# Web App Front-end Testing with Cypress

Software Product Assurance Workshop 2023
European Space Astronomy Centre, Spain

Marian Cuevas, Fernando Guerrero
*RHEA Group*

ESA ESAC

25/09/2023

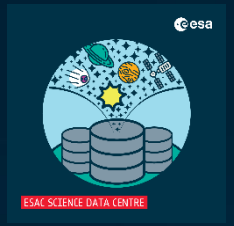
ESA UNCLASSIFIED – For ESA Official Use Only

# Table of Contents

# Web App Front-end Testing with Cypress

RHEA GROUP for esa
SSC AURORA TECHNOLOGY

# About ESAC Science Data Centre





ESAC Science Data Centre (https://www.cosmos.esa.int/web/esdc) is located a ESAC facilities in Villanueva del Castillo and is responsible for the ESA science missions archives.

- Over 30 archives… and more to come.
- Storage to reach the Petabyte scale.

The challenges:

- Data preservation and curation, log after mission termination.
- Data availability for the science community via several endpoints, among which are the archives' web sites.
- Cost optimization.
- Keep technologies up-to-date: deprecation of obsolete platforms and migration to modern frameworks.
- Strategy focused on long-term maintenance, reliable technologies.

# Web App Front-end Testing with Cypress

RHEA GROUP for esa
SSC AURORA TECHNOLOGY

# Cypress Overview

cypress (https://www.cypress.io/) is a **free open-source** JavaScript-based end-to-end testing designed to work with modern web development frameworks such as Angular. A commercial Cypress Cloud solution for enterprises is also available.

Recommended readings:

https://docs.cypress.io/guides

https://docs.cypress.io/api/table-of-contents

## Cypress vs Selenium

- Cypress provides a robust, complete framework for running automated tests but takes some of the freedom out of Selenium by confining the user to specific frameworks and languages
- Selenium supports various programming languages (Java, Python …), and provides a suite of tools for testing web applications, including Selenium WebDriver, Selenium Grid, and Selenium IDE
- Cypress uses a completely different approach to testing than Selenium. While Selenium WebDriver runs remotely outside the browser and executes remote commands into the browser, Cypress runs inside the browser

# Cypress Overview

RHEA GROUP for esa
SSC AURORA TECHNOLOGY

**Cypress is intended for functional testing**

- End-to-end testing
- Component testing
- Integration testing
- API testing
- Unit testing

**Cypress is not a non-functional testing tool**

- Performance testing
- Load testing
- Usability testing
- Security testing

# Web App Front-end Testing with Cypress

RHEA GROUP for esa
SSC AURORA TECHNOLOGY

# Why Cypress?

Some years ago, ESDC decided to deprecate Google Web Toolkit and use Angular for web developing as part of a long-term strategy. Consequently, the testing framework should fit the same principles. There are several well-known testing tools for web applications (e.g., Selenium) but it was finally decided to use Cypress due to the following reasons:

- ✓ **Modern** testing framework for JavaScript solutions, including Agular, and provides built-in support for them.
- ✓ Expected to be **maintained** and evolve **in the long-term**.
- ✓ Very good **documentation** and active **community**.
- ✓ **Productivity**:
  - ➢ Easy to set up.
  - ➢ Fast learning curve.
  - ➢ Easy-to-use debugging.
  - ➢ Automatic waiting: no need for complex flow coding to create tests.
- ✓ **Cross browser**: currently supporting Chrome-family browsers, including Microsoft Edge, WebKit (Safari's browser engine), and Firefox.
- ✓ Possibility to extend the functionality via **plugin extensions**. Many plugins already available. Possibility to design a custom plugin.

# Web App Front-end Testing with Cypress

RHEA GROUP for esa
SSC AURORA TECHNOLOGY

# Cypress Core Concepts

Writing tests:

Tests are grouped in **spec files**, and structured the way described following:

`describe()` or `context()` are synonyms used for grouping tests.

`it()` or `specify()` are synonyms that correspond to an individual test.

Hooks:

- `before()`: runs once before all tests.
- `after()`: runs once after all tests.
- `beforeEach()`: runs once before each test.
- `afterEach()`: runs once after each test.

Hooks are run within their describe block and subsequent nested blocks.

```
describe( title: 'Main Block',  fn: () : void  => {
  before( fn: () : void  => {
    //runs once before all tests
  });
  beforeEach( fn: () : void  => {
    //runs once before each tests in this block and nested ones
  });
  afterEach( fn: () : void  => {
    //runs once after each tests in this block and nested ones
  });
  after( fn: () : void  => {
    //runs once after all tests are done
  });
  it( title: 'Test case in main block',  fn: () : void  => {
    //Test code
  });

  describe( title: 'Nested Block',  fn: () : void  => {
    before( fn: () : void  => {
      //runs once before all tests in this block
    });
    beforeEach( fn: () : void  => {
      //runs once before each tests in this block
    });
    afterEach( fn: () : void  => {
      //runs once after each tests in this block
    });
    after( fn: () : void  => {
      //runs once after all tests are done in this block
    });
    it( title: 'Test case in nested block',  fn: () : void  => {
      //Test code
    });
  });
});
```

# Cypress Core Concepts

**Test isolation**: by default, browser state, including DOM, cookies and storage is cleared before each tests. Tests state, such as spies, or viewport changes are also reset.

**Automatic retries**: Cypress will retry queries on DOM elements, assertions, and actions automatically, saving the programmer time and effort.

**Cypress commands are asynchronous**: `cy.*` commands and chains of commands return immediately, after having been appended to a queue that will run after the test code has been executed. It is a common mistake to mix synchronous and asynchronous commands.

**Querying elements**: there are several ways to identify and select DOM elements. E.g., `cy.get('.element')`.

**Chaining commands**:

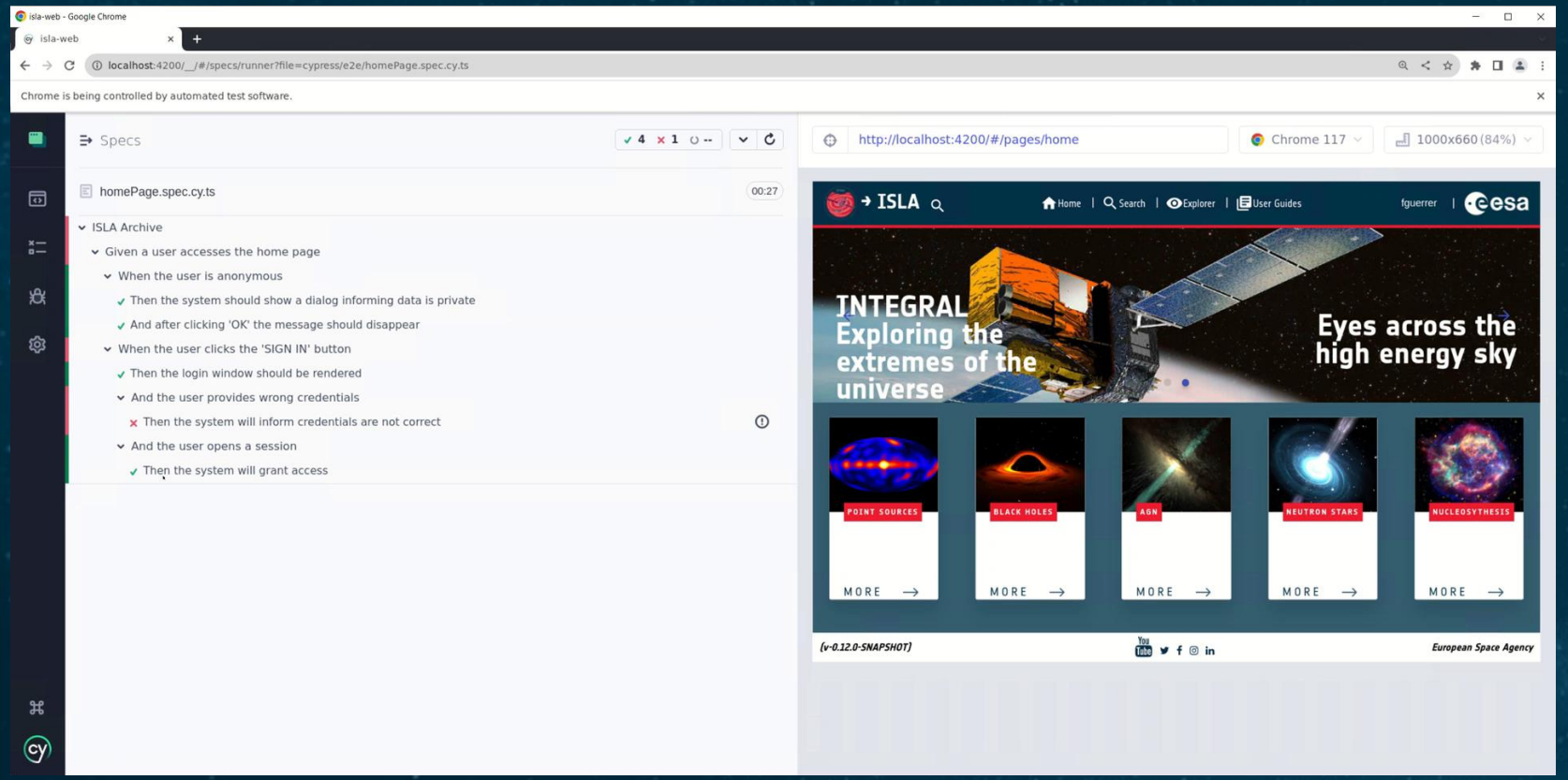
```
cy.get('#selector').then(($element) =>{$element.contains('button').click()})
```

**Assertions**: assertions can be used to evaluate the state of elements, objects or the application itself.

```
cy.get('#selector').should('contain.text', 'Welcome to the PA Workshop!')
```

Some commands have built-in assertions. E.g., `.click()` expects the element to be in an actionable state or will yield an error.

# Cypress Core Concepts




**Time travel**: Cypress test runner has a command log that is a representation of the test suite.

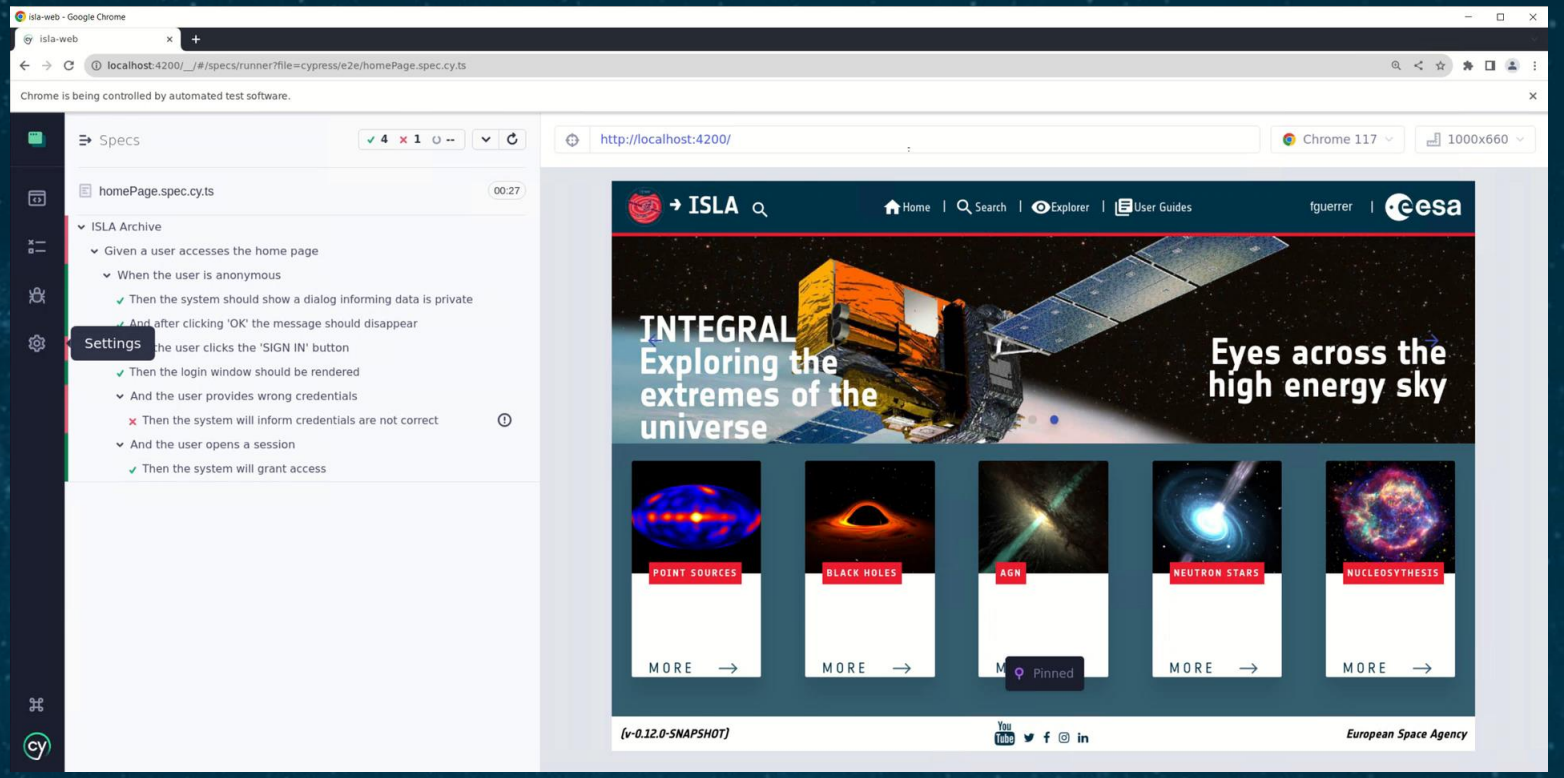
On the right it shows the application under test.

Hovering over each command restores the application to the state it was when the command was executed.

# Cypress Core Concepts

**Native browser inspection**: Open developer tools and inspect your testo or application as you would do in a normal application.

- ✓ Get logs and outputs for your commands in the console.
- ✓ Access the DOM.
- ✓ Set breakpoints in your test or application code and debug.
- ✓ Review network requests and responses.



# Cypress Core Concepts

Practical example of basic user test: access to a web site that requires a login.

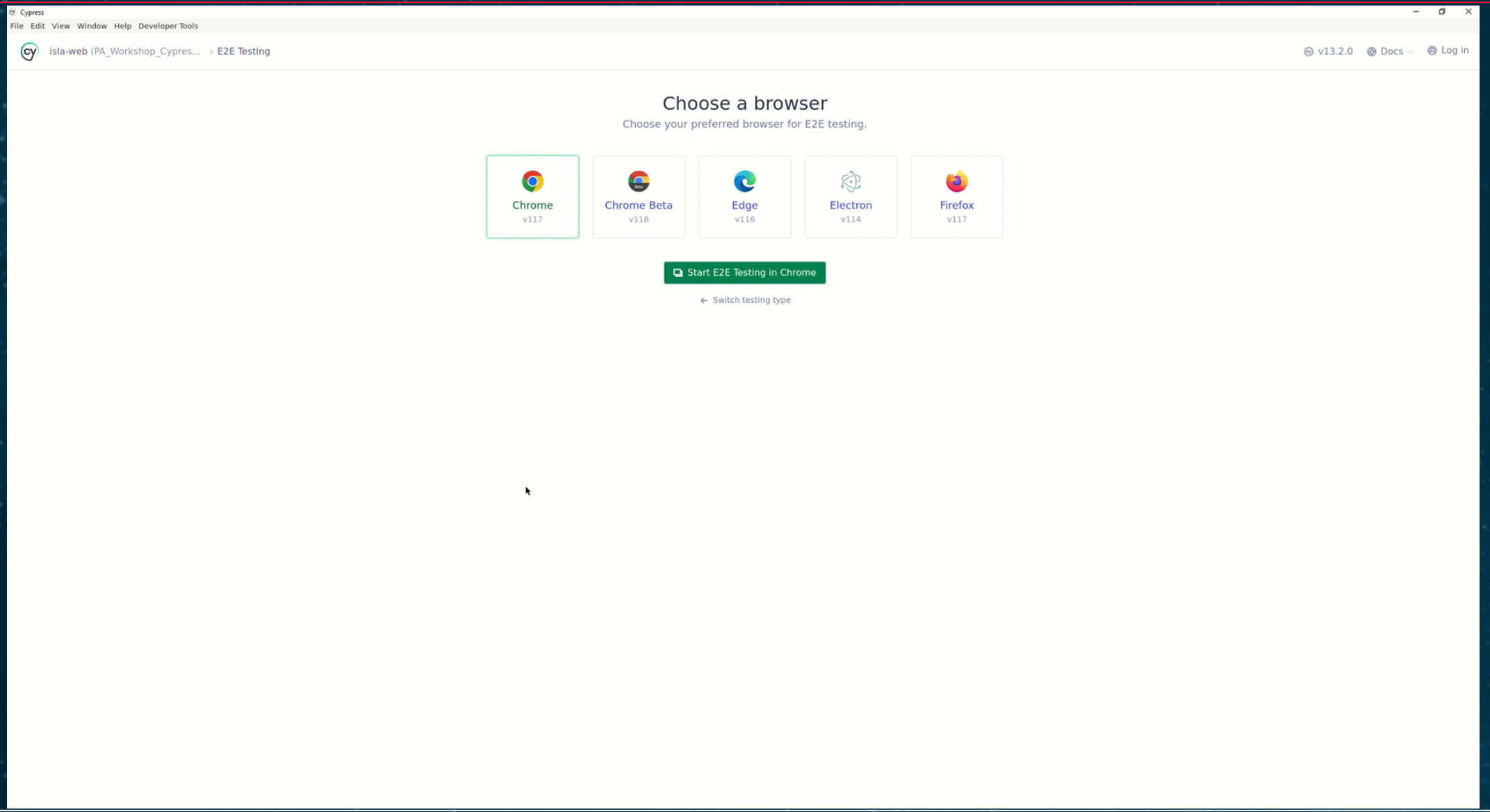
In this example:

- ✓ Running tests interactively. Browser selection
- ✓ Cypress front end
- ✓ Command log: reviewing test suite's steps
- ✓ Assertions
- ✓ Selectors

If you have any question, please use the following link or QR code to submit it and we will try to answer it:
https://forms.office.com/e/sXW1NfK2Tt

**Hands-on Session Questions**

# Cypress Core Concepts



# Web App Front-end Testing with Cypress

RHEA GROUP for esa
SSC AURORA TECHNOLOGY

# Session Handling

Cypress provides a cy.session() command to cache and restore cookies, localStorage and sessionStorage so that **browser context is recreated between tests**:

- Programmatically executes a log in and cache session data.
- Optional validation method to trigger re-creation of a failing restored session.
- Possibility to switch between different sessions in the same tests.
- Ability to modify session data before caching.

When working with `cy.session()`:

1. If the session is not cached, then a **new session** is created and cached.
2. If the session is cached and valid, session is **restored**.
3. If the session is cached and invalid, then session is **re-created**.
4. If session cannot be created, restored, or re-created, **test fails**.

# Session Handling

Example of session cache.

In this example:
- ✓ Caching a session
- ✓ Session validation
- ✓ Session creation, re-creation and restoring

If you have any question, please use the following link or QR code to submit it and we will try to answer it:
https://forms.office.com/e/sXW1NfK2Tt

**Hands-on Session Questions**

# Session Handling

isla-web - Google Chrome

isla-web

localhost:4200/__/#/specs

isla-web
PA_Workshop_Cypress_...

Specs

Runs

Debug

Settings

Specs

v13.2.0 Chrome 117 Docs Log in

Optimize and record your CI test runs with Cypress Cloud

When you configure Cypress to record tests to Cypress Cloud, you'll see data from your latest recorded runs in the Cypress app. This increased visibility into your test history allows you to debug your tests faster and more effectively, all within your local workflow.

Get started with Cypress Cloud

Search specs 7 matches + New spec

E2E specs Component specs Run 7 specs Last updated Latest runs Average duration

cypress / e2e

| | | | |
|---|---|---|---|
| homePage.spec.cy.ts | 2 weeks ago | -- | -- |
| intercept.spec.cy.ts | 3 weeks ago | -- | -- |
| reusingSessionsInTests.spec.cy.ts | 9 weeks ago | -- | -- |
| search.spec.cy.ts | 3 weeks ago | -- | -- |
| spies.spec.cy.ts | 4 weeks ago | -- | -- |
| usingCustomLoginAndLogoutCommands.spec.cy.ts | 9 weeks ago | -- | -- |
| visualTesting.spec.cy.ts | 3 weeks ago | -- | -- |

# Web App Front-end Testing with Cypress

RHEA GROUP for esa
AURORA TECHNOLOGY SSC

# Custom Commands

Cypress provides an API for **creating custom commands** or overwriting existing ones.

Following the instructions provided in cypress/support/commands.ts a new command can be created calling

```
Cypress.Commands.add()
Cypress.Commands.overwrite()
```

Custom commands are very useful to code a series of repetitive actions that will be repeated across a lot of tests. A login custom command is a good example: it avoids lots of repetitive actions and can be invoked from the API using `cy.login()`.

Custom commands can be declared following the instructions provided in the documentation.

# Custom Commands

Example of custom login and logout commands.

In this example:

- ✓ Extending the previous example: creating a custom command for session cache that can be used by any spec.
- ✓ Adding custom commands and extending the Cypress API: `cy.tapLogin()` and `cy.tapLogout()`
- ✓ Setting a breakpoint in the test code and interacting with the browser's development tools.

If you have any question, please use the following link or

QR code to submit it and we will try to answer it:

https://forms.office.com/e/sXW1NfK2Tt

**Hands-on Session Questions**

# Custom Commands

RHEA GROUP for esa
SSC AURORA TECHNOLOGY

isla-web - Google Chrome

isla-web

localhost:4200/__/#/specs

isla-web
PA_Workshop_Cypress_...

Specs
Runs
Debug
Settings

Specs

v13.2.0 Chrome 117 Docs Log in

Optimize and record your CI test runs with Cypress Cloud

When you configure Cypress to record tests to Cypress Cloud, you'll see data from your latest recorded runs in the Cypress app. This increased visibility into your test history allows you to debug your tests faster and more effectively, all within your local workflow.

Get started with Cypress Cloud

Search specs 7 matches + New spec

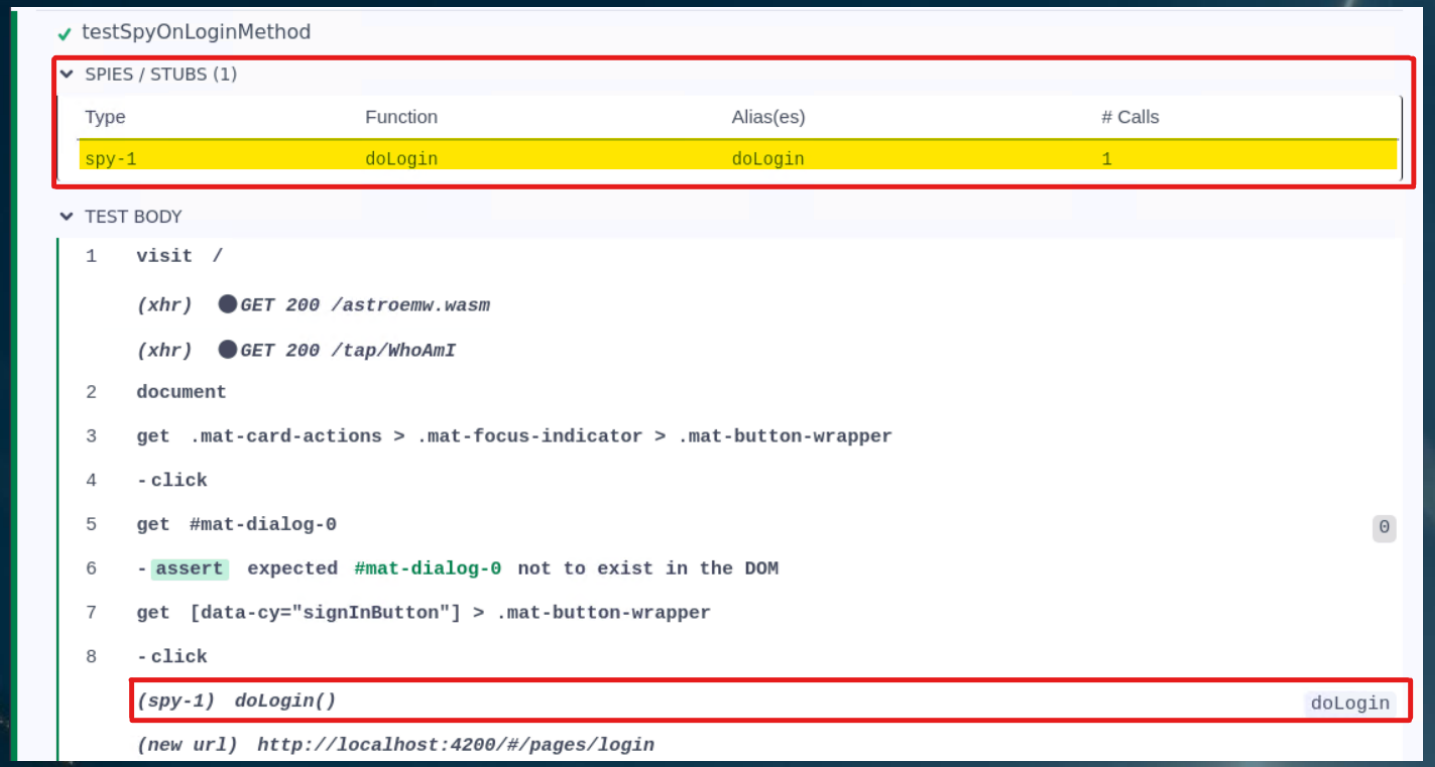
E2E specs Component specs Run 7 specs

| | Last updated | Latest runs | Average duration |
|---|---|---|---|
| cypress / e2e | | | |
| homePage.spec.cy.ts | 2 weeks ago | -- | -- |
| intercept.spec.cy.ts | 3 weeks ago | -- | -- |
| reusingSessionsInTests.spec.cy.ts | 9 weeks ago | -- | -- |
| search.spec.cy.ts | 3 weeks ago | -- | -- |
| spies.spec.cy.ts | 4 weeks ago | -- | -- |
| usingCustomLoginAndLogoutCommands.spec.cy.ts | 9 weeks ago | -- | -- |
| visualTesting.spec.cy.ts | 3 weeks ago | -- | -- |

# Web App Front-end Testing with Cypress

RHEA GROUP for esa
SSC AURORA TECHNOLOGY

# Spies and Stubs




**Spies** are used to **track the execution** of a determined code. cy.spy() can wrap a method to record calls and arguments passed to the function.

**Stubs** are used to **replace a method**, recording its usage and controlling its behaviour or returned value.

Asserts can be made on spies and stubs.

When tests are run, spies and stubs are displayed

in the command log.

# Spies and Stubs

Example of spy.

In this example:

- ✓ Obtaining a reference to an Angular component in the application
- ✓ Setting a spy on a method
- ✓ Asserting on a spy
- ✓ Observing the spy in the command log

If you have any question, please use the following link or QR code to submit it and we will try to answer it:

https://forms.office.com/e/sXW1NfK2Tt

**Hands-on Session Questions**

# Spies and Stubs

RHEA GROUP for esa
SSC AURORA TECHNOLOGY

isla-web - Google Chrome

isla-web

localhost:4200/__/#/specs

isla-web
PA_Workshop_Cypress_...

Specs
Runs
Debug
Settings

Specs

v13.2.0 Chrome 117 Docs Log in

Optimize and record your CI test runs with Cypress Cloud

When you configure Cypress to record tests to Cypress Cloud, you'll see data from your latest recorded runs in the Cypress app. This increased visibility into your test history allows you to debug your tests faster and more effectively, all within your local workflow.

Get started with Cypress Cloud

Search specs | 7 matches | + New spec

E2E specs | Component specs | Run 7 specs

| | Last updated | Latest runs | Average duration |
|---|---|---|---|
| cypress / e2e | | | |
| homePage.spec.cy.ts | 2 weeks ago | -- | -- |
| intercept.spec.cy.ts | 3 weeks ago | -- | -- |
| reusingSessionsInTests.spec.cy.ts | 9 weeks ago | -- | -- |
| search.spec.cy.ts | 3 weeks ago | -- | -- |
| spies.spec.cy.ts | 4 weeks ago | -- | -- |
| usingCustomLoginAndLogoutCommands.spec.cy.ts | 9 weeks ago | -- | -- |
| visualTesting.spec.cy.ts | 3 weeks ago | -- | -- |

# Web App Front-end Testing with Cypress

RHEA GROUP for esa
SSC AURORA TECHNOLOGY

# Visual Testing

Cypress is a functional test runner designed to validate an application functions as expected, but it **cannot see how the application is rendered**.

The approach of visual testing is using one of the visual testing plugins to compare an **image snapshot** of the application at a certain stage with a previously stored one. If there is little or no difference, assumption can be made that the application works properly.

Visual testing can be a very useful tool, but it must be considered that:

- The need for visual testing must be justified: a great number of assertions validating styles or data.
- Visual testing can lead to **flaky tests** if not designed properly. Image snapshots must be taken when the page is done changing.
- Comparing **individual elements** is preferred to comparing the whole page.

# Visual Testing

Example of visual testing.

In this example:

- ✓ Changing the screen resolution: Viewport
- ✓ Dynamically creating tests based on a predefined input
- ✓ Asserting on the application display: pre-recorded snapshots

If you have any question, please use the following link or QR code to submit it and we will try to answer it:

https://forms.office.com/e/sXW1NfK2Tt

**Hands-on Session Questions**

# Visual Testing

RHEA GROUP for esa

AURORA TECHNOLOGY SSC

isla-web – visualTesting.spec.cy.t

isla-web | PA_Workshop_Cypress_Demo_Day | Angular CLI Server

spies.spec.cy.ts | visualTesting.spec.cy.ts

Fernando Guerrero

```
describe( title: 'Visual testing examples', fn: () : void => {
  const resolutions: any = [
    { size: 'iphone-6' },
    { size: 'iphone-6', orientation: 'landscape' },
    [1280, 900],
  ];

  2 usages  Fernando Guerrero
  function composeTestLabel(
    resolution: Array<number> | { size: string; orientation?: string }
  ) : string  {
    let testLabel: string;
    if (Cypress._.isArray( value: resolution)) {
      testLabel = resolution[0].toString() + 'x' + resolution[1].toString();
    } else {
      if (resolution.orientation) {
        testLabel = resolution.size + ' ' + resolution.orientation;
      } else {
        testLabel = resolution.size;
      }
    }
    return testLabel;
  }

  2 usages  Fernando Guerrero
  function setViewportAndVisitPage(
    resolution: Array<number> | { size: string; orientation?: string }
  ): void {
    if (Cypress._.isArray( value: resolution)) {
      cy.viewport(resolution[0], resolution[1]);
    } else {
      if (resolution.orientation) {
        // @ts-ignore
        cy.viewport(resolution.size, resolution.orientation);
      } else {
        // @ts-ignore
        cy.viewport(resolution.size);
      }
    }
    cy.visit('/');
    cy.get('#mat-dialog-0');
    cy.get('.mat-card-header-text').should( chainer: 'contain.text',  value: 'Warning');
    cy.get("[data-cy='dialog-message']").should(
```

callback for describe() › callback for describe() › callback for resolutions.forEach() › callback for it()

isla-web > cypress > e2e > visualTesting.spec.cy.ts | TypeScript 4.8.4 | 55:25 (11 chars) | LF | UTF-8 | 2 spaces | 907 of 2048M

# Web App Front-end Testing with Cypress

RHEA GROUP for esa
SSC AURORA TECHNOLOGY

# Intercepting Network Requests

Cypress provides **access to HTTP requests** made during the tests via the `cy.intercept()` command. It is possible to **stub or mock responses**, make **assertions**, and simulating network delays.

The most common way for stubbing responses is using **fixtures**: fixed sets of data that are returned after a certain request is made without getting to the server. The best strategy is usually a combination of true end-to end tests and stubbed ones considering on the pros and cons.

### Stubbing

- Good for testing edge cases 👍
- Simulate network conditions 👍
- Faster 👍
- Use for the majority of test cases 👍

- Less test coverage on server 👎
- Stubbed response may differ from the server one 👎

### Full end-to-end

- Likely to work in production 👍
- Test coverage around server endpoints 👍
- Use sparingly: for testing critical paths 👍

- Requires seeding and keeping data unchanged 👎
- Much slower 👎

# Intercepting Network Requests

Example of network request interception.

In this example:

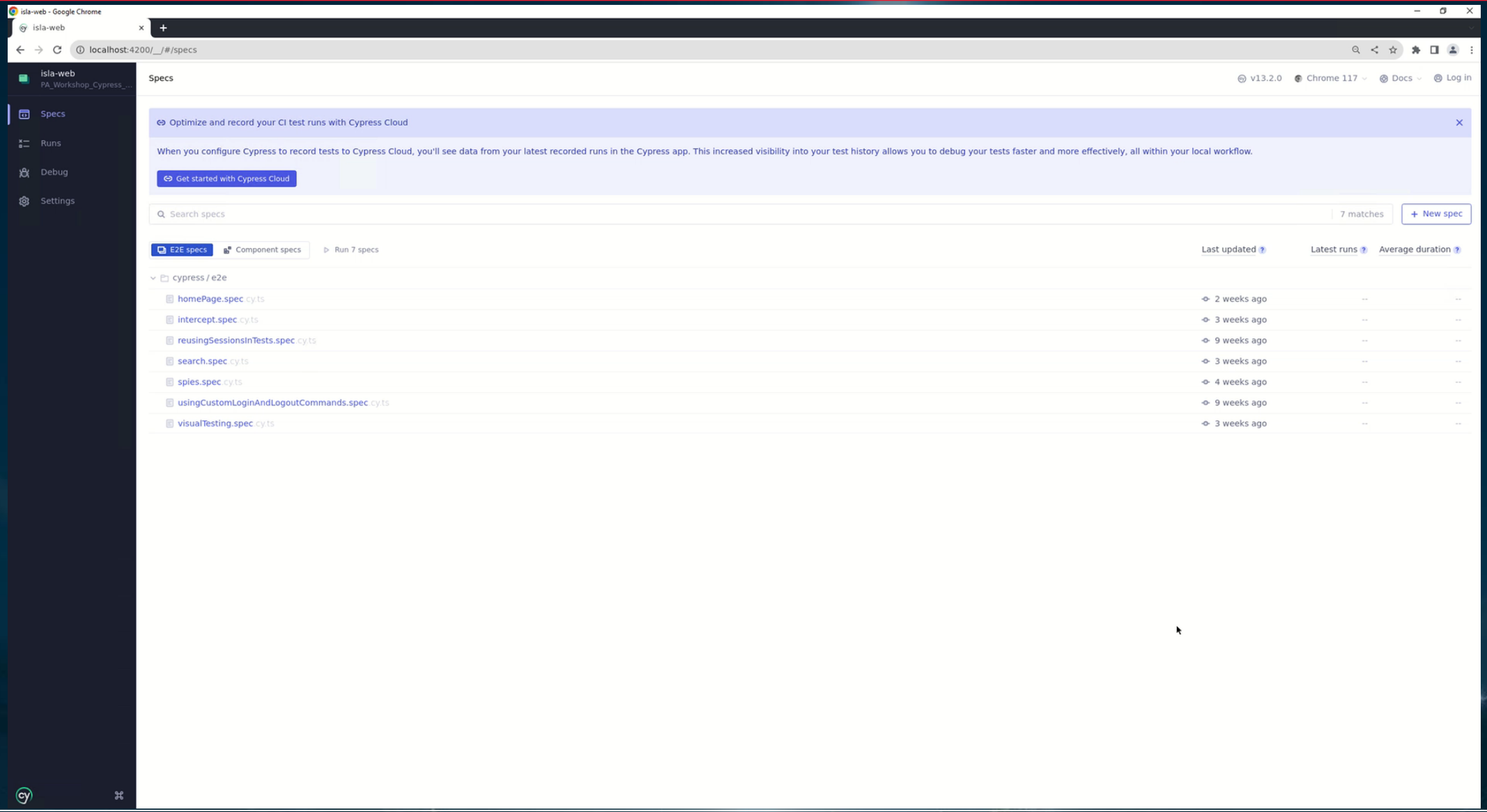
- ✓ Full flaky e2e test combining all features discussed previously
- ✓ Converting e2e to front-end testing using fixtures
- ✓ Observing interceptions, and stubbed responses in the command log
- ✓ Returning HTTP codes for simulating server error

If you have any question, please use the following link or QR code to submit it and we will try to answer it:

https://forms.office.com/e/sXW1NfK2Tt

**Hands-on Session Questions**

# Intercepting Network Requests



# Web App Front-end Testing with Cypress

RHEA GROUP for esa
SSC AURORA TECHNOLOGY

# Code Coverage

Determining code coverage is essential to assess what our tests are really testing using it as a **guide** for identifying where test effort is needed. Page https://docs.cypress.io/guides/tooling/code-coverage documents set up code coverage in Cypress.

Several coverage report formats can be configured to be either human-readable output or information for third party services such as CI/CD tools.

```
     /**
      * Checks whether the user session has expired, if so an error message is showed and the logout is
      * executed to clean the HttpOnly cookie JSESSIONID.
      * @param modifiedReq
      * @param status
      */
     handleUserSessionTimeoutIfProceeds(
       modifiedReq: HttpRequest<any>,
       status: number
     ): void {
5x     if (this.isUserSessionTimeoutError(modifiedReq.url, status)) {
3x       const dialogRef = this.matDialog.open(InfoDialogComponent, {
           data: this.UNAUTHORISED_SER_MSG,
         });

3x       dialogRef.afterClosed().subscribe((confirm) => {
           window.location.href = '';
         });
       }
     }

```

```
TN:
SF:src/main.ts
FN:13,(anonymous_0
FNF:1
FNH:0
FNDA:0,(anonymous_
DA:7,35
DA:8,0
DA:11,35
DA:13,0
LF:4
LH:2
BRDA:7,0,0,0
BRDA:7,0,1,35
BRF:2
BRH:1
end_of_record
TN:
SF:src/polyfills.t
FNF:0
FNH:0
DA:54,35
LF:1
LH:1
BRF:0
BRH:0
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| src/app/pages/portals/header | 3.57% | 1/28 | 0% | 0/6 | 0% | 0/6 | 6.25% | 1/16 |
| | | | | | | | | 3/5 |
| | | | | | | | | 38/46 |
| | | | | | | | | 2/2 |
| | | | | | | | | 210/1457 |
| | | | | | | | | 24/39 |
| | | | | | | | | 6/9 |
| | | | | | | | | 25/35 |
| | | | | | | | | 34/34 |
| | | | | | | | | 20/20 |
| | | | | | | | | 48/96 |
| | | | | | | | | 48/51 |
| | | | | | | | | 1/16 |
| | | | | | | | | 1/13 |
| | | | | | | | | 16/22 |
| | | | | | | | | 1/13 |
| | | | | | | | | 1/53 |
| | | | | | | | | 1/16 |

# Code Coverage

Example code coverage report.

In this example:

- ✓ E2e code coverage report
- ✓ Use the html report to assess actual code covered by tests and design new ones for untested code.

If you have any question, please use the following link or QR code to submit it and we will try to answer it:

https://forms.office.com/e/sXW1NfK2Tt

**Hands-on Session Questions**

# Code Coverage

RHEA GROUP for esa
SSC AURORA TECHNOLOGY

isla-web – index.htm

isla-web | PA_Workshop_Cypress | e2e

Pause | 00:00:00 | Select Area | Audio | Record Pointer

Project

- isla-web [isla-app-web] ~/develop-local/astro-obs-arch
  - .angular
  - .husky
  - .idea
  - .nyc_output
  - __image_snapshots__
  - coverage
  - coverage-e2e
    - lcov-report
      - src
      - base.css
      - block-navigation.js
      - favicon.png
      - index.html
      - prettify.css
      - prettify.js
      - sort-arrow-sprite.png
      - sorter.js
    - clover.xml
    - coverage-final.json
    - coverage-summary.json

index.html

~/develop-local/astro-obs-archives/isla-web/coverage-e2e/lcov-report/index.html

```
<!doctype html>
<html lang="en">

<head>
    <title>Code coverage report for All files</title>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="prettify.css" />
    <link rel="stylesheet" href="base.css" />
    <link rel="shortcut icon" type="image/x-icon" href="favicon.png" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style type='text/css'>
        .coverage-summary .sorter {
            background-image: url(sort-arrow-sprite.png);
        }
    </style>
</head>
```

html > body > div.wrapper > div.status-line.low

Run | e2e | Angular CLI Server

```
GET /NotesESAbold.woff 304 13.486 ms - -
GET /NotesESAregular.woff 304 19.679 ms - -
GET /fontawesome-webfont.woff2?v=4.7.0 304 20.722 ms - -
```

isla-web > coverage-e2e > lcov-report > index.html | TypeScript 4.8.4 | 64:23 (6 chars) | LF | UTF-8 | 2 spaces | 658 of 2048M

# Web App Front-end Testing with Cypress

RHEA GROUP for esa
SSC AURORA TECHNOLOGY

# Screenshots and Videos

Specs ✓ 3 ✗ -- ○ --

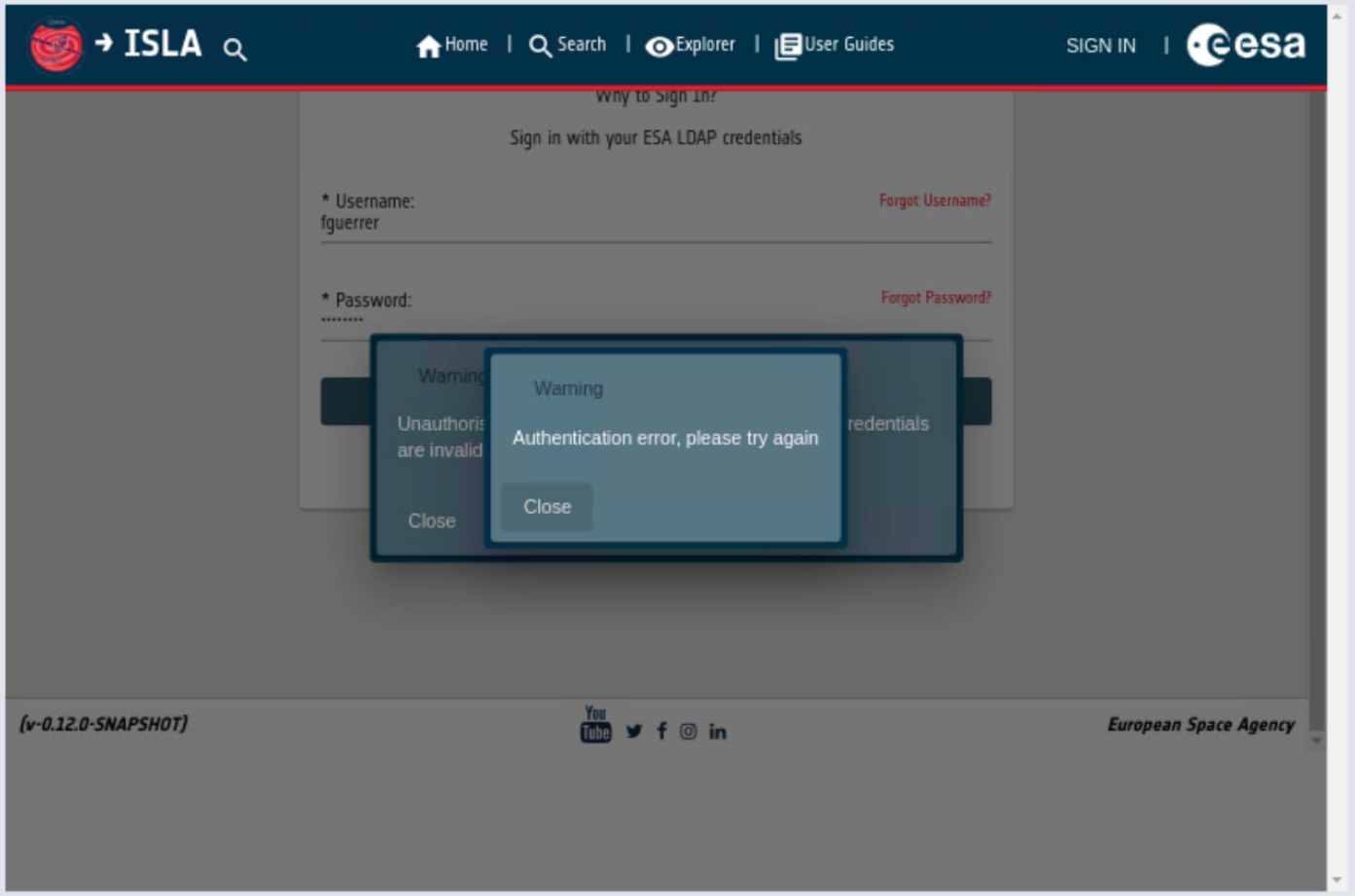
homePage.spec.cy.ts 00:15

```
13  - assert expected 401 to equal **401**
14  get .ng-star-inserted > .mat-card
15  - assert expected [ <mat-card.mat-card.mat-focus-indicator.dialog-container>, 1 more... ] to be visible
16  get [data-cy="dialog-message"] > p
17  - assert expected [ <p>, 1 more... ] to be visible
18  - assert expected [ <p>, 1 more... ] to have text Authentication error, please try again, but the text was Unauthorised: either your session has expired or the credentials are invalid, please log in againAuthentication error, please try again
```

And the user opens a session

Then the system will grant access

http://localhost:4300/#/pages/login Chrome 117 1000x660 (76%)



# Screenshots and Videos

Example of screenshot and video generation for a test suite.

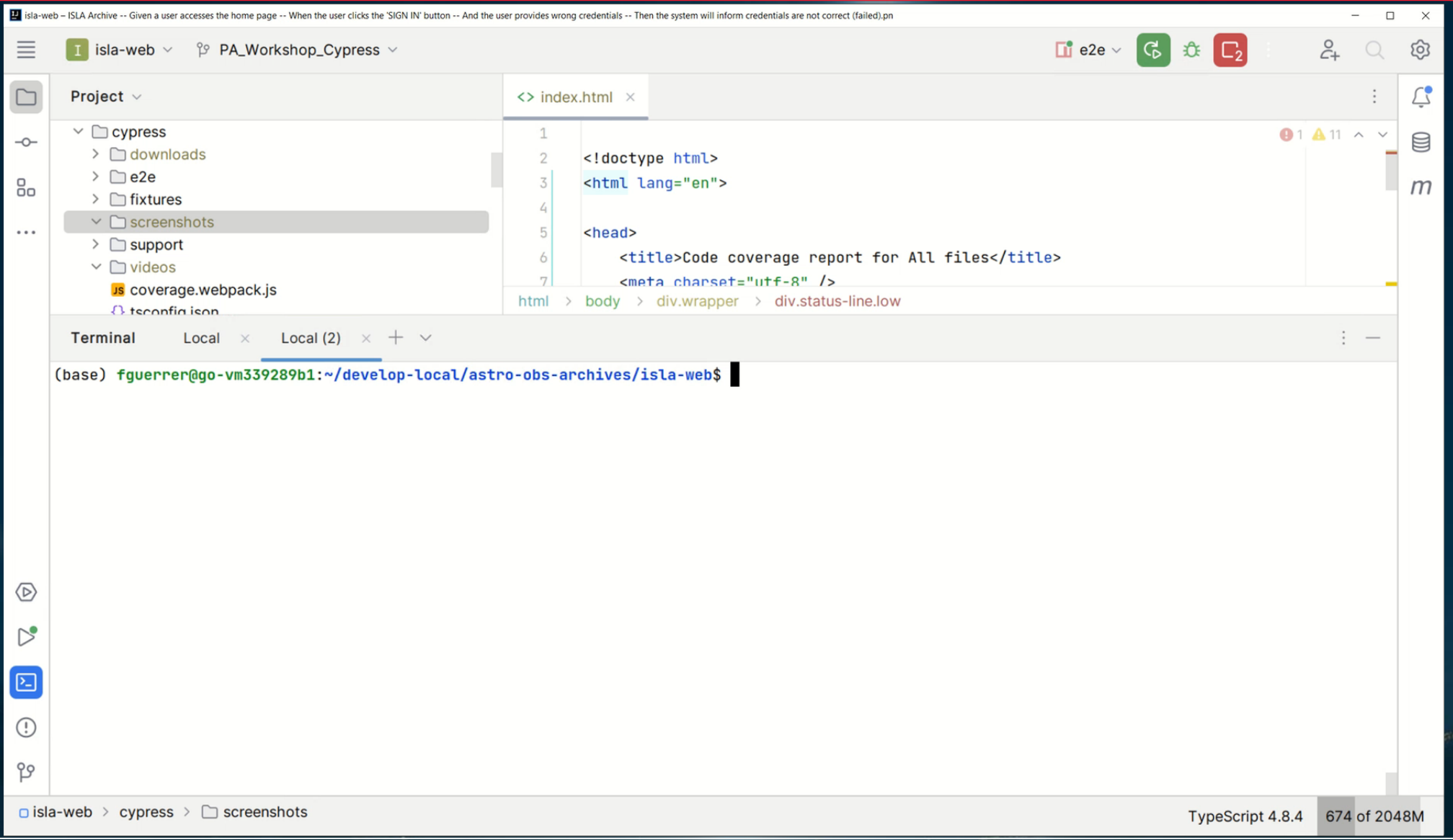
In this example:

- ✓ Running a spec in headless mode
- ✓ Review the screenshot and video generated when a test fails

If you have any question, please use the following link or QR code to submit it and we will try to answer it:
https://forms.office.com/e/sXW1NfK2Tt

**Hands-on Session Questions**

# Screenshots and Videos



# Web App Front-end Testing with Cypress

RHEA GROUP for esa
SSC AURORA TECHNOLOGY

# Recording Tests with Cypress Studio

Cypress Studio is a **Beta** feature that allows generating new code from the Cypress Application user interface.

Cypress Studio can be used to:

- **Extend existing tests**
- **Create new tests**

Studio Beta — Learn more

Generate and save commands directly to your test suite by interacting with your app as an end user would. Right click on an element to add an assertion. Studio will track events that generate the following commands:

```
.check()
.click()
.select()
.type()
.uncheck()
```

This feature is currently experimental and we will be adding more commands and abilities in the future. Your feedback will be highly influential to our team.

```
/* ==== Test Created with Cypress Studio ==== */
it( title: 'Cypress Studio Example', fn: function() : void {
  /* ==== Generated with Cypress Studio ==== */
  cy.get('.mat-card-actions > .mat-focus-indicator > .mat-button-wrapper').click();
  cy.get('.mission-label').should( chainer: 'have.text', value: '→NBSPISLA');
  cy.get('.mat-toolbar-row > :nth-child(1) > .mat-focus-indicator > .mat-button-wrapper >
  .mat-icon').should( chainer: 'be.visible');
  cy.get('[data-cy="signInButton"] > .mat-button-wrapper').should( chainer: 'have.class',
  value: 'mat-button-wrapper');
  /* ==== End Cypress Studio ==== */
});
```

# Recording Tests with Cypress Studio

Example of test recording.

In this example:

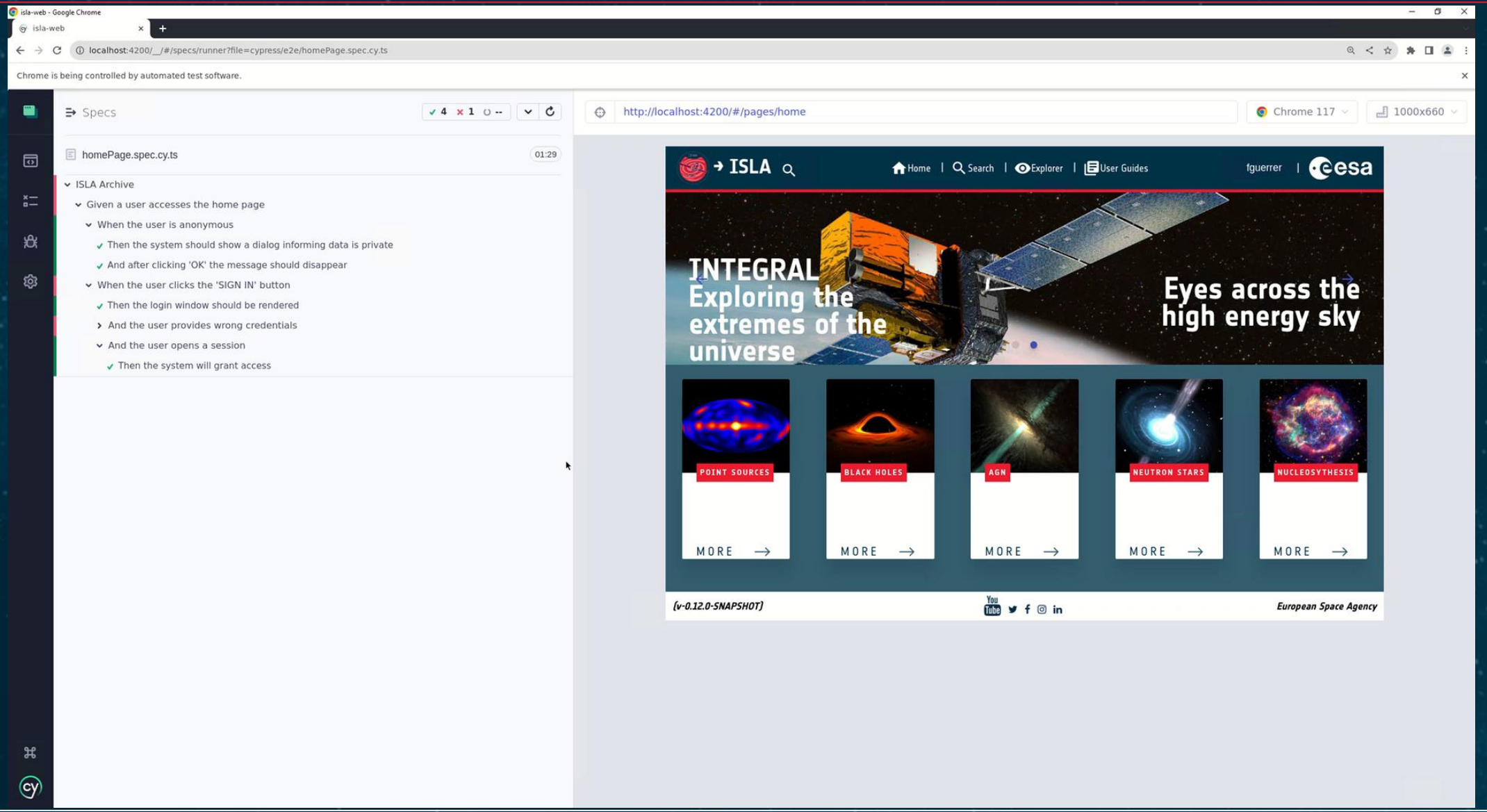
- ✓ Calling Cypress Studio
- ✓ Creating new test code recording user interaction with the Cypress Application

If you have any question, please use the following link or QR code to submit it and we will try to answer it:

https://forms.office.com/e/sXW1NfK2Tt

**Hands-on Session Questions**

# Recording Tests with Cypress Studio



# Web App Front-end Testing with Cypress

RHEA GROUP for esa
AURORA TECHNOLOGY SSC

# Acknowledgements

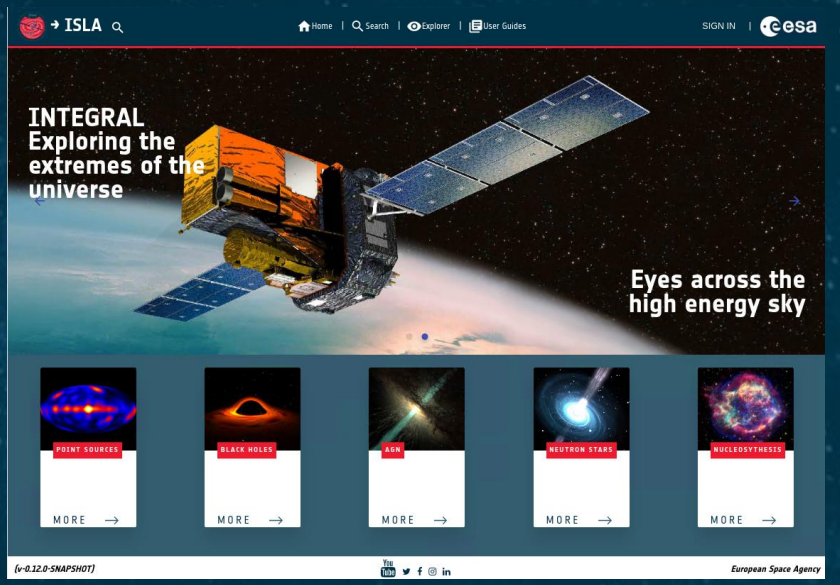
This session has been possible thanks to the cooperation of

The ESDC Team

ESAC SCIENCE DATA CENTRE

And particularly the Integral Science Legacy Archive

integral



# Thank you for your attention