The main logic and important modules are the individual methods that handle different tasks in the amazon automation script. Here are the main parts:

```
            if os.path.exists(cls.workbook_path):
                cls.wb = load_workbook(cls.workbook_path)
            else:
                cls.wb = Workbook()
            cls.ws = cls.wb.active
            if cls.ws.max_row == 1:
                cls.ws.append(["Test Case", "Status", "Details", "Timestamp"])

    @classmethod
    def teardown_class(cls):
        cls.wb.save(cls.workbook_path)

    def setup_method(self, method):
        self.driver = webdriver.Chrome()
        self.vars = {}

    def teardown_method(self, method):
```

Fig 4.3.1: Layout

```
            self.log_result( test_case: "test_navigate",  status: "PASSED",  details: "Navigated to Amazon In
        except Exception as e:
            self.log_result( test_case: "test_navigate",  status: "FAILED", str(e))
        finally:
            self.driver.close()

    def test_category(self):
        try:
            self.driver.get("https://www.amazon.in/")
            category_dropdown = self.driver.find_element(By.ID,  value: "searchDropdownBox")
            category_dropdown.click()
            time.sleep(2)  # Wait for dropdown options to appear
            category_dropdown.find_element(By.XPATH,  value: "//option[text()='Books']").click()
            time.sleep(2)  # Wait for the page to load
            self.log_result( test_case: "test_category",  status: "PASSED",  details: "Category 'Books' click
        except Exception as e:
            self.log_result( test_case: "test_category",  status: "FAILED", str(e))
        finally:
            self.driver.close()

    def test_sortproduct(self):
```

Fig 4.3.2: Sort Product

```
            search_box.send_keys(Keys.ENTER)
            sort_dropdown = self.driver.find_element(By.ID, value: "s-result-sort-select")
            sort_dropdown.click()
            self.driver.find_element(By.XPATH, value: "//option[text()='Price: High to Low']").click()
            time.sleep(5)
            self.log_result( test_case: "test_sortproduct", status: "PASSED", details: "Product sorted")
        except Exception as e:
            self.log_result( test_case: "test_sortproduct", status: "FAILED", str(e))
        finally:
            self.driver.close()

    def test_searchproduct(self):
        try:
            self.driver.get("https://www.amazon.in/")
            search_box = self.driver.find_element(By.ID, value: "twotabsearchtextbox")
            search_box.click()
            search_box.send_keys("Redmi Note 13 Pro 5G")
            search_box.send_keys(Keys.ENTER)
            time.sleep(5)
            self.log_result( test_case: "test_searchproduct", status: "PASSED", details: "Product searched")
        except Exception as e:
```

Fig 4.3.3: Search and Add to cart

5. Results/Output

5.1 Screenshots

1. Opening Website

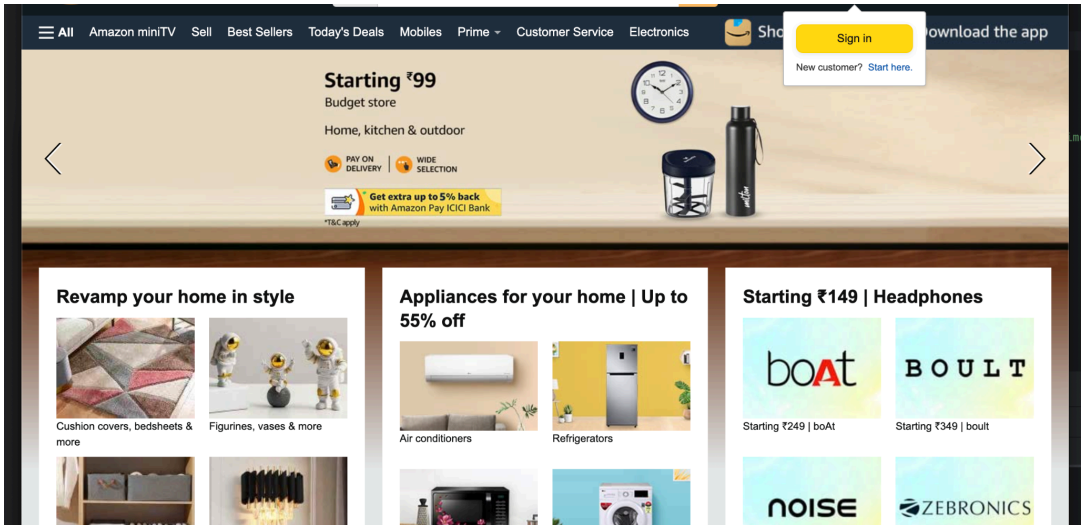
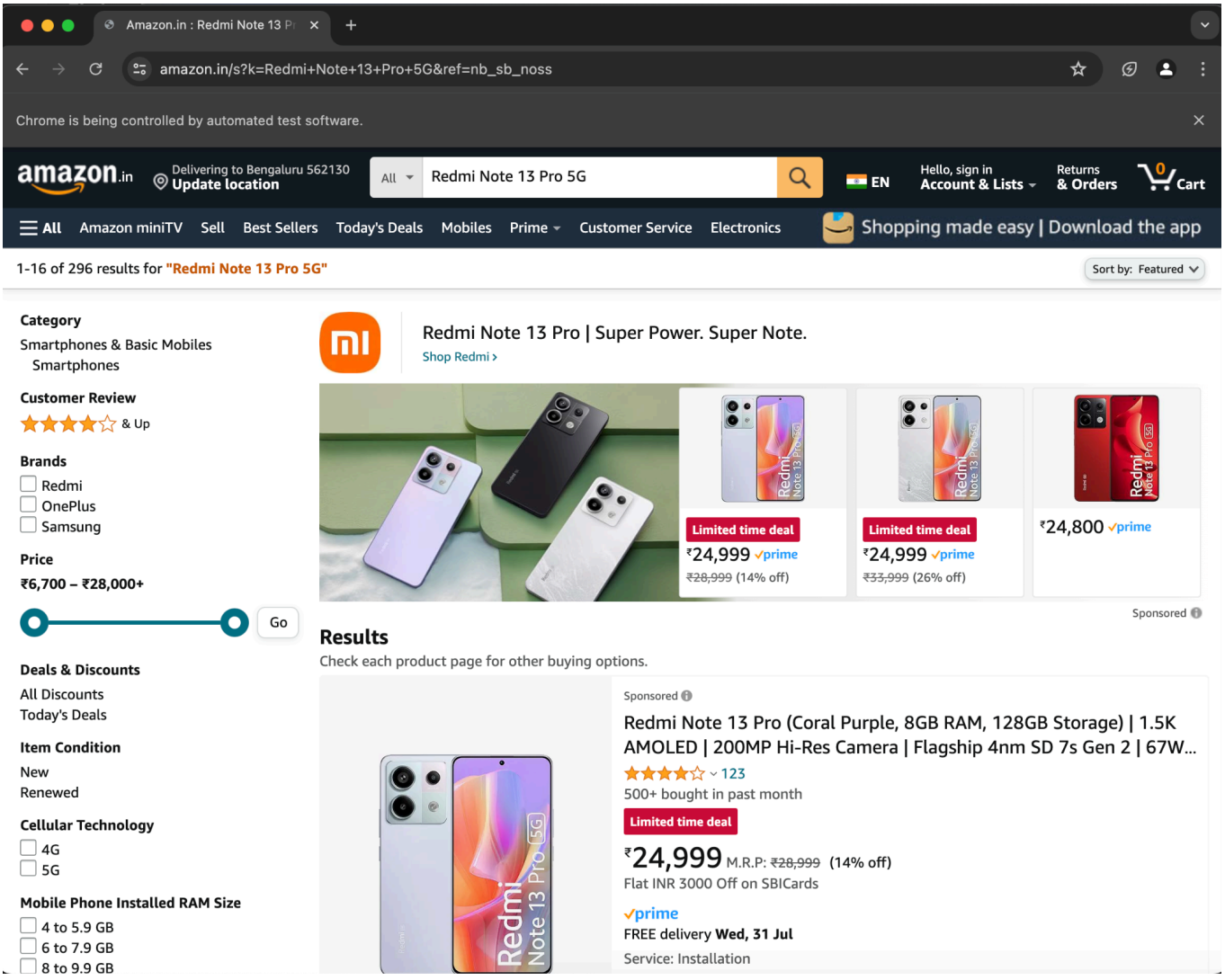


Fig 5.1.1: Opening Website

The above Fig 5.1.1 shows the initial page of amazon. The automated script enters into the Webpage of amazon. The Web page opens successfully and performs further operations.

The above Fig 5.1.1 shows the initial login page of amazon. The automated script enters the phone number and waits for otp to enter manually into the appropriate fields and clicks the login button to sign in. The successful login is confirmed by the subsequent navigation to the user's homepage.

2. Searching for an item



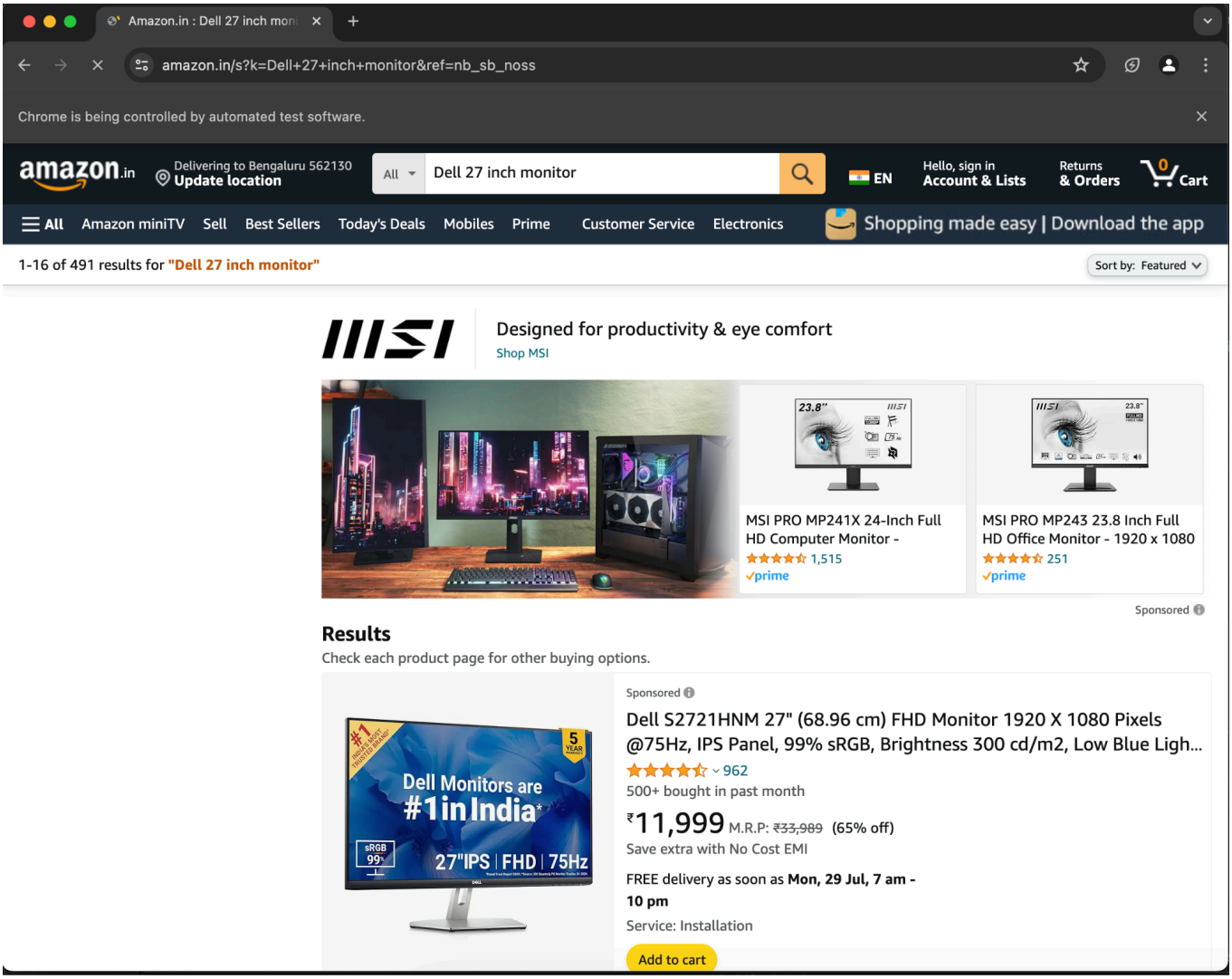


Fig 5.1.3: Searching in Item

The above Fig 5.1.2 shows the process of Searching for item. The automation script navigates to the search box, clicks on the 'box', and types an item name to be searched into the text area. Then click on search button.

6. Conclusion and Future Scope

6.1 Conclusion

In this project, we successfully demonstrated the automation of various interactions on the Amazon India website using Selenium WebDriver and Python with PyTest. The automated script performed tasks such as navigating to the Amazon homepage, selecting product categories, sorting products, and searching for specific items. These tasks were executed efficiently, showcasing the capabilities of Selenium for e-commerce website automation. The use of explicit waits and appropriate element locators ensured reliable execution even when dealing with dynamic web elements. This project highlights the potential of using automation for repetitive tasks on e-commerce platforms, improving efficiency in testing and potentially in customer service and product management.

6.2 Future Scope

The future scope of this Amazon India automation project can be expanded in several ways:

1. **Enhanced Test Coverage**: Implementing more comprehensive test scenarios, including edge cases and negative testing, to ensure thorough validation of the Amazon platform's functionalities.
2. **Performance Testing**: Integrating performance testing tools to measure load times, response times, and server performance under various conditions, ensuring the platform's robustness and scalability.
3. **Data-Driven Testing**: Implementing data-driven testing approaches, allowing for the easy scaling of test cases with different input data sets, enhancing the thoroughness and variety of tests.
4. **API Testing Integration**: Expanding the framework to include API testing for Amazon's backend services, ensuring both frontend and backend consistency and integration.
5. **Mobile App Automation**: Extending the automation framework to cover Amazon's mobile applications on iOS and Android platforms, ensuring a consistent user experience across all devices.
6. **Continuous Integration/Continuous Deployment (CI/CD)**: Integrating the automation suite into a CI/CD pipeline for automated testing with each new deployment, facilitating faster and more reliable releases.
7. **Reporting and Analytics**: Developing comprehensive reporting mechanisms with detailed analytics on test results, including visual representations of test coverage and performance metrics, to provide better insights into the quality and performance of the platform.

By pursuing these future enhancements, the project can evolve into a comprehensive testing and automation solution for Amazon India, potentially extending its use beyond testing to support various aspects of e-commerce operations. This could lead to improved quality assurance, more efficient operations, and an enhanced user experience on the Amazon platform.

7. References

- Selenium WebDriver - https://www.selenium.dev/documentation/en/webdriver/
- Explicit and Implicit Waits in Selenium -https://www.baeldung.com/selenium-waits
- ChromeDriver Documentation -https://sites.google.com/a/chromium.org/chromedriver/
- AutoIt Documentation - https://www.autoitscript.com/site/autoit/
- XPath and CSS Selectors -https://www.seleniumeasy.com/selenium-tutorials/xpath-css-selectors
- Automating File Uploads Using AutoIt - https://www.softwaretestinghelp.com/autoit-tutorial-1/
- Handling Web Elements - https://www.tutorialspoint.com/handling-web-elements-in-selenium
- Google - https://www.google.co.in/