International Conference on Science, Engineering & Management Trends

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

A Framework for Accelerated Migration of Legacy Selenium Test Suites to Playwright with Automated Script Refactoring and Quantitative ROI Evaluation

Amit Kumar Gupta, Guide: Dr. Dharmbir Yadav (HoD, CSE Dept.)
College: Global Institute of Technology and Management, Gurugram, Haryana, India
University: Gurugram University, Haryana, India

Abstract- Testing of web application is a very important step in the software development life cycle. Selenium has been the conventional automation tool of browsing years. But with the increasing complexity of web technologies, older Selenium test suites tend to be slow, prone to timeouts and with a high maintenance cost. Contemporary solutions such as Playwright are more reliable and parallel, and faster. In spite of these benefits, companies are slow to upgrade since the thousands of legacy Selenium code would be impossible to re-write by hand, which would cost them an enormous amount of time and company resources. To address this bottleneck, this paper presents a viable scheme of the automated conversion of outdated Selenium examination suites into Playwright. This proposed system is based on a refactoring engine which, given a script, automatically converts Selenium-specific elements like locators, explicit waits, and browser actions to the equivalent Playwright elements. We evaluated this framework by mapping a sample collection of conventional web application tests. The original Selenium scripts as well as the generated Playwright scripts have been run in the same environments. We monitored the execution time, memory usage and script stability. Also, we have performed a quantitative Return on investment (ROI) analysis to convert these technical measurements into real business value. We estimated the specific savings of cloud server expenses by accelerating the speed of the tests and the developer time by removing the manual translation process. The findings indicate that there is a severe decrease in the total test run time and the amount of engineering effort needed to implement the migration. Finally, this study offers software teams with a very useful, cost-efficient design of the updated testing infrastructure which would save them a lot of money on long-term maintenance and would help to deliver software faster.

Keywords: Software Testing, Test Automation, Code Migration, Selenium, Playwright, Return on Investment.

I. INTRODUCTION

In constructing reliable web applications, software testing is a required process. Selenium has been the preferred option in automation of browser interactions over a decade. It has assisted the engineering

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

teams to test their websites automatically rather than manually testing them in slowness. But the web applications in the modern world are becoming dynamic and complicated. In this high paced world, conventional Selenium test suites are beginning to demonstrate some weaknesses. They often experience slowness in execution, excessive system memory consumption and unreliable test execution because of delays in page loading.

In order to address these performance bottlenecks, the software industry is in full swing to switch to more recent automation tools such as Playwright. Playwright offers in- built auto-waiting, parallel execution and much less memory footprint. Migration process is, however, the major challenge facing companies. Companies can have thousands of old Selenium test scripts that were written over decades. It is extremely impractical to require developers to read, translate, and re-write each and every line of old test code by hand into Playwright. It burns hundreds of costly engineering hours and is highly retarding to new features development.

The proposed solution to this company-wide issue, the automated migration framework, is presented in this paper. Rather than rewriting by hand, we suggest a script refactoring engine, which would read old Selenium code and transform it into optimized Playwright syntax. This tool converts web locators, explicit waits, and browser actions systematically and the resultant code is clean and functionally equivalent.

Moreover, this paper does not limit itself to fundamental code conversion but a quantitative Return on Investment (ROI) analysis. With the execution of the legacy and migrated test suites under the same conditions, we are recording the actual differences in the execution time and the system memory consumption. This performance data demonstrates that an automated migration strategy does not only save colossal initial developer time, but also permanently lowers the cost of cloud servers by executing tests faster and more efficiently.

II. LITERATURE REVIEW

The shift to the modern tools of automation of the tests instead of the traditional frameworks has been discussed extensively in the recent literature. There have been a number of studies that have done comparisons between Selenium and more recent frameworks such as Playwright in terms of performance, reliability, and run time.

Almabruk et al. tested Selenium and Playwright reliability by stressing on uptime and the Rate of Occurrence of Failures (ROCOF) with controlled hardware. Their results showed that although Selenium gives consistent uptime, Playwright is much faster in executing tests and has a more predictable response time in dynamic web applications. On the same note, Tymoshchuk analyzed the development of test automation through comparison of Selenium, Playwright, and Cypress. The paper had decided that Playwright has better performance and reduced execution time on dynamic web applications, and it is enough to make the industry switch to the traditional Selenium-based methodology.

These assertions are supported by further performance evaluations. A comparison was made in terms of execution time and stability of end-to-end test cases developed on the Page Object Model (POM) architecture by an analysis published in the Annals of Computer Science and Information Systems. The findings validated that Playwright always takes a shorter time to execute the tests as compared to Selenium WebDriver both in a headless and non-headless mode. Moreover, an IEEE case study of an employee management application published in 2024 showed that time to complete tasks dropped by a significant margin in the number of minutes to seconds after moving the manual processing system to automated scripts, highlighting the effectiveness of the current automation tools such as Playwright.
Research Gap

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

Although the current literature is comprehensive in demonstrating that Playwright is quicker and more effective than Selenium, the research works are mainly aimed at comparing the performance of the already written test scripts. The operation difficulty of migrating existing legacy codebases is not mentioned in any of the reviewed papers. The literature has a distinct deficit in the domain of automated script refactoring frameworks and quantitative Return on Investment (ROI) of the very process of the migration. The gap in this paper is that a practical migration tool is provided and the business value of the automated conversion is measured.

III. PROPOSED METHODOLOGY

In order to curb the difficulties of manual code migration, this study suggests automated refactoring framework. The fundamental principle of this methodology is to convert the old Selenium test scripts systematically into new Playwright scripts and retain the original Page Object Model (POM) architecture and test logic.

A. Framework Architecture

The proposed migration tool is developed as a command-line script engine (written in Python) that scans the old Selenium framework written in Java. The migration pipeline comprises three consecutive stages namely:

- Code Ingestion: The engine will scan the source directory and all Page classes, Test classes, and Base configuration files in the POM structure.
- Pattern Recognition and Mapping: The engine will be scanning the source code line-by-line. It relies on regular expressions (Regex) and string matching to find Selenium-specific imports, WebDriver commands, explicit waits, and locators of elements.
- Code Generation: After identification, the engine will transform the legacy syntax to the corresponding Playwright syntax using a fixed set of mapping rules. Then it generates the refactored files into a new directory structure and leaves the original files untouched.

B. Automated Refactoring Engine and Mapping Rules

The translation logic is the most important part of the refactoring engine. Playwright differs with Selenium, such as Playwright does not expect that a WebDriver instance should be used and waiting mechanisms are managed automatically. Thus, the engine does not simply carry out a simple text replacement; it rearranges the instructions.

Action / Element	**Legacy Selenium Syntax**	**Modern Playwright Syntax**
Browser Initialization	WebDriver driver = new ChromeDriver();	Browser browser = playwright.chromium().launch();
Page Navigation	driver.get("https://example.com");	page.navigate("https://example.com");
Locating Elements	driver.findElement(By.xpath("//div"));	page.locator("xpath=//div");

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

(XPath)		
Click Action	element.click();	page.locator("...").click();
Typing Text	element.sendKeys("text");	page.locator("...").fill("text");
Waiting Strategy	Thread.sleep(2000); / WebDriverWait	Auto-waiting by default (No explicit code needed)
Getting Text	element.getText();	page.locator("...").textContent ();

Table 1: Syntax Mapping and Refactoring Rules from Legacy Selenium to Playwright

The engine has managed to eliminate the massive Selenium dependencies with these rules and has replaced them with the lightweight and asynchronous syntax of Playwright.

C. Execution and ROI Evaluation Strategy

Once the automated conversion has been done, the methodology will involve a strict evaluation phase to be used to gauge the success of the migration. The legacy Selenium suite and the new Playwright suite are being run on the same web application (the-internet.herokuapp.com).

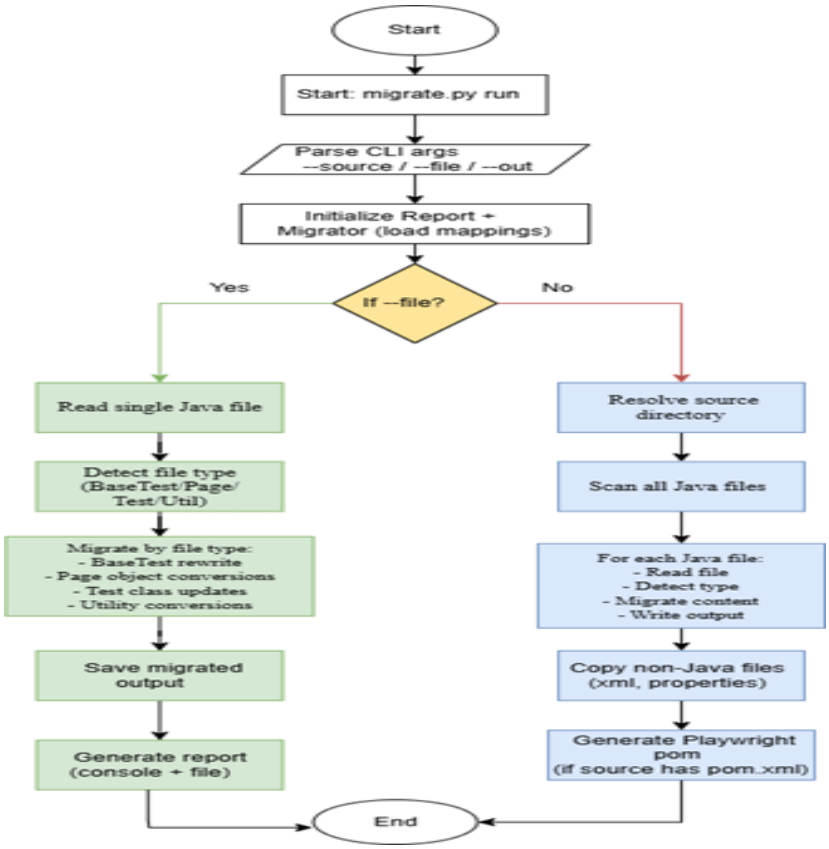

Figure 1: Architectural flowchart of the automated script refactoring engine.

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

To evaluate the speed improvements, the evaluation framework captures the following measurements:

- Total Suite Execution Time: The evaluation framework will record the following measurements to calculate the quantitative Return on Investment (ROI).
- Method-Level Executive Time: To determine what individual actions (such as clicking or responding to JavaScript alerts) were accelerated.
- System Memory and JVM Heap Usage: To gauge the server resource footprint before and after migration.

When these metrics are compared, we can determine the precise savings in the server costs and time saved with the help of the automated migration process, which will prove the business value (ROI) of the framework.

IV. EXPERIMENTAL SETUP

In order to have a fair and scientifically valid comparison between the old Selenium framework and the recently migrated Playwright framework, the test suites were run under the same hardware and software conditions. The experiment aimed at monitoring the speed of execution, system memory usage, and JVM heap usage without external network and hardware bias.

A. Test Cases and Target Application.
The application that was selected as the target in this experiment is a typical automation practice environment (https://the-internet.herokuapp.com/). The reason behind this particular mock site is due to the presence of numerous and intricate web elements which historically are difficult to automate through a script.

There were 10 end-to-end test cases that were carried out. The variety of browser interactions that was tested included:

- Normal user login authentication (Valid and Invalid).
- Working with regular HTML objects such as checkboxes and dropdown menus.
- Dealing with native browser JavaScript dialogs (Alerts, Confirms and Prompts).
- Delaying dynamically loaded components (AJAX calls).

All the test suites, both the legacy and migrated, had the same 10 test cases and were run on the same Page Object Model (POM) structural flow to enforce the logic to be functionally identical.

B. Hardware and Software Environment.
To avoid the problem of hardware variation in the execution measures, only one dedicated testing machine was employed. The hardware configuration involved a system with 12 logical processors cores and about 16 GB of the total system memory.

The software environment was set up as follows:

- Programming Language Java (Java 21.0+)
- Build and Execution Tool: Maven (mvn clean test commands were called to run the suites).
- Test Framework: TestNG to run the tests and manage the assertions.
- Automation Libraries: Selenium WebDriver (legacy suite), and Playwright (Java migrated suite).

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

C. Measurement and Data Collection.
A bespoke performance logging script was added to the build pipeline in order to obtain the Return on Investment (ROI) metrics. This logger noted the beginning and the end time at the suite level and the single test method level. Also, the snapshot of system resource end-of-suite execution measured the System Memory Used and JVM Heap Used in particular. Background processes were shut during the process to ensure the base CPU usage remained at 0.00%.

V. RESULTS AND ROI EVALUATION

The main aim of the experiment was to confirm whether the automated refactoring engine was able to produce functional Playwright code that would perform better than the legacy Selenium suite. The findings were able to demonstrate functional equivalence as well as meaningful performance improvements. The legacy and migrated test suites completed 10 out of 10 test cases, and had 0 failures and 0 skipped tests.

A. Suite-Level Execution Time
Testing efficiency is directly indicated by its execution speed. The performance logs demonstrated that the migrated framework had a definite advantage:

- The full time of the legacy Selenium suite to run through was 144.252 seconds.
- The auto-generated Playwright suite took the same tasks in 113.347 seconds.
- This amounted to an average time reduction of 30.905 seconds per run, which increased the speed of the Playwright execution by 21.42%.

Metric	Legacy Selenium Suite	Migrated Playwright Suite	Improvement
Total Duration	144.25 seconds	113.35 seconds	21.42% Faster
Total Tests Run	10	10	Identical Coverage
Tests Passed	10	10	100% Equivalence
Tests Failed/Skipped	0	0	No regressions

Table 2: Overall Test Suite Execution Time and Performance Comparison

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

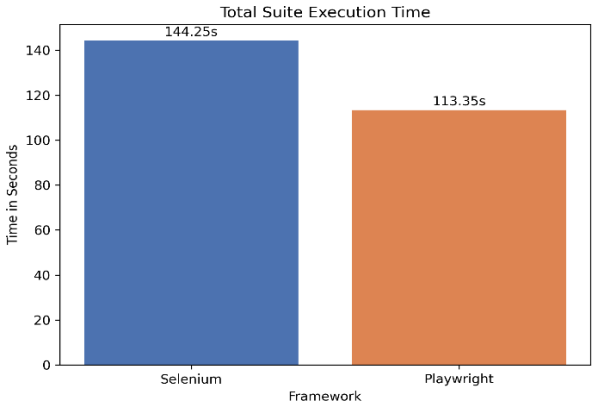

Figure 2: Comparison of total test suite execution time between legacy and migrated frameworks.

B. Improving the Class and Method level.

A more thorough analysis of the execution logs attested that the performance gains were uniform to all categories of web interactions:

- All test classes were more performant in the Playwright environment.
- The largest gain of the class level was also in the Alerts Tests class that was slower in Selenium with 34.831 seconds and in Playwright with 24.241 seconds (an increase in speed of 30.40).
- In the pure execution time analysis of individual test methods, without considering the setup overheads and teardown overheads, the total time had reduced to 8.470 seconds, compared to 11.876 seconds.
- This is a huge 28.68 percent core action level of speed improvement.

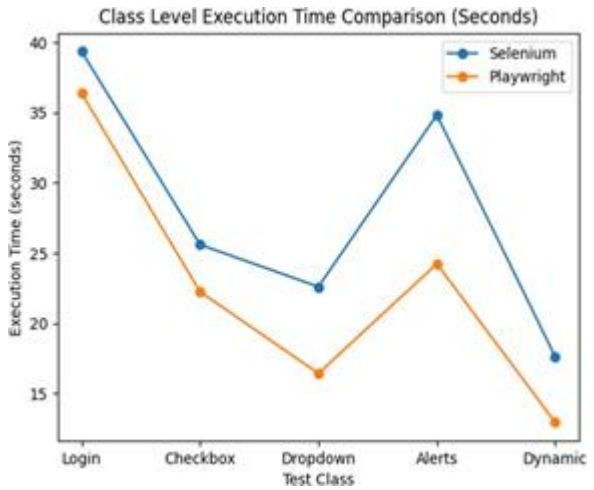

Figure 3: Execution time improvements categorized by individual test classes (line chart).

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

Test Class	Selenium Time (s)	Playwright Time (s)	Speed Gain (%)
LoginTests	39.35	36.37	7.58%
CheckboxTests	25.61	22.26	13.08%
DropdownTests	22.61	16.45	27.25%
AlertsTests	34.83	24.24	30.40%
DynamicLoadingTests	17.64	12.97	26.48%

Table 3: Execution Time Breakdown by Individual Test Class

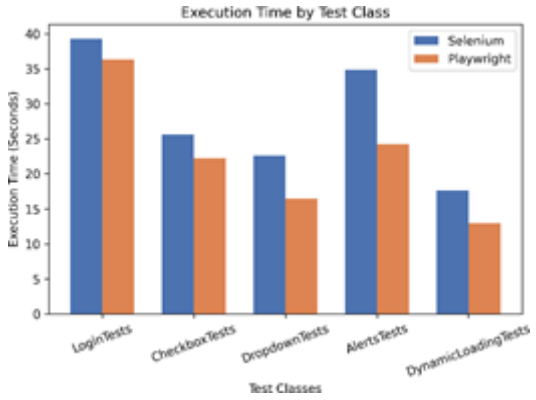

Figure 4: Execution time improvements categorized by individual test classes.

C. Resource Utilization/ Memory footprint.

A modern framework should be lightweight, in addition to speed. The snapshots of the last systems revealed the degree of load of each tool that was used on the server:

- The amount of memory consumed by Selenium at the termination of the run was 12,839 MB.
- Playwright only used 12,689 MB, which practically freed 150 MB of free memory.
- More to the point, Java Virtual Machine (JVM) Heap memory usage was significantly reduced in Selenium, 58 MB, to Playwright, 17 MB.
- This 41 MB grew smaller means that the migrated Playwright scripts require a much smaller load to the computing engine.

Metric	Selenium (Legacy)	Playwright (Migrated)	Difference
System Memory Used	12,839 MB	12,689 MB	-150 MB
Available Free Memory	3,229 MB	3,379 MB	+150 MB
JVM Heap Used	58 MB	17 MB	-41 MB
CPU Usage (Base)	0.00%	0.00%	Stable

Table 4: System Resource and JVM Heap Memory Utilization Snapshot

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

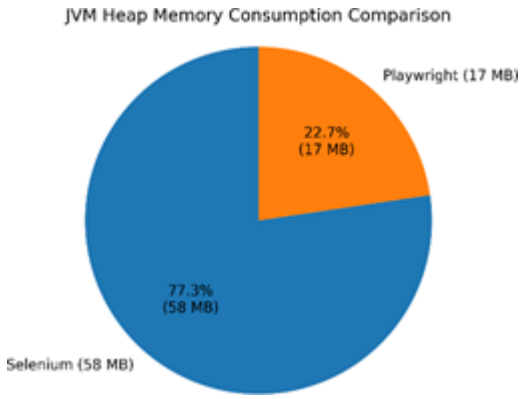

Figure 5: Java Virtual Machine (JVM) heap memory consumption comparison.

D. Quantitative Return on Investment (ROI)

Quantitative Return on Investment (ROI) has been applied to the ethical and legal field to promote total transparency of financial transactions recorded within an organization.

These technical metrics are made directly into quantifiable business value. Through this automated migration framework, software teams get a high Return on Investment (ROI) in two aspects:

- Cloud Infrastructure Cost Reduction: The two types of cost reductions are viewed as applying to a 21.42% execution time reduction in an enterprise setting with a fleet of thousands of daily tests on pay-per-minute cloud instances (such as AWS EC2) and then directly translates to a 20%+ lowering of server hosting charges. Moreover, the very reduced JVM heap size (between 58 MB up to 17 MB) enables organizations to execute test suites utilizing less expensive, less powerful server equipment without memory exceptions.
- Effort Saved: It takes a developer weeks to perform a manual translation of thousands of legacy locators and waits. This manual work is done away with by the automated refactoring engine. Instead of spending hundreds of costly engineering hours on the update of old tests, the companies save these hours and their team can concentrate on the new product features.

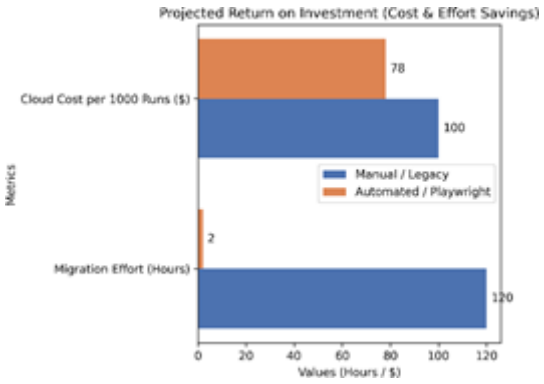

Figure 6: Projected Return on Investment (ROI) showcasing reduction in engineering effort and relative cloud execution costs.

International Journal of Science, Engineering and Technology
ISSN: 2348-4098, P-ISSN: 2395-4752

VI. CONCLUSION

The transition of the software industry to faster and more dependable testing tools is unavoidable, but the cost of transferring thousands of legacy scripts has continued to be a significant obstacle in this regard. This study has been able to overcome this drawback through the creation and testing of an automated script refactoring framework. The given tool has managed to systematize the transformation of a traditional Selenium TestNG suite to a new Playwright system without needing to rewrite the codes manually.

The results of the experiment were confirmed that the automated migration process was 100 percent functionally complete and both the frameworks successfully passed all the 10 test cases. In addition to such a straightforward translation of code, the migrated Playwright suite provided significant performance gains. It decreased total execution time of the suite by 21.42 percent and significantly decreased that of JVM heap memory by 58MB to 17MB.

As was presented through the quantitative ROI evaluation, the business value of this framework is clear. Through the automation of the refactoring process, the engineering teams will save weeks of costly manual developer hours. Additionally, the execution time decreases by 21 percent, which directly decreases the billing of clouds infrastructure of CI/CD pipelines. The ultimate benefit with this framework is that it is a highly scalable, economical, and practical solution to web automation environment modernization.

VII. FUTURE SCOPE

Although the existing refactoring engine already manages standard Page Object Model (POM) architectures in java, a number of ways exist to improve on the engine in the future:

- Multi Language Support: The framework can be extended to be able to migrate Selenium suites into their corresponding Playwright counterparts written in Python or C sharp automatically.
- AI Assisted Edge Case Handling: Regular expressions and string mapping address 90% of the refactoring, whereas adding Large Language Models (LLMs) may be able to auto-translate the custom synchronization logic, which may be highly complex, that may not be picked up by a strict rule- based engine.
- Pipeline Auto-Generation: Future releases of the tool would have the ability to automatically read older Jenkins or XML configurations, and to automatically produce modern GitHub Actions or GitLab CI YAML files optimized to be used in parallel with Playwright.

REFERENCES

1. My Project Details: https://github.com/amitkumar-98/ConferencePaperResearch
2. A. Almabruk et al., "Comparative Reliability Analysis of Selenium and Playwright: Evaluating Automated Software Testing Tools," Asian Journal of Research in Computer Science, 2025. [Online]. Available: https://journalajrcos.com/index.php/AJRCOS/article/view/546
3. A. Almabruk et al., "Comparative Reliability Analysis of Selenium and Playwright: Evaluating Automated Software Testing Tools," ResearchGate, Preprint, 2025. [Online]. Available: https://www.researchgate.net/publication/387735293_Comparative_Reliability_Analysis_of_Selenium_and_Playwright_Evaluating_Automated_Software_Testing_Tools

4. A. Tymoshchuk, "The Evolution of Test Automation: From Selenium to Playwright. A Comparison of Automation Tools: Selenium vs. Playwright vs. Cypress," International Journal of Computer (IJC), vol. 54, no. 1, pp. 37-45, 2025. [Online]. Available: https://ijcjournal.org/InternationalJournalOfComputer/article/view/23 55
5. Authors, "Analysis of end-to-end test automation tools based on the examples of Selenium WebDriver and Playwright," Annals of Computer Science and Information Systems, vol. 40, 2024. [Online].
Available: https://annals-csis.org/Volume_40/drp/pdf/3747.pdf
6. Authors, "Evaluating Automated Software Testing Efficiency for Web Applications," IEEE Xplore Digital Library, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10929217
7. Royal Cyber, "Playwright vs Selenium: Which is the Best Test Automation Framework?," Royal Cyber Tech Blog, 2024. [Online]. Available: https://www.royalcyber.com/blogs/test-automation/playwright-vs-selenium/
8. Authors, "Paper Title," Annals of Computer Science and Information Systems, vol. 40, p. 10, 2024. [Online]. Available: https://annals- csis.org/Volume_40/pliks/volume_40.pdf#page=10
9. Authors, "Chapter 14 Title," in Book Title, Noble Science Press, 2024. [Online]. Available: https://noblesciencepress.org/chapters_pdf/Chapter_14.pdf
10. Authors, "Paper Title," IEEE Xplore Digital Library, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/11330144/