SELENIUM

(A Comprehensive Report)

By:

Tharaneeshwaran V U (CS22B1056)

Naghul Pranav (CS22B1037)

Krishna J (CS22B1023)

Table of Contents

Introduction

Selenium stands as a robust open-source software testing framework specifically designed for automating web applications. Its primary role is to facilitate functional and regression testing, supporting various programming languages such as Java, Python, C#, and more. Selenium plays a crucial role in the automation of repetitive testing procedures by enabling the creation of scripts that simulate user interactions with web applications.

Historical Development

Selenium's origins date back to 2004 when Jason Huggins initially developed it as an internal tool for web application testing at ThoughtWorks. He humorously named it "Selenium" after the element selenium, which is utilized in certain medicines for treating mercury poisoning. The software was open-sourced in the same year, gaining swift adoption and fostering a vibrant Selenium community. Over the years, Selenium has evolved, with various versions and components being introduced.

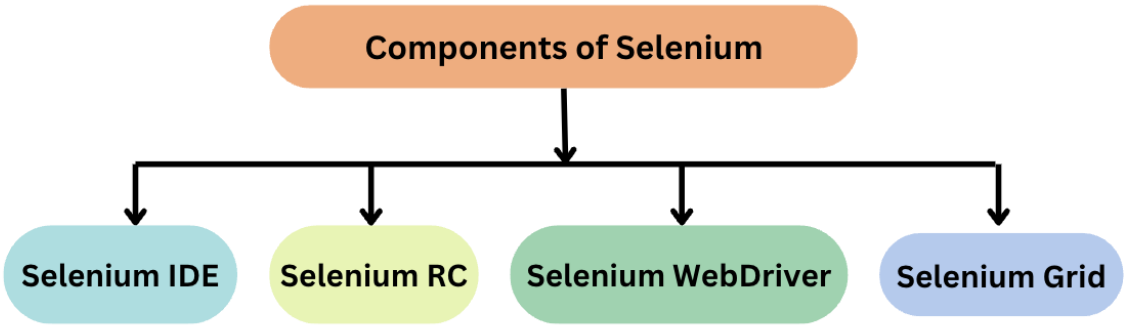
Challenges with Manual Testing

Manual testing is one of the primitive ways of software testing. It doesn't require the knowledge of any software testing tool and can practically test any application. The tester manually executes test cases against applications and compares the actual results with desired results. Any differences between the two are considered as defects and are immediately fixed. The tests are then re-run to ensure an utterly error-free application.

Manual testing has its drawbacks:

- It's extremely time-consuming and there's a high risk of error
- It requires the presence of a tester 24/7
- Requires manual creation of logs
- It has a limited scope

Components



Selenium has a dedicated suite that facilitates easy testing of web applications.

Selenium IDE

It is a Chrome and Firefox plugin. The primary use of a Selenium IDE is to record user interactions such as clicks, selections etc in the browser and plays them back as automated test. It then generates the test script (of the automated tests) in programming languages like C#, Java, Python, and Ruby and Selenese (Selenium's scripting language).

It helps in:

- Creating automated test scripts and validating them at speed
- Identifying and highlighting errors during the replay of interactions
- Cross Browser Testing

Selenium RC

Selenium RC was built to automate the testing of web applications by simulating user interactions across different browsers and platforms. It provided a way to browser automation remotely and execute test scripts written in various programming languages.

Limitations of Selenium RC:

- **Browser Limitations:** Selenium RC had to work with browsers using a JavaScript-based "proxy" mechanism, which introduced potential instability and limitations, especially when working with modern web applications.
- **Speed and Performance:** The use of a JavaScript proxy added overhead and affected the speed and performance of test execution.
- **Maintenance and Compatibility:** Selenium RC required separate "drivers" for each browser, making maintenance and compatibility challenging as browsers continued to update and evolve.

- **Synchronization Issues:** Selenium RC often faced synchronization problems, where test scripts had to wait for the browser to respond before proceeding to the next step.
- **Complex Setup:** Setting up Selenium RC involved multiple components, which could be complex and difficult to configure correctly.

Selenium WebDriver

It is a powerful and enhanced version of Selenium RC which was developed to overcome the limitations of Selenium RC. WebDriver communicates with browsers directly with the help of browser-specific native methods, thereby eliminating the need of Selenium RC. WebDriver works closely with Selenium IDE and Selenium Grid resulting in reliable test execution at speed and scale.

Selenium Grid

This is a smart proxy server that allows QAs to run tests in parallel on multiple machines. This is done by routing commands to remote web browser instances, where one server acts as the hub. This hub routes test commands that are in JSON format to multiple registered Grid nodes.

Practical usage & Applications

Selenium's utility stems from its ability to automate web applications by emulating user actions. It empowers testers and developers to draft scripts that instruct Selenium on how to undertake tasks like clicking buttons, inputting text, navigating web pages, and verifying expected outcomes. Subsequently, Selenium executes these instructions on various web browsers.

Selenium finds extensive application within the software industry for the following purposes:

- Functional Testing: Guaranteeing that the application fulfils its intended functions accurately.
- Regression Testing: Verifying that new code changes do not disrupt existing functionality.
- Load Testing: Simulating multiple users to evaluate the application's performance under various conditions.
- Cross-Browser Testing: Ensuring uniform functionality across different web browsers.
- Automating Repetitive Tasks: Automation of monotonous tasks like data entry and report generation.

Advantages

The advantages associated with Selenium encompass:

- ✓ **Open-Source Nature**: Selenium is freely available and benefits from an active and expansive community.
- ✓ **Multi-Browser Compatibility**: It extends support to a variety of web browsers, including Chrome, Firefox, Safari, and Edge.
- ✓ **Multilingual Support**: Test scripts can be authored in diverse programming languages.
- ✓ **Parallel Test Execution**: Selenium Grid permits concurrent test execution on multiple machines.
- ✓ **Integration with CI/CD**: It seamlessly integrates with Continuous Integration and Continuous Deployment pipelines.
- ✓ **Extensive Documentation and Support:** Selenium users have access to abundant resources and forums for guidance.

Limitations

Despite its myriad advantages, Selenium presents some limitations:

- ➢ **Complex Setup**: The initial setup of Selenium can prove to be intricate.
- ➢ **Maintenance Requirements**: Frequent script updates are necessary due to changes in the application's user interface.
- ➢ **Limited Desktop Application Support**: Selenium primarily focuses on web applications and offers limited capabilities for testing desktop applications.
- ➢ **Reporting Challenges**: Selenium lacks built-in reporting capabilities, necessitating the integration of supplementary tools for comprehensive reporting.
- ➢ **Mobile Testing Limitations**: While Selenium WebDriver supports mobile app testing to some extent, its capabilities in this domain are not as extensive.

Conclusion

Selenium emerges as a potent tool for automating web application testing. Its historical development, versatile components, and support for multiple programming languages make it invaluable for manual testers and automation engineers. Nevertheless, users should remain cognizant of its limitations and the challenges associated with maintaining test scripts, particularly in the context of continually evolving web applications. As the software industry progresses, Selenium retains its pivotal role in ensuring the quality and reliability of web applications.

References

- https://www.selenium.dev/
- https://www.webomates.com/blog/software-testing/selenium-testing/
- https://www.itview.in/blog/selenium-automation-testing-overview-and-its-benefits/
- https://www.browserstack.com/selenium
- https://www.simplilearn.com/tutorials/selenium-tutorial/what-is-selenium