LogRocket Frontend Monitoring

START MONITORING FOR FREE

BLOG

Cypress vs. Selenium: Why Cypress is the better option

January 1, 2021 · 8 min read

The problem with Selenium

Editor's note: This post was updated on 19 January 2021 to reflect the changes and improvements introduced with Cypress 6.1.0.

Before I start, I want to emphasize that this post is not about one particular project or any automation testers that I have worked with. I have seen this behavior in three recent projects, and nearly every automation tester that I have worked with has busted a gut to make this faulty machine work.

I am fairly sure that a memo has gone out to every contract that I have worked on recently stipulating that a million automation tests are required to guarantee success. We must not stop to question the worth of these tests. We must protect them like our children.

These tests must be written in Selenium, despite nearly everyone having a pretty grim experience due to the inherent known issues that I will state later. According to their docs, Selenium provides a range of tools and libraries to support the automation of web browsers and provides extensions that emulate user interaction with browsers, as well as a distribution server for scaling browser allocation. It also has the infrastructure for implementations of the W3C WebDriver specification that lets you write interchangeable code for all major web browsers.

Selenium tests are insanely challenging to write, but we won't let that hold us back. Instead, we will get our testers who have maybe come into programming late or are new to development. We'll get these less experienced developers to write these difficult tests.

Selenium tests might be difficult to write, but they are straightforward to copy and paste. This, of course, lead to all sorts of problems.

We often hear, "If it moves, write a Selenium test for it". Automation tests must be written for the API, the frontend, the backend, the middle-end, the happy path, the sad path, the upside-down path, etc.

We won't have any time for manual testing, and how could we? We have all these flakey Selenium tests to write and maintain. We are already late for this sprint, and every story must have an automation test.

After a year or so and an insanely long build, we will decide that this was a bit silly and delete them all. Or worse — start again.

Why does everyone still use Selenium?

I think I would be closer to understanding the true nature of our existence if I could answer the above question. All jokes aside, why is the use of Selenium so widespread? It does stagger me, but here are a few suggestions:

- It is the industry standard, lots of online resources and a vast community to lean on
- It works across multiple OS, and multiple languages, your language and platform of choice are more than likely covered
- Cross-browser testing. Selenium supports all the major browsers so you could test on Chrome, Firefox, Safari, IE, Edge, and many more

To be fair, the sudden surge of writing a million acceptance tests is not Selenium's fault. For my money, the correct number of automation tests is *one* happy path test, no sad paths or upside-down paths. This one test is a smoke test to ensure that our system is open for business.

Unit tests and integration tests are cheaper to run, implement, and maintain and should be the bulk of our tests. Has everyone forgotten about the test pyramid?

Reasons why Selenium may not work for you

The problems with Selenium can be expressed in one word: *timing*.

Before we can even start writing code to assert that our test is correct, we need to ensure that whatever elements we need to interact with are visible and are in a state to accept simulated input. Remote API calls will need to have resolved, animations and spinners need to have concluded. The dynamic content that now makes up the majority of our apps will need to have finished rendering from the currently retrieved data of the API calls.

So what do we do while this macabre pantomime of asynchronicity is occurring? How do we stop our tests from just finishing or bottoming out because a particular text input is disabled until an API call has finished or a beautiful SVG spinner overlay has put a veil of darkness over our virtual world?

In layman's terms, we wait for the HTML elements to be in a ready state. In Selenium speak, we write many custom `waitForXXXXX` code helpers, e.g.

`waitForTheFullMoonAndTheWereWolvesHaveFinishedEating` or more realistically…

```
wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath("//inpu
```

One of the worst crimes to commit is to use `Thread.sleep`. This is a heinous crime where a random number is plucked from thin air and used as a wild guess for when we think the UI is in a ready state. Please, never do this.

Below are my all-time favorite Selenium exceptions that I have found while wading through a CI build report:

- `NoSuchElementException` – move along, you'll not find your input here
- `ElementNotVisibleException` – this cheeky scamp means you are tantalizingly close but not close enough, it is in the DOM, but you can't do a single thing with it
- `StaleElementReferenceException` – the element has finished work for the day and gone to the pub. Please try again tomorrow
- `TimeoutException` – you could wait until the end of time and whatever you are trying to do is just not going to happen. You just rolled a seven

Behold: The flake

One of the most soul-destroying moments that I have experienced is having a build fail due to a failing automation test only for it to magically pass by just rerunning the build again. This phenomenon or zombie automation test is often referred to as *a flake*.

The main problem with *the flake* is that it is non-deterministic, which means that a test can exhibit different behavior when executed with the same inputs at different times. You can watch the confidence in your regression test suite go up in smoke as the number of non-deterministic tests rises.

A flakey test is more than likely down to timing, latency and the macabre opera of asynchronicity that we are trying to tame with our `Thread.sleep` and `waitForAHero` helpers that we need to keep writing to try and keep sane.

Just think how much easier this would be if we could somehow make all this asynchronous programming go away and if our world started to behave linearly or synchronously. What a natural world to test we would have.

Cypress.io sets out to do just that.

Cypress.io: The replacement for Selenium

What is Cypress?

Cypress is a JavaScript-based framework for end-to-end testing. It’s built on top of Mocha and runs in the browser, enabling asynchronous testing. According to the Cypress docs, Cypress can help you write integration tests and unit tests in addition to end-to-end tests.

Cypress includes the following features:

- **Time travel:** Cypress takes snapshots as your tests run
- **Debugging:** Readable errors and stack traces make debugging easier
- **Automatic waiting:** Automatically waits for commands and assertions before moving on
- **Spies, stubs, and clocks:** Verify and control the behavior of functions, server responses, or timers
- **Network Traffic Control:** Control, stub, and test edge cases without involving the server
- **Screenshots and videos:** View screenshots taken automatically on failure, or videos of your entire test suite when run from the CLI
- **Cross browser Testing:** Run tests within Firefox and Chrome-family browsers (including Edge and Electron) locally

The differences between Cypress and Selenium

One of the main differences between Cypress.io and Selenium is that Selenium executes in a process outside of the browser or device we are testing. Cypress executes in the browser and in the same run loop as the device under test.

Cypress executes the vast majority of its commands inside the browser, so there is no network lag. Commands run and drive your application as fast as it is capable of rendering. To deal with modern JavaScript frameworks with complex UI's, you use assertions to tell Cypress what the desired state of your application is.

Cypress will automatically wait for your application to reach this state before moving on. You are completely insulated from fussing with manual waits or retries. Cypress automatically waits for elements to exist and will never yield you stale elements that have been detached from the DOM.

This is the main take away. Cypress has eliminated the main problem with Selenium by executing in the same run loop as the device. Cypress takes care of waiting for DOM elements to appear.

I repeat: Cypress takes care of all this waiting business. No `Thread.sleep`, no `waitForTheMoon` helper. Don't you see what this means?

To really grasp how good this is, you have to have experienced the pain.

Below are a few examples of Cypress tests.

One thing synonymous by their absence is any timing or obscene `waitFor` helpers:

```
context("Login", () => {
  beforeEach(() => {
    cy.visit("localhost:8080/login");
  });

  it("can find and type in email", () => {
    cy.get("#email")
      .type("fake@email.com")
      .should("have.value", "fake@email.com");
  });

  it("can find and type in password", () => {
    cy.get("#password")
      .type("fakepassword")
      .should("have.value", "fakepassword");
  });

  it("will fail when type invalid user credentials", () => {
    cy.get("#email").type("fake@email.com");
```

I like these tests. They clearly state their purpose and are not obfuscated by code that makes up for the limitations of the platform.

Below are some tests I wrote to run the axe accessibility tool through Cypress:

```
import { AxeConfig } from "../support/axeConfig";

describe("Axe violations", () => {
  beforeEach(() => {
    cy.visit("/");
    cy.injectAxe();
  });

  it("home page should have no axe violations", () => {
    cy.configureAxe(AxeConfig);
    cy.checkA11yAndReportViolations();
  });
});
```

And here is a similar test using `webdriver`:

```
// in e2e/home.test.js
import assert from 'assert';
import { By, until } from 'selenium-webdriver';
import {
    getDriver,
    analyzeAccessibility,
} from './helpers';

describe('Home page', () => {
    let driver;

    before(() => {
        driver = getDriver();
    });

    it('has no accessibility issues', async () => {
        await driver.get(`http://localhost:3000`);

        // The dreaded wait until.  Abandon hope
        await driver.wait(until.elementLocated(By.css('h1')));
```

The main striking difference and the worrying thing to me is the latency. There are two `await` calls and the dreaded `wait(until.elementLocated)`. This is a simple test, but the more interactions you have, the more `waitFor` helpers you will need, and the flakiness starts spreading.

Here's a tutorial for writing end-to-end tests in Cypress if you're interested in learning more.

JavaScript all the way down

Cypress is clearly aimed at the frontend developer. Installing Cypress is a breeze and performed via your favorite package manager choice: npm or yarn.

```
npm install cypress --save-dev
```

It really could not be any easier. Compare that with downloading the Chrome WebDriver and friends in the world of Selenium.

There is no multi-language support like Selenium. You can have any programming language you like as long as it is JavaScript or TypeScript.

Cypress cons

Of course, there are drawbacks, and some of them are notable so I would be remiss not to list these.

- Cypress is relatively new, and it does not have the vast community that Selenium does
- As stated earlier, it's JavaScript or bust. You won't write Cypress tests in the tired old static languages of C# and java

- Because it runs inside the browser, you won't be able to support multiple tabs

It's also important to note that Cypress does not support native mobile apps. However, you can use Cypress to test some functionality of mobile web browsers and test mobile applications that are developed in a browser using frameworks like Ionic.

Will Cypress replace Selenium?

As much as I would like to say yes, I have my doubts. There is an army of automation testers who have not known any other world than selenium, and it may be difficult to move away from soon.

Testing is just the beginning – ensure passed tests mean happy users

While Cypress introduces a compelling new testing framework, it's important to take testing one step further. LogRocket monitors the entire client-side experience of your application and automatically surfaces any issues (especially those which tests might have missed). To gain valuable insight into production environments with frontend monitoring, try LogRocket.

https://logrocket.com/signup/

LogRocket is like a DVR for web apps, recording literally everything that happens on your site. Instead of guessing why problems happen, you can aggregate and report on performance issues to quickly understand the root cause.

LogRocket instruments your app to record requests/responses with headers + bodies along with contextual information about the user to get a full picture of an issue. It also records the HTML and CSS on the page, recreating pixel-perfect videos of even the most complex single-page apps.

Make performance a priority – Start monitoring for free.

Conclusion

As I stated at the start of this article, my experience with automation testing is not a good one. A lot of money, time, and pain are spent keeping thousands of hard-to-maintain tests afloat for a less-than-gratifying payout. Automation testing has only ever guaranteed a long CI build in my experience.

We, as developers, need to be better at automation testing. We need to write fewer tests that do more and are useful. We've left some of the most difficult code to write to some of the least experienced developers. We've made manual testing seem outdated when, for my money, this is still where the real bugs are found.

We need to be sensible about what automation testing can achieve.

Cypress is great because it makes things synchronous. This eliminates a whole world of pain, and for this, I am firmly on board. This, however, is not the green light to write thousands of Cypress tests. The bulk of our tests are unit tests with a layer of integration tests before we get to a few happy path automation tests.

This, of course, is far too sensible a strategy ever to happen.

LogRocket: Full visibility into your web apps

LogRocket is a frontend application monitoring solution that lets you replay problems as if they happened in your own browser. Instead of guessing why errors happen, or asking users for screenshots and log dumps, LogRocket lets you replay the session to quickly understand what went wrong. It works perfectly with any app, regardless of framework, and has plugins to log additional context from Redux, Vuex, and @ngrx/store.

In addition to logging Redux actions and state, LogRocket records console logs, JavaScript errors, stacktraces, network requests/responses with headers + bodies, browser metadata, and custom logs. It also instruments the DOM to record the HTML and CSS on the page, recreating pixel-perfect videos of even the most complex single-page apps.

LogRocket: Debug JavaScript errors more easily by understanding the context

Debugging code is always a tedious task. But the more you understand your errors the easier it is to fix them.

LogRocket allows you to understand these errors in new and unique ways. Our frontend monitoring solution tracks user engagement with your JavaScript frontends to give you the ability to find out exactly what the user did that led to an error.

LogRocket records console logs, page load times, stacktraces, slow network requests/responses with headers + bodies, browser metadata, and custom logs. Understanding the impact of your JavaScript code will never be easier!

Try it for free.

Try it for free.

Share this:

Twitter Reddit LinkedIn Facebook

Paul Cowan Follow

Contract software developer.

#vanilla javascript

43 Replies to "Cypress vs. Selenium: Why Cypress is the better option"

Erica Kane Says:
July 30, 2019 at 7:44 am

Reply

Write Selenium tests properly — with a proper class for pages that can encapsulate most of the complexities, and then additional tests become a breeze, fragility addressed in one place. Use the object oriented properties of those "old" enterprise languages. The author lost me anyway by saying "only test the happy path." If you believe in a world where errors never happen, and want to irritate your customers who hit them even more, then go ahead and serve up broken web pages. I will stick with a well-tested system.

Dagda1 (@Dagda1) Says:
July 30, 2019 at 8:49 am

Reply

Hi Erica,

when I said only test the happy path, I meant only write acceptance tests for the happy path. The vast majority of our tests should be fine grained inexpensive unit tests with a layer of integration tests. Acceptance tests with selenium are too brittle and too expensive to maintain.

Erica Kane Says:
July 31, 2019 at 9:01 am

Reply

The problem is brittle tests, not their usefulness. I assure you that testing of user experience during error handling matters. Other types of tests, while also important, are not enough.

Fix your brittle tests — write a proper Selenium Page, encapsulate common user functions, and see how easy it is to write robust tests.

GB Says:
July 30, 2019 at 8:43 am

Reply

Cypress.io. Just what Selenium used to be, like 15 year ago: A sandbox contained JavaScript library, embedded in your page. Now,w use Webdriver/Wire Protocol technology.

No multi-browser testing, regardless of how much HTML and JavaScript have evolved, still ends on products that work on Chrome but not on Firefox. I have seen this happening regularly for the last 20 years.

AD Says: Reply
July 30, 2019 at 9:01 am

Well written but my biggest complaint is that you make judgement with your title but don't really make that point. It should be a question.

Cypress has its uses but I don't think it's going to be the tool to end all tools, specifically selenium. Selenium written with custom attributes and proper encapsulated classes be they page level or component level can be a very effective and valuable tool still.

Cypress is doing some sort of waiting, it's just not explicitly done by the client wiring up the tests. What does Cypress do if it truly can't find the element you are looking for? It probably waits up to a certain time and fails.

Milos Miletic Says: Reply
July 30, 2019 at 9:05 am

I think that you didn't understood the author.
"Only test the happy path" – means that UI should be automated as less as it could. Much more to be covered in unit and integration tests, and then just automate the happy path with UI. I agree with this part.

Dagda1 (@Dagda1) Says: Reply
July 30, 2019 at 9:26 am

absolutely Milos. You got it

Nick Mennen Says: Reply

July 30, 2019 at 9:06 am

I agree with the other commenter. Use page objects and proper wait logic and selenium isn't that difficult. This article also left out a very big concern of mine for Cypress. Cypress uses semantic click and keystroke events through the DOM API, and doesn't actually interact with the UI itself. For a quality acceptance test Cypress won't work due to this constraint. Cypress is great for a front end developer integration test, but I cannot see it's place for a quality group.

Dagda1 (@Dagda1) Says: Reply
July 30, 2019 at 9:28 am

cypress runs in the same run loop as the browser, there is no waiting, this is the kicker

Dagda1 (@Dagda1) Says: Reply
July 30, 2019 at 4:04 pm

Hi Add, Cypress runs in the same run loop as the browser, there really is no waiting https://docs.cypress.io/guides/overview/key-differences.html#Flake-resistant

Dagda1 (@Dagda1) Says: Reply
July 30, 2019 at 4:15 pm

Hi Nick, I think wait logic is a bit of an oxymoron here. Selenium executes at break neck speed and not anything remotely like a user. you simply have to add lots of waitForXXX helpers. network latency in the CI environment quickly lead to non deterministic tests. you are left with a bunch of brittle and hard to maintain tests that make change really difficult, hence my call for 1 happy path acceptance test is sensible but I can see rushing to call this heresy by people who make a living adding a mountain of these tests.

Filip Says: Reply
July 30, 2019 at 5:06 pm

So.. how does this compare to testcafe?

Nef Says: Reply
July 30, 2019 at 7:01 pm

I think you are blaming selenium for developers bad practices in most cases. The waits are there for a good reasons. Im sure there is a thread.sleep buried in your dlls.

Mansoor Shaikh Says: Reply
July 31, 2019 at 1:55 am

The title should be a question mark. The content has not convinced me that it is a selenium killer.

Benjamin Bischoff Says: Reply
July 31, 2019 at 7:48 am

From my point of view this article makes the wrong assumption from the start. It is not Selenium's problems if tests are flaky – you need a good test framework to handle this well. Also he totally dismisses the point that Webdriver is now a W3C standard that vendors have to follow. Of course, if you have to test in Chrome only, it will be much faster as it is running inside of it. The same applies if you use an ios or android specific test framework as opposed to a generic one for all.

Bruce Williams Says: Reply
August 2, 2019 at 12:26 am

How does that "logon error" test work? Clicking the submit button is likely to kick off some asynç work, but you check for the error message immediately after clicking it.

Techgirl1908 Says: Reply

August 2, 2019 at 4:42 am

That's a pretty substantial list of drawbacks and the only pro that I take away from this article is that I don't have to explicitly handle any waiting in my code. I'd gladly add the one-liner wait statements here and there as a tradeoff for that laundry list of drawbacks.

Benjamin Bischoff Says: Reply
August 2, 2019 at 9:08 am

Well said 🙂

Jordan Says: Reply
August 5, 2019 at 5:45 pm

Cypress waits for a page load event before firing the next enqueued command. In this case either a toast message is displayed and cypress will find it or the page would attempt to log in and be redirected back to a failed login page. He kind of glossed over how being in the same run-loop as the application gives Cypress access to the network. Thus it can ensure (unless your app doesn't send a page load event) in most cases that next command waits for the proper state.

Jordan Says: Reply
August 5, 2019 at 5:54 pm

Also in defense of a different way: POM abstraction in most large case test suites tends to be a detriment for upkeep and onboarding. I can't tell you how many times an entire suite is thrown away because reverse engineering it can be maddening for a new SDET. So to those who think page objects solves the selenium problem are missing the point. Cypress stresses quick test creation and state management + easy ci/cd setup.

Dagda1 (@Dagda1) Says: Reply
August 6, 2019 at 10:26 am

well said Jordan

Michael B. Says:
August 8, 2019 at 11:07 pm

Reply

Cypress.io is not a replacement or killer for Selenium WebDriver it's just another cool and trending tool to use, which looks awesome and really have a lot of cool features. One of the big problems of cypress is inability to do cross browser testing, so currently only chrome browser and electron framework are supported.
Selenium WebDriver on the other hand is a W3C standard (https://www.w3.org/TR/webdriver1/) and all the major web-browsers creators supporting it by developing dedicated WebDrivers which are W3C standard compliant.
Now cypress would like to support all the other browsers in the future, and they figured out that to do this they will need to use the same dedicated WebDrivers which Selenium is currently using (https://github.com/cypress-io/cypress/issues/310).

Will Says:
September 3, 2019 at 7:53 am

Reply

However, cypress has lots of limitations, for instance you can't upload or download files, no multiple tabs or iframes support. While Cypress is a good tool for components testing, for general end 2 end tests I prefer Puppeteer from Google Chrome. I also use CodeceptJS which has a very simple syntax and brings cypress-like experience to Pupteer.

Aron Says:
October 2, 2019 at 11:59 am

Reply

Cypress does have a lot of limitations, however some of which you mentioned aren't or can be circumvented at least.

You can upload files by trigger a drop event on the desired input field (https://github.com/cypress-io/cypress/issues/170#issuecomment-311196166)

And you can download files and make assertions on them
You can catch the name of the downloaded file by catching the event https://docs.cypress.io/api/events/catalog-of-events.html#Event-Types

And since cypress runs on node you can read directories and make assertions on the existence of the downloaded file in the appropriate folder.

Imo cypress won't replace selenium, but it is an easily graspable and implementable tool which can ease development for the developers, since they can debug the app during the tests

Rich Moss Says:
October 22, 2019 at 11:30 am

Reply

The lack of Cross Browser support is a deal killer, and will also be a Cypress killer if they don;t figure it out.

Sarang Says:
November 10, 2019 at 7:46 pm

Reply

I agree with this. In my current project first end to end test was developed in 2 hours by newly joined SDET. With existing POM selenium framework it would have been min 2 days to understand the framework then start using existing "reusable" methods to develop first script.

Sean Says:
November 27, 2019 at 8:34 am

Reply

Cypress is not a Selenium killer, rather it adds to the stack of tools to use. Cypress is very good at quickly testing components on a page and proving that they work. You can then use Selenium to test the e2e flows through the web app hitting the top 3-5 flows that users take. With less flows through the app the Selenium framework can be smaller and therefore less complicated. If there is a failure you know it is to do with the component integration or (if you are doing cross-browser testing) the browser's implementation as the cypress tests have already tested the logic.

Kelvin Says:
January 3, 2020 at 6:40 am

Reply

This comment is absolutely true for me. These tools are complimentary. Cypress needs to work across browsers though for QA Engineers to be engaged with it.

Jan Vondrouš Says:
January 6, 2020 at 7:01 am

Reply

true true

Wpgens Says:
February 13, 2020 at 5:31 am

Reply

I see that most comments are regarding the Cypress only testing in Chrome, now with 4.0, Cypress supports Edge and Firefox. Article should be updated.

Dagda1 (@Dagda1) Says:
February 13, 2020 at 2:27 pm

Reply

nice one, thanks for the heads up

Anu Says:
February 23, 2020 at 5:48 pm

Reply

I can still see my cypress test runner, running only on chrome or electron browsers. Can you please provide any more documentation regarding this please. That would be appreciated..

Pushpanjali Says:
April 22, 2020 at 10:14 am

Reply

I run through all the comments , and i am more confused now 🙁 I am an automation tester, currently working with selenium web driver but i want to replace the existing selenium test framework with Cypress, can i go ahead with this ?

Robert Maclean Says: Reply
June 18, 2020 at 9:00 am

I’m pretty sure this piece is not that relevant anymore. I’ve used new versions of Selenium IDE. It’s flexible, you barely need any code, everyone can work with it. It’s great.
I’ve been creating Automated Tests for a few years, using different methods and yes, “oldschool” Selenium is quite time-consuming.

I’ve been looking for differences between Selenium IDE and Cypress.io to see which works best, but thusfar I don’t see it.
I’ve set up a project with a lot of tests which can be run simultaneously. I’ve created a PowerShell to handle them in TFS and create useful test output. It was easy, fast and security-wise it seems to be the best option as well.

Can someone convince me Cypress.io is fast to learn and better to use?

Gabriels Says: Reply
November 10, 2020 at 4:40 am

Well, I rarely use waits while automating a very old and very bad written application.
Using POM for better test clarity and only disabling waits where needed to speed up execution, with perhaps adding some waits where we switch between apps or display very long lists (we don’t have 100% control of test data!).
Newest Selenium has build in waits which solves all the problems earlier versions had.

Its good, when used properly. And yes it needs to be maintained, both TF and tests.
I just think the author isn’t doing his selenium the right way, hence the mess he is experiencing.

Gabriels Says: Reply
November 10, 2020 at 4:45 am

The moment you write first ten tests and you end up with enough objects and methods that writing 40 new equals to creating 40 new xml files with test data and maybe a single new line of code per test, you will know you did you abstraction the right way.
Naming conventions and object pages organisation is the key. If you don't have these, you will end up with a mess of illegible code. Do POM the right way and it will lead you to the success.
This is important. Proper conventions, clean code, code reviews and maintenance for automation tests and framework are vital for a long term project success.

Gabriels Says:
November 10, 2020 at 4:59 am

Reply

Same here – Cypress is a novelty, something good to learn to know why Selenium remains an industry gold standard, if it is behind a proper Automation Framework.
I'm working on TestNG+Selenium combine framework where we also have some API and Unit tests.
All can be done from a single framework where needed, as needed.

I guess it's just a matter of how people work. Our tests for UI follow POM and we map test steps to code lines (usually few lines of reusable methods per step), so they look very different than this 'it' convention that I somehow always relate to Protractor.

But the way ours are written, you can read the code aloud and they make perfect english sentences describing what user is doing. API and unit tests are simpler (send request, check response, assert method results) but for system tests you could have zero coding skills and you should still be able to read what we coded and understood what happens.

// Step 1
logIntoApp

// Step 2
navigateToUserProfile

// Step 3
selectProfileDetails
assertTrue(userIdEqualsTestData)

Etc. Easy enough. And the reason we write it that way is exactly to allow any newcomers to understand what is going on in the test and in the code.
Maintainability is second most important quality of test automation framework

after usability 🙂

Robert Maclean Says: Reply
November 11, 2020 at 5:20 am

We did a study to compare Cypress.IO, Selenium IDE with other options and chose WebdriverIO instead. It's easy to use, flexible and stable.
It's not easy to use Gherkin, but it's possible.
There are easy-to-use packages, you only need NPM and drivers (I use Visual Code for editting).

Although I spent a lot of time setting up my Selenium IDE, Selenium, MS Coded UI Testing and learned a lot, this is the best choice to be able to run locally for us with a great reporting Allure add-on.

This is no ad because it's free to use, but I'm glad we're using it. I created 300 tests in a short period and it runs within a few minutes headless (and they aren't small either).

Sava Says: Reply
January 1, 2021 at 6:59 pm

Well I used both Cypress and Selenium. Both have the adventages and down sides but major problem of Cypress is stability because cypress code is constantly changed which causing much bigger problem with the maintanace of the tests over time on complex project. In some versions they literaly depreacated a lot of methods which made tests written in the previous version useless. Also if you are using the cypress in regular mode without any patterns enforced, as it is suggested by cypress team, it is absolut nightmare to mantain the large number of tests over time and there will be a lot of overlappings in the tests. Also cypress has the severe issues with the React and interaction with React elements is also nightmare especially if the App code isn't perfect. Due to limited number of browser supported and tab limitation it is also useless in complex flows or multiple app testing. Thus from my experiance so far cypress will never be able to replace selenium. At the moment it is just one of the fancy tools which is hyped. Cypress is ok only for quick validation of the simple apps during the sprints or for some test which will not be used anymore in the future. For all other projects that required constant regressions and monitoring Selenium will be always better option. Just design appropriate selenium framwork with appropriate and simple granular page object with methods that work with each element on the page seperatelly and remove driver from the tests by hiding the driver handling behind generic classes like base suite or the base test and it will be as simple for usage as Cypress but much better option for long term project.

William Hruska Says:
March 4, 2021 at 8:35 am

Reply

Great read.

Martin Says:
May 11, 2021 at 5:43 am

Reply

Thoughts on Cypress vs Selenium

Selenium has failed me in spectacular fashion trying to write automated UI tests for an Ext6 JS based product. Totally flaky. I had to rely on complex xpaths to locate elements that were plagued with being hidden, detached you name it. Used POMs, waits, eventually just gave up. In defense of Selenium Ext6 JS DOM is notoriously bloated and our developers didn't add unique IDs or attributes to elements (a big help surely).

I tried Cypress yesterday and got further in half a day than in a week with Selenium. Plus I drove Selenium with Python which I have a lot of experience with. Cypress uses Javascript which I am not as well versed in, I still got a lot further and faster than with Selenium.

I noticed there is also SeleniumBase so I gave it a whirl. Same agony. In exasperation I wrote a Tornado server to serve up various pages and explore Cypress vs Selenium. Again the latter is just useless re stability. I had a simple click-a-button-increase-font-size page and Selenium repeatedly crashed clicking the button to grow the text (connection time out or some such nonsense). Cypress not once. Never. That is surely the starting point you evaluate any automation tools with ! You can't afford false negatives or positives !!

Selenium is hyped beyond belief, at least for me it is one of the worst tools I have ever had to use. I suspect the success stories are because developers designed for testability and added unique IDs to elements so were easier to find and whatever JS library was used worked better with Selenium. But what to do if they didn't? and a code base is over a decade old? I have to conclude Selenium is the most hyped and soul destroying product I have worked with in a decade of automation.

Rhino API Says:
May 14, 2021 at 3:28 am

Reply

Usually I do not leave comments on blogs, but A LOT of misunderstanding is going on here.

First, about the element lifecycle exceptions (NoSuchElement, StaleElememtReference, etc.) and Selenium waiters
1. These exceptions helps to verify negative tests and are an integral part of healthy testing design.
2. Waiters are not part of the WebDriver protocol, and are just a client implementation and can be easily replaced or customized.
3. The full control over the element life cycle is very important when testing dynamic applications and I don't want "silent" waiters in some cases.

Second, Cypress is based on JavaScript which means it is bound to all JS restrictions including CORS policies (WebDriver is not bound to these restrictions). You will not be able to work cross pages or frames which violates the CORS policy. For instance, try to use Cypress with MS Dynamic CRM products.

Third, about flaky tests, the only thing it indicates is a bad implementation. I had these problems when I began automation but when I really understood how things works I stopped experience it and I do not have flaky tests in the past 8 years(!!!). Cypress implements some of the flakiness mitigating under the hood, while in Selenium you need to implement it by yourself, while having a full control over the element lifecycle.

In general, Selenium is basically just a WebDriver client… IT DOES NOTHING but sending POST/GET/DELETE requests to the WebDriver.

It is not a perfect client, but it does the work. The whole point of WebDriver is to create a unified protocol for automation (that is why it is the industry standard), all that is WebDriver including Appium and some Windows/OSX drivers were designed to give a single unified client for different automation platforms.

Lets say, I have a web application and native mobile application and I need to test my application on
1. All browsers
2. At least 5 last versions of Android
3. At least 5 last versions of iOS

The WebDriver protocol allows me to do it with a single framework and a single tech stack.

As long as tools like Cypress will interact directly with the browser they will not have this capability since they do not have the abstraction layer to communicate with the application under test.

Furthermore, anything can be WebDriver and the protocol can be (and does) implemented for different application types such as back-end testing, database testing and anything you can imagine. By doing so, I can run all my tests in a distribute way on and against any platform – independent from code or tool, using a single tech stack and unified standard.

Forth, Cypress cannot distribute tests out of the box as you would have in Selenium grid. You can use 3rd party like https://www.browserstack.com/ to achieve that, but it might be very expansive. You can also implement your own orchestrator (good luck with that) – it might sounds not so important, but lets say you have 6000 tests and your want to parallel them over 100 pods in Kubernetes.

Conclusion
Cypress is good and powerful, but it have it's own use case. It cannot replace Selenium, since it lacks the necessary abstraction, flexibility and functionality and it works on completely different paradigm.

It can be a very useful tool for front developers who wants to write components or integration tests assuming no CORS limitations and no cross browser testing is needed.

Most automation engineers, do not write automation in JavaScript (for a good reason, I will write a different post about it), which makes platforms like Cypress and Oxygen less appealing.

Daniel Says:
May 24, 2021 at 2:37 pm

Reply

There are already different solutions to run parallel cypress tests, even cypress has it's own but is paid. Saying so, there are some others that are free.

Leave a Reply

Enter your comment here...