Semester	S.E. Semester IV – Information Technology Engineering
Subject	Computer Networks and Network Design Lab
Subject Professor In-charge	Unnati Gohil
Assisting Teachers	-
Laboratory	MS Teams

Student Name	Sanika Kate	
Roll Number	22101A2005	
Grade and Subject Teacher's Signature		

Experiment Number	6	
Experiment Title	Case Study on Selenium	
Resources / Apparatus Required	Hardware: Basic Desktop with Windows or Linux.	Software: Java/ Python/Wireshark/Cisco Packet Tracer
Objectives (Skill Set / Knowledge Tested / Imparted)		
Theory:	**Selenium :** Selenium is one of the most widely used open source Web UI (User Interface) automation testing suite.It was originally developed by Jason Huggins in 2004 as an internal tool at Thought Works. Selenium can be easily deployed on platforms such as Windows, Linux, Solaris and Macintosh. Moreover, it supports OS (Operating System) for mobile applications like iOS, windows mobile and android. Selenium supports a variety of programming languages through the	

	use of drivers specific to each language.Languages supported by Selenium include C#, Java, Perl, PHP, Python and Ruby.Currently, Selenium Web driver is most popular with Java and C#. Selenium test scripts can be coded in any of the supported programming languages and can be run directly in most modern web browsers. Browsers supported by Selenium include Internet Explorer, Mozilla Firefox, Google Chrome and Safari. ## Challenges with Manual Testing **Prone to Human Error**: Manual testing is susceptible to human error, which can lead to inconsistent results and missed defects. **Time-Consuming**: Manual testing can be time-consuming, especially for repetitive tasks and regression testing, leading to longer release cycles. **Resource-Intensive**: It requires a dedicated team of testers, which can be costly for large projects. **Not Suitable for High-Volume Testing**: Manual testing is not efficient for high-volume testing or continuous testing in a CI/CD environment. ## Automation Testing beats Manual Testing **Repeatability and Consistency**: Automated tests can be executed repeatedly without errors, ensuring consistent and reliable results. In contrast, manual testing is prone to human errors, which can lead to inconsistent test outcomes.

	Efficiency: Automation is highly efficient for repetitive and high-volume testing tasks. It can execute a large number of test cases quickly and accurately, saving time and resources. **Regression Testing**: Automated tests are ideal for regression testing, where existing functionalities are tested to ensure they haven't been affected by code changes. Manual regression testing can be time-consuming and error-prone when compared to automation. **Parallel Execution**: Automation can execute tests in parallel across different browsers, devices, and environments. This capability is essential for testing cross-browser compatibility and scalability, which is challenging to achieve manually. **Continuous Integration/Continuous Delivery (CI/CD) Integration**: Automated tests can be seamlessly integrated into the CI/CD pipeline, providing rapid feedback to developers. This integration helps catch issues early in the development process, leading to faster and more reliable releases. **Resource and Cost Savings**: While there is an initial investment in setting up and maintaining an automation framework, it can lead to significant cost savings over time. Automated tests can be run 24/7 without requiring additional human resources. **Detailed Reporting**: Automation provides detailed test reports and logs, making it easier to identify the root cause of failures. ## Different Parts of Selenium **Selenium RC**

	Selenium RC, short for Selenium Remote Control, is an older version of the Selenium automation testing framework. It was one of the initial Selenium tools developed to automate web applications for testing purposes. Selenium RC has since been deprecated in favor of WebDriver, which offers improved functionality and is now the standard for Selenium-based automation testing. **Selenium IDE** Selenium IDE (Integrated Development Environment) is a browser extension and graphical user interface (GUI) tool for building and running Selenium test cases for web applications. It's one of the components in the Selenium suite of tools, along with Selenium WebDriver and Selenium Grid. Selenium IDE is designed primarily for users who are new to automated testing or for those who want to create simple automation scripts quickly without writing code. **Selenium Grid** Selenium Grid is a component of the Selenium automation testing framework that allows you to distribute and parallelize test execution across multiple machines, browsers, and operating systems. It is especially valuable when you need to perform cross-browser testing or when you want to execute a large number of tests quickly and efficiently. Selenium Grid helps save time and resources by enabling concurrent test execution on multiple remote machines, making it an essential tool for automated testing in a scalable and parallelized manner. **Selenium WebDriver** Selenium WebDriver is a widely used open-source automation testing

framework that allows testers and developers to automate interactions with web applications. It provides a programming interface to interact with web browsers, enabling the automation of tasks such as clicking buttons, filling out forms, navigating web pages, and validating page content. Selenium WebDriver is a core component of the Selenium automation testing suite and is available in various programming languages, making it accessible to a wide range of users.

Different WebDrivers

WebDriver for Chrome (ChromeDriver): ChromeDriver is used to automate interactions with the Google Chrome web browser. It is the official WebDriver for Chrome and is maintained by the Chrome team at Google.

WebDriver for Firefox (GeckoDriver): GeckoDriver is the WebDriver for the Mozilla Firefox browser. It is maintained by Mozilla and is used to automate Firefox browser interactions.

WebDriver for Safari (SafariDriver): SafariDriver is used for automating the Safari web browser on macOS. Starting from Safari 10, SafariDriver is bundled with Safari, so you don't need to download a separate driver.

WebDriver for Edge (EdgeDriver): EdgeDriver is used to automate the Microsoft Edge browser. EdgeDriver can be used to automate interactions with both the legacy EdgeHTML-based Edge and the newer Chromium-based Edge.

Working Of Selenium

Here's a step-by-step explanation of how Selenium works:

	Test Script Creation: Testers and developers write test scripts using programming languages like Java, Python, C#, Ruby, and others. These test scripts define a series of actions that simulate user interactions with a web application, such as opening a browser, navigating to a URL, clicking buttons, filling out forms, and verifying page content. **WebDriver Initialization:** Test scripts include code to initialize a WebDriver instance, which acts as a bridge between the script and the web browser. The WebDriver instance is chosen based on the browser you want to automate (e.g., ChromeDriver for Google Chrome, GeckoDriver for Firefox). **Interaction with the Browser:** The WebDriver communicates with the selected web browser using a browser-specific driver (e.g., ChromeDriver, GeckoDriver). The WebDriver sends commands and instructions to the browser's native automation interface (e.g., Chrome DevTools Protocol for Chrome). These commands include actions like clicking elements, sending keystrokes, navigating between pages, and more. **Web Page Interaction:** The browser executes the actions specified in the test script, mimicking user interactions with the web page. The WebDriver can locate web elements (e.g., buttons, input fields)

	using various strategies like XPath, CSS selectors, or element IDs, and interact with them as needed. **Data Validation:** Test scripts may include assertions and validations to check whether the web page behaves as expected. These assertions compare the actual results or page content with expected values or conditions. **Logging and Reporting:** Selenium provides mechanisms for generating logs and reports during test execution. This helps testers and developers monitor the progress of test runs and identify any issues. **Repeatable and Automated Testing:** Test scripts can be executed repeatedly to perform regression testing, ensuring that new code changes do not introduce defects or regressions in existing functionality. Selenium can be integrated into continuous integration (CI) and continuous delivery (CD) pipelines to automate testing as part of the development process. **Parallel Execution:** Selenium Grid or cloud-based testing services can be used to run tests in parallel on multiple browsers, versions, and platforms simultaneously, allowing for faster test execution. **Cross-Browser and Cross-Platform Testing:** Selenium supports various web browsers (e.g., Chrome, Firefox, Safari, Edge) and platforms (e.g., Windows, macOS, Linux), enabling

comprehensive cross-browser and cross-platform testing.

Integration with Testing Frameworks:

Test scripts created with Selenium can be integrated with testing frameworks like JUnit, TestNG, NUnit, or PyTest to organize tests, manage test data, and generate detailed reports.

Drawbacks Of Selenium

Complexity for Beginners: Selenium can be complex for beginners, especially those who are new to programming or automation testing. Writing and maintaining automation scripts may require a significant learning curve.

No Built-in Reporting: Selenium itself does not provide built-in reporting capabilities. Test reporting and result analysis need to be implemented separately using additional libraries or tools, which can be a challenge for some users.

No Built-in Test Data Management: Selenium does not offer built-in solutions for managing test data. Handling test data, such as test inputs and expected results, is typically the responsibility of the test script developer.

Flakiness with Web Element Locators: Selenium relies on web element locators (e.g., XPath, CSS selectors) to interact with elements on a web page. If web pages change frequently, the locators may become outdated, leading to test script failures.

Similar Tools for used for Testing:

Several testing tools and frameworks are similar to Selenium in their

capabilities for automating web applications. These tools can be alternatives or supplements to Selenium, depending on your specific testing needs and preferences.

Katalon Studio: Katalon Studio is a comprehensive automation testing solution that supports web, mobile, and API testing. It offers a user-friendly interface for both beginners and experienced testers. While it provides built-in support for Selenium, it also offers an alternative scripting mode for advanced users.

Cypress: Cypress is a JavaScript-based end-to-end testing framework that focuses on making web testing easy and fast. It provides a simple API and allows you to see what happens in the application as the tests run in real-time. Cypress is known for its speed, ease of setup, and the ability to perform both unit and end-to-end testing.

Playwright: Playwright is a cross-browser automation framework developed by Microsoft. It supports multiple browsers, including Chromium, Firefox, and WebKit. Playwright provides a more robust and reliable automation solution compared to Selenium, with built-in support for modern web features like single-page applications (SPAs) and browser contexts.

Appium: Appium is an open-source tool for automating mobile applications on Android and iOS platforms. It is similar to Selenium WebDriver but is tailored specifically for mobile application testing. Appium supports native, hybrid, and mobile web applications.

Comparison of Selenium with other tools

Selenium vs. Katalon Studio:

	Scope: Katalon Studio is an all-in-one automation solution that includes built-in test recording, scripting, and reporting, making it suitable for users with varying levels of technical expertise. Selenium is primarily a scripting-focused tool. **Learning Curve:** Katalon Studio is designed to be user-friendly and may have a shallower learning curve compared to Selenium. **Community and Support**: Selenium has a larger and more active community, while Katalon Studio provides commercial support and additional features. **Selenium vs. Cypress:** **Ease of Use**: Cypress is known for its user-friendliness, offering an easy-to-understand API and real-time test execution. Selenium may have a steeper learning curve, especially for beginners. **Speed**: Cypress is faster than Selenium due to its architecture, which allows it to execute tests directly in the browser. Selenium uses WebDriver, which communicates with the browser over a network connection, potentially causing delays. **Browser Support**: Selenium supports multiple browsers, while Cypress primarily focuses on Chromium-based browsers (although it has been expanding support). **Cross-Browser Testing**: Selenium is better suited for cross-browser testing because of its broad browser support.
Output	
Conclusion	In conclusion, Selenium played a pivotal role in helping address web application testing challenges, improving website quality, efficiency, and cross-browser compatibility while seamlessly integrating with their development workflow. This case study demonstrates how Selenium

	can be a valuable tool for businesses seeking to enhance their web application testing processes.