Eduvest – Journal of Universal Studies
Volume 6 Number 6, June, 2026
p- ISSN 2775-3735 - e-ISSN 2775-3727

Comparative Analysis of Automated Testing Tools (Automation Testing) Using Cypress and Selenium with Testng Framework on Web Portal Testing Performance in A Banking Company

Agam Aprianto*, Gerry Firmansyah, Agung Mulyo Widodo, Habibullah Akbar
Universitas Esa Unggul, Indonesia
Email: agamaprianto@student.esaunggul.ac.id*, gerry@esaunggul.ac.id, agung.mulyo@esaunggul.ac.id, habibullah.akbar@esaunggul.ac.id

Keywords:
Automated testing, Cypress, Selenium, TestNG, banking sector, testing efficiency.

ABSTRACT

The increasing complexity of web-based banking applications has intensified the need for efficient and reliable software testing methods to ensure system quality, security, and operational stability. Traditional manual testing often faces limitations in terms of execution speed, consistency, and scalability, making automated testing an essential approach in modern software development. This study aims to compare the performance of two widely used automated testing tools, Cypress and Selenium WebDriver integrated with the TestNG framework, in testing a banking web portal application. A quantitative experimental approach was employed by executing 104 automated test scenarios, consisting of 41 Login scenarios and 63 Customer Care scenarios, under identical testing conditions. The evaluation focused on execution time, test success rate, stability through repeated testing, and reporting capabilities. The findings reveal that both tools achieved a 100% test success rate across all scenarios. However, Cypress demonstrated significantly better performance in terms of execution efficiency, requiring only 511 seconds to complete all test cases, compared to 2,184 seconds required by Selenium WebDriver with TestNG. Repeated testing also confirmed the stability and consistency of Cypress results. In terms of reporting, Cypress provided a simpler and more user-friendly reporting mechanism, whereas Selenium TestNG with Allure offered more detailed analytical reporting features. In conclusion, Cypress is a more efficient automated testing solution for banking web portal applications, particularly when execution speed, ease of implementation, and testing productivity are prioritized.

INTRODUCTION

Automation testing has now become an important component in the development process of modern software because it provides benefits in increasing efficiency, accuracy, and quality of developed applications (Faisal et al., 2025). The automation testing process aims to reduce the possibility of human error, allowing software to be tested more quickly, more consistently, and more reliably. Especially in the banking industry, which has mission-critical web applications, effective testing is essential for ensuring the security, performance, and functionality of applications (Viharika Bhimanapati, 2024).

There is a continuous need in the industrial world to do more with fewer resources (Mamede, Gonçalves, & Mira, 2023). Bank XYZ is facing a serious problem related to the limitations of manual testing, which is only capable of handling five projects per month (Chen et al., 2024; Lau et al., 2026; Mulyawan & Mauritsius, 2025). This causes delays in project completion, which in turn results in significant losses, estimated to reach 10 billion per month if five projects are not completed on schedule (Chadee et al., 2023; Kermanshachi & Pamidimukkala, 2023; Schumacher & Schumacher, 2023; Welde & Bukkestein, 2022). These limitations have a significant impact on operational speed and efficiency, especially in the banking sector, which requires applications to be tested quickly and precisely to maintain stability. To overcome this problem, Bank XYZ decided to implement automated testing. One of the most promising tools for guaranteeing quality is automation testing for banking software (Bhanushali, 2023; Gangabhathina, 2025; Itkin & Treshcheva, 2024; Kolawole et al., 2024). Automation testing not only speeds up the error identification process but also improves testing accuracy by minimizing the influence of human factors (Bolgov, 2025), and is expected to increase speed, consistency, and scalability in the testing process, thereby allowing projects to be completed more quickly and reducing losses caused by delays.

Testing web applications in the banking industry must fulfil strict standards related to application security and performance, because negligence in testing can result in financial losses and reputational damage (Al-Ajily, 2022). The banking sector worldwide continues to introduce various web-based applications, which allow customers to access banking services online. The banking sector faces 300 times more cyber attacks compared to companies in other sectors, with 31% of breaches targeting web applications, emphasizing the need for robust security testing throughout the software development cycle (Gangabhathina, 2025). This increases the complexity of testing because banking web applications often involve a large number of transactions and sensitive data that need to be secured. Therefore, it is important to test not only the functionality of web applications but also their performance and security.

Cypress and Selenium TestNG are two of the most frequently used tools in automation testing for web-based applications. Cypress is an automated testing tool that supports various programming languages and browsers, while Selenium is a testing tool that facilitates test management, such as running tests in parallel and setting the order of test execution. Both are often used together to provide more efficient automated testing, which becomes increasingly important in a banking environment that relies heavily on stable and secure web applications.

Selecting the right testing tool, one that can handle these challenges effectively, is very important. Previous research has largely focused on the advantages and disadvantages of each tool separately. For example, some studies conducted comparative reviews of automated testing tools involving Selenium and other tools; however, they did not focus their analysis on performance in the banking sector. This creates a gap that needs to be bridged, because each sector has its own characteristics and different testing needs, especially regarding the security and scalability of web applications.

In addition, the banking sector has very specific needs, including performance testing under high-traffic conditions and testing involving sensitive financial transactions. Therefore, this research aims to fill that gap by conducting a comparative analysis of the performance of Cypress and Selenium TestNG in testing a web portal application at a banking company. The focus of this study is to evaluate aspects such as execution time, stability, test success rate, and

ease of maintenance of the testing tools in the context of a web portal application used in the banking sector.

This study is expected to make a significant contribution to the selection of more optimal tools for testing banking web portal applications, with a focus on banking web application testing performance. Theoretically, this research is expected to enrich the literature on automated testing tool comparisons, while practically, the results can be used by banking companies to increase the efficiency and effectiveness of their software testing.

The specific issue addressed in this research is the limitation of manual testing in supporting the development and maintenance of banking web portals. In the case of Bank XYZ, manual testing can only handle a limited number of projects each month, causing delays in project completion and potentially creating significant financial losses. This condition shows that manual testing is no longer sufficient to meet the speed, accuracy, and scalability demanded by modern banking systems. As banking applications become more complex, the need for automated testing tools becomes increasingly urgent to reduce repetitive work, minimize human error, and accelerate the testing process.

Automation testing offers several advantages for software quality assurance, including faster execution, consistent test results, repeatability, and better test coverage. In web-based application testing, tools such as Cypress and Selenium with the TestNG framework are widely used because they support automated execution of test scenarios and generate structured testing reports. Cypress is known for its fast execution, auto-waiting mechanism, and simpler configuration, while Selenium combined with TestNG provides flexibility, cross-browser support, and detailed reporting when integrated with tools such as Allure.

Several previous studies have discussed automated testing tools in software development. Ateşoğulları and Mishra compared several automation testing tools and emphasized the importance of selecting tools based on project needs. Pelivani and Cico (2021) also examined automation testing tools for web applications and highlighted differences in usability, performance, and implementation complexity. Garcia et al. discussed the Selenium ecosystem and showed that Selenium remains widely adopted in web automation testing due to its flexibility and broad community support. Meanwhile, studies by Morales and Taky (2023) focused on Cypress and showed its potential for modern front-end and end-to-end testing.

Other relevant research has also examined testing efficiency and automation frameworks. De Brito et al. compared Cypress, Playwright, and Selenium WebDriver, showing that each tool has different strengths depending on execution speed, stability, and ease of use. Bhimanapati discussed the use of Selenium and Cypress for comprehensive web testing and emphasized their relevance for improving software quality. Faisal et al. highlighted the importance of maturity in software testing automation, while Bolgov discussed the role of automation testing in banking system development. These studies provide a strong foundation for understanding automation testing, but they have not fully addressed comparative testing performance in a banking web portal context.

The research gap lies in the limited number of studies that specifically compare Cypress and Selenium TestNG in banking web portal testing using measurable performance indicators. Many previous studies compare automation tools in general web applications, but banking systems have more specific requirements, such as high reliability, sensitive data handling, transaction accuracy, and strict operational standards. Therefore, a sector-specific comparison

is needed to determine which tool is more effective for banking web portal testing, especially in terms of execution time, test success rate, reporting capability, and stability.

The urgency of this research is based on the practical need of banking companies to improve testing efficiency and reduce delays in software project completion. Inefficient testing can slow down service innovation, increase operational costs, and expose banking institutions to system failures. By identifying the most suitable automation testing tool, banking companies can improve the quality assurance process, accelerate project delivery, and support more reliable digital banking services. This research is also important because tool selection directly affects testing productivity, maintenance effort, and the accuracy of defect detection.

The novelty of this research lies in its comparative analysis of Cypress and Selenium TestNG in a real banking web portal testing environment using actual test scenarios. This study does not only describe the features of each tool but also evaluates their performance based on execution time, repeated testing stability, test success rate, and reporting quality. By using Login and Customer Care scenarios, this research provides a more contextual and practical evaluation of how both tools perform in banking application testing.

The purpose of this research is to compare the performance of Cypress and Selenium with the TestNG framework in automated testing of a banking web portal. The objective is to determine which tool provides better efficiency, stability, and reporting capability for banking web application testing. The contribution of this research is both theoretical and practical. Theoretically, it enriches the literature on automation testing tool comparison in a sector-specific context. Practically, it provides recommendations for banking companies, QA engineers, and software developers in selecting appropriate automated testing tools to improve testing effectiveness, reduce execution time, and support the development of reliable digital banking systems.

METHOD

Study This use approach quantitative with method experiment comparative engineering focused device software, especially in field testing automatic (automation testing). The purpose of study This is for do testing automated testing tools, namely Cypress and Selenium TestNG, in context testing application Web Portal that simulates service banking. Research This done through a number of stages.

TestNG Framework Workflow

TestNG framework workflow for testing automatically in the system banking started from stage initialization projects and configurations environment, where the QA team prepares folder structure, dependencies such as Selenium WebDriver and TestNG, and set up browser drivers and connections to the non- production banking environment so that the test data is not influence system actual. After that, done design test scenario with compile test cases based on business processes important such as login, checking balance, fund transfers, and payments bill. Stage next is implementation script automatic using Java and TestNG with apply Page Object Model (POM) approach so that the structure code more organized as well as use data-driven testing concept so that testing more flexible to various data inputs. After script finished made, done test suite execution via testng.xml configuration file for arrange order, grouping, and test parameters, even allows testing parallel to speed up the process. During the test process, TestNG performs validation and logging using assertions to verify suitability results

current with what is expected, and if happen failure, system automatic record logs as well take a screenshot. After testing finished, the framework generates report automatic in HTML format or use tool like ExtentReport and Allure Report which displays results testing in a way interactive and informative. Report the Then sent automatic to stakeholders such as QA Lead or Project Manager via email integration or CI/CD systems like Jenkins so that test results can be quick reviewed without manual execution. Stage final is analysis results and actions next, where the QA team reviews report for find a bug or error, report it to team developer through system tracking like JIRA, and do testing repeat (regression test) after repairs, so that the entire testing process walk repeatable and measurable until system banking confirmed stable, safe , and in accordance standard operational.

Figure 1TestNG Framework Workflow

Data collection technique

Study This use approach quantitative, so that technique Data collection is directed at obtaining numeric data that can be obtained measured, analyzed, and compared in a way statistics. Data collected through a number of method as following:

1. Experiment Live, Testing done in a way direct to application Web Portal used in the XYZ Bank environment with use automated testing tools, namely Cypress and Selenium-TestNG. Each test case is executed on the tool with scenario identical tests, and the results noted for analyzed in a way quantitative.

2. Execution Time Measurement, Time required For finish all test cases on each tool noted in a way automatic through feature reporting from each framework. Measurement done in unit seconds (s), and the result use analysis statistics descriptive.
3. Calculation of Test Success Rate, Data on the number of successful (pass) and failed (fail) test cases is collected from report results testing automatic. Percentage success counted as indicator reliability of each test tool.
4. Evaluation Stability (Repeatability), Testing done in a way repetitive five times on the same test case For observe consistency results. Amount inconsistency or difference results between test recorded and used as numeric data For measure level stability test equipment .
5. Reporting Test Results, Reporting results testing done For presenting the data obtained from the testing process automatic using Cypress and Selenium TestNG. Reported data covers results testing, total time test results, time test script unit, as well as amount successful error identified by each automated test tool.

All quantitative data collected will served in form tables, graphs, and done analysis comparison use method statistics descriptive, such as mean value, deviation standard, and percentage. This is done for give description objective about performance automatic test equipment to application Web banking portal.

RESULTS AND DISCUSSION

Code Snippet

During the implementation phase, automation testing of the banking web application was conducted using Cypress. This implementation aims to automatically run functional test scenarios in a web browser. This code snippet demonstrates the implementation of testing using Cypress, which was used to verify the main functions of the banking web application. The implemented test scenarios were tailored to the research needs and thus served as the basis for the comparative analysis process.

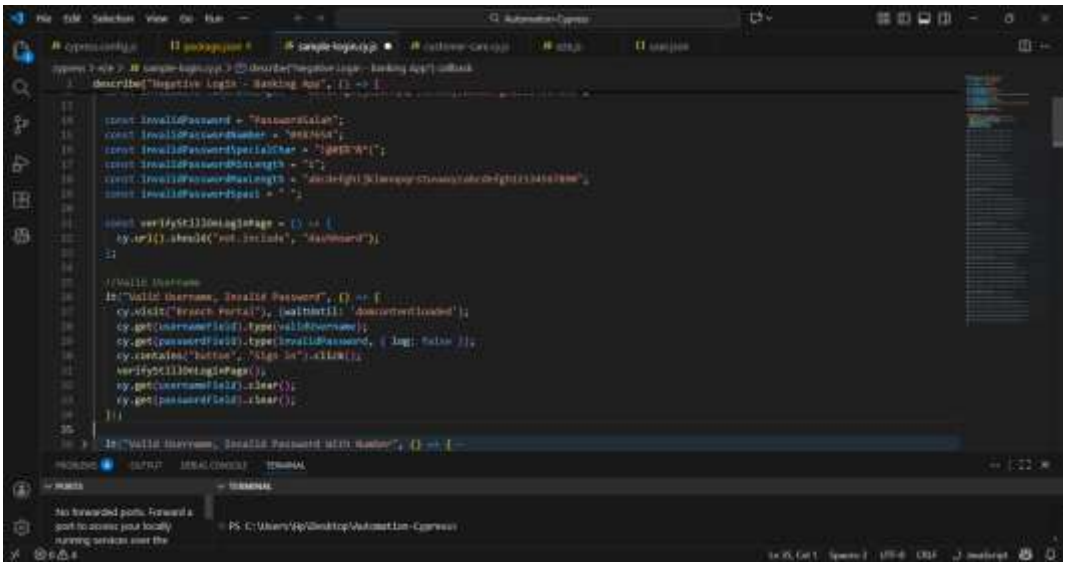

Figure 2 1Code

Further implementation of automation testing done using Selenium as comparator against Cypress. Selenium is used for operate scenario the same testing on banking web applications

via web browser. Snippet this program code show implementation testing using Selenium, which aims for ensure consistency and validity results testing. The results of implementation This used in the analysis process comparative between Cypress and Selenium.

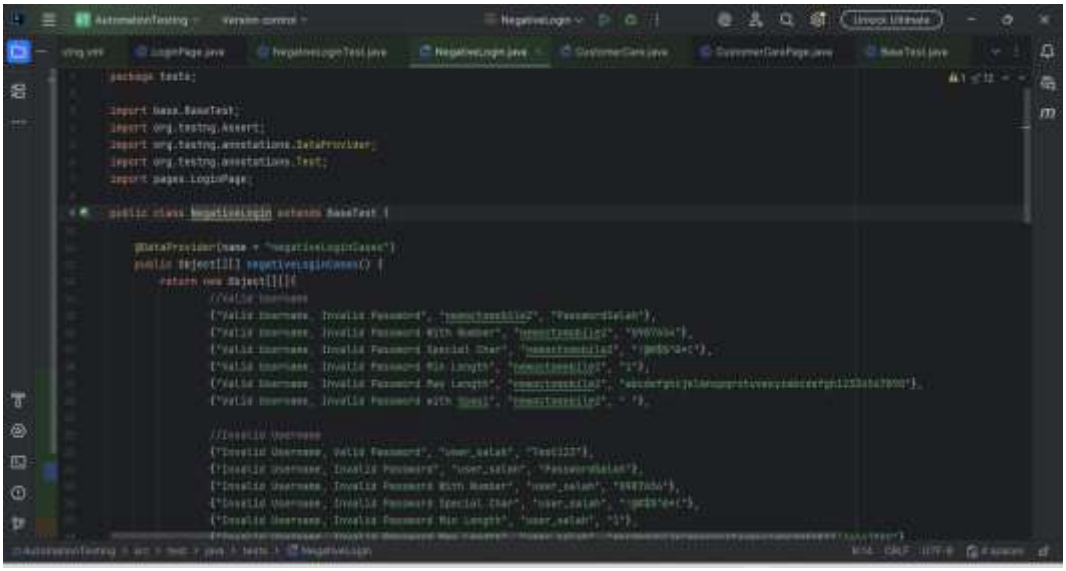

Figure 3 2Code

Cypress Implementation

The Cypress implementation is structured as a collection of JavaScript-based test cases with a test directory structure that separates specification files (spec), supporting utilities, and test data. Test cases are written using the describe/it approach, ensuring each scenario has a clear identity. Test execution is performed in open mode (for interactive debugging) and run mode (for automated execution in the terminal).

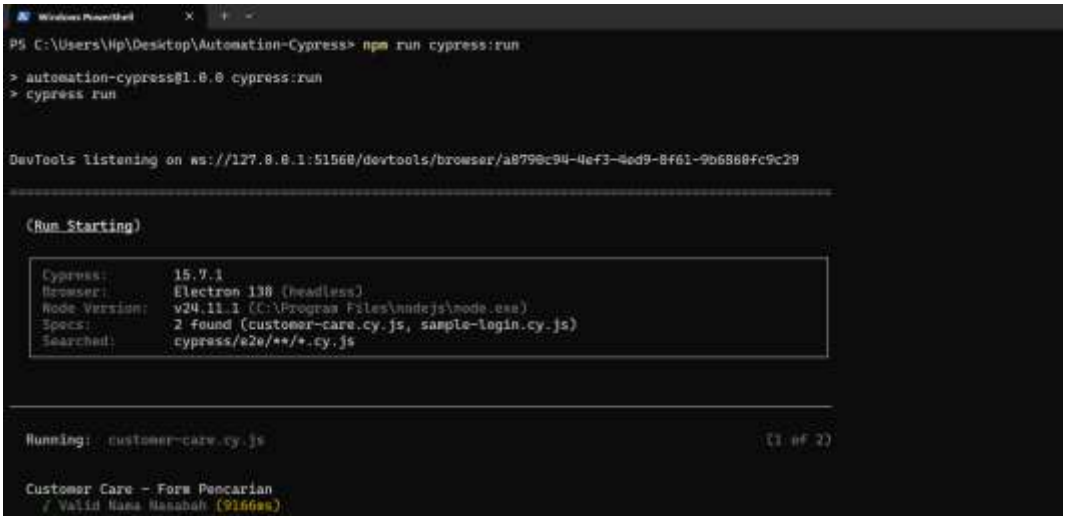

Figure 3When Running Cypress

For reporting results, Cypress can generate HTML/JSON-based reports using Mochawesome. Reports are typically generated during run mode, allowing for automatic recapitulation of all execution results.

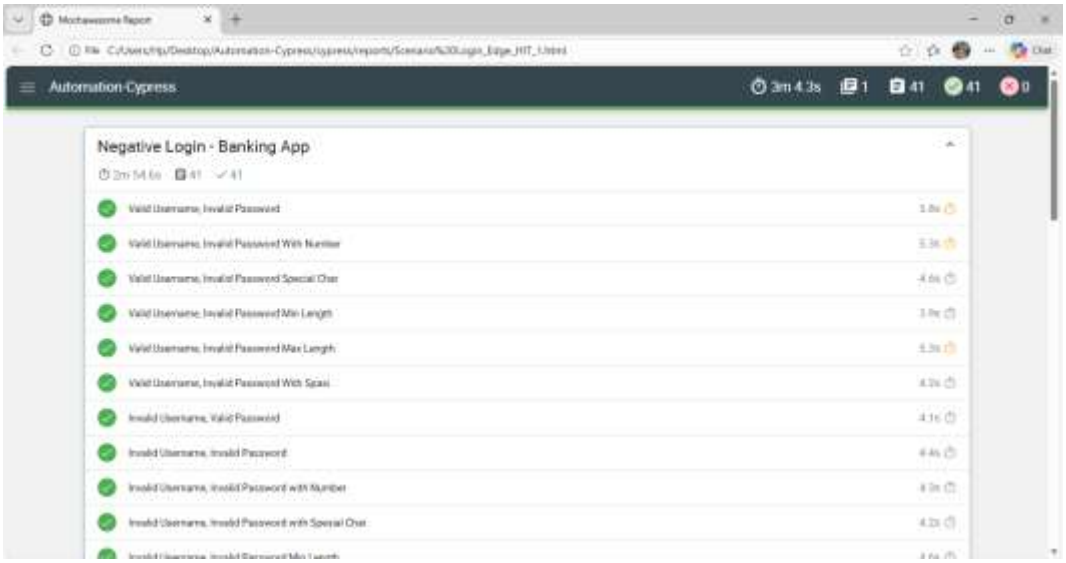

Figure 5 4Scenario Login Report Results

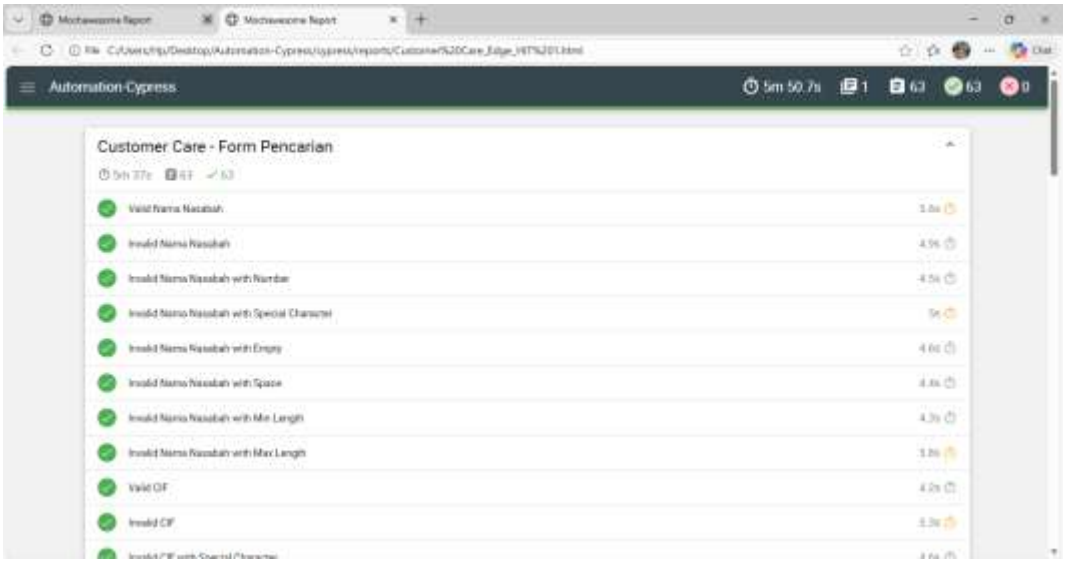

Figure 6 5Customer Care Report Results

Selenium WebDriver Implementation with TestNG

The Selenium WebDriver + TestNG implementation uses Java and is managed through Maven. The framework is structured using a Page Object Model (POM) approach to separate page locators and actions from testing logic. The BaseTest class is used to manage WebDriver initialization, wait configuration, and teardown. Test case grouping and execution are done using the @Test annotation and suite configuration in testng.xml.

Figure 7 When Running Selenium + TestNG

For reporting results, Allure integration is done via the TestNG listener so that every execution generates a results file (allure-results) that can be generation become HTML report (allure-report). Report This make it easier analysis results execution through pass/fail summary , error details, as well as history execution.

Figure 8 Selenium Report Results with Allure Scenario Login

Figure 9 Selenium Report Results with Allure Customer Care

Recapitulation of Test Results

The following table serve recapitulation results testing for each test tool . The values in the table filled based on results execution that has been carried out , including amount scenario as well as time execution required in Login and Customer Care scenarios .

Table 1 1Scenario Test Results

No	Login Scenario	Cypress	Selenium
1	Valid Username, Invalid Password	7.6s	18.6
2	Valid Username, Invalid Password With Number	6s	18.3
3	Valid Username, Invalid Password Special Char	5.7s	18.1
4	Valid Username, Invalid Password Min Length	6s	18.2
5	Valid Username, Invalid Password Max Length	6.6s	18.7
6	Valid Username, Invalid Password With Space	5.8s	18.4
7	Invalid Username, Valid Password	5.7s	18
8	Invalid Username, Invalid Password	5.6s	18.1

9	Invalid Username, Invalid Password with Number	5.6s	18.2
10	Invalid Username, Invalid Password with Special Char	5.5s	17.9
11	Invalid Username, Invalid Password Min Length	5.7s	18.1
12	Invalid Username, Invalid Password Max Length	6.5s	18.6
13	Invalid Username, Invalid Password with Space	5.9s	18.3
14	Invalid Username with Number, Valid Password	5.9s	18.2
15	Invalid Username with Number, Invalid Password	5.7s	18
16	Invalid Username with Number, Invalid Password with Number	5.6s	18.1
17	Invalid Username with Number, Invalid Password with Special Char	5.6s	18.2
18	Invalid Username with Number, Invalid Password Min Length	5.2s	17.8
19	Invalid Username with Number, Invalid Password Max Length	6.3s	18.5
20	Invalid Username with Number, Invalid Password with Space	5.2s	17.9
21	Invalid Username with Special Char, Valid Password	5.8s	18.2
22	Invalid Username with Special Char, Invalid Password	5.9s	18.3
23	Invalid Username with Special Char, Invalid Password with Number	6.9s	18.7
24	Invalid Username with Special Char, Invalid Password with Special Char	6.1s	18.4
25	Invalid Username with Special Char, Invalid Password with Min Length	5.6s	18.1
26	Invalid Username with Special Char, Invalid Password with Max Length	6.3s	18.5
27	Invalid Username with Special Char, Invalid Password with Space	5.4s	18
28	Invalid Username with Min Length, Valid Password	5.4s	18.1
29	Invalid Username with Min Length, Invalid Password	5.5s	18
30	Invalid Username with Min Length, Invalid Password Number	5.6s	18.2
31	Invalid Username with Min Length, Invalid Password Special Char	5.8s	18.3
32	Invalid Username with Min Length, Invalid Password Min Length	5.3s	17.9
33	Invalid Username with Min Length, Invalid Password Max Length	6.3s	18.4
34	Invalid Username with Min Length, Invalid Password with Space	5.5s	18.2
35	Invalid Username with Max Length, Valid Password	6.3s	18.4
36	Invalid Username with Max Length, Invalid Password	6.7s	18.6
37	Invalid Username with Max Length, Invalid Password with Number	6.5s	18.5
38	Invalid Username with Max Length, Invalid Password with Special Char	6.4s	18.3
39	Invalid Username with Max Length, Invalid Password with Min Length	6.3s	18.4
40	Invalid Username with Max Length, Invalid Password with Max Length	6.8s	18.6
41	Invalid Username with Max Length, Invalid Password with Space	6.5s	18.7

Table 2 2Care Scenario Test Results

No	Customer Care	Cypress	Selenium
1	Valid Customer Name	5.2 s	21.5 s
2	Invalid Customer Name	5.1 s	20.3 s
3	Invalid Customer Name with Number	4.3 s	20.7 s
4	Invalid Customer Name with Special Character	4.3 s	21.0 s
5	Invalid Customer Name with Empty	4.8 s	21.1 s
6	Invalid Customer Name with Space	4.1 s	19.2 s
7	Invalid Customer Name with Min Length	4 s	17.3 s
8	Invalid Customer Name with Max Length	5.5 s	21.0 s
9	Valid CIF	4.6 s	22.6 s
10	Invalid CIF	5.1 s	17.0 s
11	Invalid CIF with Special Character	5 s	19.9 s
12	Invalid CIF with Space	4.3 s	20.7 s
13	Invalid CIF with Min Length	4 s	18.9 s

14	Invalid CIF with Max Length	4.8 s	22.1 s
15	Valid Customer ID	4.8 s	22.9 s
16	Invalid CustomerId	4.3 s	20.8 s
17	Invalid CustomerId with Special Character	4.7 s	21.3 s
18	Invalid CustomerId with Space	4.4 s	21.7 s
19	Invalid CustomerId with Min Length	4.9 s	20.4 s
20	Invalid CustomerId with Max Length	5 s	21.8 s
21	Valid Number Telephone Customer	4.5 s	20.7 s
22	Invalid Number Telephone Customer	4.3 s	20.6 s
23	Invalid Number Telephone Customer with Special Character	5.3 s	18.4 s
24	Invalid Number Telephone Customers with Space	4.3 s	21.1 s
25	Invalid Number Telephone Customers with Min Length	4.3 s	20.1 s
26	Invalid Number Telephone Customers with Max Length	5.5 s	21.5 s
27	Customer Credit Card	4.5 s	21.4 s
28	Customer Credit Card	4.9 s	21.1 s
29	Customer Credit Card with Special Character	4.4 s	21.0 s
30	Customer Credit Card with Space	4.2 s	21.6 s
31	Customer Credit Card with Min Length	4.5 s	20.3 s
32	Customer Credit Card with Max Length	4.6 s	20.7 s
33	Customer Debit Card	4.4 s	21.8 s
34	Invalid Customer Debit Card	5.1 s	20.8 s
35	Customer Debit Card with special character	4.2 s	21.5 s
36	Customer Debit Card with Space	4.2 s	20.1 s
37	Customer Debit Card with Min Length	4.4 s	22.4 s
38	Customer Debit Card with Max Length	5 s	18.5 s
39	Valid Number Account Customer	4.5 s	21.7 s
40	Invalid Number Account Customer	5 s	21.6 s
41	Invalid Number Account Customer with Special Character	4.3 s	20.0 s
42	Invalid Number Account Customers with Space	4 s	21.0 s
43	Invalid Number Account Customers with Min Length	4.2 s	18.3 s
44	Invalid Number Account Customers with Max Length	5.1 s	22.0 s
45	Valid Number Reference Customer	4.3 s	22.2 s
46	Invalid Number Reference Customer	4.2 s	20.3 s
47	Invalid Number Reference Customer with Special Character	4.7 s	18.0 s
48	Invalid Number Reference Customers with Space	6 s	20.5 s
49	Invalid Number Reference Customers with Min Length	7 s	21.5 s
50	Invalid Number Reference Customers with Max Length	6.2 s	20.9 s
51	Valid Number Identity Customer	7.1 s	20.5 s
52	Invalid Number Identity Customer	6.3 s	16.4 s
53	Invalid Number Identity Customer with Special Character	7.3 s	21.3 s
54	Invalid Number Identity Customers with Space	5.6 s	20.5 s
55	Invalid Number Identity Customers with Min Length	6.3 s	21.1 s
56	Invalid Number Identity Customers with Max Length	7.9 s	21.8 s
57	Valid Customer Email	6.6 s	22.6 s
58	Invalid Customer Email with No At	6.6 s	21.7 s
59	Invalid Customer Email with No Domain	6.8 s	18.7 s
60	Invalid Customer Email with Special Character	6.7 s	17.8 s
61	Invalid Customer Email with Space	6.3 s	22.4 s
62	Invalid Customer Email with Min Length	5.9 s	20.9 s
63	Invalid Customer Email with Max Length	7.9 s	21.1 s

Execution Time Comparison

Comparison time execution done with compare total duration of testing as well as duration per module. For make it easier interpretation, value time execution can served in form table.

Table 3 Test Results per Test Script

Test Script	Cypress (seconds)	Selenium + TestNG (seconds)	Difference (seconds)	Information
Login (41 scenarios)	174	747	573	Cypress more fast
Customer Care (63 scenarios)	337	1,437	1100	Cypress more fast

Comparative Reporting

Comparative reporting is done with evaluate quality and completeness information provided by the report results execution. Observed aspects covering convenience generate reports, summaries results (summary), failure details, and supporting evidence such as screenshots or logs.

Table 3Comparative Reporting Results

Aspect	Cypress (Mochawesome)	Selenium + TestNG (Allure)	Notes
Ease of generating reports	Report can produced in a way automatic after execution testing without configuration complex additions .	Need configuration additions to the TestNG listener and Allure integration before report can produced .	Cypress more superior from side ease of initial setup .
Pass/fail summary	Provide summary number of successful and failed test cases in a way clear on the page main report .	Provide summary results structured and detailed test case execution on the Allure dashboard.	Both of them You're welcome display summary , but Allure is more informative .
Error details / stack trace	Displays error messages automatically concise with relevant stack trace snippets .	Displays error details and stack traces complete until to the method level.	Selenium + TestNG more superior For analysis failure deep .
Evidence (screenshot / log)	Screenshots can be produced in a way automatically when the test fails and is displayed directly in the report .	Screenshots and logs can be attached as an attachment to the Allure report .	Allure more flexible in evidence management .
Convenience navigation report	Navigation report simple and easy understood , but filtering feature still limited .	Provide navigation interactive with filtering, grouping, and execution history features .	Allure more superior For analysis scale big .

Test Results Repeatedly

Repetitive testing was performed on every test scenario (T1 to T5) to evaluate the consistency of execution time for both Selenium and Cypress. Each scenario was executed several times under the same configuration and test environment. The repetitive test results are presented in the form of an average time table and an execution graph to facilitate comparative analysis between the tools.Based on the repetitive testing results table and the execution time comparison graph, it is visible that Selenium has a relatively consistent execution time across

every test scenario, with an average execution time in the range of 11 to 13 minutes. Although there is slight variation in time between repetitions, the differences do not show significant fluctuations.Meanwhile, Cypress shows shorter execution times across all test scenarios, with an average execution time in the range of 1.7 to 2.9 minutes. Based on the graphs, the Cypress testing results also show a good level of stability, characterized by relatively small differences in execution time between repetitions.The comparison of repetitive testing results on the average graph shows that the difference in execution time between Selenium and Cypress remains consistent with the previous testing results. This indicates that the efficiency of Cypress is not only observed in a single test but is also maintained across subsequent repeated tests.

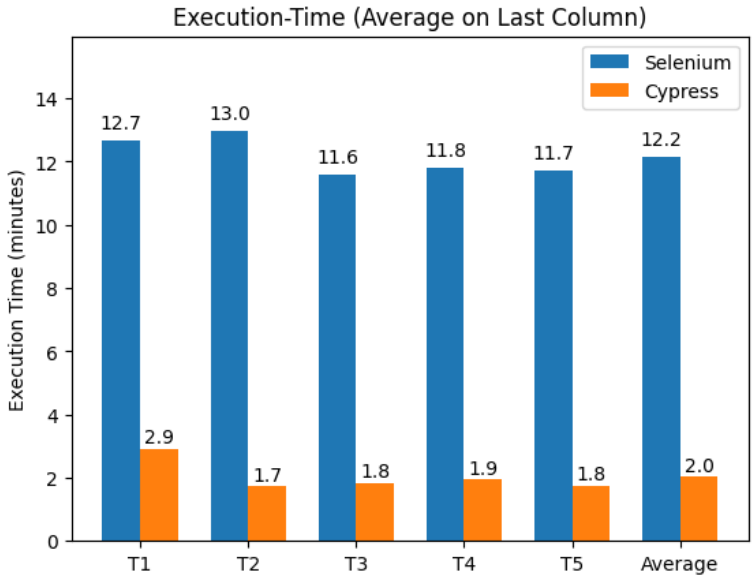

Figure 10 Testing Diagram Repeated Login Scenario

Discussion Findings Important

Based on the implementation process and results testing, there is a number of findings important things that influence stability execution and ease test case development, including:

1) Difference waiting mechanism: Cypress has auto-waiting, while Selenium requires explicit wait for element dynamic.
2) Element locator selection: changes DOM structure (eg. text knob is on the span element) can influence Selenium stability test.
3) validation in Customer Care: some fields are rejecting character special / space, while certain fields (customer name and email) accept such input so that expectation testing need adjusted.
4) Reporting: Allure requires a report generation process to be saved. Permanent, while Cypress reporting is generally produced during run mode.
5) Environmental constraints testing (eg SSL certificate in UAT environment) can influence automatic browser connection on Selenium and need configuration addition.

CONCLUSION

Based on the findings of this study, both Cypress and Selenium WebDriver integrated with the TestNG framework successfully executed all automated testing scenarios on the banking web portal with a 100% test success rate. However, significant differences were observed in execution performance and reporting efficiency. Cypress demonstrated superior performance by completing 104 test scenarios in only 511 seconds, whereas Selenium WebDriver with TestNG required 2,184 seconds for the same workload. The repeated testing results also confirmed that Cypress maintained a high level of execution stability while consistently delivering faster testing cycles. In terms of reporting, Cypress provided a simpler and more user-friendly reporting mechanism, while Selenium with TestNG offered more comprehensive reporting features through Allure, including detailed failure analysis and execution history. These findings indicate that Cypress is a more efficient solution for automated testing of banking web portal applications, particularly when execution speed and ease of implementation are prioritized.

For future research, it is recommended to expand the scope of testing by including additional performance indicators such as memory consumption, resource utilization, scalability under high-traffic conditions, and integration with Continuous Integration/Continuous Deployment (CI/CD) pipelines. Future studies may also compare Cypress and Selenium TestNG with other emerging automation testing frameworks, such as Playwright or Robot Framework, across different domains beyond banking. Furthermore, incorporating security testing, cross-browser compatibility testing, and large-scale enterprise application environments would provide a more comprehensive evaluation of automated testing tools and contribute to the development of more effective software quality assurance strategies.

REFERENCE

Al-Ajily, M. (2022). *Automated testing for React web application with Cypress* [Bachelor's thesis]. Computer Applications -koulutus.

Ateşoğulları, D., & Mishra, A. (2020). Automation testing tools: A comparative view. *International Journal on Information Technologies & Security, 12*.

Bhanushali, A. (2023). Ensuring software quality through effective quality assurance testing: Best practices and case studies. *International Journal of Advances in Scientific Research and Engineering, 26*(1), 1–18.

Bolgov, S. (2025). The role of test automation in the development of banking systems. *Norwegian Journal of Development of the International, 72–75*. https://doi.org/10.5281/zenodo.15657632

Chadee, A., Ali, H., Gallage, S., & Rathnayake, U. (2023). Modelling the implications of delayed payments on contractors' cashflows on infrastructure projects.

Chen, S., Jiang, M., & Luo, X. (2024). Exploring the security issues of real world assets (RWA). In *Proceedings of the Workshop on Decentralized Finance and Security* (pp. 31–40).

de Brito, A. B., dos Santos, R. M., & de Medeiros, S. N. (2023). *Testing tools: A comparative study of Cypress, Playwright and Selenium WebDriver*. Instituto Federal de Educação, Ciências e Tecnologia de Pernambuco.

Faisal, M., Alharbi, Y., Alsaffar, M., Hussain, S., Saqib, M., Khan, J., & Lee, Y. (2025). Developing SPIM-TA: A maturity-level framework for systematic process improvement in software testing automation. *Ain Shams Engineering Journal*. Advance online publication. https://doi.org/10.1016/j.asej.2025.103472

Gangabhathina, P. M. (2025). Enterprise test automation strategy for scalable banking

innovation. *Journal of Multidisciplinary, 5*(7), 756–765.
Garcia, B., Gallego, M., Gortazar, F., & Organero, M. (2020). A survey of the Selenium ecosystem. *Electronics, 9*(7), Article 1067. https://doi.org/10.3390/electronics9071067
Itkin, I., & Treshcheva, E. (2024). Enhancing the quality of banking technology platforms through a hybrid AI testing approach. *Journal of Digital Banking, 9*(1), 86–95.
Kermanshachi, S., & Pamidimukkala, A. (2023). Robustness analysis of total project cost and schedule delay and overrun indicators of heavy industrial projects. *Journal of Legal Affairs and Dispute Resolution in Engineering and Construction, 15*(2), 04523005.
Kolawole, I., Osilaja, A. M., & Essien, V. E. (2024). Leveraging artificial intelligence for automated testing and quality assurance in software development lifecycles. *International Journal of Research Publication and Reviews, 5*(12), 4386–4401.
Lau, E., Dücker, M., Chaudhary, R., Goh, H. W., Wei, R., Kumar, V., Qunbar, S., Gogia, G., Liu, Y., & Millslagle, S. (2026). *BankerToolBench: Evaluating AI agents in end-to-end investment banking workflows* (arXiv Preprint No. 2604.11304). arXiv.
Mamede, H., Gonçalves, C. M., & Mira, M. (2023). A lean approach to robotic process automation in banking. *Heliyon*. https://doi.org/10.1016/j.heliyon.2023.e18041
Morales, A. M. (2023). *Automated front-end website testing with Cypress*.
Mulyawan, H., & Mauritsius, T. (2025). Bank XYZ case study. In *Advances in Smart Knowledge Computing: Towards Post Artificial Intelligence Era* (Vol. 2020, p. 179).
Pelivani, E., & Cico, B. (2021). A comparative study of automation testing tools for web applications. In *Mediterranean Conference on Embedded Computing (MECO)*. https://doi.org/10.1109/MECO52532.2021.9460242
Schumacher, A., & Schumacher, M. (2023). *The road to project management excellence: Understanding the reasons and impacts of delays within project organizations*.
Taky, & Malika, T. (2021). *Automated testing with Cypress* [Bachelor's thesis]. Vaasan Ammattikorkeakoulu University of Applied Sciences. https://www.theseus.fi/handle/10024/495908
Viharika Bhimanapati, P. (2024). Leveraging Selenium and Cypress for comprehensive web testing. *Journal of Quantum Science and Technology, 1*(1), 66–79. https://doi.org/10.36676/jqst.v1.i1.10
Welde, M., & Bukkestein, I. (2022). Over time or on time? A study of delays in large government projects. *Procedia Computer Science, 196*, 772–781.