BNFIN Automated Testing Documentation

Robot Framework, Selenium

Contents

[Introduction 3](#_Toc167421113)

[Features that will be tested: 4](#_Toc167421114)

[Testing Limitations: 4](#_Toc167421115)

[Running the tests: 5](#_Toc167421116)

[Instructions: 5](#_Toc167421117)

[Interpreting Results 7](#_Toc167421118)

[Test Contents 9](#_Toc167421119)

[System Alternatives 12](#_Toc167421120)

[Cypress.io 13](#_Toc167421121)

[Playwright 17](#_Toc167421122)

[TestCafe 20](#_Toc167421123)

[Testim.io 22](#_Toc167421124)

[WebdriverIO 25](#_Toc167421125)

[Selenium 27](#_Toc167421126)

[Comparison Chart 28](#_Toc167421127)

## Introduction

This documentation will outline the testing procedure for the automated testing suites used to test the BNFIN platform and its functionality.

Technologies used: Robot Framework, Selenium, Python.

**Robot Framework** is an open-source automation testing framework that uses a keyword-driven approach for acceptance testing and acceptance test-driven development (ATDD). It's designed to be easy to use by allowing users to write tests in a natural language-like syntax using keywords. Making for a flatter learning curve and easier for all stakeholders (whether competent programmers or not) to be able to read and understand the testing process.

**Selenium** is a widely used automation and test writing tool widely used for web browser automation. While Selenium itself can be integrated and used with a wide variety of programming languages (Python, Java, and C# being the most commonly used), it can be imported and integrated with Robot Framework via the **SeleniumLibrary** plugin, which allows the user to access the Selenium functionalities (such as clicking buttons, entering text, navigating through pages etc.) but using the simple Keyword syntax provided by Robot. Essentially eliminating the need to use any of the abovementioned programming languages.

Example of Selenium Syntax:

A black background with white text

Description automatically generated

Example of Robot Syntax for comparison:

A screen shot of a computer

Description automatically generated

**Python** will be integrated with Selenium for certain tests which may be difficult to achieve using Robot Framework alone. (Such as spell checker, file saving mechanisms, or complex conditional statements).

## Features that will be tested:

* Login/Logout/Password reset
* Registration
* Navigation between links
* Pagination
* Grant display
* Grant Reviewer assignment process
* Grant Application Stage 1 (incomplete)
* Field validation
* Draft creation and editing
* Filtering
* Functionality of eligibility quiz

## Testing Limitations:

Due to the nature of Robot script and Selenium, and the use of “locators” (elements that tell the framework where to find specific elements on a web page, those could be a CSS element, a text element, xpath, link, ID etc...), the tests in this system heavily rely on the structure of the HTML in the source code of the original application. Should that source code change, some of the tests in this system may be rendered unusable or require an update.

Robot Framework does not deal very well with Dynamic content. It needs explicit instructions of locators and if those locators do not yet exist – the tests may fail or need updating.

Due to the above reason, certain test elements may need to be modified during the course of development. For instance, if a test currently has to navigate to the last page to see the newest grants in order to select one; when this issue is fixed (i.e. the newest grants will show on the first page, as per industry standards), the test code will need to be modified to reflect that.

Slow in testing dynamic content. Due to the various loading times of dynamic site content and the time it takes for items to appear on the screen, the testing speed needs to be manually adjusted (i.e “set selenium speed 1s” in Common.robot) which greatly slows down the testing process. A single test suite containing 20+ tests may take up to 20-25minutes to run. Further steps that slow down the testing speed (such as setting keywords to the likes of “Unselect Frame”, “Wait Until Element Is Visible” etc.) are also required when moving between certain testing fields, otherwise the tests may fail.

System is slightly finicky and inconsistent. A test that could be ran consecutively without any changes to the inner working of the web-app and the tests could produce different results with each run.

A common occurrence is the following type of error:

“Element with locator 'xpath=//\*[@id="app"]/div[4]/div[2]/div/div/div[1]/section/main/div/div[2]/div[2]/div[1]/input' not found.”

This error indicates that the element or input field locator is not pointing to the right place or cannot be seen. However, a test can be re-run and the locator would work.

Or a different test with this same locator could be ran and it would work. The locator could be changed to a different type (from “xpath” to “Full xpath” or “value” or “text”) and that would run sometimes – others it won’t.

## Running the tests:

While the tests can be ran using a variety of means, from individual tests to group runs using a selected Tag, to scheduling specific tests using the task scheduler, the method chosen for this system is through the running of batch files that contain groups of related tests.

Clicking on each batch file will run the tests defined in it and open the “log.html” report that would display the test results.

### Instructions:

1. Prerequisites:

In order for the testing software to take control of your chosen web browser, web drivers for this particular browser must be installed.

For Chrome-

Check your current browser version by pressing the Options menu -> Help -> About Google Chrome.

Download the driver pertaining to your version here:

<https://chromedriver.chromium.org/downloads>

Note: the PATH to your web driver will need to be added to your environment variables. The actual driver .exe does not need to be executed. Just included in the PATH folder.

Follow a similar process for any additional browser.

1. Getting the files-

Go to “https://github.com/nomanymore/RobotFramework-Testing”

This repository contains keyword files (in “Resources”), test suite files (In “Tests”), result logs (in “Results”) as well as the batch files.

NOTE: The password and login to access the staging site should be entered in a few files in the project folder after cloning it to your local machine, those files are “Spellchecker.py”, “AccessibilityChecker.py” and “Common.robot”. Those have been removed from the repository due to security concerns. Without them, the BNFIN initial site will not load, and all tests will fail.

A black screen with white text

Description automatically generated

To run the batch files, the path pointing to the test must first be changed to the path of your local test files (currently pointing to my personal computer). That can be done by editing the .bat file in a text editor.

A screen shot of a computer

Description automatically generated

1. Execution

To run the full test suite, double click on the .bat file.

Please note a full suite of over 20 tests may take over 30 minutes to complete.

To run individual (or group) tests within a suite, first open the test file in your preferred IDE, note each individual test has “Tags”. Locate the desired tests, either copy a common tag, or add your own tag to the list to identify the tests you want to run. I like to use “Run”, and add it to each test I want to run. Save the test file.

Open and edit the batch file for that test, add “-i” and the tag name before the test location/name. Save file and double click to run.

A screen shot of a computer

Description automatically generated

**A screen shot of a computer

Description automatically generated**

## Interpreting Results

Once the tests have been completed, the “log.html” file for that test run will open automatically. Note this file overwrites itself with each run, unless a new path for the result file is indicated in the .bat file. (i.e results/grantLead1/log.html)

The log file provides an easy overview with an option to drill down into each step of the test, and find exactly which step passed and which did not.

The log is colour-coded for easy visualization, with passed tests marked in green and failed marked in red.

**A screenshot of a computer

Description automatically generated**

The tests have descriptive names to easily identify what each test is meant to do.

When drilling down into each test, each step should also easily indicate what is being done and where and how it failed or succeeded. If a test fails in one of the steps in the middle, it’s execution is stopped and no further steps will be taken. A failed test will also produce an automatic screenshot of the problem area.

**A screen shot of a computer program

Description automatically generated**

Passed Test

**A screen shot of a computer

Description automatically generated**

Failed Test

## Test Contents

**The contents and tests that can be found in each file are as follows:**

|  |  |  |
| --- | --- | --- |
| File Name | Overview of included tests | Detailed tests |
| {Sign in, sign out, registration, forgot password.bat}    **Total test cases: 23** | This testing suite will check for the user’s ability to Login, Logout, Reset Password and Register | * Login in as: Reviewer BBI-1, BBI-2. Reviewer Tropicana-1, Tropicana-2. Applicants 1,2 and 3.  Lead reviewers- BBI, Tropicana.  Admin. * Login with invalid fields. * Logout. (As admin) * Successful password reset with valid fields. * Unsuccessful password reset with empty fields. * Unsuccessful password reset with invalid fields. * Unsuccessful password reset with invalid format fields. * Navigation from “forgot password” back to “login” page. * Sign up with empty form fields. * Sign up with invalid form fields. * Sign up with valid form fields. |
| {Public Page Navigation.bat}  **Total test cases: 16** | This testing suite will check for the user’s ability to reach all the publicly available links on the website while logged out | * Test links for * Top nav menu:   Homepage,  About Us,  Contact Us,  Sign Up,  Grants.   * Footer links:   Cookie policy,  Privacy Policy,  Disclaimer,  Terms of use,  Join now.   * Social networks:   Facebook,  Twitter/X  Instagram   * Language toggle * Navigation between “Sign In” and “Sign Out” pages and vice versa. |
| {Grants main page, Lead Reviewer 1.bat}  **Total test cases: 22** | This testing suite will test for the functionalities of the main page displaying the grants as seen on the Lead Reviewer’s account. | * Display Grants page * Choose how many grants to display per page: 6,12,24 * Pagination, flipping from first page to last * Archiving grants * Searching grants, valid and invalid values * Filtering grants by each filter * Creating new grants (of both types) with valid values. * Creating a draft * Editing a draft * Creating new grant with invalid values:   Empty fields.  Max length check.  Text data entered into numeric fields.  Relationships between amounts (i.e. min amount not higher than max, max not higher than budget).  Date field relationships (Finish date cannot be before start date). |
| {Submitted Grants page, Lead Reviewer.bat}  **Total test cases: 12** | This testing suite will test for the functionalities of the “submitted” section of the application stage, where the submitted applications can be viewed, an eligibility test completed to mark as eligible or declined, filter through the applications and search for applications. | * Navigation to page * Filtering (By Intermediary, Region, Organizational Area Of Focus, Task/Stage Completion) * Submission of Eligibility quiz both valid and declined. * Check that eligibility quiz is disabled after submission so it cannot be edited. * Search in search bar for valid and invalid terms. * Assigning reviewers. * Check that reviewers can only belong to the selected intermediary. |
| {Application Stage 1.bat}  **Total test cases: 29…** | This testing suite will test the functionality and validity of the fields in the “application stage 1” form, divided by pages. (7 in total) | * Navigation * Empty Field Validation * Email and numeric field validation * Date Field Validation * Max Char Validation * Adding and removing fields using “+” and “-“ buttons * Draft saving |

## System Alternatives

Due to the limitations described earlier in this document, a number of system alternatives have been researched and tested. The findings are as follows:

### Cypress.io

A screenshot of a computer

Description automatically generated

Cypress is a free, open source web testing tool that provides both unit testing and e2e testing capabilities. While the unit (and component) testing is integrated directly into the actual code-base and requires access to it, the E2E testing can be done via the remote browser provided by the tool itself and does an excellent job of simulating the user experience when going through the app.

The size of the test is an individual preference and there is no issues with testing just one input field – for instance, bypassing the limitation.

Cypress uses **JavaScript** as the base language for its operations, making it easy to learn for anybody with previous development experience and is overall very simple to use.

Advantages:

* **FAST**. Tests are executed at top speed without compromising quality and reliability.
* **Time Travel:** Let’s you see snapshots of every stage in the testing process, which helps with debugging and ensuring the tests were executed correctly.
* **Easy to install**, minimum setup required. Easy to learn.
* **Easy to switch between** **browsers**, so long as you have them installed on your system to begin with.
* Hassle free locator selection – Cypress provides a “**Selector Playground**” which allows you to capture the locators for the elements you wish to interact with, without having to use the dev tools and extract them manually. Though you still can, if you wish to.
* **Reliable** – The “selector playground” makes the tests a lot more stable, as it follows one unique pattern of choosing a selector, instead of using ever-changing elements which may break the next time you run the test.
* **Provides Asynchronous Solutions –** While Cypress executes the tests sequentially (as does Robot), it handles asynchronous events internally, not requiring any special input from the user.

Disadvantages:

* **Navigation between different domains impossible.** Once your environment is set up, you provide a “starting point” with the domain you will be testing. While a different domain can be used for a different testing suite, each suite can only navigate within its own domain, making it difficult if you have links leading outside the main domain that need to be tested in the same test.
* **Authentication issues.** Due to the staging site requiring an initial login to even access the site, this creates a problem which requires manual login before any test can be performed. I’ve looked at multiple ways of going around it (for instance including the username and password inside the URL: a strategy that worked wonders in Robot Framework, not for Cypress) but to no avail.
* **Error messages** on test fail can be ambiguous, or at times – not shown at all.

For example, I created a loop that should have gone through 3 fields and entered first names in all 3, one after another. The test executed and the loop only filled in the first field, leaving the rest blank and moved on – the test was marked as “pass” and I got no indication that the loop failed.

Comparison to Robot Framework:

Both tools share the same simple setup and flat learning curve.

While Robot provides a more user-friendly interface as far as error reporting and uses natural-language based keywords, Cypress leads in reliability and speed. A lot of the time it takes to write Robot code is due to the search, identification (and potentially RE-identification due to bugs and errors) of element locators, while Cypress’ selector playground solves it very quickly, combined with a good multi-copy-clipboard and the test code can be ready in seconds.

Being based on JS, Cypress’ more advanced functions are also a lot more reliable (if statements, loops, reusable code etc.) eliminating the need to write the same line over and over again and leaving the code cleaner. While Robot also offers those functions, they tend to be buggy, and for the most part – I had to get rid of them or rewrite them.

Asynchronous handling- Robot requires a lot of “checks” for dynamic content. If an element is not visible on the page, or hasn’t loaded yet, manual keywords (like “Wait Until Element Is Visible”, and “Unselect Frame” when jumping between frames) need to be added to every step of the test that may require it – hurting performance and slowing it down even more. Cypress handles those internally and automatically.

Cypress definitely wins in terms of speed: I compared the same test (filling in the first page of the Application Stage 1 form) using both tools:

Cypress took 22 seconds to execute it. (including the manual site credentials verification that had to be done in the beginning)

Robot Framework took 4 minutes(!) and 2 separate tests (one for text fields and one for select fields, as together they would constantly run into bugs when ran as one test).

Screenshots:

A screenshot of a computer

Description automatically generated

Selector Playground and test logs

A screenshot of a computer

Description automatically generated

Cypress Syntax and use of loops

### PlaywrightA screenshot of a computer Description automatically generated

Playwright Codegen

Playwright is an opensource automation tool that makes e2e testing easy.

It supports most modern browsers (Chromium, Firefox, and WebKit engines) as well as headless browser testing. And while tests can be written in any of the 5 supported languages: Java, JavaScript, TypeScript, Python or C# it also includes a “Codegen” that “codes” and records a tester’s interactions with the browser and turns them into code that can then be copied and turned into a script. Meaning that the tests can be written by people with minimal coding experience. However, the Codegen has it’s own challenges which will be described in more detail below.

Advantages-

**Fast**- Tests are performed at high speed.

**Code-Free testing** – The Playwright Codegen allows a user to interact with the UI directly and the interactions will be recorded as code that can then be copied into the actual test file.

**Dynamic Element Handling** – Playwright has in-built mechanisms to auto-wait for elements to appear and be ready before performing interactions, removing the need for manual waits.

**Fairly easy installation process** – only requires Node.js, an IDE and playwright as a dependency.

Note: While the installation is easy, there are still issues that may arise along the way between installation and running the tests. Having copied the code from Codegen, I had to go through a few more steps and configurations before my tests would actually run from the IDE itself.

**Multiple Context Testing-** One of the biggest selling points of Playwright is the ability to use multiple contexts, since each testing suite doesn’t only open a browser, but also a context inside of a browser. (either headed or headless)

This greatly speeds up the process of testing as it allows multiple tests to run at once and in different environments, since each context is completely independent from the other. This kind of setup can help in testing various users independently but at the same time, without interfering with each other.

Disadvantages:

**Playwright Codegen**- Although an advantage in some ways (such as not having to go into source code to find manual locators), the Codegen records every interaction, needed or unneeded and a lot of unnecessary actions seem to be recorded in the code, making it messy. For instance, if I were to clear a field (that had previously retained values) instead of recording one action (or two: click and clear), the Codegen records every move of an arrow or button and the directions to locate an element are far too “precise”, a good example is the following code:

await page.locator('div').filter({ hasText: /^Apr 25, 2024 - May 30, 2024Application - Stage 1$/ }).getByRole('link').click();

This line directs the program to click the link (that should say “Application Stage 1”) but instead of choosing to go by the link text “Application Stage 1”, it captures the exact application and the text that application has “Apr 25, 2024 - May 30, 2024Application” which could be an issue if the application has since expired, and the date is now after May 30th.

While Codegen is useful for extremely simple tests and interactions, any more complex code will need to be manually written, rendering this feature useless.

**Test Tracking** – While video capture and screenshots are available in Playwright, those need to be manually entered and added to each testing suite and do not occur automatically like they do in Robot Framework or Cypress.

**Result log** – After running the tests, the results are shown in the console directly. Besides listing if the exam passed or failed, feedback is available regarding the point where the issue occurred, however, debugging may be a bit difficult as the errors are not very descriptive. Other methods of reporting are available via additional external libraries; however, those need to be installed and integrated and do not come with playwright right out of the box.

**Steeper Learning Curve** – To fully leverage Playwright's capabilities, the user needs to install various libraries, set up certain configurations and setting in the working environment, and have at least some coding experience and knowledge, all of which take time and effort. Making playwright slightly more complicated to get started with when compared to its competitors.

A screen shot of a computer program

Description automatically generated

Playwright results log

Compared to Robot Framework and Cypress:

When running a single e2e testing suite, Playwright’s speed is comparable to that of Cypress and much faster than Robot Framework. However, due to its ability to test multiple tests simultaneously using different contexts, one would argue Playwright surpasses Cypress in terms of speed.

Browser support is near identical in all 3 tools. Playwright has more language support making for a more flexible environment than Cypress that only uses JS and Robot that only uses keywords.

Authentication does not seem to be an issue with Playwright (as opposed to Cypress) and credentials can be passed directly in the URL.

When compares to Cypress and Robot, Playwright’s reporting capabilities are poor and require more skill to read and solve any errors. Playwright has a steeper learning curve.

### TestCafe

A screenshot of a computer

Description automatically generated

TestCafe is an open source, mostly free (the TestCafe Studio which doesn’t require coding is a paid product) e2e testing tool that works with Node.js.

Setup is fairly simple and just requires adding the TestCafe dependencies in your project folder in your chosen IDE. Much like Cypress, TestCafe only supports Javascript/Typescript.

Advantages:

Very simple **installation**. No need for web drivers or browser plugins.

**Live Update/Reload** – TestCafe allows you to keep the testing browser open after a test and re-run it automatically when any changes are made, saving time on opening/closing the browser and restarting the whole process.

**Integrated Wait Mechanism** – No need to add explicit waits for elements to be visible/interactable.

**Speed**- TestCafe performs fairly well in terms of execution speed.

Disadvantages:

**Manual Selector Search** – Since TestCafe doesn’t include a free screen recorder type tool, all coding and element locations need to be done manually. Not only is it time consuming but also more difficult to get started for anybody not familiar with JavaScript.

**Reports/logs** are console based and not very user-friendly. External libraries can be added to improve reporting.

**Test Tracking** – Much like with Playwright, while video capture and screenshots are available in TestCafe, those need to be manually entered and added to each testing suite and do not occur automatically like they do in Robot Framework or Cypress.

**Steeper Learning Curve** – To fully leverage TestCafe’s capabilities, the user needs to install various libraries, set up certain configurations and setting in the working environment, and have at least some coding experience and knowledge, all of which take time and effort.

A computer screen with text on it

Description automatically generated

TestCafe error log

Compared to Robot Framework, Cypress and Playwright:

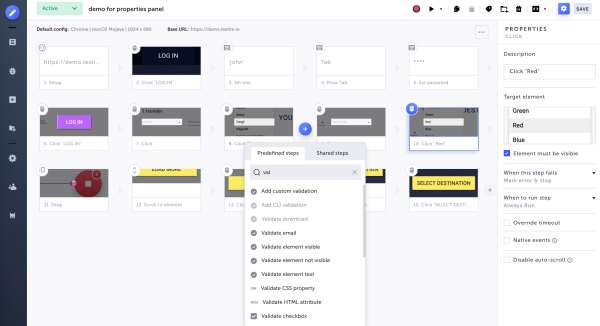
In terms of installation, out of the box features and learning curve, TestCafe is very similar to Playwright. Both are based on Node.js and require a simple project folder and dependency installation. Both have very simple features available from the offset that could be improved by installing 3rd party libraries. Both (for the most part) require coding and have a bit of a steeper learning curve. Both also have fairly limited test logs/reports that could be hard to read and debug.

One thing TestCafe offers that other don’t is the live tracker for test changes and retesting on the spot.

In terms of speed, while it is still faster than Robot Framework, it’s not by much and it is still much slower when compared to Cypress or Playwright.

No issues with authentication.

### Testim.io



Testim recorded test steps. Courtesy of methodsandtools.com

Testim is a cloud-based, paid platform that helps with automation of web applications. Testim uses a “recording” feature which follows and records the user’s clicks and interactions as they go through their test and creates separate “cards” in a drag&drop interface, that can be moved, set and manipulated.

Installation is not required, just an account setup and a browser extension.

Testim supports most modern browsers and includes a coding as well as a codeless environment.

Advantages:

**User Specific** – Testim is suitable both for non-technical testers as well as more experienced coders. The record, drag and drop feature allows the user to create tests without having to write a single line of code. Meanwhile, the user is also able to “add a JavaScript step” and write their own JS code to improve the test’s performance and include more complex logic.

**AI powered** - One of the main selling points of Testim is the AI powered locators. Testim uses machine learning algorithms to analyse multiple attributes (ID, class, text, position, etc.) of UI elements and automatically detects if those attributes have changed – modifying the test on the go. Reducing the need for maintenance.

**Advanced reporting dashboards** - Testim took reporting to a whole new level by introducing data visualization-like dashboards that show anything from pie charts successful/failed tests, test duration charts, failure types, company level reports, various filtering capabilities etc. Allowing the users and developers gain better insight into their app’s performance.

**Great Documentation** – The [Testim Docs](https://help.testim.io/docs) has easy to read, step-by-step documentation complete with screenshots to walk the user through the software. Overall Testim has a very flat learning curve.

A screenshot of a computer

Description automatically generated

Testim Reports, courtesy of Testim.io

Disadvantages:

**Expensive** – While Testim offers a 14-day free trial, once the trial expires, the prices can be fairly steep. The Starter plan includes 2 parallel tests and 1,000 monthly test minutes and costs $99 per month or $1,188 per year.

**Limited** – While Testim does offer integration of JS to create more robust tests, being a cloud-based software as opposed to a library or framework that can be opened and easily manipulated and configured in an IDE means it will eventually have its limitations for any use cases beyond what the standard user may need.

Compared to other tools:

In terms of user friendliness and pleasant UI, Testim is the only other tool that can be comparable to Robot Framework , if not surpassing it. Both tools make it extremely easy for a non-technical user to build, maintain and run tests, while showing detailed and easy to understand reports and logs after every test run. No more ambiguous console errors that point to some mysterious line in your code, that then points to some hard-to-find locator you’ve long forgotten about since writing the test…

Testim’s advanced AI locators make it much more reliable than Robot Framework (and any of the other tools, I must say) and make maintenance a breeze.

At the same time, Testim runs much faster than Robot, essentially making it the only rival that can surpass it in every single one of the scoring criteria. However, this performance comes at a cost.

Testim’s only pitfall in compared to other testing tools is the lack of ability to customize certain features as it can only use what it provides out-of-the-box and personal additions are limited.

### WebdriverIO

A screenshot of a computer

Description automatically generated

WebdriverIO with Allure reporting

WebdriverIO is another open-source, free, Node.js based web automation tool that uses JavaScript to run e2e tests. While not much different than Cypress, TestCafe or Playwright, in that it’s a Node.js based system, which is easy to install but requires various plugins and libraries if one would like to add extra functionality to it (like more intricate reports than a console log, or video capture of the test runs etc.) WebdriverIO makes it a bit easier by providing configuration options upon creation of the project. So, the other libraries and dependencies can be installed along with the testing software, and all configurations are included in the config file. It supports most modern browsers.

It has no issues with web authentication to access the website.

A screenshot of a computer program

Description automatically generated

WebdriverIO config upon project creation

Advantages:

**Very Fast** – Probably one of the fastest testing tools I’ve checked. Even when running on a headed browser, WebdriverIO completed our standard “Application Stage 1” test in about 13seconds.

**Extensive plugin ecosystem** – has a variety of plugins and services that can be installed along with the main project folder setup. So while it doesn’t come “out of the box” with these features, it does help install them so they do not have to be added one by one manually, saving time.

**Parallel Executions** – Users can configure the number of parallel tests they’d like to run directly in the config file, speed doesn’t seem to be affected too much by it.

Disadvantages:

**Steeper learning curve** – WebdriverIO uses only JavaScript to write tests and does not actually provide a screen recorder or selector finer feature to help identify or speed up the selector finding process. Nor is it non-coder friendly.

**Reports are complicated –** Although WebdriverIO provides a few different reporting mechanisms, neither provide very easy to understand insightful reports. For instance, Spec is just a simple command line log. Allure and Timeline provide slightly more impressive, visualization graph style reports, but still don’t show clear step by step go-through of the test, but rather “POST” and “GET” requests and responses). Although a meaningful error message is still displayed, in a large test, it would be hard to pinpoint where the error occurred and what needs to be changed.

**A screenshot of a computer

Description automatically generated**

Allure error log

**Compared to Robot Framework and other tools:**

WebdriverIO is pretty much on par with the rest of the open source, Node.js based automation tools, with the added bonus of configuration on install but with the disadvantage of not having a “locator helper” to speed up the test writing.

Reports are on part with the rest of the Node.Js tools as well and far inferior to those of Robot or Testim.

Speed-wise, it’s faster than most of the tools presented so far.

### Selenium

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Comparison Chart | *Languages* | *Speed* | *Installation/*  *Setup* | *Complexity of learning/using* | *Logging/reporting* | *Code Assists/Element Location Tools* | *Free/Paid* |
| Robot Framework | Natural Language. Keywords. | Slow. | Medium | Very Easy | Very detailed. Easy to read. Well-presented.  Includes Screenshots on error. Even an unrelated 3rd party could read and understand. | None. Manual element location. However, no coding required. | Free |
| Cypress | JavaScript. | Very Fast. | Easy | Easy | Time travel snaps. A bit more complex to read, but detailed enough. | “Selector Playground” for element location. No coding assist. | Free with paid tier available. |
| Playwright | Javascript, Typescript, Java, C#, Python. | Very Fast. | Medium | Medium | In-console logs. Presented like developer logs. Hard to decipher for anybody who hasn’t actually written the code itself. Screenshots/video require configuration. | “Codegen” code assist. Generates the code for you and “translates” to any of the supported languages. Can record your actions in the UI. | Free |
| TestCafe | JavaScript  /Typescript | Medium. | Easy. | Medium. | In-console logs. Presented like developer logs. Hard to decipher for anybody who hasn’t actually written the code itself. Screenshots/video require configuration. | None. All coding and element locations need to be done manually. | Free |
| Testim | JavaScript | Fast. | Easy. | Very Easy | Detailed step by step logs during execution, intricate data visualization charts in general reports. | Drag & Drop style dashboard that record user interactions. AI powered locators that change dynamically if the locators change in the code. | Paid with 14 day free trial |
| WebdriverIO | JavaScript | Very Fast. | Easy. | Medium. | In console logs.  Hard to decipher for the most part.  Can use several plugins like allure, spec, timeline etc. all available to install during setup of WebdriverIO project. | None. All coding and element locations need to be done manually. | Free. |
| Selenium |  |  |  |  |  |  |  |