Position Papers of the 19th Conference on Computer Science and Intelligence Systems (FedCSIS) pp. 1–8
DOI: 10.15439/2024F3747
ISSN 2300-5963 ACSIS, Vol. 40

Analysis of end-to-end test automation tools based on the examples of Selenium WebDriver and Playwright

Agnieszka Antonczak, Beata Bylina
Institute of Computer Science,
Marie Curie-Sklodowska University
Pl. M. Curie-Skłodowskiej 5, 20-031 Lublin, Poland
Email: agnieszka.w.antonczak@gmail.com, beata.bylina@mail.umcs.pl

***Abstract*—In the digital era, ensuring software reliability and effectiveness is crucial for business operations and daily life. This article analyzes and compares two prominent end-to-end test automation tools, Selenium WebDriver and Playwright. By examining their architecture, functionality, and browser interaction, the study provides practical guidance on tool selection. It includes theoretical foundations of testing, the importance of manual testing, and a detailed analysis of both tools. The findings suggest that Playwright generally offers faster test execution, particularly in headless mode, and simpler configuration. In contrast, Selenium benefits from a mature community and extensive documentation. The choice of tool should be based on project-specific needs, with Playwright favored for speed and simplicity, and Selenium for community support and integration capabilities.**

I. Introduction

IN THE digital age, where technology is an integral part of everyday life, software quality and reliability are crucial for both business and personal efficiency. In response to these needs, end-to-end (E2E) testing has gained prominence as a method of software verification aimed at ensuring that IT systems perform to users' expectations under real-world conditions. By simulating the end user's interactions with an application, E2E testing allows for comprehensive verification of a system's functionality [6]. Automated software testing tools are critical in performing extensive system tests, reducing errors that could lead to financial losses, and improving reliability and performance [8]. Test automation, using tools such as Selenium WebDriver and Playwright, offers efficiency, speed and accuracy that are difficult to achieve with manual approaches. A comparative analysis [7] reveals that Selenium, a widely adopted open-source tool, offers extensive support for multiple browsers and platforms, which enhances its utility for diverse testing scenarios. The study highlights Selenium's flexibility and robustness, making it a prevalent choice among testers for complex web application environments. Conversely, commercial tools like HP QuickTest Professional (now HP UFT) provide strong integration capabilities with enterprise environments, offering built-in object repositories and comprehensive technical support, which can be crucial for continuous integration and extensive test management. Another study [4] underscores that while no single tool can meet all testing requirements, TestComplete ranked highest in effectiveness among the evaluated programs, suggesting the importance of aligning tool capabilities with team skills and project needs.

Similarly, the article by E. Pelivani and B. Cico [2] compares Selenium with Katalon Studio, highlighting the strengths and weaknesses of both tools in various testing scenarios. This study underscores the necessity of selecting the right tool based on specific project requirements, further validating the approach taken in our current comparative study between Selenium WebDriver and Playwright.

This article focuses on the analysis of Selenium WebDriver and Playwright tools in the context of their use for end-to-end test automation, motivated by the absence of comparative studies between these two tools, especially given the recent introduction of Playwright in 2020. Playwright has been increasingly mentioned as a potent automation tool in the tech community, prompting an evaluation of its capabilities relative to the well-established Selenium WebDriver.

The article begins with an introduction to Selenium, focusing on its architecture and how Selenium WebDriver communicates with browsers. It then moves on to Playwright, describing it as an advanced tool capable of complex web application testing across browsers and platforms. The article continues with a discussion of developing 15 detailed test cases for an online pet store, using the Page Object Model design pattern and tools such as Maven and TestNG. This is followed by a comparative analysis of test execution times, showing that Playwright generally has shorter test execution times. The stability of tests in headless and non-headless modes is investigated. Personal experiences with the ease of use and configuration of both tools are shared. The availability of documentation, community support and educational resources for both tools is compared. Differences in extensibility and integration capabilities are described, focusing on how each tool can be extended and integrated with other systems. The article concludes with a summary of the findings, offering final thoughts on the strengths and weaknesses of Selenium WebDriver and Playwright, and providing recommendations for choosing the right tool based on specific project requirements

and team expertise.

II. Selenium

Selenium WebDriver [14] is an advanced browser test automation tool that simulates user interactions with a web application. The development of Selenium, initiated in 2004 by Jason Huggins, was a response to the limitations of manual user interface testing. Originally an internal project of ThoughtWorks, it quickly gained popularity and became open-source, allowing it to evolve rapidly and adapt to new web browser technologies [1]. Selenium's architecture is characterized by its modularity and ability to integrate with various programming languages such as Java, C#, Python, which significantly extending its versatility and accessibility. The core component, Selenium WebDriver, communicates directly with the browser using dedicated drivers, such as ChromeDriver for Google Chrome or GeckoDriver for Firefox. This direct communication allows Selenium to accurately mimic user behavior, from clicks and text entry to advanced interactions with various web elements. Selenium enables tests to be run on a wide range of browsers, which is crucial for cross-browser compatibility verification of web applications. Selenium is one of the prominent automated GUI testing tools, widely used for its capability to test web applications across different browsers [8]. Its modularity also allows tests to be easily scalable and integrated into existing frameworks, significantly improving development and testing processes in dynamically changing production environments. Setting up a test environment with Selenium is relatively straightforward, typically involving the setup of appropriate browser drivers and selecting a development environment for executing test scripts. This aspect makes Selenium an attractive solution for organizations with varying degrees of technical sophistication, allowing even less technical users to effectively design and execute automated tests [5].

III. Playwright

Playwright [12], a state-of-the-art browser test automation tool, has been introduced by Microsoft as a framework that provides comprehensive capabilities for web application testing. The development of Playwright is a response to the growing need for end-to-end testing that can be executed reliably and reproducibly across multiple platforms and browsers simultaneously. Since its debut, Playwright has gained recognition for its ability to work with Chromium, Firefox, and Safari, thus offering extensive cross-browser testing capabilities. Playwright's architecture is built around the idea of a "browser context," which enables the simulation of multiple sessions in a single browser instance, which is particularly useful for testing scenarios that depend on user sessions. In addition, Playwright integrates with a variety of development and testing environments providing developers with the flexibility to choose tools tailored to their specific projects. It is also distinguished by its support for programming languages such as JavaScript, Python, as well as C# and Java, which accounts for its versatility. Setting up a test environment with Playwright is intuitive and mainly involves installing the appropriate npm package or equivalents in other programming languages. Playwright provides custom drivers for browsers that are automatically managed by the framework, eliminating the need to manually configure and update browser drivers. In terms of testing capabilities, Playwright offers features such as screenshot generation, video recording of test sessions, and advanced context and session management. These features make it uniquely suited to the dynamically changing environment of modern web applications, where there are often requirements for testing application state-dependent functionality and user interaction.

IV. Test data

The analysis utilizes the example of the online pet store *https://petstore.octoperf.com* to provide a practical context for this comparison. Fifteen detailed test cases were created for this store, which were used to implement automated tests using both tools. The tests were written in Java, utilizing Maven [11] for dependency management and TestNG [13] for test management, which streamlined the testing process and enhanced the execution framework. The Page Object Model (POM) design pattern was used to implement the tests, making the test scripts more readable and easier to maintain. Figure 1 illustrates the interrelationships between a defined test case titled "Verification of login process with incorrect data," its implementation within an automated test using Selenium WebDriver, and the effect of the test, as observed on the user interface of the pet store. These elements are interconnected by arrows, allowing you to follow the flow from the test specification to the specific system behavior. The test case shown in the figure is for a login process using incorrect credentials. It includes the following steps: accessing the site (orange and red arrows), navigating to the login section (green arrow), entering invalid credentials (purple arrow) and attempting authentication (blue arrow) with the expected result of a login error message (pink arrow).

As part of the article, fifteen test cases were developed. Each test case includes a detailed description of the purpose of the test, the prerequisites necessary for its execution, the detailed specified test activities and expected results, which provides a comprehensive approach to verify the functionality of the system. As noted by Umar and Chen, the creation of detailed and comprehensive test cases is fundamental to the effectiveness of automated testing frameworks, which we have applied in our analysis of the online pet store [3]. In addition, each test case is appropriately titled:

1) Verification of the login process with correct data
2) Verification of the login process with incorrect data
3) Successful creation of a user account
4) Logging out
5) Adding a product to the shopping cart without logging in
6) Removing a product from the shopping cart without logging in
7) Searching for a product from the search bar

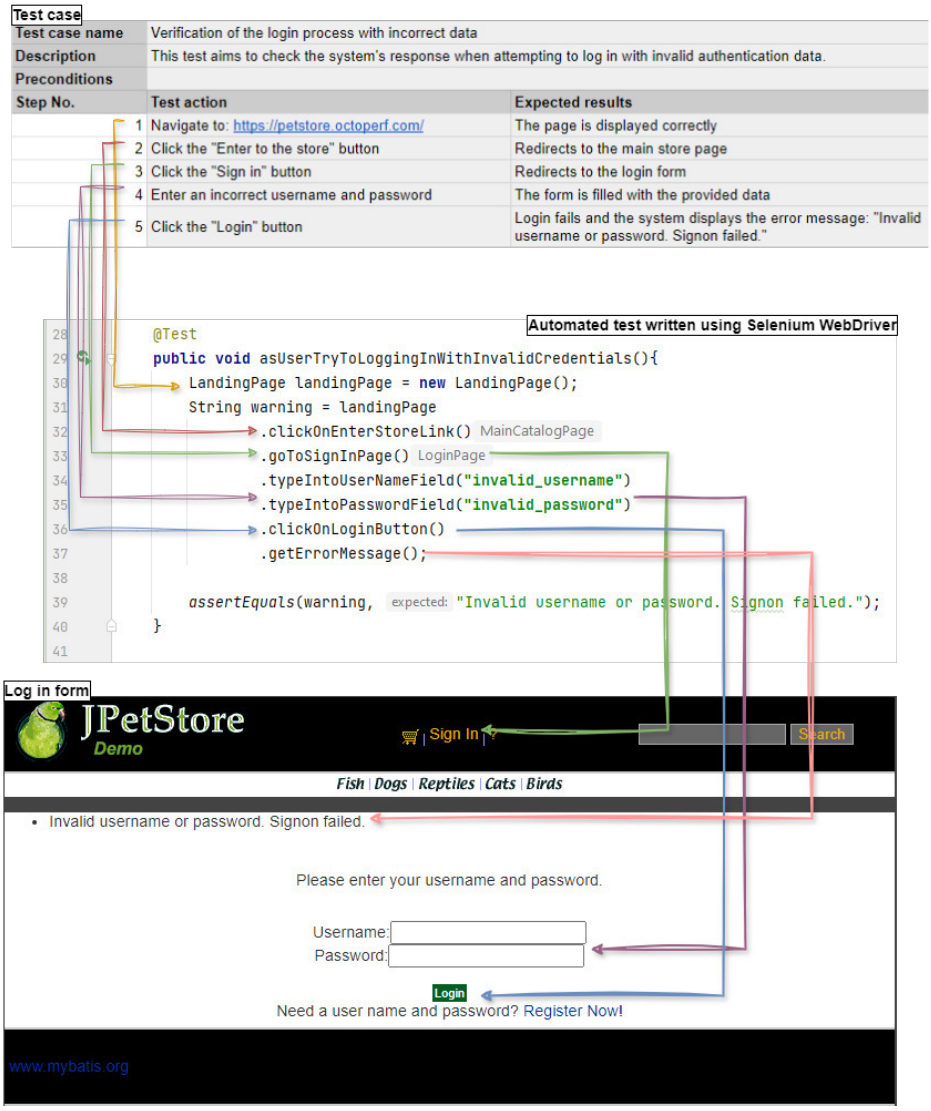

Fig. 1. Overview of the test case, automated test and login form view

8) Checking the display of the product image
9) Changing the quantity of a product in the shopping cart
10) Attempting to make a purchase without logging in
11) Making a purchase after logging in
12) Making a purchase with indicating a different delivery address
13) Return to the main menu from the category level
14) Check the display of detailed product information
15) Verifying the absence of errors with an empty search field

V. Test Execution Time

The research involved benchmarking the execution time of a set of 15 end-to-end tests using Selenium and Playwright tools. Automated tests were created based on test cases. The execution time of each test was measured directly in the Intellij IDEA environment — using the built-in functionalities of this IDE to monitor the duration of tests. The tests were executed on three major web browsers in the following versions: Google Chrome 120.0.6099.225 (Official Version) (64-bit), Microsoft Edge 121.0.2277.83 (Official Version) (64-bit) and Mozilla Firefox 122.0 (64-bit), in both standard (non-headless) and headless modes. The tests were performed on a computer running Windows 10 Home, equipped with an Intel(R) Core(TM) i5-6300HQ CPU @ 2.30GHz at 2.30GHz, with 16.0GB of RAM installed and a 64-bit, x64-based system. The overall analysis of the data shows shorter test execution times for the Playwright tool compared to Selenium in both modes. Detailed results for the Google Chrome browser in non-headless mode are shown in Figure 2, where we observe a reduction in execution time by Playwright, especially in test cases number 4, 11 and 12. Observations for the Microsoft

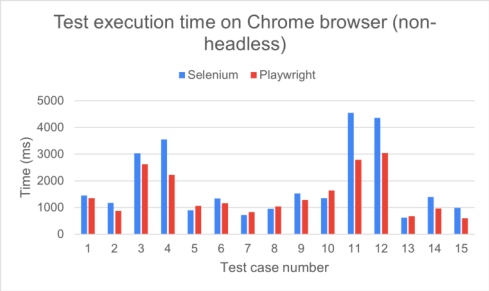

Fig. 2. Execution time of individual tests on Chrome browser (non-headless)

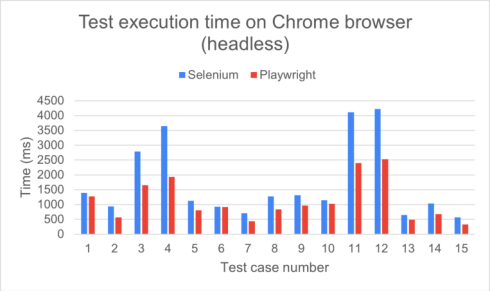

Fig. 5. Execution time of individual tests on Chrome browser (headless)

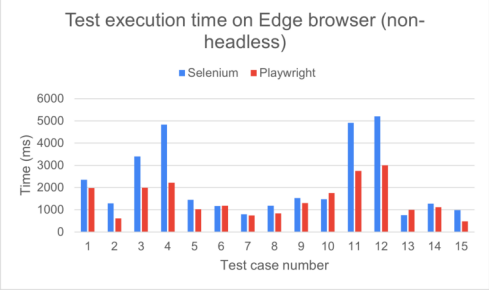

Fig. 3. Execution time of individual tests on Edge browser (non-headless)

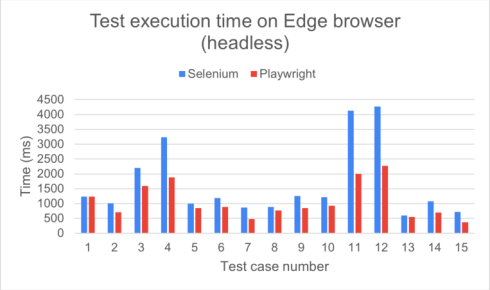

Fig. 6. Execution time of individual tests on Edge browser (headless)

Edge browser, illustrated in Figure 3, also suggest a shorter execution time for Playwright. An analysis of the results for the Mozilla Firefox browser, shown in Figure 4, also indicates an overall time advantage for Playwright over Selenium. However, test cases 1, 5, 6, 8, 9, 10 and 14 note that Selenium performs comparably or slightly better. The differences in test execution results between the tools indicate the importance of considering specific test conditions and scenarios when evaluating the performance of these tools.

In addition, analyzing the performance of tools in headless mode provides additional perspectives on their performance. As shown in Figure 5, for the Google Chrome browser in headless mode, Selenium showed longer test execution times, especially in test cases 3, 4, 11 and 12. Similar trends were observed for the Microsoft Edge browser, where Selenium also took longer to execute tests, as illustrated in Figure 6.

In contrast, Figure 7 illustrates the results for Mozilla Firefox in headless mode, which do not show an equally clear advantage for either tool. Although in some test cases, such as No. 6 and No. 9, it was Playwright that recorded longer execution times, in other cases the results of both tools are comparable. This indicates the need for a deeper analysis of the specific conditions under which the tools were tested, and potential optimizations in the test setup. These observations highlight that while Playwright generally shows better time performance in headless mode, the performance differences are dependent on the specific execution environment and can vary by browser and test scenario. Such variation in results underscores the importance of matching the test tool to the project's specifications, and shows that no tool is a one-size-fits-all solution.

Analyzing the overall execution times of all tests, we observe that Playwright performs better in both headless and standard (non-headless) modes for all tested browsers. The

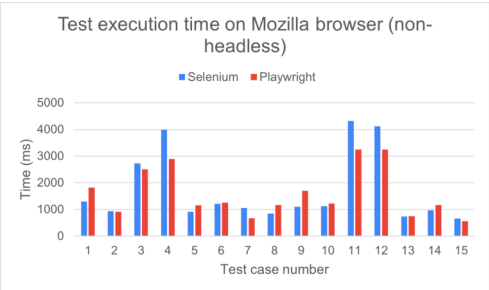

Fig. 4. Execution time of individual tests on Mozilla browser (non-headless)

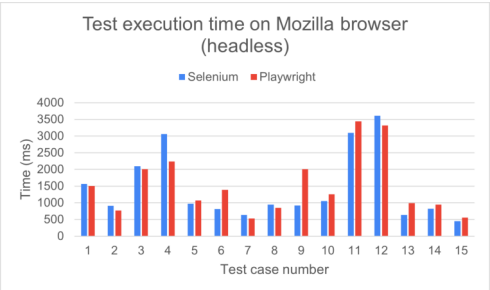

Fig. 7. Execution time of individual tests on Mozilla browser (headless)

Fig. 8. Execution time for all tests in non-headless mode

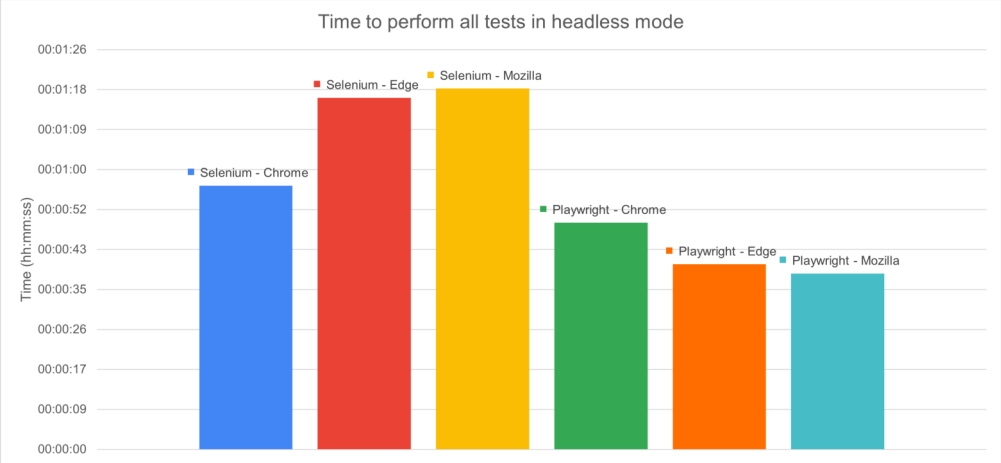

Fig. 9. Execution time for all tests in headless mode

average execution times, calculated from the three test calls, are shown in two graphs, each of which provides additional information about the effectiveness of the test tools.

In Figure 8, showing the results for non-headless mode, it can be seen that Playwright shows a significant advantage over Selenium. In particular, for the Google Chrome browser Playwright was 46% faster, for Microsoft Edge a 28% speedup was achieved, and for Mozilla Firefox a 19% speedup was achieved. These results demonstrate Playwright's superior performance in a typical browser environment, where user interactions are simulated in a full user interface.

Figure 9 illustrates the results for headless mode, where the differences in execution time are even more noticeable. In this scenario, for the Mozilla Firefox browser, Playwright was found to be faster by as much as 51% compared to Selenium, a significant time advantage of 40 seconds. For Microsoft Edge, the difference was 47%, while for Google Chrome Playwright was 14% faster. Such results in headless mode, where tests are performed without a graphical user interface, may indicate that Playwright is better optimized for performance and resource management.

The conclusions of the analysis indicate that the Playwright tool may offer better performance in terms of test execution time compared to Selenium, which may be important when choosing the right test automation tool for projects with limited time resources.

VI. Test Stability

The tests were run in two modes: headless and non-headless. In non-headless mode, all tests were successful in each of the ten run cycles. In headless mode, on the other hand, instability was observed in the case of Playwright, where test 8 (concerning the display of the product image) failed an average of four times, regardless of the browser used. It is worth noting that in the case of Selenium this problem did not occur, suggesting differences in how the two tools handle rendering of page elements.

An interesting aspect is that regardless of the tool and mode, there were cases where tests failed due to errors in the application itself, rather than the testing tool. For example, a

user registration test using random data revealed a problem in the handling of the ”Country” field, where entering a country name longer than 20 characters prevented registration. This indicates the value of automated testing in identifying application errors, regardless of the tool used.

The results show a negligible difference in test stability between Playwright and Selenium in headless mode. A possible reason for Playwright's problems could be the way the tool handles element rendering, which requires further research. At the same time, these observations underscore the importance of running tests in different configurations to better understand the limitations and potential problems associated with test automation tools.

VII. Ease of use and configuration

In terms of our experience with Selenium, our first interaction with Playwright resulted in some interesting insights. From a user's perspective, both tools presented ease of implementation; however, Playwright seemed to offer greater simplicity, especially in the aspect of browser configuration. A differentiating element was the release of the user from having to worry about browser version compatibility and the process of downloading drivers, which was important in the case of Selenium, particularly in its third version.

It should be noted, however, that these perceptions may have been shaped by the sequence of tool usage — with initial use of Selenium, followed by adaptation of existing tests to the Playwright environment. It is possible that the reimplementation of the tests ran with greater efficiency and fluidity. The initial configuration for both Selenium and Playwright is relatively similar, but there are significant differences, especially when considering different versions of Selenium. Selenium 3 requires downloading and configuring the appropriate drivers for each browser, in addition to one added dependency. Example 7.1 shows the configuration code for the Chrome browser. With Selenium 4, on the other hand, the process is somewhat simplified, as there is no need to manually download browser drivers. However, it is required to add an additional dependency — *WebDriverManager*. Nonetheless, the number of lines of code remains similar, as you can see in the example of 7.2.

Unlike Selenium, Playwright does not require adding additional dependencies or manually configuring drivers, which greatly simplifies the configuration process. Implementing the same goal in Playwright requires only one line of code as you can see in the example 7.3.

In summary, both Selenium and Playwright offer satisfying experiences for users with varying levels of experience. Personal observations suggest a preference toward Playwright's greater intuitiveness, primarily due to its simplified browser configuration. However, it should be stressed that these conclusions are subjective in nature and may be partially determined by the order in which these tools are used. An analysis of the number of lines of code needed to execute tests shows Playwright's advantage. For Selenium, the number of lines of code is about 2,300 for Selenium 3 and remains similar for Selenium 4, despite the ease of driver management. In comparison, Playwright requires only about 1,700 lines of code.

VIII. Community and support

In the digital age, where software development is happening at a rapid pace, community support and the availability of learning resources are becoming key factors in the selection of test automation tools. An analysis of the tools from the standpoint of documentation availability, size and activity of the community, support for new users, and training resources reveals significant differences between them.

Selenium, which has been around since 2004, has established itself as one of the most mature tools in the test automation field. Its documentation is not only extensive, but also regularly updated, with numerous examples to help users understand various aspects of the tool. This maturity is also reflected in the size and activity of the community, which is evident in forums, newsgroups and social media. Selenium is also a leader in educational resources, with numerous online courses, tutorials and webinars, reflecting its long presence in the market. On the Udemy platform, the number of courses on Selenium is 1,881, which far exceeds those dedicated to Playwright, of which there are 126 [15, 16].

On the other hand, Playwright, despite being a newer player on the market, has gained a rapidly growing community. Its documentation, while less extensive than Selenium's, is well organized and includes numerous examples. Playwright also stands out for its development activity, as evident in the number of commits on GitHub, surpassing those for Selenium in 2023 [9, 10]. Despite the smaller number of educational resources available, Playwright is rapidly gaining popularity, indicating its future potential.

In terms of online presence, a search for Selenium brings up significantly more results than Playwright, reflecting its long-standing presence and established position in the industry. Similarly, on Stack Overflow, the number of discussions and questions about Selenium far exceeds those devoted to Playwright.

IX. Extensibility and Integration

Selenium and Playwright offer significant integration capabilities with a variety of development tools and environments. Selenium, being a mature tool, is well-established among developer tools, offering integration with popular database management systems, reporting tools and version control systems. Playwright, while newer to the market, also demonstrates impressive integration capabilities, especially with modern development environments and frameworks. Both tools feature support for multiple programming languages. Selenium has traditionally supported languages such as Java, C#, Python, which contributes to its versatility. Playwright, on the other hand, initially focused on Node.js, has also extended its support to other languages, including Java and Python, which increases its appeal in development environments.

Example 7.1 (Java): Driver configuration in Selenium 3

```
System.setProperty("webdriver.chrome.driver", LocalWebDriverProperties.getChromeWebDriverLocation());
ChromeOptions options = new ChromeOptions();
options.addArguments("--remote-allow-origins=*");
```

Example 7.2 (Java): Driver configuration in Selenium 4

```
WebDriverManager.chromedriver().setup();
ChromeOptions options = new ChromeOptions();
options.addArguments("--remote-allow-origins=*");
```

Example 7.3 (Java): Driver configuration in Playwright

```
browser = pw.chromium().launch(new BrowserType.LaunchOptions().setHeadless(false));
```

Example 7.4 (Java): Adding the ability to record tests in Playwright

```
context = browser.newContext(new Browser.NewContextOptions().setRecordVideoDir(Paths.get("videos/")));
```

Playwright is distinguished by the simplicity of using built-in tools, such as screen recording, which is accomplished through short code snippets, which is represented by the example 7.4. This capability significantly simplifies the process of documenting tests and analyzing their progress.

Such solutions built into Playwright minimize the need to look for external tools and speed up the configuration process.

On the other hand, Selenium, although it may require the use of external tools for some advanced features such as screen recording, offers better extensive documentation, making it easier to integrate these tools. The availability of extensive documentation and a user community often makes adding and configuring external tools in Selenium more intuitive and less time-consuming. The choice between Selenium and Playwright should be dictated by the specific needs of the project and the preferences of the development team. Playwright offers simplicity and speed of configuration with its built-in tools, while Selenium provides greater flexibility in integrating external tools through better documentation and community support.

X. Conclusion

The present work aimed to thoroughly analyze and compare two end-to-end test automation tools, Selenium WebDriver and Playwright. In pursuit of this goal, a series of comparative tests were conducted, focusing on aspects such as test execution time, stability, ease of use, configuration and integration capabilities with other tools. Fifteen test scenarios were developed and implemented, which included test automation for an online pet store, available at https://petstore.octoperf.com/, using Java. These scenarios involved key store functionalities, such as logging in, logging out and placing orders, which allowed a deeper evaluation of the performance and effectiveness of Selenium WebDriver and Playwright tools under real-world usage conditions. The execution of these test tasks was an integral part of the research, allowing a direct comparison of the tools in question in terms of their suitability for end-to-end test automation.

According to the study, Playwright generally offers faster test execution times compared to Selenium, which is particularly evident in headless mode. This time advantage can be significant in projects where time constraints are crucial. As shown in Table I, Playwright completed the test suite in 2 minutes and 4 seconds, whereas Selenium took 2 minutes and 38 seconds. In terms of test stability, both tools demonstrated a high level of reliability, although Playwright tended to be unstable in specific scenarios in headless mode, as indicated by the one false negative test out of 15, while Selenium had none.

As for ease of use and configuration, Playwright seemed to offer a simpler approach, especially in terms of browser configuration. This is reflected in the number of lines of code required for the project, with Playwright needing 1700 lines compared to Selenium's 2300 lines, suggesting a more efficient and streamlined setup. However, Selenium, with its maturity and community support, maintains a strong position, offering rich educational resources and better documentation. In terms of integration with other systems, both tools present extensive capabilities, however Selenium stands out with better documentation and support, making it easier to use external libraries.

Based on the analysis, I recommend choosing a test automation tool based on the specific requirements of the project. Playwright may be preferred in projects where short test execution times and simple configuration are a priority, while Selenium will be a better choice in situations where rich community support, high configuration flexibility and the ability to integrate with external tools are key.

There are a number of opportunities for further development of this topic. Future research could focus on an extended comparison of the performance of the two tools in different environments and applications, including more complex testing scenarios. In addition, analysis of the long-term stability and scalability of these tools in large software projects could provide more valuable information. It's also worth exploring how developments in technologies such as artificial intelligence could affect the future of test automation, which could open up new perspectives in the field.

In conclusion, although both tools — Selenium WebDriver

TABLE I
COMPARISON OF SELENIUM WEBDRIVER AND PLAYWRIGHT BASED ON KEY CRITERIA

Criteria	Selenium WebDriver [14]	Playwright [12]
Browser compatibility	Chrome, Safari, Firefox, Edge	Chromium, Safari, Firefox, Edge
Programming language support	Java, JavaScript, Python, Ruby, C#	Java, Node.js, Python, .NET
Cost	Open source, free to use.	Open source, free to use.
Operating System compatibility	Windows, Linux, and macOS	Windows, Linux, and macOS
Total test execution time	2 Minutes 38 Seconds	2 Minutes 4 Seconds
False negative tests	0/15	1/15
Lines of code in project	2300	1700
Number of courses on Udemy [15, 16]	1881	126
Number of threads on StackOverflow [17, 18]	57696	3213

and Playwright — have their strengths and weaknesses, the choice between them should always be made taking into account the specific needs and limitations of the project. Umar and Chen in their study conclude that the success of automated testing projects heavily relies on the appropriate selection of testing tools and frameworks, a perspective that our comparative study of Selenium WebDriver and Playwright supports [3]. The research and analysis carried out provides valuable input to the evaluation of these tools, and the conclusions drawn from this work can be helpful in deciding on the appropriate test automation tool.

REFERENCES

[1] A. Holmes, M. Kellogg, *Automating functional tests using Selenium*, 2006, doi: 10.1109/AGILE.2006.19

[2] E. Pelivani and B, Cico *A comparative study of automation testing tools for web applications*, 2021 doi: 10.1109/MECO52532.2021.9460242.

[3] M. A. Umar, Z. Chen, *A study of automated software testing: Automation tools and frameworks*, International Journal of Computer Science Engineering 2019, doi: 10.5281/zenodo.3924795.

[4] M. Psujek, A. Radzik, G. Kozieł, *Comparative analysis of solutions used in automated testing of Internet applications*, Department of Computer Science, Lublin University of Technology, Lublin, Poland, 2021, doi: 10.35784/jcsi.2373.

[5] P. Ramya, V. Sindhura, P. V. Sagar, *Testing using selenium web driver*, 2017, doi: 10.1109/ICECCT.2017.8117878.

[6] R. Dahiya, A. Shahid, *Importance of manual and automation testing* Department of Information Technology, AGI Institute, Auckland, New Zealand, 2019, doi: 10.5121/csit.2019.91719.

[7] R. K. Lenka, U. Satapathy, M. Dey, *Comparative Analysis on Automated Testing of Web-based Application*, Department of CSE, 2018, doi: 10.1109/ICACCCN.2018.8748374.

[8] S. K. Alferidah, S. Ahmed *Automated Software Testing Tools*, International Conference on Computing and Information Technology, ICCIT 2020, doi: 10.1109/ICCIT-144147971.2020.9213735.

[9] https://github.com/microsoft/playwright/graphs/commit-activity (06.01.2024).

[10] https://github.com/SeleniumHQ/selenium/graphs/commit-activity (06.01.2024).

[11] https://maven.apache.org/

[12] https://playwright.dev/docs/intro

[13] https://testng.org/

[14] https://www.selenium.dev/documentation/

[15] https://www.udemy.com/courses/search/?src=ukw&q=Playwright (06.01.2024).

[16] https://www.udemy.com/courses/search/?src=ukw&q=Selenium (06.01.2024).

[17] https://stackoverflow.com/questions/tagged/selenium-webdriver?tab=Newest (10.07.2024).

[18] https://stackoverflow.com/questions/tagged/playwright (10.07.2024).