WHITE PAPER

HEADLESS TESTING - WHEN, WHERE AND HOW?

A BRIEF ON BEST PRACTICES TO ENSURE EARLY PIPELINE TESTING SUCCESS

As testing development organizations look to shift left and integrate more quality checks into their software delivery pipeline, headless browser testing has emerged as an effective strategy to increase release velocity and improve productivity. By providing developers with a lightweight and easily scalable infrastructure, headless browser testing allows for fast feedback on the code quality, and keep features moving through the pipeline quickly. However, it is important to understand the specific use case for headless browsers, and how to effectively integrate them into a larger continuous testing strategy. This technical paper will answer three key questions teams must answer when considering headless browsers as an option, and offer practical advice on how to successfully implement early pipeline testing.

SAUCELABS

TABLE OF CONTENTS

INTRODUCTION

As testing development organizations look to shift left ("test early and often") and integrate more quality checks into their software delivery pipeline, headless browser testing has emerged as an effective strategy to increase release velocity and improve productivity. By providing developers with a lightweight and easily scalable infrastructure, headless browser testing allows for fast feedback on the code quality, and keep features moving through the pipeline quickly.

However, this also means that teams that have traditionally implemented testing by a QA function at the end of the development pipeline are finding new challenges as they attempt to shift quality left to the developer. These include:

- How do I get developers to test their own code, or even write their own tests?
- Which of my existing tests can be run in headless browsers?
- What about mobile?

This technical paper will answer these three questions, and offer practical advice on how to successfully implement early pipeline testing with headless browsers.

DEVELOPERS AND TESTING - A PERFECT MATCH

In the old days of software development, there was a very tight connection between developers and the quality of their code. With the rise of "shift left" testing methods, this culture is starting to make its return. Now more than ever, developers are being asked, and in some cases required, to take responsibility for the quality of the code that they write, and ensure that it works as expected before merging into the larger pipeline.

This doesn't mean that traditional QA is going away. To the contrary, seasoned QA professionals, especially those with automation experience, are leading the charge to implement some of these cultural shifts. In modern development organizations, testers are often embedded in Agile teams, or are seen as subject matter experts who train multiple groups on testing best practices. QA professionals are invaluable to development organizations that want to shift left because of their knowledge on how the application is supposed to behave in its entirety, along with their focus on the end-user and understanding their behaviors. Because of this, they often become the stewards of quality in the organization, and are uniquely positioned to lead

the charge in enabling developers to test earlier and more often, leading to fewer bugs deployed to production, and increasing release velocity.

Providing development teams with headless browsers to test against can ease that adoption barrier. By giving them access to dedicated resources that are lightweight and easily scalable, headless allows for fast feedback on code quality, and can show developers the benefits of testing earlier in the pipeline - fewer rollbacks, faster releases, and improved productivity.

HEADLESS VS CROSS-BROWSER TESTS

Since they are a more lightweight, cost-effective solution, some might consider moving all of their functional tests to headless browsers. However, it is important to remember that headless browsers serve a much different purpose than full, cross-browser testing. Headless browsers are perfect for developers who are working on smaller components of an application, and gives them the basic infrastructure they need to confirm the question, "Does my code work as expected?" Further along in the pipeline, after everything as been integrated, QA or SDETs need to run their end-to-end regression tests on every browser and operating system available to mimic real-world conditions and confirm, "Does the application in its entirety work as expected?". This distinction means that complete continuous testing pipelines will contain a mix of headless and full cross-browser testing strategies that give everyone the tools they need to run their tests at scale, without slowing down pipeline velocity.

When expanding testing to include early pipeline component tests or sanity checks, many teams also ask the question, "Should I just take all of my UI and regression tests and run them on headless browsers?" And while the answer to this question varies from team to team, there are some things to consider broadly.

First, we should eliminate the thought process of "headless tests" versus "cross-browser tests." Instead we should realize that there is quite a bit of overlap in the types of tests that developers should implement as part of their workflow. Remember, headless browsers are great for early pipeline testing because they provide fast feedback and answer the question: does my code work as expected before I merge? To that end, consider what already existing tests you have, and whether they would be good candidates to shift left with headless browsers. In general, these tests should have a few key traits:

1. **Short** - if the key to early pipeline testing is to give fast feedback, it is imperative that the tests that developers run should be capable of doing so. Long tests lead to long wait times for them to pass, effectively counteracting the benefits of headless testing.

2. **Atomic** - because we are testing smaller components of an application, it's important that early pipeline tests are focused on a single function.

3. **Autonomous** - in the same vein, autonomous tests are great candidates for shifting left because they ensure that there are no dependencies (either within the application or outside of it). The more complicated a test becomes, the more chance for flakes, timeout errors, or long run times (which translate to long wait times for feedback).

Of course, we recommend you follow these best practices with all of your tests throughout your development pipeline. But especially in the case of early pipeline testing, it's crucial that these tests be optimized to give developers fast feedback. This not only keeps stable code moving through the pipeline smoothly, it also breaks down barriers that prevent developers from adopting testing by giving them the fast feedback they require.

```
import AppHeader from '../../../page-objects/appHeader';
import InventoryListScreen from '../../../page-objects/inventoryList';

describe('Parallelization - Inventory list', () => {
    beforeEach(() => {
        // Load the url
        browser.url('');

        // Set the storage
        browser.execute('sessionStorage.setItem("session-username", "standard_
user")');

        // Now got to the inventory page
        browser.url('/inventory.html');
    });

    it('should validate that all products are present', () => {
        // Wait for the inventory screen and check it
        expect(InventoryListScreen.waitForIsDisplayed()).toEqual(
            true,
            'Inventory List screen was not shown',
        );
        expect(InventoryListScreen.swagItems.length).toEqual(
            6,
            'Amount of items was not equal to 6',
        );
    });

    it('should validate that a product can be added to a cart', () => {
        // Wait for the inventory screen and check it
        expect(InventoryListScreen.waitForIsDisplayed()).toEqual(
            true,
            'Inventory List screen was not shown',
        );
        expect(AppHeader.getCartAmount()).toEqual(
            '',
            'The amount of cart items is not equal to nothing',
        );

        // Add an item to the cart
        InventoryListScreen.addSwagItemToCart(0);
        expect(AppHeader.getCartAmount()).toEqual(
            '1',
            'The amount of cart items is not equal to 1',
        );
    });
});
```

The above is a typical test that a developer would choose to run, pre-commit, through a headless browser. Set to run against the Sauce Labs demo app, this test succinctly checks a single function in a component of a retail web application, "add item to cart". Running this test on a cross-browser platform would still give the same feedback, but the developer might be resource constrained, as QA uses most cross-browser infrastructure for end-to-end regressions. Using headless browsers, a developer can get the basic feedback on the quality of their code without having to wait in line, and then check in their code with confidence knowing that this particular component works before merging into the master.

HEADLESS BROWSERS AND MOBILE

Headless browsers are obviously a great tool for testing desktop web applications. But what about mobile? Of course, there isn't a way to use headless browsers for native mobile applications, but is a use case for using headless for mobile web testing.

Kwo Ding's Mobile Testing Pyramid provides a practical strategy for building mobile test automation, along with insight into what kind of infrastructure requirements you need for testing your apps throughout the pipeline. While real devices are still necessary to mirror real-world conditions, Ding recommends re-sized browser windows can offer a scalable and cost-effective solution when testing earlier in the pipeline. To that effect, headless browsers' responsive designs allows for playing with user agents and checking for compatibility with different screen sizes at the click of a button, or by dragging to resize, and are ideal for quick sanity checks or component tests.

So while there is still a need for emulators, simulators, and real devices at different points in the pipeline, headless browsers can offer a scalable and cost effective solution for validating functionality earlier in the pipeline for all web applications, no matter the screen size.

CONCLUSION

When considering implementing headless browsers, remember that they are only a single tool in a larger toolbox of test platform options. None of your users are navigating to your apps with a headless browser, and so there is still a need for cross-browser testing and real devices for E2E regressions and other tests later in the pipeline to mimic real-world conditions. Headless browsers have a specific place in the pipeline to help developers embrace "shift left" testing. By providing them with their own scalable infrastructure, they are able to get fast feedback on the quality of their code, keep stable features moving through the pipeline faster, and deploy with confidence knowing that they can continue developing new features without fear of rollbacks. This allows for every team in the organization to deliver on their core promise - delight users with high quality applications.

Sauce Labs has recently released a headless browser testing platform - Sauce Headless. To learn more, visit our website or contact our team to get started with a free trial.

ABOUT SAUCE LABS

Sauce Labs ensures the world's leading apps and websites work flawlessly on every browser, OS and device. Its award-winning Continuous Testing Cloud provides development and quality teams with instant access to the test coverage, scalability, and analytics they need to rapidly deliver a flawless digital experience. 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

SAUCELABS

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

SAUCE LABS EUROPE GMBH
Stralauer Allee 6
10245 Berlin DE

SAUCE LABS INC. - CANADA
128 West Pender Street, 8th Floor
Vancouver, BC V6B 1R8, Canada

SAUCE LABS - POLAND
Złota 59 St., 4th Fl.
00-120 Warsaw, Poland