International Journal of Leading Research Publication (IJLRP)

E-ISSN: 2582-8010 ● Website: www.ijlrp.com ● Email: editor@ijlrp.com

DEBUGGING SELENIUM FAILURES WITH INTELLIJ STACK TRACES AND CHROME DEVTOOLS: CASE STUDIES IN RETAIL QA

Udayan Verma

Denver, USA
udayanverma7@gmail.com

Abstract:
Automated testing is an essential component of retail quality assurance (QA), wherein checkout procedures and product catalogues, as well as transactional workflows, have to be maintained during the times of high demand. Regression and functional testing have gained popularity with selenium, but it is prone to fail due to dynamic content, asynchronous updates, and environmental variability, instead of actual defects. Classical logs are not as diagnostic as possible, which makes the triage process longer and lowers the trust in automated pipelines. The paper is a case-based analysis of debugging the failures with Selenium failures using IntelliJ IDEA and Chrome DevTools in a retail QA setting. IntelliJ supports stack trace for forensics, breakpoint analysis, and dependency management whereas Chrome DevTools provides runtime visibility with network, console, and DOM inspection support. There are recurrent problems in case studies including stale element reference, timing issues and latency-induced false negative and how joint diagnostic methods led to shortened triage and enhanced reproducibility. The research will introduce a viable framework of evidence-based debugging, increasing the stability of tests, collaboration, and CI/CD resiliency to retail pipelines.

Keywords: Selenium, IntelliJ, Chrome DevTools, Retail QA.

I. Introduction

The other fundamental concept of quality assurance (QA) in regards to technology in retail is the concept of automated testing so that the checkout systems, search features and inventory tracking systems are certain to perform well under the heavy load. Selenium is a free browser automation solution, which has been extensively used in functional testing and regression testing, but the actual implementation of the scripts can very easily reveal difficulties when it is executed without errors. The old systems of debugging cannot be used in the retail environment anymore because of the elements of web dynamics, asynchronous JavaScript, and ever-changing interfaces. The toolkit has room to be enhanced on its resolution. IntelliJ IDEA has debugging features and stack traces to analyse root causes of test scripts, and Chrome DevTools to get run time data on front-end behaviour, such as network traffic, console errors and element states. All these tools are complementary to one another when they are used simultaneously thereby helping the QA teams to figure out and rectify failures. It is a case study of the findings that can be used to ensure that the process of debug, test stability, and scalability of QA is more effective in retail automation environments.

II. Background and Rationale

The complexities of retail QA environment are far more than the other industries. The applications should be able to support the changing traffic at the seasonal peaks, secure transactions, and flawless cross-device experiences. This working environment demanded automation systems that can perform

huge regression suites in a short time and with consistency. Nevertheless, Selenium test suites are often faced with challenges that undermine their performance [1].
Common failures are element not found failures due to dynamic loading of elements, stale element reference exceptions due to repetitive loading of the DOM, and timeouts due to asynchronous processes that may involve product catalogue refresh loading or payment authorisation. Whereas the Selenium logs can tell the type of failure, it rarely has enough diagnostic information so the QA staff is left with the debugging by trial and error.

The logic behind focusing on IntelliJ and Chrome DevTools is the complementary nature of these specific tools in terms of their diagnostic functions. IntelliJ is able to group stack traces with breakpoints and log output and do root cause analysis both in test code and inside external libraries. By comparison, Chrome DevTools reveals behaviour of applications at runtime, providing insight into network requests, console errors, and DOM state. As an example, the delay in checkout due to the slow response of AJAX would not be displayed in the Selenium logs but would be easily identified in DevTools [2].
These views allow retail QA teams to fix problems more quickly, stabilise CI/CD pipelines and maintain high-volume retail systems reliability.

III. System Architecture & Pipeline Design

The concept of continuous integration and continuous delivery (CI/CD) pipeline is increasingly becoming common among retail organisations in an attempt to encourage a rapid feature delivery without compromising the quality of the application. Such pipelines have embedded Selenium tests at various stages: build verification, pre-release regression testing, and others. The common pipeline architecture consists of a version control (e.g. Git), build servers (e.g. Jenkins or GitLab CI) and containerised environments (e.g. Docker or Kubernetes). The hierarchical organization enables the scalable and repeatable implementation of the automated testing suites, ensuring the consistency of the validation process of different retail workflows.

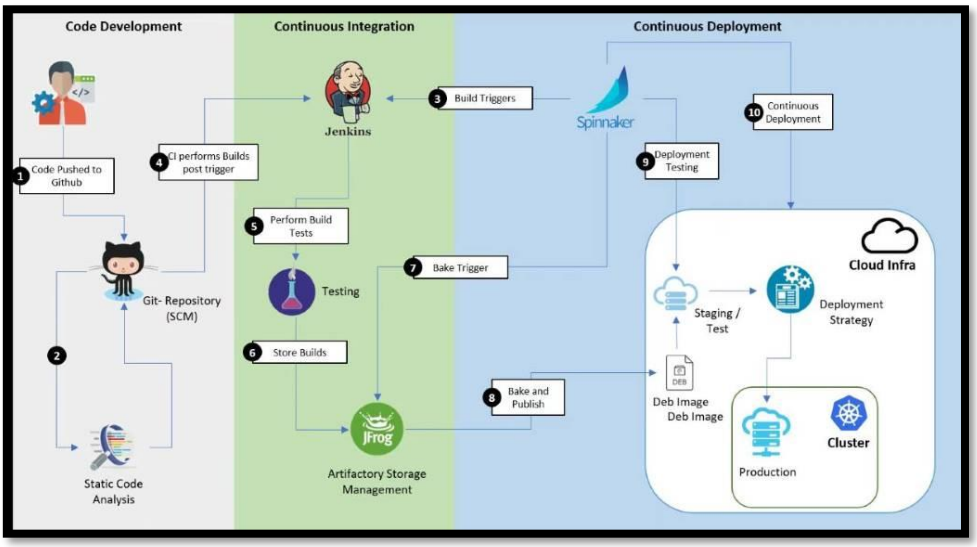

Fig 1. CI/CD Pipeline

IntelliJ is a development hub that is part of this architecture. Test engineers write and debug Selenium scripts in IntelliJ using dependency management of frameworks like TestNG or JUnit and real-time debugging tools [3]. In case of failures when executing a pipeline, the stack trace analysis of IntelliJ will allow engineers to replicate the problem on their local machine, track method hierarchies, and match

statements with code paths. This gives a richer context as compared to generic pipeline logs which tend to hide the root causes.

Chrome DevTools is used along with IntelliJ to provide visibility into the running application. Network analysis, performance profiling, and DOM inspection identify bottlenecks Selenium logs is unable to capture. Syntactically correct scripts failed repeatedly in a single retail case study with checkout automation. The traces made by IntelliJ showed a time out exception, whereas DevTools revealed an intermittently unresponsive external payment gateway API which postponed the update of the DOM.

Combining IntelliJ code-level information with DevTools application-level evidence creates an entire debugging system in retail QA pipelines. This architecture does more than identify failures, it puts them into the context of the system behaviour, allowing its failures to be fixed precisely and its tests to be more resilient.

IV. Integration & Maintenance Strategy

Retail QA pipelines based on Selenium tests should be explicitly integrated and actively maintained to be effective with the changing technology. Retail applications are regularly modified in user interface design, backend architecture and integration of third-party services and each of these can interfere with the existing test scripts. Therefore, it is necessary to have a formal maintenance plan to maintain reliability of automated testing [4].

One of the major practices is to synchronise the Selenium, ChromeDriver, and browser versions to avoid compatibility errors. IntelliJ simplifies this task with dependency management and support of the plugins, engineers can see the progress of libraries and update them regularly. This is augmented by Chrome DevTools that checks rendering when the browser on release is altered in CSS or layout behaviour. In the absence of such alignment, teams are likely to have continued failures due to environmental mismatches as opposed to actual defects.

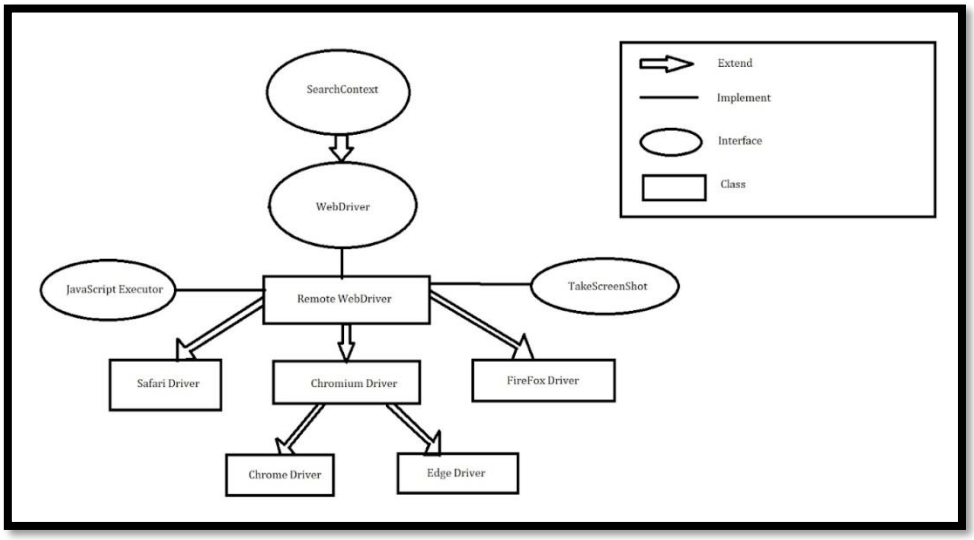

Fig 2. Architecture of Selenium WebDriver

Authentication workflows are found to be areas of weakness in case studies. In a case, there was failure of tests after the initiation of OAuth-based multi-factor authentication. It was observed that the asynchronous token refresh calls were causing elements to be unavailable and that this was showing up in the DevTools rather than in IntelliJ stack traces which showed that there were null pointer exceptions [5]. By reorganizing the scripts with explicit waits and configuration optimization QA engineers put the suites back on their feet and reduced the maintenance overhead.

Therefore, an IntelliJ–DevTools coordinated plan lets retail organisations reduce disruption, speed up recovery and ensure fidelity to testing through high release rates.

V. Scalability & Optimisation

The retail QA environment needs to support test execution scale, especially during peak seasons when the stability of the application has a direct influence on revenue. To achieve thousands of runs per day Selenium test suites, scaling is necessary as well as architectural efficiency and focused optimization.
Parallel execution is a leading way to do so, which is typically performed via Selenium grid, Docker or cloud BrowserStack, Sauce Labs. With the distribution of workloads over several nodes, organisations reduce feedback by covering a wider range of browsers and devices [6].

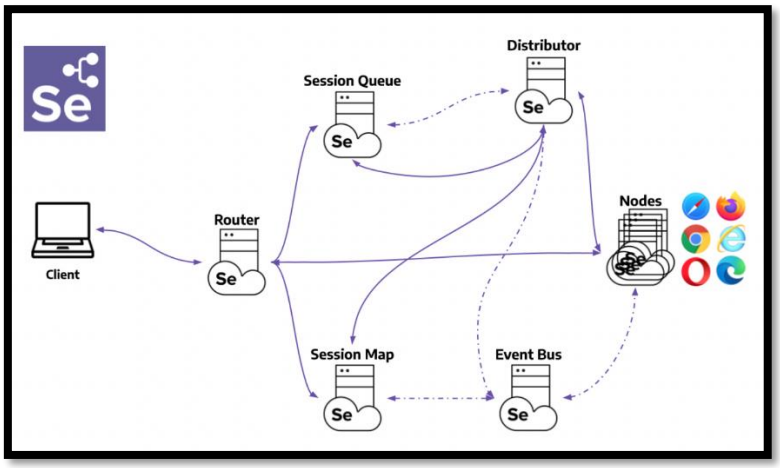

Fig 3. Selenium Grid Parallel Execution Architecture

It also is optimisation that is involved in debugging. The IntelliJ stack traces are organized in a way that allows the engineers to filter failed tests in a more effective way without increases in false positives by scale. It forms the basis of Chrome DevTools, which offers indications of bottlenecks in the runtime of a client-side environment, e.g. slow JavaScript execution or stalled network requests. In case of a checkout regression case study, IntelliJ showed the same phenomenon of repeated timeouts of elements as well as DevTools showed third-party tracking scripts that require too long to load the web pages. Those scripts would have been delayable and this would have made it more stable and the tests less flaky.

VI. Reporting & Monitoring

It is not merely that effective QA needs that failures are identified, but also how the insights are formatted and followed up. Selenium generates execution logs though these may be too small to provide a business or product decision. IntelliJ stack traces can be included in structured reporting systems to enhance the readibility of such reports, by giving a mapping of errors to a specific location within the code, and identifying script defects and actual application problems. Monitoring is improved when the test reporting includes the Chrome DevTools outputs. Raw Selenium results are unable to reflect any context, as it is presented in network logs, console errors, and performance measurements. In one of the case studies, the instability of the Selenium scripts was the source of constant checkout failure. Upon closer inspection, the DevTools logs showed API latency with a third-party payment provider, enabling the QA team to go upstream [7].

Through code-level and application-level reporting, retail organisations create a feedback mechanism that can be used to support root cause analysis, trend monitoring, and continuous improvement in their QA pipelines.

VII. Conclusion

The Selenium failures in retail QA need a two-angle viewpoint, which is a combination of IntelliJ (code-focused stack trace viewer) and Chrome DevTools (runtime visibility). The case studies explain how this combined strategy provides strength to the test reliability, scalability, and improvement of monitoring practices to guarantee that the automated pipelines are resilient in the dynamic retail circumstances.

REFERENCES:

[1] Strandberg, P.E., Afzal, W. and Sundmark, D., 2022. Software test results exploration and visualization with continuous integration and nightly testing. *International Journal on Software Tools for Technology Transfer*, *24*(2), pp.261-285.

[2] Google Chrome Developers, *"Inspect Network Activity – Chrome DevTools Documentation,"* Google LLC, Mountain View, CA, USA, 2023. [Online]. Available: https://developer.chrome.com/docs/devtools/network/.

[3] García, B., Ricca, F., del Alamo, J.M. and Leotta, M., 2023. Enhancing web applications observability through instrumented automated browsers. *Journal of Systems and Software*, *203*, p.111723.

[4] Romano, A., Song, Z., Grandhi, S., Yang, W. and Wang, W., 2021, May. An empirical analysis of UI-based flaky tests. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)* (pp. 1585-1597). IEEE.

[5] Ricca, F. and Stocco, A., 2021, January. Web test automation: Insights from the grey literature. In *International Conference on Current Trends in Theory and Practice of Informatics* (pp. 472-485). Cham: Springer International Publishing.

[6] Mattiello, G.R. and Endo, A.T., 2022. Model-based testing leveraged for automated web tests. *Software Quality Journal*, *30*(3), pp.621-649.

[7] Elsner, D., Hauer, F., Pretschner, A. and Reimer, S., 2021, July. Empirically evaluating readily available information for regression test optimization in continuous integration. In *Proceedings of the 30th ACM SIGSOFT international symposium on software testing and analysis* (pp. 491-504).