JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 36 (2025) 320–327
Received: 9 May 2025
Accepted: 15 June 2025

Comparative analysis of Cypress and Playwright frameworks in end-to-end testing for applications based on Angular

Analiza porównawcza frameworków Cypress i Playwright w testach end-to-end dla aplikacji opartych na Angularze

Przemysław Gosik*, Marek Miłosz

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
The aim of this article is to present the results of a comparative analysis of the Cypress and Playwright frameworks, focusing on their efficiency in the end-to-end testing of Angular-based web applications. At the outset, a hypothesis was formulated that Playwright surpasses Cypress in testing Angular-based applications due to lower memory consumption, faster execution of end-to-end tests, and better support for tests across multiple browsers. Subsequently, a study was conducted, which involved implementing defined test scenarios in both tools. The time and memory consumption of each automated test were measured. Additionally, the ease of use of the tools was evaluated by analysing the length of the code required for each test and the resulting file size. The experiment fully confirmed the formulated hypothesis.

Keywords: Cypress; Playwright; end-to-end testing; test automation

Streszczenie
Celem artykułu jest analiza porównawcza frameworków Cypress i Playwright pod kątem ich efektywności w testach end-to-end aplikacji webowych wykorzystujących szkielet Angular. Na początku postawiono tezę, że Playwright przewyższa Cypress w testowaniu aplikacji wykorzystujących szkielet Angular, dzięki niższemu zużyciu pamięci, szybszemu wykonaniu testów end-to-end oraz lepszemu wsparciu dla testów w wielu przeglądarkach. Następnie przeprowadzono badanie, które obejmowało implementację zdefiniowanych scenariuszy testowych w obu narzędziach. Zmierzono czas oraz zużycie pamięci podczas wykonywania każdego automatycznego testu. Dodatkowo oceniono wygodę korzystania z narzędzi, uwzględniając długość kodu oraz rozmiar poszczególnych testów. Przeprowadzony eksperyment w pełni potwierdził postawioną tezę.

Słowa kluczowe: Cypress; Playwright; testy end-to-end; automatyzacja testów

*Corresponding author
Email address: przemyslaw.gosik@pollub.edu.pl (P.Gosik)

1. Introduction

With the increasing complexity of web applications, ensuring their quality and stability has elevated the role of test automation in software engineering. The diversity of devices, browsers, and environments in which users interact with applications forces development teams to employ advanced testing methods. While traditional approaches, such as unit testing, remain a valuable component of quality assurance, their effectiveness can be limited, especially in the case of complex applications composed of numerous subsystems and interfaces. In such scenarios, alternative approaches like end-to-end (E2E) testing become essential. E2E testing allows for a comprehensive system verification from the end-user's perspective, enabling realistic simulation of real-world application use and identifying potential issues at the intersections of various system components [1].

Applications built using Angular are a particularly noteworthy case in this context, as Angular is one of the most popular front-end frameworks. Thanks to its popularity and the modular nature of Angular applications, teams face specific testing challenges and requirements that must be addressed when selecting automation tools. Therefore, this article presents a comparative analysis of two popular E2E test automation tools, Cypress and Playwright, focusing on their application in Angular-based projects. The analysis considers aspects such as support for different browsers, stability, test execution speed, resource consumption, and ease of implementation. The comparison aims to identify the tool best suited to the needs of modern web applications.

In order to thoroughly evaluate each tool, this study puts forward three main hypotheses:

H1. Playwright consumes less memory during testing compared to Cypress.
H2. Playwright is faster in executing E2E tests compared to Cypress.
H3. The simpler code structure in Playwright is expected to facilitate easier maintenance and implementation of new tests, while also providing greater stability in multi-browser environments compared to Cypress.

By examining these hypotheses, the study aims to highlight the unique strengths and limitations of each tool, ultimately helping teams select the most suitable solution for their Angular-based projects.

2. Literature review

This chapter provides a literature review of key topics relevant to the experimental study. It includes an overview of what E2E tests are and their role in quality assurance, a review of insightful comparative analyses of test automation tools, and a discussion of common challenges in web application testing. Additionally, it presents best practices for implementing reliable tests and explores approaches aimed at optimizing test performance and stability.

The review begins with a BrowserStack article [2] that explains what E2E testing is. It is described as simulating complete application flows using realistic user scenarios to verify system integration. Furthermore, the article presents tools like Selenium and Cypress along with best practices and challenges. The article [3] demonstrates the importance of E2E testing by showing that automating tests validates the entire software stack while overcoming manual testing drawbacks, significantly reducing time, errors, and costs.

There are several publications offering comparative analyses of test automation frameworks. One such study is a conference paper [4] that provides an in-depth comparison of four widely used tools: Selenium, Cypress, Puppeteer, and Playwright, by examining their architectures, strengths, weaknesses, and key features. In another study [5], the authors conduct a comparative analysis of both server-side and client-side testing. However, this research focuses solely on time measurements, leaving room for memory performance evaluation, a gap addressed in another article [6]. It is important to note that both analyses primarily target limited component testing. Additionally, the article [7] not only serves as a solid example of comparative analysis but also introduces an innovative method for assessing test case complexity based on non-comment lines of code (NLOC) and file size. Collectively, these studies underscore the need for a comprehensive comparative analysis of E2E tests and provide a solid foundation for establishing the evaluation criteria required for such research.

One of the key topics explored during the experiment preparations was the analysis of Angular-based applications. To gain a deeper understanding of Angular's specifications, a comparative review of the official Angular documentation [8] was undertaken. Developed and maintained by Google, this documentation is widely regarded as an indispensable resource for both novice and experienced developers due to its comprehensive coverage of modules, components, routing, form handling, and backend API integration. In addition, the case study "Web Development Using Angular: a Case Study" [9] demonstrates the framework's practical application in real-world scenarios. The study highlights its effectiveness in supporting both rapid prototyping and large-scale enterprise projects, despite challenges such as a steep learning curve and certain backend limitations. Moreover, the chapter "Angular Adventures: Crafting Smart User Interfaces" from the book "CodeMosaic" [10] offers insights into the innovative aspects of Angular. It showcases how the framework appeals to developers across all experience levels and underscores its significance in modern web development.

Testing web applications is fraught with challenges, as highlighted in "Web Application Testing: Challenges and Opportunities" [11]. This study emphasizes the difficulty of evaluating quality attributes-such as interoperability, reusability, transparency, maintainability, precision, and performance, that are crucial for application stability. It also points out that variations in the effectiveness and resource requirements of testing techniques complicate the selection of an optimal approach. In another paper, "Web Application Testing: a Systematic Literature Review," [12] the research is categorized into methods addressing the distributed, dynamic, and diverse nature of web systems. This review examines model-based testing approaches and fault taxonomies, covering issues like navigation errors and session management, to provide an overview of current methods and identify areas for further investigation.

Following best practices is essential when implementing each E2E test to ensure optimal results. The book by W. Mwaura [13] offers comprehensive guidance on the Cypress framework, covering everything from installation and debugging to developing advanced test cases. Similarly, the book by K. Pathak [14] not only details effective techniques for writing robust E2E tests, but also delves into Playwright's powerful features. It presents a variety of tools that enhance the professionalism and effectiveness of testing, while also introducing advanced techniques, such as building frameworks using the Page Object Model (POM), performing API testing, intercepting HTTP requests, and conducting accessibility and visual tests. Pathak also examines AI-driven test execution and professional tooling that further enhance test reliability and efficiency.

Another critical consideration is minimizing test execution time and resource consumption. Augusto [15] proposes reducing E2E test duration and cloud costs by grouping tests with similar resource requirements and sequencing them to avoid redundant deployments, complete with a simple cost model for selecting the most economical cloud configuration. In contrast, Fasolino and Tramontana [16] address test fragility by embedding lightweight, uniquely identifiable hooks into page templates, ensuring reliable element selection and significantly reducing both failures and maintenance overhead.

This literature review not only highlights the significance of the topic discussed in this article but also provides an essential foundation for designing the most effective experiment possible. The reviewed articles offer valuable guidance on selecting appropriate criteria for comparing web automation tools, as well as best practices for creating reliable and standards-compliant test scenarios. Additionally, insights from the literature contributed to improving the quality and reliability of the conducted experiment.

3. Presentation of the technologies under study

3.1. Cypress

Cypress [17], created by Cypress.io, a company founded by Brian Mann in 2015, is a modern testing tool built to simplify and streamline web application testing. Designed with ease of setup and integration in mind, it enables developers to write and execute tests directly within the browser, delivering fast, reliable results. Unlike traditional tools like Selenium, Cypress operates inside the browser environment, allowing real-time interaction with the application under test and providing advanced debugging capabilities without requiring additional servers or complex configurations. Its tight integration with the browser ensures tests are executed more naturally, closely mimicking user behaviour.

Cypress primarily supports Chromium-based browsers like Chrome and Edge, with partial support for others like Firefox and Safari. This focus on specific browsers makes it exceptionally fast and efficient, but introduces limitations for teams needing extensive cross-browser testing. Despite these constraints, Cypress has gained widespread adoption for its intuitive interface, real-time test monitoring, and robust support for modern JavaScript frameworks.

3.2. Playwright

Playwright, [18] is a versatile tool that supports a wide range of browsers, including Chrome, Firefox, and WebKit [19], making it especially flexible for cross-browser testing. Its architecture enables testing applications in diverse environments, both on desktop and mobile platforms, while also allowing the automation of complex user interactions. Playwright stands out due to its advanced debugging and error-logging capabilities, which make it an excellent choice for identifying and resolving issues in sophisticated web applications.

It was created by a team at Microsoft, including engineers who previously worked on Puppeteer, a similar tool developed at Google. Building on their expertise, they designed Playwright to offer broader browser support and additional functionality. Since its initial release in 2020, Playwright has gained popularity for its ability to manage multiple contexts, execute parallelized tasks efficiently, and provide a comprehensive toolkit for web automation and testing.

4. Research methodology

4.1. Description of the research stand

To minimize the risk of disruptions, such as server delays, the experiment was conducted locally. The Table 1 presents the parameters of the device used.

Table 1: Research environment parameters

Disc	SSD 512 GB
Operating System	Windows 10 64-bit
Processor	Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz
RAM	8 GB

4.2. Test application

The study was conducted using application named**“**Web application for learning fast reading, writing and memorizing**”,** which was presented by authors in their thesis [20], which offers a comprehensive user history and allows testing various features of Angular technology. It was selected for its rich scenario coverage, which enables the creation and execution of comprehensive end-to-end test cases and, consequently, a robust comparison of Cypress and Playwright**.** The Figure 1 shows a screenshot of the main view of the application.

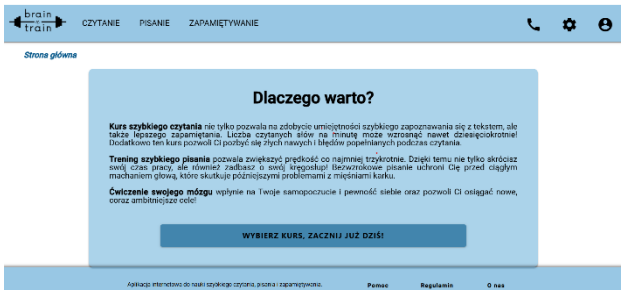

Figure 1: Main screen of the test application.

The web application designed for improving skills in fast reading, writing, and memorization is built to cater to diverse user needs and skill levels. Its interactive features are organized into three distinct modules, each containing multiple exercises targeting a specific skill area. Many of these exercises support result saving, enabling users to monitor and track their progress over time.

In the fast reading module, users can enhance their visual span and processing abilities through exercises like the "Schultz Tables" [21] and reaction-based tasks such as identifying words and numbers. These exercises are complemented by performance statistics, providing users with real-time feedback on their progress.

The typing module develops the user’s proficiency in touch typing through structured lessons. These lessons include exercises on specific key groups and full-text typing tasks, reinforced with gamified elements such as points and progress tracking. Detailed statistics after each session-errors, speed, and completion time-allow for targeted improvement.

The memory training module employs mnemonic techniques, offering games and exercises like chain associations and image-based tasks to aid in memorizing sequences. Users can adapt the difficulty level to their preferences, ensuring a personalized learning curve.

Additionally, the application includes robust user account management features, enabling registration, data editing, and progress tracking. a dedicated admin panel allows issue management and communication with users, ensuring seamless maintenance. Accessibility is a key focus, with customizable themes and text sizes catering to users with disabilities. All functionalities are integrated with an intuitive interface, supported by backend service for data processing and synchronization. This comprehensive approach ensures that the application serves as a versatile tool for skill development.

Additionally, to run the application and prepare the tests, it was necessary to set up a local database. For this purpose, the MySQL tool version 8.0 was used. The specification of the remaining parameters of the application, including both the frontend and server-side, are presented in the Table 2.

Table 2: Test application specification

Tool	Version
Node.js	16.13.0
npm	8.1.0
Angular	14.0.0
CLI Angular	14.2.0
TypeScript	4.7.2
Java	17
Spring Boot	2.7.3
Apache Maven	3.5.1

4.3. Evaluation criteria

The expected outcomes of the experiment aimed to assess the efficiency, stability, and usability of the Cypress and Playwright frameworks in testing Angular applications. The study focused on these key aspects:

- **Ease of implementation**: The analysis focused on the simplicity and efficiency of implementing tests in both frameworks. To assess code complexity, two specific metrics were applied: the number of non-commented lines of code (NLOC) and test file sizes. The findings showed which tool enabled developers to implement E2E tests more effectively and with fewer complications.
- **Test execution time**: The experiment compared the speed of both tools in running E2E tests. The results determined which framework was more efficient in terms of execution time, a critical factor for projects where rapid testing cycles were essential.
- **Test stability**: The stability of test results was assessed by analyzing variations in the outcome distributions across multiple runs. Since false negatives should be rare in a controlled local environment, stability was mainly assessed by measuring the dispersion of key performance metrics, including execution time, memory usage and CPU utilization across multiple runs.
- **Cross-browser support**: The ability of both tools to execute tests across various browsers was evaluated. This included their performance in Chrome, Firefox, and WebKit. The results indicated which tool was better suited for ensuring compatibility in multi-browser environments.
- **System resource usage**: The experiment compared the consumption of system resources, such as RAM usage and CPU load, during test execution. The findings identified the tool that performed more efficiently in resource-constrained environments, offering an advantage for teams operating in limited hardware settings.

4.4. Tests scenarios

The research scenarios were designed to enable a comprehensive evaluation of the Cypress and Playwright frameworks in the context of testing Angular-based web applications. The key scenarios considered various aspects of application performance, such as form handling, dynamic interactions, backend integration, and multitasking. The tests were also selected to reflect the most important functionalities of the test application.

Test 1: User registration

The first scenario involved testing user registration and its validation, including the following steps:

1. Open the registration page.
2. Fill out the form with invalid data and verify that appropriate error messages are displayed.
3. Fill out the form with valid data (username, password, email).
4. Verify that now the “Register” button is enabled
5. Click the "Register" button.
6. Verify that a success message is displayed.
7. Enter the verification code and complete the registration process.
8. Log out and log back in to ensure the registration process was successful.

Test 2: Typing course

The next scenario involved testing the typing course. The test steps included:

1. Open the application and log into your user account.
2. Navigate to the typing course module.
3. Select a difficulty level and a specific lesson.
4. Start the exercise
5. For the first four inputs, type the displayed text, entering correct characters.
6. Pause the game
7. Check if time counting has stopped
8. Verify that the displayed statistics much expected results
9. Click button to save the attempt
10. Check if the course progress bar is updated to reflect the completion of the lesson.

Test 3: Schulz tables

The tests also included the educational game "Schulz Tables." The test steps included:

1. Open the application and log into your user account.
2. Navigate to the games module and select "Schulz Tables."
3. Choose medium level of difficulty
4. Verify that game shows expected number of fields for this level
5. Verify that statistics of gained points and time are visible
6. Start the game
7. Click the correct number and verify point has been added
8. Click the wrong number and verify point has been abducted
9. Complete the game and check the displayed statistics (completion time and points)
10. Save results

Test 4: Number sorting

The objective of this exercise in the application was to strengthen the user's memory by presenting numbers one at a time for memorization, and then requiring the user to reorder the entire sequence at once. This scenario tested the stability of the drag-and-drop mechanism. The test steps included:

1. Open the application and navigate to the games module without logging into a user account.
2. Select the game "Number Sorting."
3. Select the easiest level of difficulty
4. Start the game
5. Verify that all displayed numbers are rendered and visible correctly.
6. Verify that the drag and drop mechanism functions correctly by rearranging one of the numbers
7. Complete the game
8. Verify that there are displayed statistics, such as the number of correctly sorted numbers and the completion time.

Test 5: Issue management

The final scenario involved issue management by an administrator, including the following steps:

1. Open the application and log in as a regular user.
2. Navigate to the issue reporting form.
3. Fill out the form, providing a topic and description of the issue.
4. Submit the issue and verify the confirmation message is displayed.
5. Log in as an administrator.
6. Navigate to the issue management panel
7. Open the issue
8. Verify that details of the issue contains correct data

The adopted research methodology involves thorough and multifaceted testing of the application under various conditions to ensure reliable and meaningful results. Each test was executed in a headless mode with the appropriate browser option enabled. Running tests in headless mode allows them to be executed without a graphical user interface, optimizing performance by reducing visual overhead. This approach ensures that tests can focus purely on backend interactions and overall application behaviour without the influence of rendering issues.

Each test scenario was repeated 50 times in each browser environment, allowing for a robust analysis of test stability and performance. Repeated execution not only helps detect potential inconsistencies or anomalies in test results but also provides a solid dataset for evaluating the reliability and efficiency of the testing tools. This repetition yielded the following total test executions:

- 50 runs in Chrome,
- 50 runs in Firefox,
- 50 runs in WebKit,
- 150 runs per scenario for each framework,
- 300 runs per scenario in total for both frameworks,
- 1500 measurement across all scenarios.

Additionally, the ease of implementing test scenarios was assessed by comparing the amount of code required to write each test. This comparison focused on the number of lines of code necessary for test implementation, providing insights into the simplicity and efficiency of the tools from a development perspective.

5. Research preperation

The implementation process required small modifications to the application to facilitate resetting test data before each test execution. This was achieved using the Flyway database migration tool [22], which allowed the management of database schema and data resets in a structured way. a dedicated endpoint was created in the application to enable the reset of test data on demand, ensuring consistent and clean test conditions before every run. This approach was essential for maintaining the integrity of test results and avoiding interference from leftover data between test executions.

Implementation began with defining robust locator strategies using the data-test HTML attribute. During configuration, Playwright provided a wide range of options, including configurable retry counts for tests that failed and automatic screenshot capture when errors occur. This capability makes Playwright highly adaptable to a variety of project requirements. In addition, Cypress requires each target browser to be installed on the host operating system for testing across multiple browsers, whereas Playwright bundles browser binaries by default, simplifying setup.

During test development, Playwright provided a broader set of built in methods that simplified implementation. For instance, tests involving drag and drop interactions or text input operations required significantly fewer lines of code. Additionally, the ability to use multiple browser contexts enabled more effective execution of issue management scenario, which involved two actors. In contrast, Cypress often required manual insertion of wait periods to avoid timing errors, a step that was unnecessary when using Playwright.

To streamline the test execution process, a dedicated Python script was developed. Its primary function was to automate the repetitive execution of tests and systematically gather pertinent data. The Python library psutil [23] was utilized, as it provides methods for monitoring and collecting detailed information about system processes. For each test execution, the following data points were collected:

- The outcome of the test, including detailed error messages and causes in the event of failure.
- The elapsed execution time, measured in seconds from the initiation to the termination of the test process.
- The average CPU usage percentage attributed specifically to the test execution process.
- The total memory consumption of the test process, quantified in megabytes (MB).

The experimental runs were conducted in a controlled environment, free from extraneous software execution, to mitigate external influences and ensure result accuracy. The collected data from each test iteration were saved in CSV format, facilitating subsequent analysis

through an additional Python script designed explicitly for data processing. Its role was to remove outliers and preprocess the data for more straightforward interpretation and effective graphical presentation.

6. Research results

The collected data was compiled and then processed into values that allowed for the verification of the hypotheses posed at the beginning of the study. The presentation and discussion of the results are structured according to the previously defined evaluation criteria.

6.1. Ease of implementation

Initially, implementation complexity was assessed by quantifying NLOC in each test file. As shown in the Table 3, Playwright consistently exhibited a smaller NLOC metric than Cypress across all scenarios. The only deviation occurred in the final test case, where Playwright's use of dual browser contexts produced a marginal increase in NLOC, this trade-off nevertheless yielded superior execution speed and reduced resource utilization in later presented results.

Table 3: Comparison of NLOC per test file

	Playwright	Cypress
Test 1	33	44
Test 2	50	58
Test 3	48	52
Test 4	45	61
Test 5	43	35

To provide additional clarity, since NLOC alone may not fully reflect the complexity or conciseness of test implementations, the actual file sizes of the test scripts were also compared. As shown in Table 4, Playwright test files were smaller in every scenario except for Test 5. This result aligns with the earlier NLOC findings and further supports the observation that Playwright generally allows for more compact and potentially more maintainable test implementations.

Table 4: Comparison of test file sizes

	Playwright	Cypress
Test 1	2.17 kB	2.59 kB
Test 2	3.2 kB	3.4 kB
Test 3	2.98 kB	3.3 kB
Test 4	2.97 kB	3.33 kB
Test 5	2.56 kB	2.89 kB

6.2. Test execution time

After comparing the code complexity, the focus shifted to execution times. The Figure 2 presents the average execution duration for each test scenario, clearly showing that Playwright consistently completed tests faster than Cypress. This performance advantage was observed across all five scenarios, with Playwright's execution times significantly lower in each case.

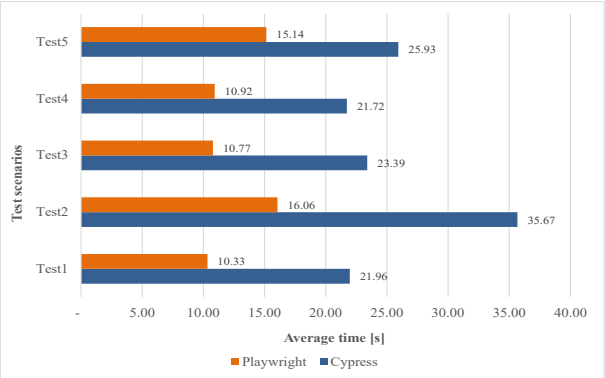

Figure 2: Comparison of Average Times for Test scenarios.

6.3. Test stability

To assess the stability of test execution times, a box plot was prepared and is presented in the Figure 3. The visualization clearly illustrates that Playwright demonstrated significantly lower variability between test runs. In contrast, Cypress showed a broader distribution of execution times, indicating less consistent performance and greater sensitivity to runtime conditions.

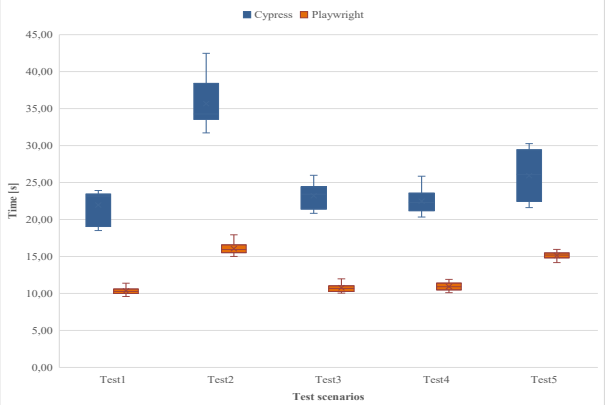

Figure 3: Comparison of tests execution times across multiple iterations.

6.4. Cross-browser support

Expanding the analysis to browser-specific performance, the next Figures 4-8 presents results of comparing average execution times across examined browsers for each test scenario. The results show that Playwright maintained consistent performance across all browsers, with much smaller differences in execution times.

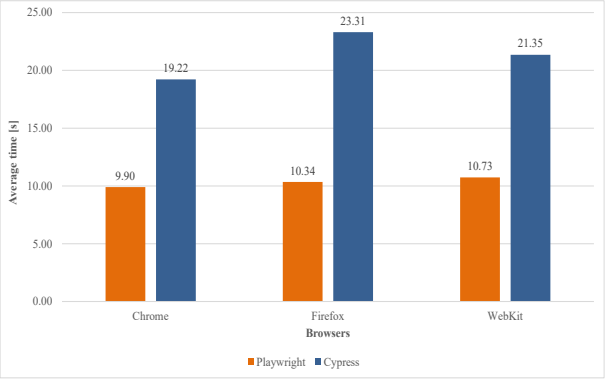

Figure 4: Average time for each browser for the first test scenario.

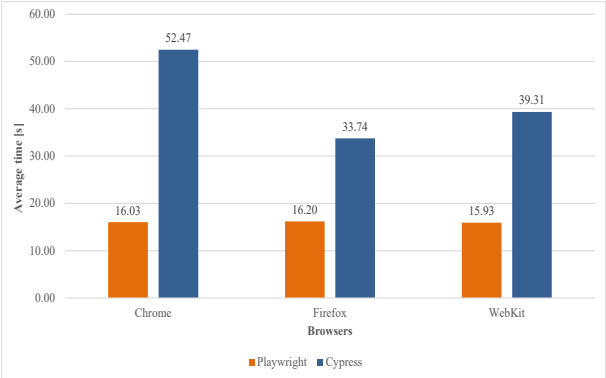

Figure 5: Average time for each browser for the second test scenario.

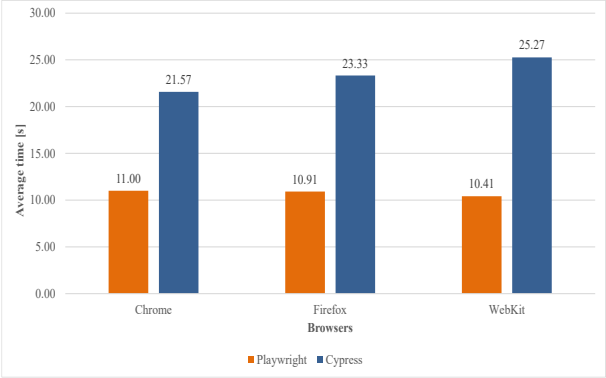

Figure 6: Average time for each browser for the third test scenario.

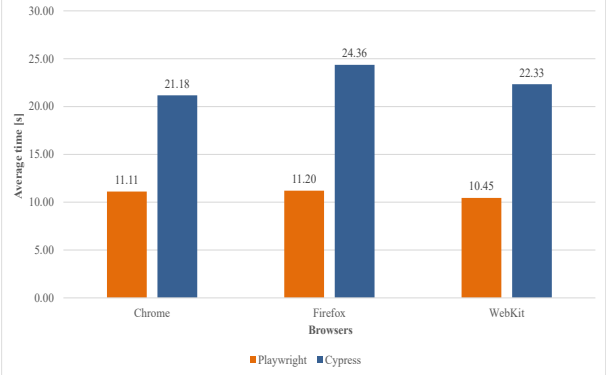

Figure 7: Average time for each browser for fourth test scenario.

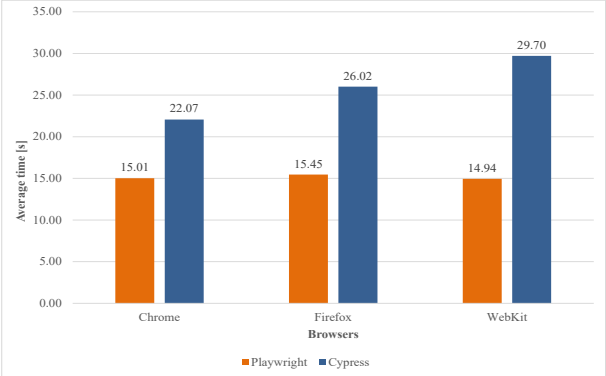

Figure 8: Average time for each browser for the fifth test scenario.

6.5. System resource usage

The next stage involved evaluating resource consumption, focusing on metrics such as RAM usage and CPU load during test execution. The Figure 9 illustrates CPU usage during test execution, showing that Cypress consumed less disc space compared to Playwright.

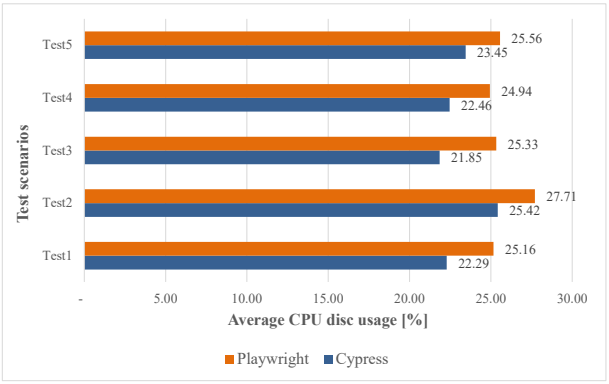

Figure 9: Comparison of Average CPU consumption for Test Cases.

The Figure 10 depicts RAM consumption during test execution, revealing that Playwright required much less memory than Cypress.

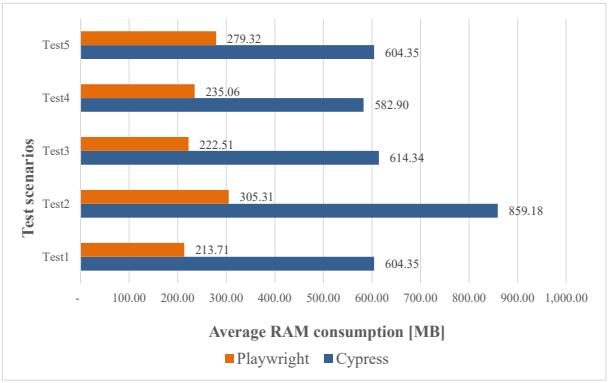

Figure 10: Comparison of Average RAM consumption for Test Cases.

7. Conclusions

In conclusion, the study unequivocally confirmed all hypotheses. Playwright demonstrated superiority in several key aspects, making it a more effective tool for E2E testing in Angular applications. Tests implemented with Playwright were not only shorter and easier to maintain, thanks to its built-in features, but also faster and more stable, as confirmed by the analysis of execution time distributions.

In multi-browser environments, Playwright once again proved to be better, showing minimal differences in results across browsers. Cypress, on the other hand, exhibited greater discrepancies, indicating its lower versatility in such scenarios.

Additionally, the analysis of system resource consumption revealed that although Playwright exhibited marginally higher CPU utilization, its markedly lower RAM footprint across all test scenarios highlights its suitability for memory-constrained environments.

The findings presented throughout the study confirm the validity of all three research hypotheses. The first hypothesis, concerning Playwright's memory efficiency, was supported by consistent evidence of lower RAM consumption across all tests. The second hypothesis,

which predicted faster execution times, was also confirmed, as Playwright clearly outperformed Cypress in every scenario. Finally, the third hypothesis, which assumed that Playwright's simpler code structure would result in easier implementation and greater cross-browser stability, was validated by both the NLOC and file size comparisons, as well as by the consistent results observed across all tested browsers.

Admittedly, the extensive configuration surface and rich API offered by Playwright may at first seem formidable, however, this very versatility makes it well suited to a wide spectrum of projects. This comparative analysis has demonstrated Playwright is the preferred framework for comprehensive E2E testing of Angular applications.

References

[1] W. T. Tsai, X. Bai, R. Paul, W. Shao, V. Agarwal, End-to-end integration testing design, In 25th Annual International Computer Software and Applications Conference (COMPSAC) (2001) 166–171, https://doi.org/10.1109/CMPSAC.2001.960613.

[2] BrowserStack - practical guide to E2E testing. End to End testing: Tools, Types and Best Practices, https://www.browserstack.com/guide/end-to-end-testing, [24.11.2024].

[3] R. Soundarya, B. K. Srinivas, Automation of End-to-End Testing and Their Importance, Int. J. Res. Appl. Sci. Eng. Technol. 10(7) (2022) 4280–4283, https://doi.org/10.22214/ijraset.2022.45940.

[4] B. García, J. M. del Alamo, M. Leotta, F. Ricca, Exploring Browser Automation: a Comparative Study of Selenium, Cypress, Puppeteer, and Playwright, Proceedings of Quality of Information and Communications Technology (QUATIC 2024), Communications in Computer and Information Science 2178 (2024) 142–149, https://doi.org/10.1007/978-3-031-70245-7_10.

[5] F. Wąsik, M. Pojęta, M. Plechawska-Wójcik, Comparative analysis of selected tools for automation testing of web applications, J. Comput. Sci. Inst. 28 (2023) 229–235, https://doi.org/10.35784/jcsi.3689.

[6] P. Pasławski, M. Pańczyk, Comparison of selected tools for automation testing of web applications, J. Comput. Sci. Inst. 31 (2024) 145–150, https://doi.org/10.35784/jcsi.6238.

[7] M. Kuutila, M. Mäntylä, P. Raulamo-Jurvanen, Benchmarking web-testing: Selenium versus Watir and the choice of programming language and browser, arXiv Preprint (2016) arXiv:1611.00578, https://doi.org/10.48550/arXiv.1611.00578.

[8] Angular documentation. Home – Angular, https://angular.dev, [24.11.2024].

[9] A. K. Sahani, P. Singh, Web development using Angular: a case study, J. Inf. Electr. Electron. Eng. 1(2) (2020) 1–7, http://doi.org/10.54060/JIEEE/001.02.005.

[10] A. Dwivedi, Angular adventures: crafting smart user interfaces, In CodeMosaic, Apress Berkeley CA (2024) 147-174, https://doi.org/10.1007/979-8-8688-0276-8_5.

[11] S. Balsam, D. Mishra, Web application testing—challenges and opportunities, J. Syst. Softw. 219 (2025) 112186, https://doi.org/10.1016/j.jss.2024.112186.

[12] S. Doğan, A. Betin-Can, V. Garousi, Web application testing: a systematic literature review, J. Syst. Softw. 91 (2014) 174–201, https://doi.org/10.1016/j.jss.2014.01.010.

[13] W. Mwaura, End-to-End Web Testing with Cypress: explore techniques for automated frontend web testing with Cypress and JavaScript, Packt Publishing, Birmingham, 2021.

[14] K. Pathak, Web automation testing using Playwright, BPB Publications, New Delhi, 2025.

[15] C. Augusto, Toward an efficient end-to-end test suite execution, In 2023 IEEE 34th International Symposium on Software Reliability Engineering Workshops (2023) 26–29, https://doi.org/10.1109/ISSREW60843.2023.00038.

[16] A. R. Fasolino, P. Tramontana, Towards the generation of robust E2E test cases in template-based web applications, In 2022 48th Euromicro Conference on Software Engineering and Advanced Applications (2022) 104–111, https://doi.org/10.1109/SEAA56994.2022.00024.

[17] Cypress documentation. Why Cypress?, https://docs.cypress.io/guides/overview/why-cypress, [19.10.2024].

[18] Playwright documentation. Fast and reliable end-to-end testing for modern web apps, https://playwright.dev, [19.10.2024].

[19] WebKit project site. The WebKit open source project, https://webkit.org, [19.10.2024].

[20] P. Gosik, D. Grzejszczyk, Web application for learning fast reading, writing and memorizing, Bachelor thesis, Lublin University of Technology, Lublin, 2023.

[21] Schultz Table game example. Schultz Table, https://schultetable.web.app, [30.12.2024].

[22] Flyway documentation. Redgate Flyway documentation, https://documentation.red-gate.com/fd/redgate-flyway-documentation-138346877.html, [30.12.2024].

[23] Python package documentation. psutil, https://pypi.org/project/psutil/, [30.12.2024].