Cross-Validation of UI and API STATES in Cloud-Based test Automation: A Json vs UI Discrepancy Resolution Framework

Udayan Verma
Denver, USA

* Corresponding Author: **Udayan Verma**

Article Info

ISSN (online): 2582-7138
Impact Factor: 5.307 (SJIF)
Volume: 04
Issue: 06
November-December 2023
Received: 23-10-2023
Accepted: 20-11-2023
Published: 23-12-2023
Page No: 1546-1548

Abstract
This paper introduces a systematic approach for executing cross-validation on User Interface (UI) states and the respective JSON payloads on a cloud-based test automation platform. At Charter communications, this methodology was created to address discrepancies stemming from cache lag time, async rendering, and data transformation issues that quickly lead to the UI being out of sync with the canonical output from the API. The solution proposes to embed, within Selenium WebDriver based DOM Inspection and parallel RESTful API calls, synchronization checks. By incorporating the synchronization checks into a CI pipeline, infinite scalable cross-browser nightly regression testing is achieved. Ultimately, an efficient process which generates fast, validated defect findings is achieved.

DOI: https://doi.org/10.54660/.IJMRGE.2023.4.6.1546-1548

Keywords: UI-API Validation, Synchronized Testing, JSON Discrepancy, Continuous Integration

1. Introduction

In microservice architecture today, syncing discrepancies between User Interface (UI) and back-end API Application Programming Interface (API) is next to impossible. As long as back-end systems return the intended JSON output, the data shown on the UI is potentially stale and imprecise. This is JSON drift. This may refer also to client rendering sync timing mechanism, caching policies, transformation layers and so on. This all costs a company like Charter Communications with lasting inefficiencies in operations and risk of losing customer trust. This paper will elaborate on solution framework that provides a systematic way to correct discrepancies in a synchronized approach. The process is simply taking the DOM state with Selenium WebDriver while using the same API payload from the source of truth at the same time. This double-pronged approach that is embedded into our nightly regression suites allows us to catch synchronization errors well before we would run any tests. This decreased amount of analysis removed and the overall assurance we have with releases is increased.

2. Background and Rationale

UI - API data mismatch occurs because modern web applications have decoupled architecture design. The front-end uses complex technologies like asynchronous data loading and heavy client-side caching to improve user experience. These enhancements can work, but they do create a time delay, from the state of the data represented by the backend to the DOM state it is displayed in. Current testing approaches operate in a siloed approach, so they cannot verify the above with confidence. UI tests can only verify that the DOM state is correct, but don't verify that the data that originated the state is correct. API tests can verify that the backend business logic and data definitions are correct by the specification or agreement, but cannot ensure the value is represented properly in the DOM. Validating those two approaches, separately is not practical. Validating them both is moreover possible with both an automated, synchronized approach. Thus, we can argue that the only way we would ever know data is consistent, all the way down the stack, from service to presentation would be via.

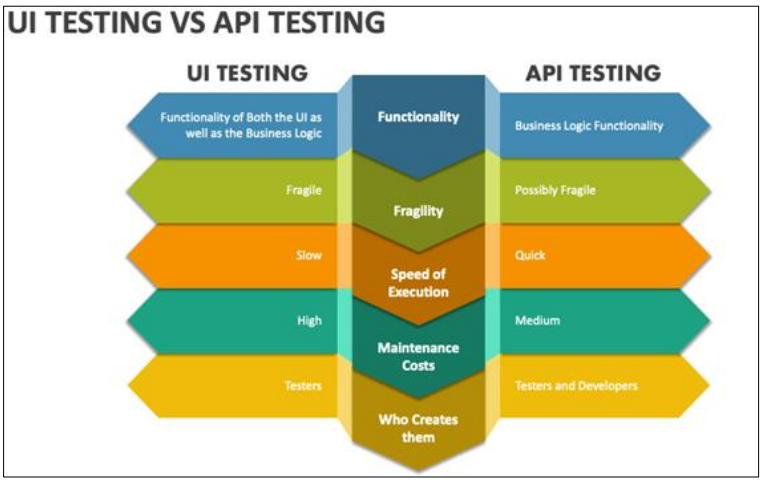

Fig 1: UI Testing Vs API Testing

3. The Discrepancy Resolution Framework: Architecture and Core Components

The core consists of three interlocking building blocks. The first block (3rd) is UI State Capture, which we utilize a selenium-WebDriver to bind our UI expectations and automation on the front end in the app. UI State Capture, the automation for creating an automation sequence, is somewhat different then UI State Capture from the User in application of design. UI State Capture captures a specific DOM element, "locates", uses a specified locator, and captures the displayed value (text, attributes, states, etc) and image at user determinable moments in time.

The 2nd building block, API Payload Copying, produces verification along with interaction with the user interface. The automation logic does this by getting to the correct REST API endpoint that would have returned back the data that is being displayed in the ui-component being tested, the logic verification gets the raw source of the json file at the service tie [1].

The last and most important component is Sync-Check Logic. This logic acts as the comparison engine that performs a systematic, field by field, comparison of the UI captured data and the JSON payload data retrieved. The Sync-Check Logic is architected to account for the needed data formatting (ex. converting epoch to regular readable dates on the UI). The logic also includes configurable thresholds for acceptance to mitigate small acceptable variances (e.g. constantly updated timestamps or calculated values). Depending on your context, this comparison engine knows when to ignore real data sync defects and undesirable but expected variances and the system will only flag the problematic data for inspection [3].

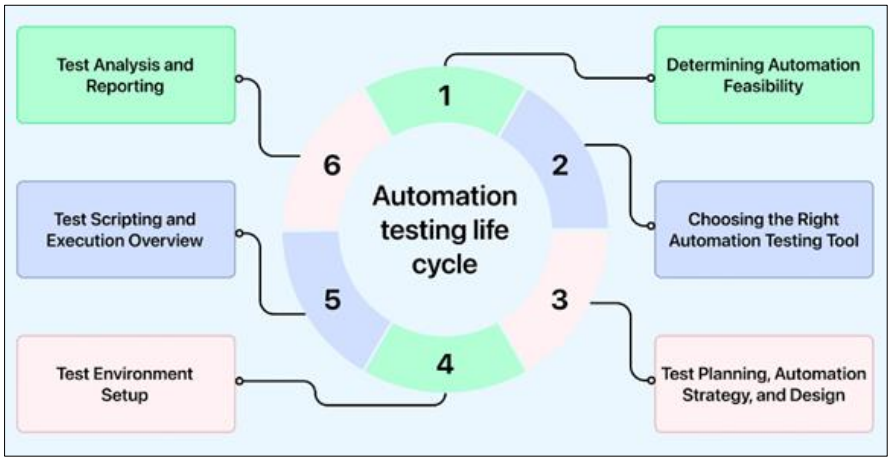

Fig 2: Automation Testing

4. Integration into the CI/CD Pipeline

The framework operates seamlessly through an embed into your existing Continuous Integration & Continuous Delivery (CI/CD) pipeline in Jenkins. The validation checks are not intended to serve as tests; they are defined to their own step in the nightly regression test suite. The Jenkins pipeline is triggering jobs defined to run based on a schedule to provide consistent automated runs. In order to provide runs for test cases for test levels in deployment, the test suite is highly parameterized [7]. So, the same test-case definition can run in some form of environment, either development, QA, or staging by just inputting in the build parameters the URL, user id/password for each environment. This does genuinely mean UI-API validation is the automated and unquestionable quality gate in the software pipeline. It gives real time and continuous feedback around the accuracy of the data you are testing and enhances the resilience of builds so ultimately you know that sync variance will never go to Production. This framework in CI/CD is not just some tool or tool, but has been intentionally developed as a part of the organizations DevOps maturity model with standards of the quality coded [4].

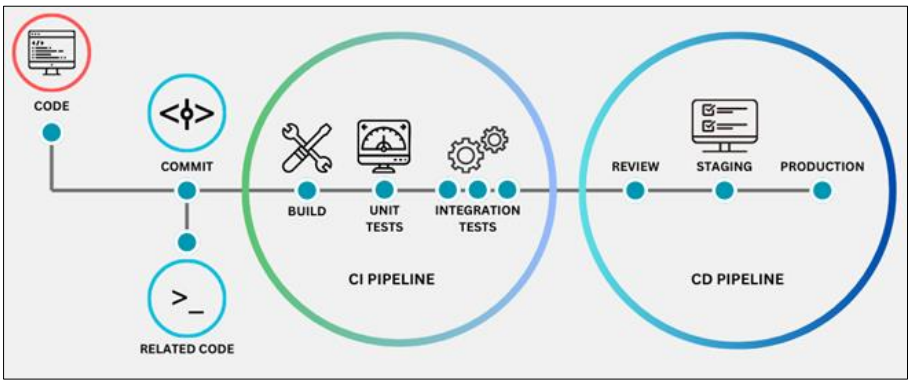

Fig 3: CI/CD

5. Scalability and Cloud-Based Execution

The framework has been designed for scalability and cloud technologies so it has provided incredible test invoking and while providing no affect to run-times. The primary intent of the framework is to invoke tests concurrently as perhaps realistically as possible with a grid of distributed browser nodes. Among other things, you can use Selenium Grid with containers so tests are actually truly running concurrently with unending matrix of browser version and OS combinations Cloud test execution provides amazing elasticity on running tests to provision or de provision resources, as appropriate, to the linear regression suite in peak times of regression testing. This on demand resource provisioning for cloud tests gives the ability to ramp resources up or down based on the financial implications.

6. Automated Reporting and Discrepancy Analysis

Where the framework brings a lot of value is in the automated reports and analysis functionalities, and exactly for defect resolution. In the case of UI vs API discrepancies, the framework generates an evidence report on its own, beyond the normal pass/fail report, it is an evidence-based report with a full diagnostics package including for each fail. The diagnostics package includes a screen capture of the user interface, and the exact validation failure point, the value copied from the DOM element and the raw JSON payload that was returned from the API. Each of these individualized reports are directly linked from the build results of CI and can easily be automated into a defect management tool or collaborative tool. This ultimately gets immediately scoped visibility to appropriate engineering teams without needing manual communications; and creates a unassailable audit trail for every defect that was raised [6].

7. Conclusion

In conclusion, the resolution framework for JSON & UI inconsistency delivers further validation for distributed applications already out in the wild. The framework offers a higher level of test consistency, zero defect detection time by legally comparing from front end to canonical back-end data, while your benefits are even more simplistic; zero time in manual check, lower false positives with justification comparison logic, and new found audacity with every release of your software. An integrity of state data, a stable back-end to UI relationship will be an eventual goal of any quality program even with released software.

References

1. Alonso JC, Martin-Lopez A, Segura S, Garcia JM, Ruiz-Cortes A. ARTE: Automated generation of realistic test inputs for web APIs. IEEE Trans Softw Eng. 2022;49(1):348-363.
2. Bajpai P, Lewis A. Secure development workflows in CI/CD pipelines. In: 2022 IEEE Secure Development Conference (SecDev); 2022 Oct. p. 65-66.
3. D'Elia DC, Nicchi S, Mariani M, Marini M, Palmaro F. Designing robust API monitoring solutions. IEEE Trans Dependable Secure Comput. 2021;20(1):392-406.
4. García JC, Bachiller P, Bustos P, Núñez P. Towards the design of efficient and versatile cognitive robotic architecture based on distributed, low-latency working memory. In: 2022 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC); 2022 Apr. p. 9-14.
5. Rangnau T, van Buijtenen R, Fransen F, Turkmen F. Continuous security testing: A case study on integrating dynamic security testing tools in CI/CD pipelines. In: 2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC); 2020 Oct. p. 145-154.
6. Türker UC, Hierons RM, Mousavi MR, Tyukin IY. Efficient state synchronisation in model-based testing through reinforcement learning. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE); 2021 Nov. p. 368-380.
7. Zampetti F, Geremia S, Bavota G, Di Penta M. CI/CD pipelines evolution and restructuring: A qualitative and quantitative study. In: 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME); 2021 Sep. p. 471-482.