End to end testing in JavaScript

Filip Andrei

Contents

[Context 3](#_Toc93499466)

[The main goal 4](#_Toc93499467)

[Research methods 5](#_Toc93499468)

[How does end-to-end testing work? 6](#_Toc93499469)

[Which are the available frameworks? 7](#_Toc93499470)

[Conclusion 12](#_Toc93499471)

[Bibliography 13](#_Toc93499472)

# Context

An application is made up of different features and components. Before releasing the application into production, testing how the features and components work together is crucial. End-to-end testing is a testing methodology that checks the flow of an application from the start until the end of the process. It is supposed to reproduce a set of actions done by a user in a real scenario. It is impossible to anticipate all the actions of a user, but at least it is possible to have an idea of what the user intends to do.

# The main goal

This study will explore end-to-end testing in relation to the application that I am building. The main question of the research is: **Which end-to-end testing library to choose for movie upload?**

During this study a few other questions will be answered:

* How does end-to-end testing work?
* Which are the available frameworks?

# Research methods

This study complies with the Development Oriented Triangulation (DOT) framework for giving structure to the research.

Icon

Description automatically generated

Text

Description automatically generatedThe preferred research strategy is Library, because of the freedom of choice for the large and vast information available on the internet. From this research strategy, the method used is Literature study for sourcing relevant information from trusted sources.

The other main research strategy used is Showroom since the best way to explain a topic is by presenting short and understandable pices of code put into context

# How does end-to-end testing work?

As any type of test run, the goal is to assure that the implemented features work accordingly, so that when new features are added they do not break the other features. End-to-end tests check the whole application through the user interface, which means they must be run in a browser, making them slower to run compared to unit or integration test that run in milliseconds or seconds.

End-to-end test could be split into 2 categories:

* **Horizontal testing:** all the components and features of the application are to be tested. For example, you test logging in or signing up, navigating through the pages, uploading a movie, then playing it. This checks the breadth of the application.
* **Vertical testing:** with this category, the depth of a certain component/feature is to be tested. For example, in order to be authenticated for an application you can log in with you credentials, or you could sign up with your email address, or you could choose to sign up with a third party account such as google or Facebook.

# Which are the available frameworks?

**Selenium WebdriverJS**

This framework is based on Selenium, which is an open-source tool that automates browsers. Its architecture is an implementation of W3C web driver JSON wire protocol, which is a “remote control interface that enables introspection and control of user agents” (WebDriver, n.d.). Given this architecture, it does not directly communicate with the browser, requiring a web driver. WebdriverJS supports Edge, Chrome, Safari, Firefox, Opera, and Internet Explorer, it has an inbuilt test runner that supports parallel execution, it supports cloud execution, but it also has mobile support. The framework makes use of a promise manager, which removes the need to write chained promises, allowing for writing the test as if the code was executed synchronous.

*Pros:*

* Wide range of browsers supported
* Large community support
* Record and playback for scripts

*Cons:*

* Takes longer to set up
* Harder to pick up

Example   
“

const {Builder, By, until} = require('selenium-webdriver');

let driver = new Builder()

.forBrowser('firefox')

.build();

driver.get('http://www.google.com/ncr');

driver.findElement(By.name('q')).sendKeys('webdriver');

driver.findElement(By.name('btnK')).click();

driver.wait(until.titleIs('webdriver - Google Search'), 1000);

driver.quit();

“ (Rodionov, 2017)

**Cypress**

It’s not based on Selenium, it supports Chrome, Electron and Firefox, it also has an inbuilt test runner that supports parallel execution. It lacks the support for cloud execution and mobile support. On the architectural level, Cypress interacts directly with the browser. The fact that it’s not based on Selenium makes the framework a more lightweight tool, requiring minimal setup. It is easy to pick up, allowing for tests to be written as if the code was run synchronously. Debugging also is easy and straightforward, given that you can view the output in the browse at each stage of the test.

*Pros:*

* User-friendly documentation
* Easy to set up and use
* Large community support

*Cons:*

* Few browsers supported
* The lack of some functionalities

Example

it(('public message test'), () => {

        // mock data

        const publicMessage = "The API doesn't require an access token to share this message.";

        // stub server request

        cy.intercept('GET', Cypress.env("react\_app\_server\_url") + '/api/messages/public-message',

        {

            statusCode: 200,

            body: {

                message: publicMessage

            }

        }).as('public');

        // visit page

        cy.visit('http://localhost:3000/');

        // navigate to the page using the menu

        cy.get('[aria-label=menu]').click();

        cy.get('[href=external-api]').click();

        //trigger the request

        cy.get('[type=button]').contains('Get Public Message').click();

        // waiting for a response

        cy.wait('@public').then(() => {

            // asserting the content of the div for a message

            cy.get('[data-testid=api-message]').contains(publicMessage);

        });

    });

**TestCafe**

This framework is like Cypress, since it is not based on Selenium, and it interacts directly with the browser. It supports the following browser: Edge, Chrome, Safari, Firefox, Opera, and Internet Explorer. Like all the frameworks described previously, it has an inbuilt test runner that allows parallel execution. Unlike Cypress it is more complete since it also supports cloud execution and mobile browsers. It is easy to set up, not needing any web drivers or specific browser versions to work. TestCafe uses an assertion query mechanism, which “which automatically waits for changes to take place and retries with checkups.” (Kupcewicz, 2020). The biggest downside to this framework is that it is not free. It has a 30 day trial, after that it costs 200$ per year.

*Pros:*

* Easy to set up
* Assertion mechanism query
* Wide range of supported browsers

*Cons:*

* Requires tests to be written asynchronously
* It is a paid service

Example

“

fixture `Pizza Palace`

.page `https://testcafe-demo-page.glitch.me/`;

test('Submit a form', async t => {

await t

// automatically dismiss dialog boxes

.setNativeDialogHandler(() => true)

// drag the pizza size slider

.drag('.noUi-handle', 100, 0)

// select the toppings

.click('.next-step')

.click('label[for="pepperoni"]')

.click('#step2 .next-step')

// fill the address form

.click('.confirm-address')

.typeText('#phone-input', '+1-541-754-3001')

.click('#step3 .next-step')

// zoom into the iframe map

.switchToIframe('.restaurant-location iframe')

.click('button[title="Zoom in"]')

// submit the order

.switchToMainWindow()

.click('.complete-order');

});

“

**Which end-to-end testing library to choose for movie upload?**

Considering all the pros and cons listed above, the best framework for writing end to end tests for movie upload would be Cypress. Even though it seemed as TestCafe is more suitable for the project given the wider browser support and more functionalities, the problem is that it is a paid service, which becomes a large impediment.

For movie upload, a crucial test would be uploading a file, since this is one of the main features of the app. Even it is not possible to upload a file just with Cypress there is a library named cypress-file-upload [[1]](https://www.npmjs.com/package/cypress-file-upload), which facilitates this task. So, if there are any other limitations encountered, finding a workaround is always possible.

Another criteria that makes Cypress a good choice is the fact that it is easy to pick up, given that it is well documented, with solid examples and that is not based on Selenium opposed to most of the end to end testing frameworks. Cypress makes use of Mocha’s behavior driven development syntax, using the essential functions: describe, it, before, before each, etc. While this provides structure to the tests, easy assertions are provided by Chai, by using the .should() command.

Even if it is missing a playback function for the tests, debugging is a simple process given to the explicit error messages shown and the fact that the developer can go step by step through each step of the test too se which is the output in the browser.

# Conclusion

To sum up, end to end tests are crucial for testing an application’s behavior before it makes its way into production. The most suitable framework for testing my application is Cypress, due to it’s simplicity and minimalist approach to testing, supporting the required feature (file upload) and easy debugging.

# Bibliography

Kupcewicz, T. (2020, February 20). *Which JavaScript UI Testing Framework to Use in 2020?* Retrieved from Netguru: https://www.netguru.com/blog/which-javascript-ui-testing-framework-to-use-in-2020

Rodionov, A. (2017, November 1). *WebDriverJS*. Retrieved from Github: https://github.com/SeleniumHQ/selenium/wiki/WebDriverJs

*WebDriver*. (n.d.). Retrieved from w3.org: https://www.w3.org/TR/webdriver/

<https://www.netguru.com/blog/which-javascript-ui-testing-framework-to-use-in-2020>

<https://aswinkumar4018.medium.com/which-e2e-testing-framework-to-use-for-js-based-client-applications-fbcac9aab680>

<https://www.testim.io/blog/end-to-end-testing-frameworks/>

<https://wanago.io/2019/12/30/javascript-testing-introduction-end-to-end-testing-cypress/>