Cypress Automation

Edit New page

Jump to bottom

Nikita Deepak Poyrekar edited this page 10 days ago · 7 revisions

INTRODUCTION

Software testing is an activity conducted in the software development lifecycle to verify that the software functionality is accurate and confines to the requirements.

You can test a particular software in many ways. The standard types of software testing used in FinApp are –

1. Unit testing
2. Smoke testing
3. End-to-end testing
4. Manual testing

This article focuses on end-to-end testing using the Cypress framework.

THE CYPRESS ECOSYSTEM

End-to-end testing aims to assert the flow of an application from start to finish. Instead of manually clicking around an application, you can write a test that runs as you build the application. It mirrors user experience as the written test acts like a real human when it tests several pieces of the application at once.

Cypress is one of those end-to-end testing frameworks which does all that clicking work for us. It consists of a free, open-source, locally installed Test runner and a Dashboard Service for recording your tests.

Features:

Cypress comes fully baked, batteries included. Some of its key features include:

1. Time travel: Cypress takes snapshots as your tests run. Hover over commands in the Command Log to see exactly what happened at each step
2. Debuggability or Debug-ability: Stop guessing why your tests are failing. Cypress provides descriptive messages at every step
3. Automatic waiting: Cypress never adds waits or sleeps to your tests. It automatically waits for commands and assertions before moving on
4. Stub responses: Cypress can verify and control the behavior of functions, server responses, or timers
5. Screenshots and videos: With Cypress, you can view screenshots that are taken automatically on failure, or videos of your entire test suite when run from the CLI
6. Cross-browser testing: Run tests within Firefox and Chrome-family browsers (including Edge and Electron) locally.

Basic constructs of Cypress:

Cypress comes with bundled tools like Mocha and Chai.

Cypress has adopted Mocha's syntax for the development of test cases, and it uses all the fundamental harnesses that Mocha provides. Below are a few of the main constructs which we majorly use in Cypress test development:

1. describe() - It is simply a way to group our tests. It takes two arguments, the first is the name of the test group, and the second is a callback function

2. it() – It is used for an individual test case. It takes two arguments, a string explaining what the test should do, and a callback function that contains an actual test
3. before() - It runs once before all tests in the block
4. beforeEach() - It runs once before all tests in the block
5. after() - It runs after all tests in the block
6. afterEach() - It runs after each test in the block
7. .only() - To run a specified suite or test, append ".only" to the function
8. .skip() - To skip a specified suite or test, append ".skip" to the function.

Learn more about Mocha here.

While Mocha provides us a framework to structure our tests, Chai gives us the ability to easily write assertions. Chai gives us readable assertions with excellent error messages. Below are a few examples of the main assertions which we majorly use in Cypress test development:

1. not - expect(name).to.not.equal('test')
2. true - expect(true).to.be.true

Click here to go through the list of commonly used assertions.

Cypress installation:

Cypress.io installs easily with NPM. Type this into your terminal to install it for your project:

```
PS C:\pro\eu-geft\src\main\webapp> npm install --save-dev cypress
```

Note: Run the above command in the '/webapp' directory of our FinApp application.

This is the final output that is displayed once Cypress is installed and up on your system.

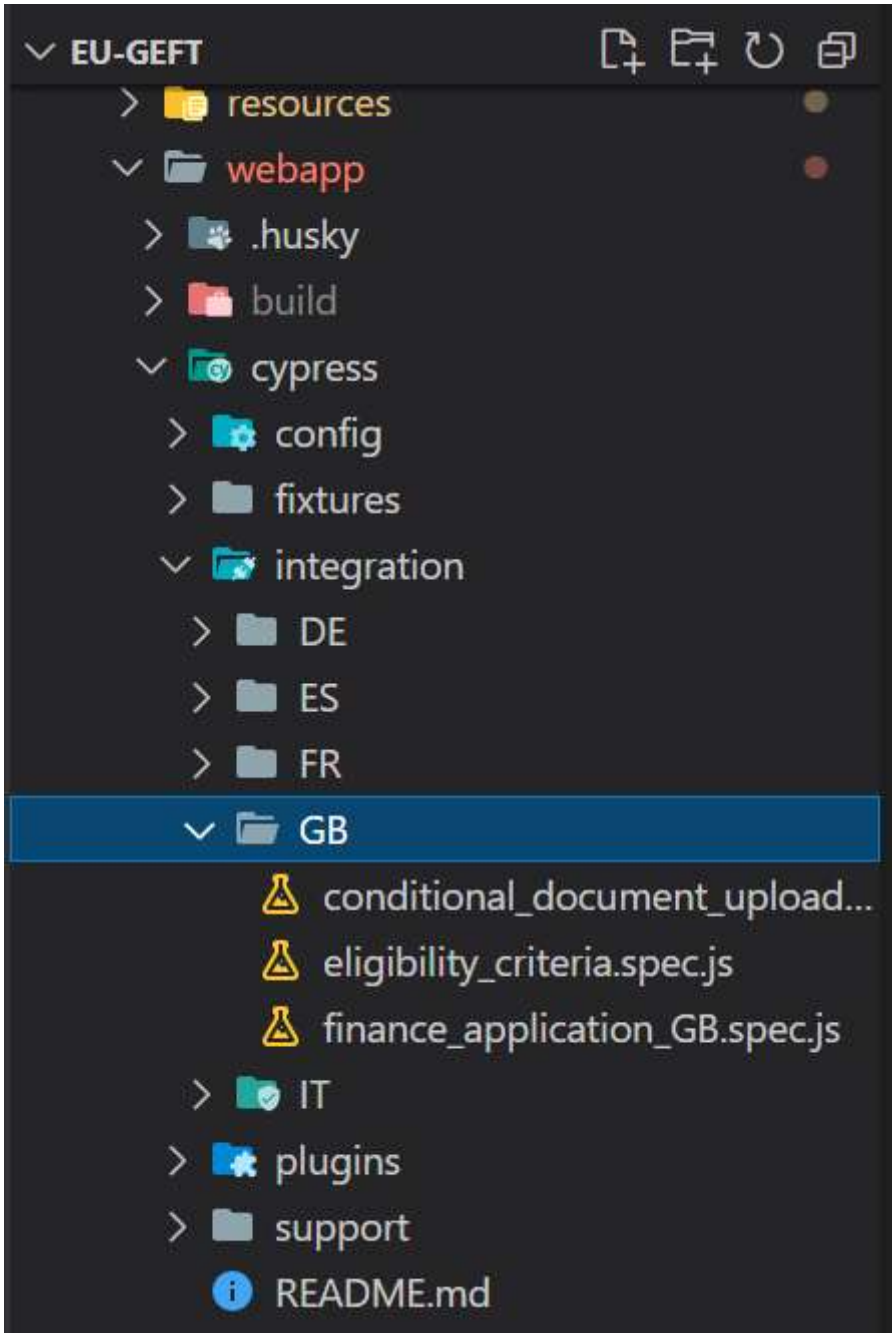
```
PS C:\pro\eu-geft\src\main\webapp> npm install --save-dev cypress

> cypress@9.5.1 postinstall C:\pro\eu-geft\src\main\webapp\node_modules\cypress
> node index.js --exec install

Installing Cypress (version: 9.5.1)

✓ Downloaded Cypress
✓ Unzipped Cypress
✓ Finished Installation C:\Users\NPOYREKA\AppData\Local\Cypress\Cache\9.5.1

You can now open Cypress by running: node_modules\.bin\cypress open
```

Cypress installation in VDI:

1. Go to C:\Users\Your_Username
2. Open the file .npmrc with Notepad
3. Replace the contents of the file with **proxy**=http://internet.ford.com:83, **https-proxy**=http://internet.ford.com:83
4. Install Cypress as mentioned: npm install cypress@9.7.0 --save-dev

Folder structure:

Once the package is installed on your system, a folder named Cypress is created under the webapp directory with a few more default folders under it.



1. Config – Config is used to store different configurations. E.g., timeout, base URL, or any other configuration that we want to override for tweaking the behavior of Cypress

2. Fixtures - Fixtures are external pieces of static data that can be used by your tests
3. Integration - The integration folder provides us a space to write our test cases
4. Plugins - Plugins contain the plugins or listeners. By default, Cypress will automatically include the plugins file "cypress/plugins/index.js" before every test it runs.
5. Support - Support writes customized commands or reusable methods that are available for usage in all your spec/test files.

Set up a simple automation test case:

**Test subject: ** Let's start with navigating on the eligibility criteria page and then land on the finance application page of FinApp. Firstly, create an eligibility_criteria.spec.js file under the cypress/integration folder and add a describe() to the spec file as shown below.

```
describe("Eligbility Screen", () => {

});
```

The describe() takes two arguments: a string, which is the "subject" of the test group, and a callback function that can run any code you want. The callback function should probably also call it() which tells us what we expect to happen in this test and checks for that outcome as shown below.

```
describe("Eligbility Screen", () => {
    it("should navigate to Finance Application when privacy checkbox is checked and Continue is clicked", () => {

    });
});
```

The it() also takes two arguments: a string or subject of the test and a callback function. The code we run inside the it callback should ultimately check our assertion (our desired result) for this test against reality. Note that the describe callback can contain multiple calls to it. Best practice says each it callback should test one assertion.

Now, let's further set the test for navigating to the finance application page. As per the subject mentioned in it callback, we have to navigate to the finance application screen and this is possible if we first land on the eligibility criteria page. Therefore, let's add a beforeEach() before actually writing the test as shown below:

```
describe("Eligbility Screen", () => {

    beforeEach(() => {
        cy.visit("/gb?countryCode=gb&orderId=sit-fake-order");
      });

    it("should navigate to Finance Application when privacy checkbox is checked and Continue is clicked", () => {

    });
});
```

Now we have successfully landed on the eligibility criteria page. Let's get hold of clickable DOM elements and then navigate to the finance application screen.

```
describe("Eligbility Screen", () => {

    beforeEach(() => {
        cy.visit("/gb?countryCode=gb&orderId=sit-fake-order");
      });

    it("should navigate to Finance Application when privacy checkbox is checked and Continue is clicked", () => {
        cy.get('[data-testid="privacyCheckbox"]').check();
        cy.get('[data-testid="continueButton"]').click();
    });
});
```

The above commands will load up the finance application screen page in the web browser. But to verify it, we need to write out an assertion as follows.

```
describe("Eligbility Screen", () => {

    beforeEach(() => {
        cy.visit("/gb?countryCode=gb&orderId=sit-fake-order");
      });

    it("should navigate to Finance Application when privacy checkbox is checked and Continue is clicked", () => {
        cy.get('[data-testid="privacyCheckbox"]').check();
        cy.get('[data-testid="continueButton"]').click();

        cy.url().should("include", "finance_application");
    });
});
```

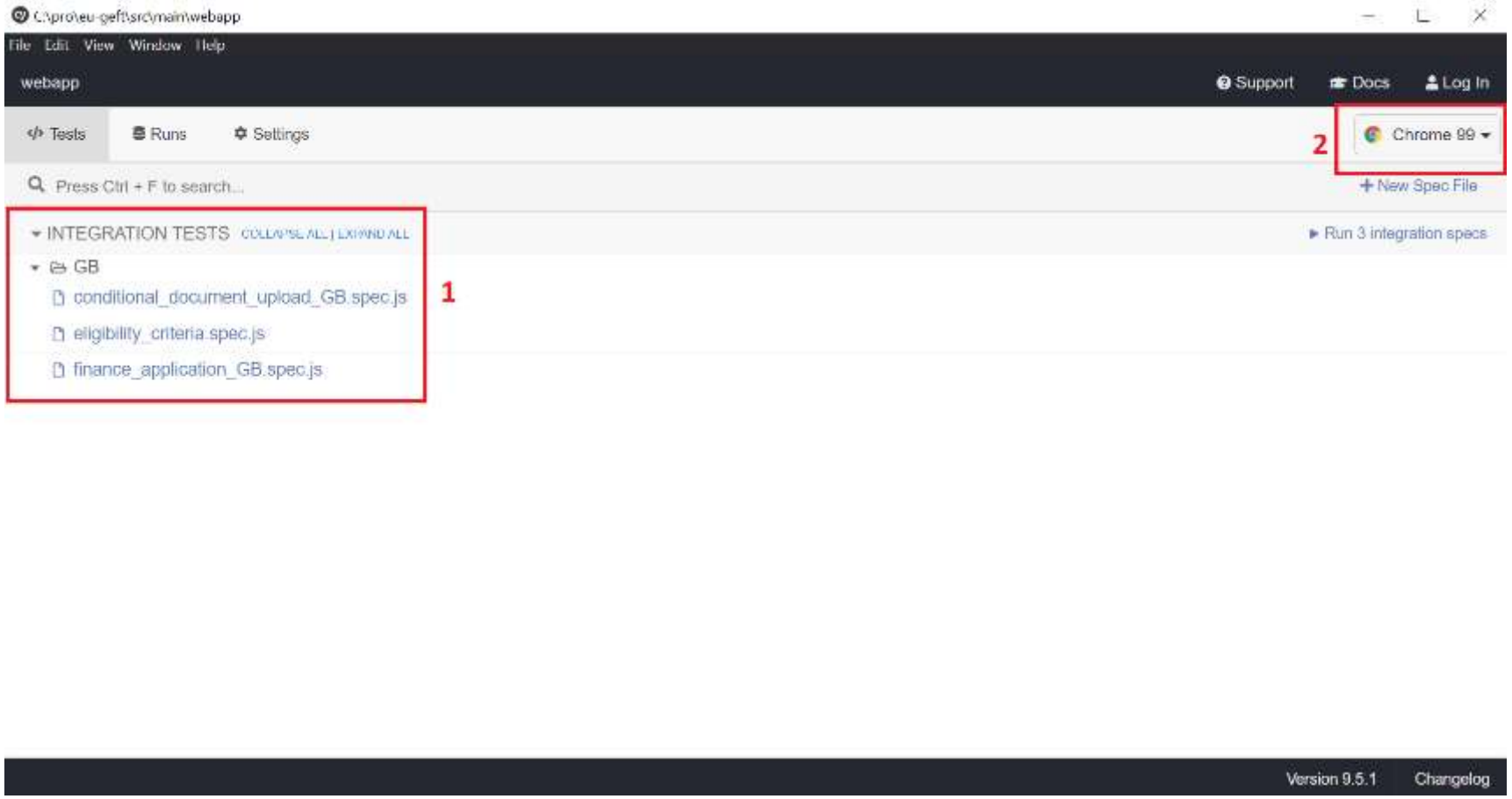
cy.url() yields the page url. We chain it with should() which creates an assertion. In this example, we pass should() two arguments: a chainer and a value. Here, the assertion then reads as – "The page URL should include finance_application".

Executing a cypress test:

Cypress tests can be executed in two ways:

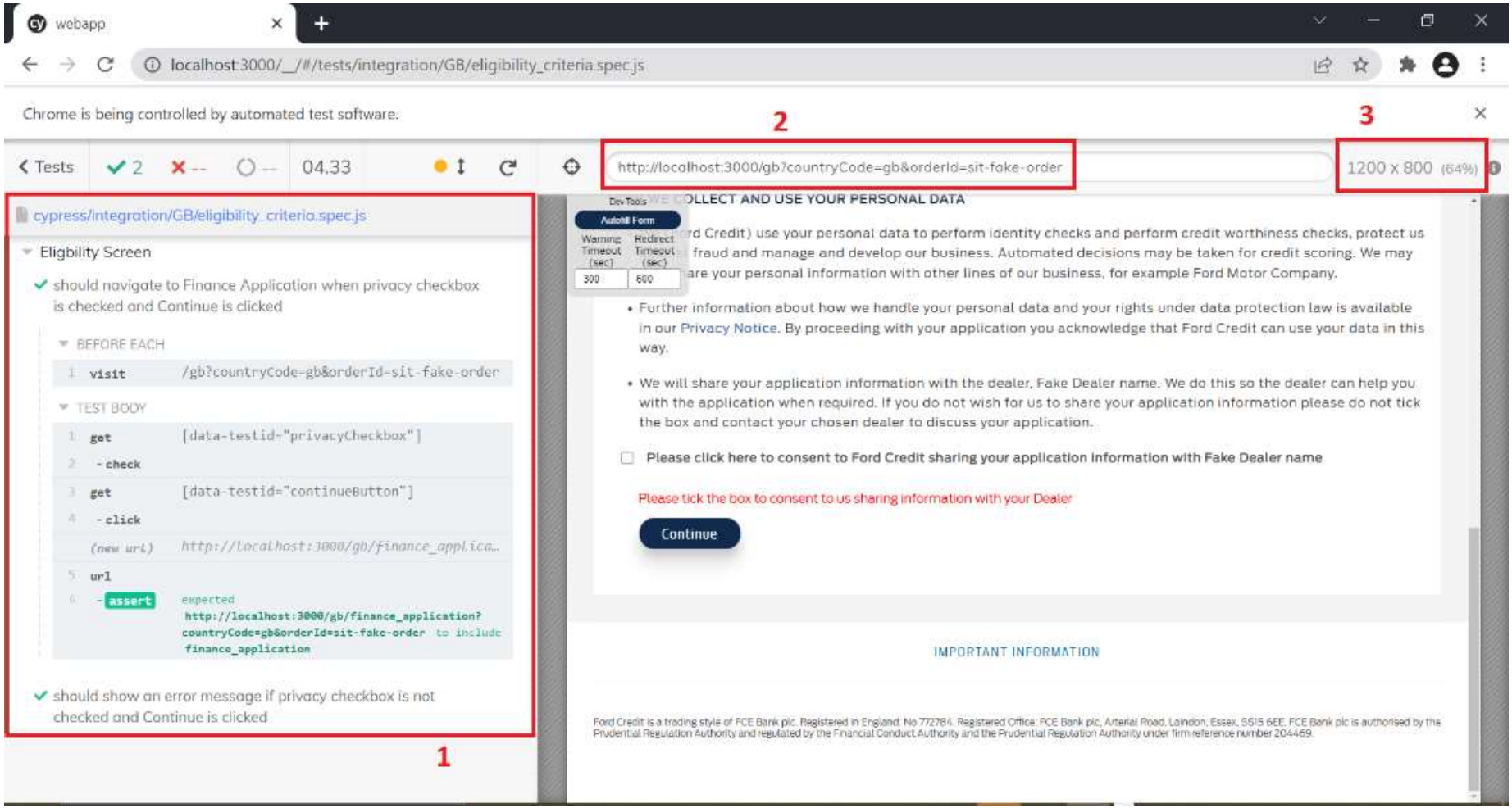
1. Via a test runner For each test automation framework, test runners provide the entry point for kicking off the execution of the test cases. Cypress has a unique test runner that allows us to see commands as they execute. Additionally, it also shows the real-time run of the application under test. Follow the steps below to open the test runner.
    - Start the backend of our application
    - Start the Frontend of our application with the command- `npm run start:noauth` . This disables authentication in the app for quick local testing
    - When both the backend and front-end are up and running, use the following command to start Cypress: `npm run cy: open` or `node_modules/.bin/cypress open`

- After you run the above command, the cypress test runner window will open as shown below. #1 represents the list of test suites. Clicking on any one of them opens the test runner for a particular spec file. You can also select the browser of your choice from the top right corner of the test runner (#2). The Cypress Test Runner tries to find all compatible browsers on the user's machine.



- Let's click on the eligibility_criteria.spec.js file

- The screenshot below shows the execution of the test case. The Cypress test runner displays the test suite and all tests belonging to it [#1] on the left side and their execution on the right-hand side panel. Details about every step in a particular test case are also highlighted on the left side. #2 states the URL of the current webpage displayed in the browser and #3 shows the browser width that helps us to understand the responsiveness of our application.



- In some cases, we tend to change our code while writing our tests or include more steps or assertions. The Cypress Test Runner is smart enough to detect any saved change in your test file and starts executing your test from the start whenever you save any code changes on your local.

2. Via command line or headless Running the test cases in UI mode is more suitable when the development of test cases is in progress. But once the development completes, the user would want to run the test cases in headless mode. Also, running test cases in UI is always slower than running in headless mode. Cypress fulfills all these needs and provides ways to execute the test cases from CLI.

We can run the following command to execute all the tests from the CMD line:

```
PS C:\pro\eu-geft\src\main\webapp> ./node_modules/.bin/cypress run
```

or

```
PS C:\pro\eu-geft\src\main\webapp> npx cypress run
```

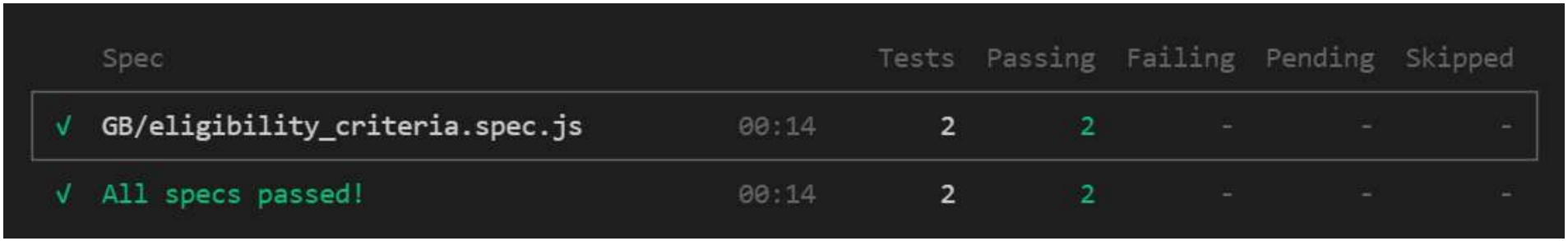
If you want to execute a specific test file, you can use the spec flag as below -

```
PS C:\pro\eu-geft\src\main\webapp> npx cypress run --spec "cypress/integration/GB/eligibility_criteria.spec.js"
```

The terminal then outputs the results with a summary of test cases that were executed.

```
PS C:\pro\eu-geft\src\main\webapp> npx cypress run --spec "cypress/integration/GB/eligibility_criteria.spec.js"



====================================================================================================

  (Run Starting)

  ┌────────────────────────────────────────────────────────────────────────────────────────────────┐
  │ Cypress:        9.5.1                                                                          │
  │ Browser:        Electron 94 (headless)                                                         │
  │ Node Version:   v14.17.0 (C:\Program Files\nodejs\node.exe)                                    │
  │ Specs:          1 found (GB/eligibility_criteria.spec.js)                                      │
  │ Searched:       cypress\integration\GB\eligibility_criteria.spec.js                            │
  └────────────────────────────────────────────────────────────────────────────────────────────────┘


────────────────────────────────────────────────────────────────────────────────────────────────────

  Running:  GB/eligibility_criteria.spec.js                                             (1 of 1)
Browserslist: caniuse-lite is outdated. Please run:
npx browserslist@latest --update-db

Why you should do it regularly:
https://github.com/browserslist/browserslist#browsers-data-updating


  Eligbility Screen
    √ should navigate to Finance Application when privacy checkbox is checked and Continue is clicked (9707ms)
    √ should show an error message if privacy checkbox is not checked and Continue is clicked (4316ms)
```

```
2 passing (14s)


(Results)

┌────────────────────────────────────────────────────────────────────────────┐
│ Tests:        2                                                            │
│ Passing:      2                                                            │
│ Failing:      0                                                            │
│ Pending:      0                                                            │
│ Skipped:      0                                                            │
│ Screenshots:  0                                                            │
│ Video:        true                                                         │
│ Duration:     14 seconds                                                   │
│ Spec Ran:     GB/eligibility_criteria.spec.js                              │
└────────────────────────────────────────────────────────────────────────────┘


(Video)

-  Started processing:  Compressing to 32 CRF
-  Finished processing: C:\pro\eu-geft\src\main\webapp\cypress\videos\GB\eligibilit   (10 seconds)
                        y_criteria.spec.js.mp4

  Compression progress:  100%

================================================================================

(Run Finished)
```



	Spec		Tests	Passing	Failing	Pending	Skipped
√	GB/eligibility_criteria.spec.js	00:14	2	2	-	-	-
√	All specs passed!	00:14	2	2	-	-	-

ENABLE AUTHENTICATION FOR E2E TESTING ON LOCAL

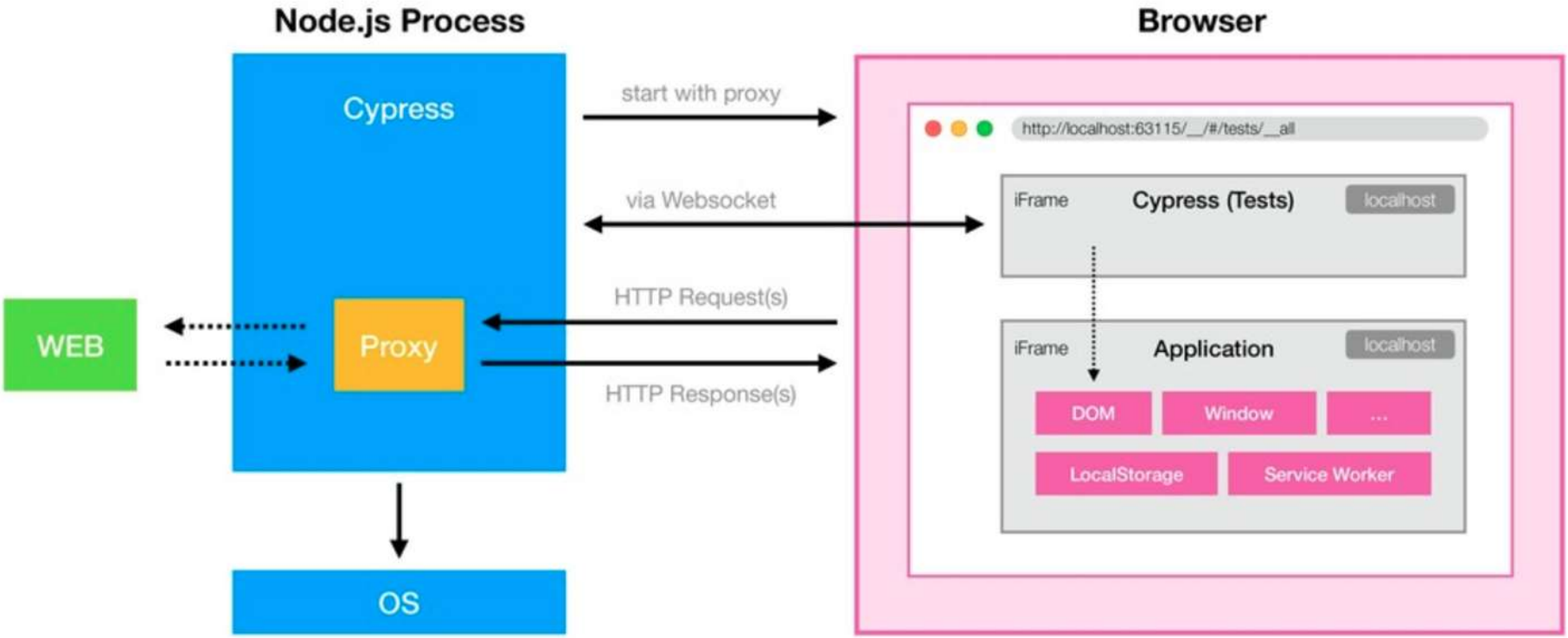
A valid token is needed to make successful API calls. To enable authentication while running Cypress tests on our local machine, follow the below-mentioned steps.

1. Update the base URL in the local.json file to point to the QA environment.
2. Open Git BASH inside the Webapp folder and run the following commands:
   - export http_proxy=http://internet.ford.com:83
   - export https_proxy=http://internet.ford.com:83
   - export no_proxy="localhost,127.0.0.1"
   - export HTTP_PROXY=http://internet.ford.com:83
   - export HTTPS_PROXY=http://internet.ford.com:83
   - export NO_PROXY="localhost,127.0.0.1"
   - export NO_PROXY="*cms.cf.app.ford.com, *.cf.ford.com, *.authagent.ford.com"
   - export no_proxy="*cms.cf.app.ford.com, *.cf.ford.com, *.authagent.ford.com"
3. Check whether all proxies are set using - env | grep -i proxy
4. In the same terminal, run cypress using - npm run cy:open.
5. Modify your test file to enter valid login credentials after hitting the respective FinApp page URL.

How to get proposal Ids for France or Great Britain?

1. Visit the QA URL for France or Great Britain
2. Click the continue button from the Eligibility criteria page to land on the finance application
3. Click on the autofill button on the top left corner of the page
4. Right-click on the page, select inspect and open the network tab
5. Submit the application and observe the sendOrderWithDocs API call
6. Upon successful submission of the application, click on the Preview tab and check the response with a format {data: XXXXXXX}. The XXXXXXX is the proposal id for the respective order id mentioned in the URL.

CYPRESS ARCHITECTURE



> Cypress' architecture during a test consists of a Node.js process (the "server-side") and a running browser. The browser, instrumentalized by Cypress, shows a page that embeds the tested application as well as Cypress as an iFrame. Test code is also executed in the Cypress iFrame so that the test code and application code run in the same browser tab and thus in the same JavaScript loop. The test code communicates with the Node.js process via WebSockets. The Node.js process acts as a proxy for every HTTP request of the browser and can even execute shell commands. [1]

Key takeaways:

1. Cypress provides a default folder hierarchy, hence it's easy to start with the test development
2. Cypress provides a global `cy` object that allows us to invoke multiple methods for testing meaningful assertions
3. Easy to launch – just type in simple `open` command through CLI or launch a UI Test Runner
4. A specific test case can be executed on the CLI using the "--spec" option.
5. Cypress uses Mocha's BDD constructs for writing test cases and Chai's assertion for validating the actions.

What next:

1. Expand test cases to mock or stub API responses
2. Enable authentication for E2E testing - test locally
3. Add Cypress to the Jenkins pipeline
4. Run automation test cases during build time – First on Dev, then QA and later on EDU
5. Improvise test cases with additional test scenarios for regression testing (optional).

Reference: Official document of Cypress

▸ **Pages** 49

- Home
  - Templates/Accelerators/KAs
    - Country Template
    - Environment Mappings
    - Finance E2E Journey Document
    - Project-Phases
    - Release-Cadence
    - SonarQube Implementation
    - User Story workflow
    - Ways of Working
  - Quality Assurance
    - QA Framework
    - Test Data
    - Test-Environment-URLs
    - Cypress Automation
      - Automate user authentication
  - Design Docs
    - E2E journey
    - Web Analytics
    - Wiremock
    - Retry Logic
      - Retry Logic POC
    - Frontend logging user actions
    - UK Adjusted First InstalmentAmount
      - Feature toggle
    - Content Authoring capability
      - Strapi CMS
      - Musings

        - Save and Return
        - Micro Frontends
        - Secrets Management
        - PostgreSQL DB
    - GCP Migration
        - Migration Strategy
    - API Security
        - 42Crunch
    - Release Notes
        - EU2022.02 Release notes
        - EU2022.03 Release notes
        - EU2022.04 Release notes
        - EU2022.05 Release notes
        - EU2022.06 Release notes
        - EU2022.07 Release notes
        - EU2022.08 Release notes
        - EU2022.09 Release notes
        - EU2023.Feb Release notes
        - EU2023.Apr Release notes
        - EU2023.May Release notes

Clone this wiki locally

```
https://github.ford.com/global-ecommerce-finance-tech/eu-geft.wiki.git
```