SAUCELABS

WHITE PAPER

Headless Testing Primer

As testing continues to shift left in the development lifecycle, it's important to understand that different stages of the pipeline have different test requirements, and need resources that allow teams to continue delivering code quickly without creating bottlenecks. Headless browser testing has emerged as an effective way to implement early pipeline testing. It's lightweight, fast, and able to scale with development as they incorporate more testing into their workflows.

This whitepaper explains how headless testing works, when to use it, and where headless testing fits within a test strategy that includes other types of testing, including cross-browser testing.

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Effective software testing requires striking the right balance between the resources expended to run tests on the one hand, and the extent of the application delivery pipeline that is covered by tests on the other. If time and infrastructure were in infinite supply, software testers would be able to test every module of code at every stage of the delivery process. In a world where testing resources are limited, however, this is not possible.

That is why it is critical to find ways of optimizing testing resources in order to get the most test coverage out of the time and infrastructure available to your organization.

One way to do this is to take advantage of headless testing. Headless testing is a software testing technique that makes it possible to expand test coverage by running more tests, without requiring a significantly greater investment of time or hosting resources for those tests. Combining this approach with other approaches to testing, organizations that adopt this blended paradigm towards testing enable teams to perform testing earlier in the delivery cycle, thereby shifting testing left with the identification of problems in the code-line where they are easier to correct before they are merged into the main branch.

Headless testing enables a more effective and expansive process, while allowing the testing team and infrastructure your organization already has in place to be more effective.

This whitepaper explains how headless testing works, when to use it, and where headless testing fits within a test strategy that includes other types of testing, including cross-browser testing.

WHAT IS HEADLESS TESTING?

Headless testing is a technique for testing applications (or individual units of code inside applications) without displaying their visual elements. Generally, applications tested operate in a web browser with a graphical user interface, or GUI. Headless testing still tests the components, but skips the time- and resource-consuming process of rendering a visual display.

Headless testing does not mean the code determining how an application is displayed is ignored. HTML is still rendered and JavaScript is still invoked by the test scripts to determine whether layout rendering and performance goals are met.

Headless testing is not an alternative to component testing, acceptance testing, end-to-end testing, or other types of software tests; instead, it is a technique that can be used to execute these types of tests.

Headless testing can be thought of as the opposite of cross-browser testing. The latter testing technique entails testing an application by running it inside a complete web browser, including all visual components.

WHY USE HEADLESS TESTING?

Agile and fast waterfall teams are struggling to develop code quickly — let alone wait for the results of testing. However, in today's development lifecycle, it is often a requirement that both the application and the tests are developed simultaneously. This means the developer has tests already written when they write new code, but often with no way to self-service execute the test. Headless does not replace the other forms of testing, but it offers several advantages that make it a better fit for tasks that require faster feedback and more efficient use of resources.

Testing Speed

Because headless testing does not require the test environment to display the visual elements of an application, headless tests can typically execute much faster than cross-browser tests.

The exact time saved by headless testing as compared to cross-browser tests depends on how complex the visual elements are and how many resources they would require to display, but in general, testing teams can expect headless test to reduce test execution time by about 10 percent.

Another speed benefit teams will see by using headless browsers for early pipeline testing is in the increased velocity at which code moves through the pipeline. With development incorporating more tests at the component level, bugs can be found and fixed much more quickly as compared to later in the development lifecycle. This results in fewer rollbacks, and better-quality software released more quickly.

Cost and Infrastructure Savings

Skipping the step of displaying an application visually also reduces the amount of resources that each test consumes. As a result, you can run more tests at once on the same infrastructure. Ultimately, this saves you money.

Enabling Earlier Testing

Because headless tests run more quickly and require less infrastructure, it's easier to perform tests early in the development cycle.

Typically, because of the time and resources required to run cross-browser tests, testing teams wait until after code has been integrated into an application's codebase to perform cross-browser tests. Although there is no rule preventing cross-browser tests from being run earlier in the development cycle, it is often not feasible to run them prior to code integration, as it requires running many tests for many different units of code, demanding a great deal of resources.

In contrast, headless tests are well suited for running early in the development cycle, even if only small units of code need to be tested. In this respect, headless tests are a good approach for performing component testing.

It's important to note headless testing is by no means limited to component testing or other types of tests that occur early in the software delivery cycle. Headless testing can be used at any point in the development cycle, but it is particularly valuable for conducting early pipeline tests that would not be feasible to perform using a cross-browser approach. With Agile development gaining more popularity, this idea of early testing is critical to support accelerated development cycles.

Shift-left and Continuous Testing

The software industry is in the middle of a paradigm shift from a traditional Waterfall approach to Agile, DevOps, and other modern development practices. These methodologies put a priority on taking steps to know more about software quality much earlier in the delivery cycle — shifting focus further to the left. In doing so, organizations find continuous testing is critical to ensuring software can move at an accelerated pace without compromising quality. This idea has given rise to strategies such as "shift-left" and "continuous testing," and headless testing offers a fast and resource-efficient choice to support both of these strategies.

Shift-left and continuous testing revolve around the idea that to deliver quality software at the pace Agile requires, testing must be incorporated earlier, and more often throughout the pipeline. This means testing responsibility cannot fall solely on the QA organization — It also falls on the developers themselves. Enabling developers to test their code before it is merged into a master branch provides the opportunity to find and fix issues before they

have a larger and negative impact on the application downstream. This means quality code can move through the pipeline more quickly and efficiently, and developers can move on to new tasks instead of chasing down bugs.

These strategies are useful for increasing the impact of software testing. By running tests earlier in the delivery pipeline, testing teams can detect software problems in individual units of code prior to integration into the larger codebase. In turn, problems can usually be solved more easily, because issues can be isolated to a specific unit of code and resolved in isolation, without requiring changes to other parts of the application.

Similarly, because continuous testing increases the frequency at which tests are run, and allows code to be tested at all stages of the delivery pipeline, it increases a testing team's ability to identify and address problems. Not only does this help to identify issues earlier in the development cycle — It can also reveal problems that may not be evident when performing testing at a particular stage of the delivery pipeline (such as pre-integration) but that can become apparent at another stage (such as after code has been integrated and can interact with other application components).

Early pipeline testing is particularly important in today's microservices-based software architectures, which consist of multiple components that depend on one another. If a problem exists within the code for one microservice and it is not detected early in the development cycle, it could easily affect the code of other integrated microservices. This would possibly require each microservice be modified to fix the problem, wasting time and resources in the process.

WHEN NOT TO USE HEADLESS TESTING

While headless testing enables faster and more frequent tests, it is not the right fit for every testing use case. Below are four scenarios where headless testing is a poor fit for a given type of test.

Tests That Evaluate Graphical Elements

Most obviously, tests requiring visual elements to be evaluated on a display that mimics real-world conditions, or which involve visual components that cannot be represented by standard web browser code cannot be run effectively using headless testing.

For example, a test that assesses how real users interact with an application interface cannot be run in a headless browser, because doing so would not

produce an interactive display. Similarly, if the application or component being tested displays streaming video, headless testing cannot confirm the video is displayed correctly and performs adequately; it can only test whether the application passes or fails the test conditions. (A reminder — the use case for headless focuses on rapid outcomes, not detailed actionable insight. For that you need much more robust cross-browser testing.

Real-World Test Conditions

Because headless environments rarely exist in the real world, headless testing is not well suited for testing application behavior under genuine real-world conditions. Cross-browser testing is the best approach for mirroring real-world conditions within a testing environment.

Performance Troubleshooting

Because headless tests don't execute the display, they may miss performance pain points related to either page interactivity or the visual components of the application.

This does not mean headless testing should never be used in conjunction with performance testing. On the contrary, headless tests are a fast and efficient way of identifying some types of performance issues. For example, if HTML is slow to render due to inefficient code design, or network-bandwidth problems are preventing an app from responding adequately, headless testing will be able to detect the root of the problem in most cases as well as cross-browser testing. But headless tests will not be able to identify performance problems stemming from a browser struggling to display information or respond to display-based input, for example.

Some Applications Won't Work in Headless Environments

In rare cases, applications or websites are designed to detect whether they are running in a headless environment, and will refuse to load under those circumstances. This does help prevent certain types of abuse, such as attempts by attackers to overload an application with repetitive requests from a headless browser to prevent it from responding to legitimate users. This type of configuration is rare, though it may be possible to disable in a pre-production application in order to perform headless tests on the application. But, if your developers have made the choice to prevent their application from running in a headless environment, your testing team will most likely need to rely on cross-browser testing.

The latter testing strategy will work for these applications because it presents the application with a complete browser environment, avoiding the risk the application will refuse to load in the absence of a full browser.

HEADLESS TESTING REPRESENTS ONE PART OF A BROADER TESTING STRATEGY

Because headless testing doesn't address all testing use cases, it's best to think of headless as one useful testing technique to include in your toolset, but not the only one that you'll use. Cross-browser tests should almost certainly remain a part of your testing routine, as should unscripted, manual tests.

Before deciding whether to use headless testing, ask the following questions.

Do display requirements prevent me from using headless testing altogether?
If the answer to this question is yes, you simply can't use a headless test. Instead, use cross-browser testing.

How significant are the time and resources saved by running a test headlessly?
Although headless testing is faster and requires fewer resources, this is not universally the case. An application or website with minimal display elements might take about as long to test using a cross-browser test as a headless test.

If you won't actually save much time or resources by using headless testing, you might be better off sticking with a cross-browser test. The latter will deliver more test data and a deeper level of coverage.

Do we already have a script for running this test in cross-browser mode?
If your team already has a solution in place for running a particular test in cross-browser mode, you might want to keep using that test script and devote your team's time to writing headless tests that cover new territory. That way, you increase your overall test coverage, rather than devoting resources to converting cross-browser tests to headless tests (without increasing the thoroughness of your testing).

Eventually, once your test coverage has been maximized, you can shift resources to converting cross-browser tests to headless tests.

How critical is the completeness of this test?
If getting test results that reflect real-world conditions is an absolute priority, or if a test is one of the last tests you are running before production, it is wise to invest in a cross-browser test, even though it will take longer to run and place a greater load on your test infrastructure.

On the other hand, tests that run earlier and that involve components that will be re-tested multiple times before release into production might not demand such a high degree of completeness.

GETTING STARTED WITH HEADLESS TESTING

Performing headless testing is simple. The only requirement is a headless web browser, such as Headless Chromium, Headless Firefox or HtmlUnit, and the infrastructure to run it. While it's possible to perform headless testing manually by running an application inside a headless browser, in most cases, you'll want to use an automated testing framework to perform headless tests quickly and at scale. Selenium WebDriver, an open source automated testing framework, offers broad compatibility with most headless browsers and types of tests. Other frameworks, including Watir and Serenity, are also compatible with headless tests, although the set of use cases they support is not as broad.

To make headless testing easier, Sauce Labs recently introduced a headless testing platform, Sauce Headless. This new product provides a turnkey solution for spinning up headless Chrome or Firefox instances on Sauce's cloud testing infrastructure, and is perfect for developers who want to incorporate testing earlier in their pipeline.

Sauce Headless saves testing teams from having to set up headless browsers or host infrastructure, placing the focus on tests and developing, coding, and creating new applications. With more insight earlier in the development process, organizations will benefit from code and application branches that are cleaner, more reliable, and more usable across the organization. Combined with a traditional cross-browser testing platform, Sauce Headless helps achieve continuous testing throughout the software development pipeline. To learn more, please visit the Sauce Labs website.

SAUCELABS

ABOUT SAUCE LABS

Sauce Labs is the leading provider of continuous testing solutions that deliver digital confidence. The Sauce Labs Continuous Testing Cloud delivers a 360-degree view of a customer's application experience, ensuring that web and mobile applications look, function, and perform exactly as they should on every browser, OS, and device, every single time. Sauce Labs is a privately held company funded by Toba Capital, Salesforce Ventures, Centerview Capital Technology, IVP, Adams Street Partners and Riverwood Capital. For more information, please visit saucelabs.com.

saucelabs.com/signup/trial

FREE TRIAL

SAUCE LABS INC. - HQ 116 New Montgomery Street, 3rd Fl San Francisco, CA 94105 USA