

Frontend Testing Libraries Review



Edita Pronckute | S8-Graduation Internship | 2023 April

Frontend Testing

There is always a lot of emphasis on testing the software, but not so much the frontend part which is what the user is visually seeing and interacting with. Commonly, this gets tested via clickthrough of a pre-set staging environment before releasing to end-users.

This is also the case in Drieam, where first line of live testing is the developers reviewing and clicking through each other's PRs (Pull Requests). The second line is for the PO and/or the UX designer to try and break the interface. However, just like backend has unit tests and automated integration tests via pipelines (actions), so should the frontend. For this reason, the frontend unit tests are written for each component of the interface. It is a relatively new practice and thus causes a lot of discussions within the company.


A frontend unit test should validate that the smallest possible module (unit) is functioning as expected independent of the other modules. They can be split into state-based and interaction-based, meaning it either checks if the state of an element changed as expected or if an interaction is calling the right methods/making the right changes. Drieam uses a *Jest* library for front-end testing. It is a simple and lightweight JavaScript testing framework that is well documented and can run tests in parallel making it fast too. Since the tests are run periodically, they are automated using CI actions (pipelines). Thus, each time the code is pushed, merged, released the actions will run all the unit tests to ensure the application is still meeting expectations.

This document is a glance at potential testing solutions (libraries and frameworks) their benefits and weaknesses. It improves my understanding of frontend testing challenges, best practices and allows exploring potential solutions.

Testing tools

Unit testing


The following Testing libraries and/or frameworks are widely popular for frontend unit tests. It is beneficial to know a bit about them as well as their pros and cons.

 **Jest** – is the framework used by Drieam, as it has a strong community and active support. It is still the most popular framework (used by Facebook and officially supported by React dev team). It runs rather fast (as it supports parallel testing) and does not require separate assertion libraries or expensive configs creation.



Advantages

- Free and open source
- Compatible with NodeJS, React, Angular, VueJS, and other
- Very fast and highly performant
- Good documentation and community support



Disadvantages

- Automocking may slow down tests as it auto-wraps all libraries



Mocha – a very flexible and customizable framework running on node.js and in browser, however, it is known to require importing other libraries to write unit tests.



Advantages

- Lightweight
- Open source
- Flexible
- Servers and browsers can be tested too



Disadvantages

- Complex set up
- Auto-mocking and snapshot testing require custom configurations
- Plugin inconsistency



Jasmine – is another node.js and browser testing framework, it does not require external dependencies, however, together with Karma test runner it is a rather default option for Angular projects.



Advantages

- Widely chosen for behaviour driven development (BDD)
- Compatible with other languages like Ruby and Python
- Relatively fast as it does not require external libraries



Disadvantages

- Complex setup
- Creation of test globals may "pollute" global environment
- Asynchronous testing is challenging



Vitest – is another reasonable option as the entire frontend is run via Vite, thus, sharing one pipeline (action), same plugins and vite.config.js. to run tests might speed things up.



Advantages

- Out of the box typescript support
- Test Runner of choice for Vite projects




Disadvantages

- Rather young tool, thus as promising as it seems, may lack support yet.


End-to-end testing

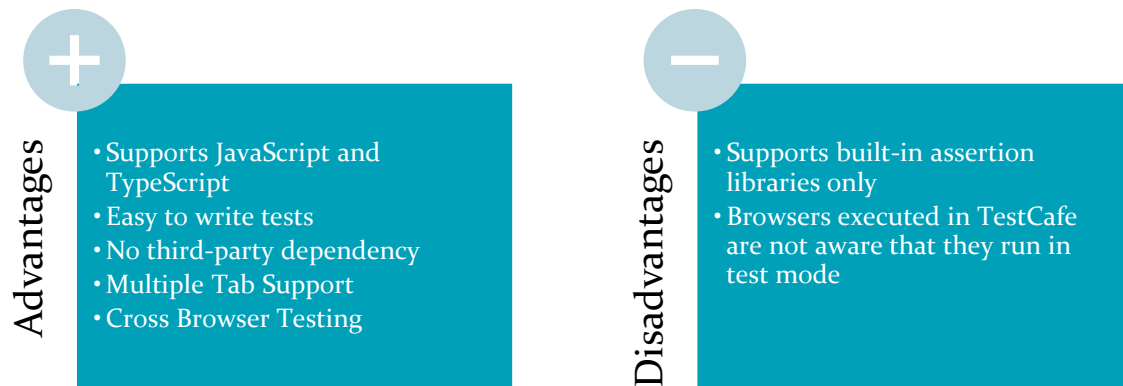
Next to unit tests it is important to have End-to-end (E2E) testing where real user scenarios could be automated for frequent rapid testing. E2E tests the end user behaviour, in the Portflow case, the interaction with the browser to use the application. In a way this is already being done live by clicking through typical user actions by the developers, PO or UX designer (Drieam does not have designated testers, the responsibility is shared among all involved). However, E2E tests provide more insight and details on how all the application parts work together to ensure the final product works as expected. It is wise to automate E2E testing and with the right tools it can be integrated in the pipelines (actions) to run


each time there are changes to the repository. There are many E2E frontend testing frameworks, several of them are analysed below:

 **Cypress** – is a web-based test suite that uses JavaScript to automate the tests. It is already used in Portflow actions for E2E testing. It is rather easy to understand and comes packaged as an npm module thus is easy to set up too.



 **TestCafe** - is another open-source JavaScript test automation framework suitable for E2E testing. It does not have its own window for debugging like cypress but provides the Live Mode feature to visualize and help debug.



 **Selenium** – is a browser automation library and mostly used for testing web-applications. There is a whole debate on library/framework categorization into selenium based and non-selenium-based ones. The selenium-based ones use a webdriver to interact with browsers, on the non-selenium ones interact directly. In example, selenium based E2E tools will allow multi-tabbing of the browser. However, tests that require leaving an application should be avoided for most cases as it adds complexity and may lead to possible test instabilities. Here are some of the advantages and disadvantages of using Selenium.

+


Advantages

- Well-established library
- Covers almost all browsers
- Community support
- Open source
- Support of multiple languages

-

Disadvantages

- Challenging to set up
- Low readability of test Scripts
- Built-in reporting facility is not directly available
- Requires 3rd party tools

 **Playwright** – is an automation framework best fit for E2E testing. It is owned and maintained by Microsoft and is a fork from Puppeteer (another famous testing tool). It is gaining attention due to its focus on testers and developers, thus also made this list.

+

Advantages

- Backed up by a trusted company
- Multi-language support
- Cross-browser
- Multiple test runner support (Mocha, Jest, Jasmine)

-

Disadvantages

- Relatively new, thus limited support
- Does not support legacy MS Edge or IE11
- Requires setting up own reporter for CI/CD builds

Common testing problems

Below are some testing challenges that arose during the course of the internship and are generally common in frontend testing:

Changing UI – upgrades to core libraries or their components requires quick response and test updates too.

Flaky tests – are the ones that sometimes fail, sometimes pass, they cause pipeline instabilities. Typically, they are appearing when small mistakes are made in the codebase. Not waiting till asynchronous action is completed, is a common example.

Execution duration – E2E can be time expensive especially when covering all possible user journeys. Best practice it to cover only the most important paths but not all edge cases (use integration or unit tests for them instead as they are faster).

Mocking – can be time consuming too, especially if some components rely on multiple data sources from outside providers. To render the full DOM and all components correctly Drieam uses Testing Library designed for React to help imitate user interactions in real browsers. This way, some load is taken of the Cypress E2E tests.

To help prevent and manage these problems better it is important to choose the testing libraries that match the code base, the application needs and are developer friendly.

Conclusion

Nowadays, most test frameworks can do the job well, the choice is either a personal preference or simply the one that is most popular and matches best with the rest of the set up.

It is rather overwhelming to know which tools to choose as some like Selenium and Jest have been there long enough and are well established. However, the newer tools often are better designed with developers in mind, they can offer easier set up, simpler syntax and new reporting options. Some tools have a wide coverage, multi-language support and wide integration options. In the end, the price of flexibility is complexity, keeping it simple can become a challenge. But keeping test simple, fast and stable is still the ultimate goal of developers and testers.

Thus, after trying out and seeing the efficiency of Cypress, Jest and React Testing Library, I believe them to be a good and well researched choice of the company. Perhaps, as the product develops, and its complexity grows there may be a need to add more to the testing suite. Luckily, there are plenty of great options out there from the ones mentioned above to many more like QA wolf, Puppeteer, Storybook etc.

References

- Appel, R. (2020, September 10). *Unit Testing in TypeScript Code*. Retrieved from JetBrains blog: <https://blog.jetbrains.com/dotnet/2020/09/10/unit-testing-in-typescript-code/>
- Błaszyński, Ł. (2021, December 3). *Front end testing frameworks for React Applications in 2022*. Retrieved from Espeo Software: <https://espeo.eu/blog/front-end-testing-frameworks-2022/>
- Bose, S. (2023, March 16). *Front End Testing: A Beginner's Guide*. Retrieved from BrowserStack: <https://www.browserstack.com/guide/front-end-testing>
- Chatterjee, K. (2023, March 17). *Playwright Testing Tutorial - A Guide With Examples*. Retrieved from LambdaTest: <https://www.lambdatest.com/playwright>
- G., A. (2022, June 7). *Vitest (Unit Testing) To Test React Application*. Retrieved from Medium: <https://waresix.engineering/vitest-unit-testing-to-test-react-application-177ade1e6c1b>
- Jones, A. (2020, September 14). *Front End Automation Testing Tools*. Retrieved from applitools: <https://applitools.com/blog/2020-front-end-automation-testing/>
- Katalon Inc. (2023). *Top 10 Best End-to-End Testing Tools and Frameworks*. Retrieved from Katalon Insights : <https://katalon.com/resources-center/blog/end-to-end-e2e-testing-tools-frameworks>
- Rajendran, A. K. (2020, June 28). *Which E2E testing framework to use for JS-based client applications?* Retrieved from Medium: <https://aswinkumar4018.medium.com/which-e2e-testing-framework-to-use-for-js-based-client-applications-fbcac9aab680>
- Sanders, M. (2021, August 25). *Challenges and Types of Front End Testing*. Retrieved from Binmile: <https://binmile.com/blog/challenges-and-types-of-front-end-testing/>
- Taleb, M. (2022, October 6). *JavaScript unit testing frameworks in 2022: A comparison*. Retrieved from Raygun: <https://raygun.com/blog/javascript-unit-testing-frameworks/>
- testing-library documentation. (2018-2023). *React Testing Library*. Retrieved from Testing Library: <https://testing-library.com/docs/react-testing-library/intro/>
- Vitest guide. (2023, March 5). *Comparisons with Other Test Runners*. Retrieved from Vitest: <https://vitest.dev/guide/comparisons.html>

Appendix A

The tables below compare the known test frameworks based on a set of characteristics.

	Jest	Mocha	Storybook	Cypress	Jasmine	Puppeteer	Testing Library	WebdriverIO	Playwright	Ava	Vitest
Ranking (Usage)**	73%	50%	48%	43%	40%	37%	35%	10%	10%	7%	3%
Multi Threading / Worker	Y	N	N	N	N	N	N	N	N	N	Y
Mocking and Spies	Y	N	N	N	Y	N	N	Y	Y	N	Y
DOM Mocking	Y	N	N	N	Y	N	Y	N	N	Y	Y
Snapshots	Y	N	Y	Y	Y	N	Y	N	Y	Y	Y
Coverage	Y	Y	Y*	Y	Y	Y	Y	Y*	Y	Y*	Y
Timer Mocks	Y	N	N	Y	Y	Y*	Y	N	N	N	Y
E2E Testing	N	N	N	Y	N	Y	Y	Y	Y	Y*	N
TypeScript Support	Y	Y*	Y	Y	Y*	Y	Y	Y*	Y	Y*	Y
Watch mode	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y
VSCode extension	Y	Y	Y	N	Y	N	N	N	Y	N	Y

* with addons / third-party library

** Data from stateofjs.com

Source: <https://raygun.com/blog/images/javascript-unit-testing-frameworks/compare-testing-frameworks.png>

Categories	WebdriverJS	Protractor	WebdriverIO	NightwatchJS	Cypress	TestCafe
Architecture	Implementation of W3C web driver JSON wire protocol	Wrapper around WebdriverJs	Custom implementation of W3C web driver JSON wire protocol	Custom implementation of W3C web driver JSON wire protocol	Interacts directly with the browser	Interacts directly with the browser
Selenium-based	Yes	Yes	Yes	Yes	No	No
Supported Browsers	Edge, Chrome, Safari, Firefox, Opera, IE	Edge, Chrome, Safari, Firefox, Opera, IE	Edge, Chrome, Safari, Firefox, Opera, IE	Edge, Chrome, Safari, Firefox, Opera, IE	Chrome, Electron	Edge, Chrome, Safari, Firefox, Opera, IE
Inbuilt test runner	Available	Available	Available	Available	Available	Available
Supported testing frameworks	Jasmine, Mocha, Cucumber	Jasmine, Mocha, Cucumber	Jasmine, Mocha, Cucumber	Mocha, Inbuilt framework	Custom implementation	Custom implementation
Parallel execution	Supported	Supported	Supported	Supported	Supported	Supported
Cloud Execution - Sauce Labs, BrowserStack, Testing bot	Supported	Supported	Supported	Supported	N/A	Supported
Mobile Support (APPIUM)	Supports mobile browsers	Supports mobile browsers	Supports mobile browsers	Supports mobile browsers	No	Supports mobile browsers
CI	Supported	Supported	Supported	Supported	Supported	Supported
Retrying flaky tests	Possible	Possible	Possible	Possible	Possible	Possible
TS support	Yes	Yes	Yes	No	Yes	Yes

Source: https://miro.medium.com/v2/resize:fit:2000/format:webp/1*I2X-aBITwXcar5_y1Efffkw.png