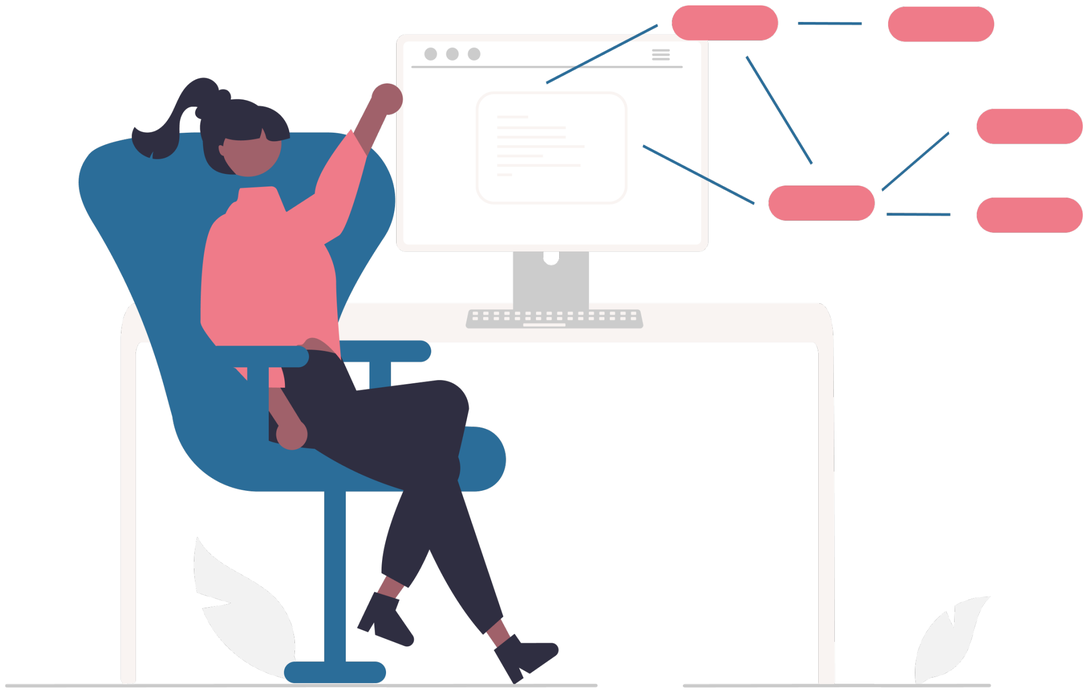


Cypress in One Day

End-to-End Testing with Cypress

Brian Love and **Mike Ryan**

Co-Founders of LiveLoveApp

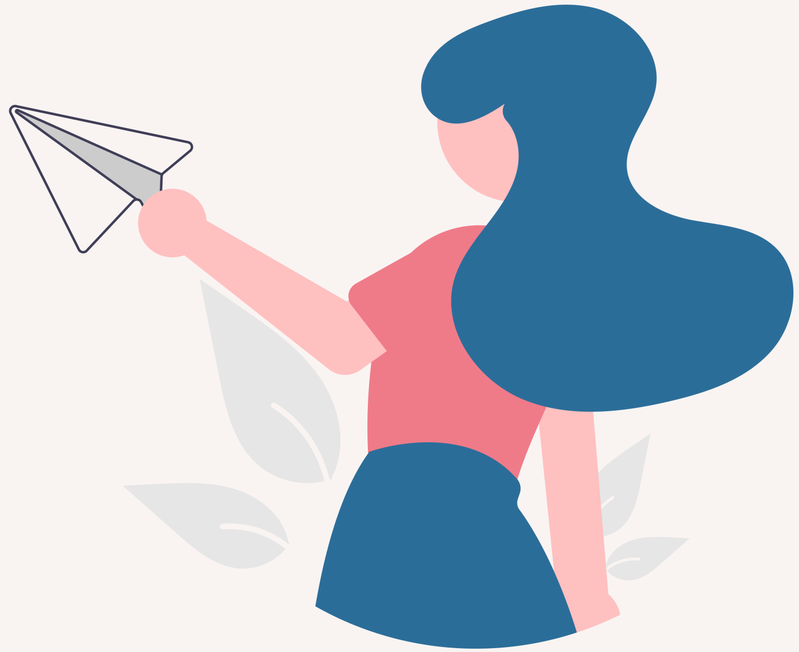
Google Developer Experts

Cypress Ambassadors

Open Source Contributors

liveloveapp

Find Absolute Joy in Shipping Apps



Cypress provides fast, easy, and reliable browser-based testing

Cypress Test Runner is free and open source


cypress.io

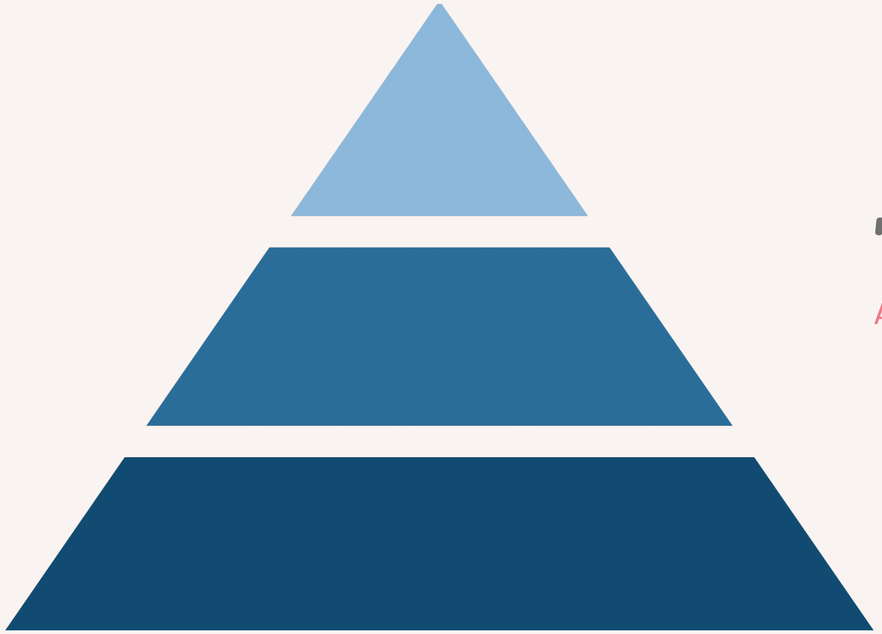
Why Cypress?

What you'll learn

- What Cypress is and why you should use it
- Our mission, and what we believe in
- Key Cypress features
- Types of tests Cypress is designed for

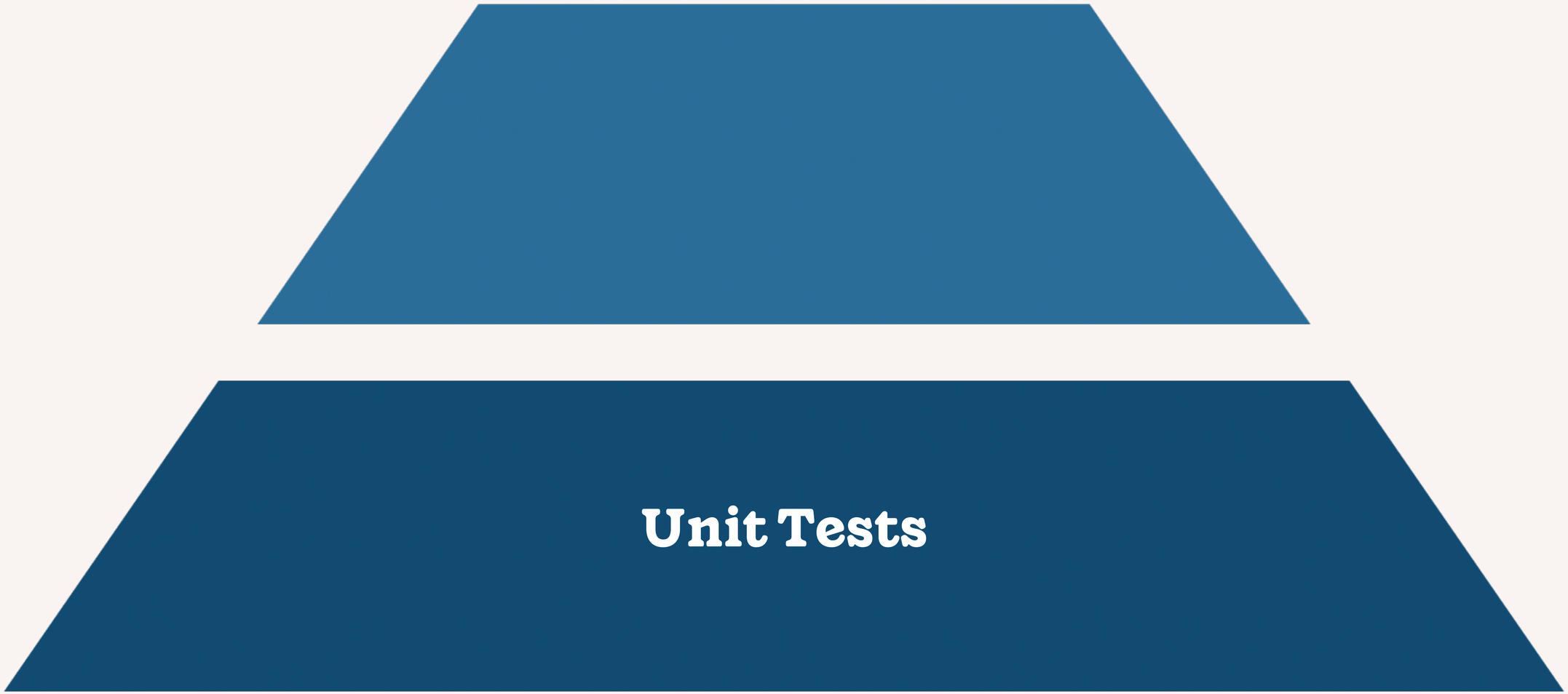
In a nutshell

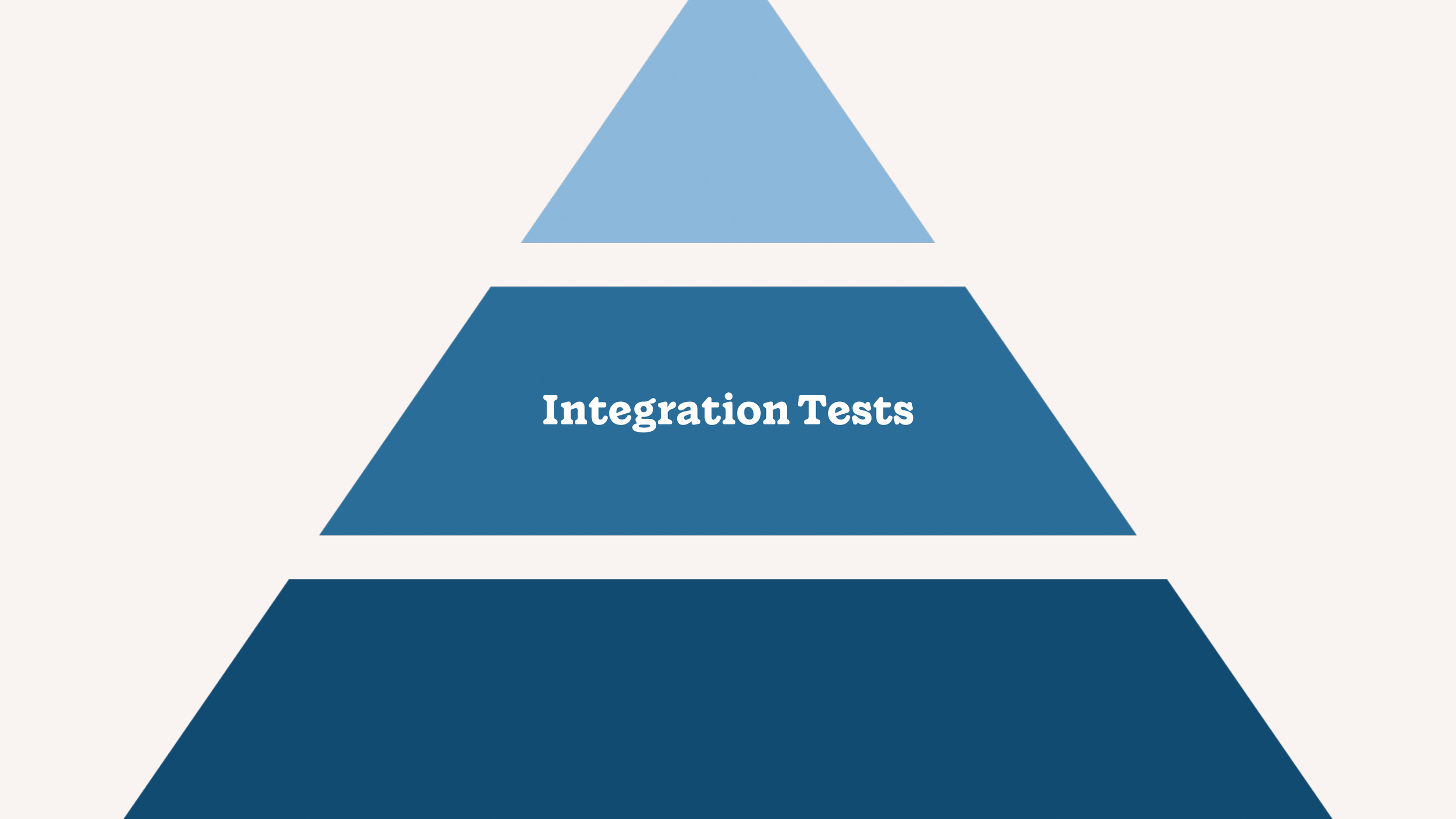
The Theory of Tests

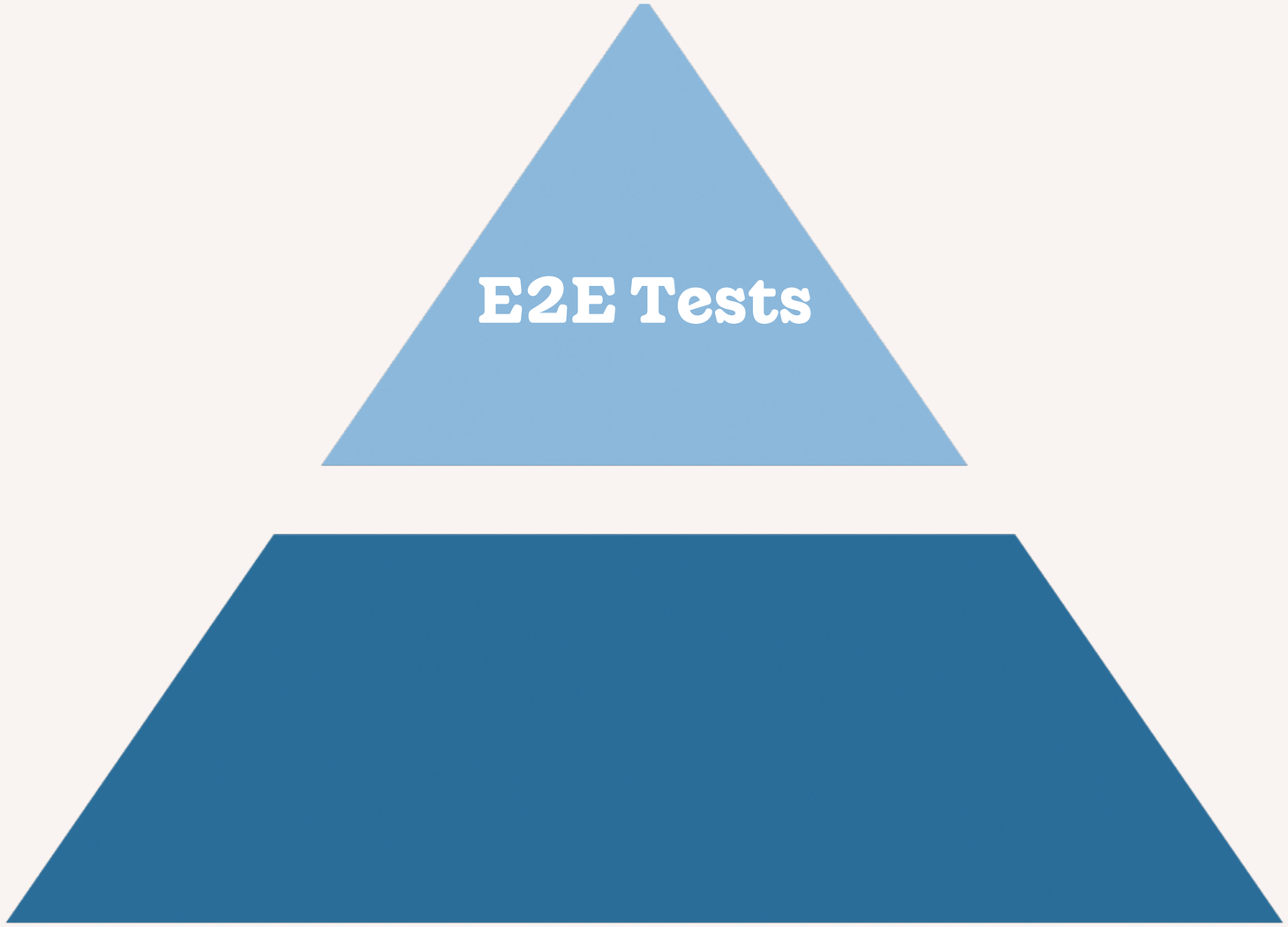


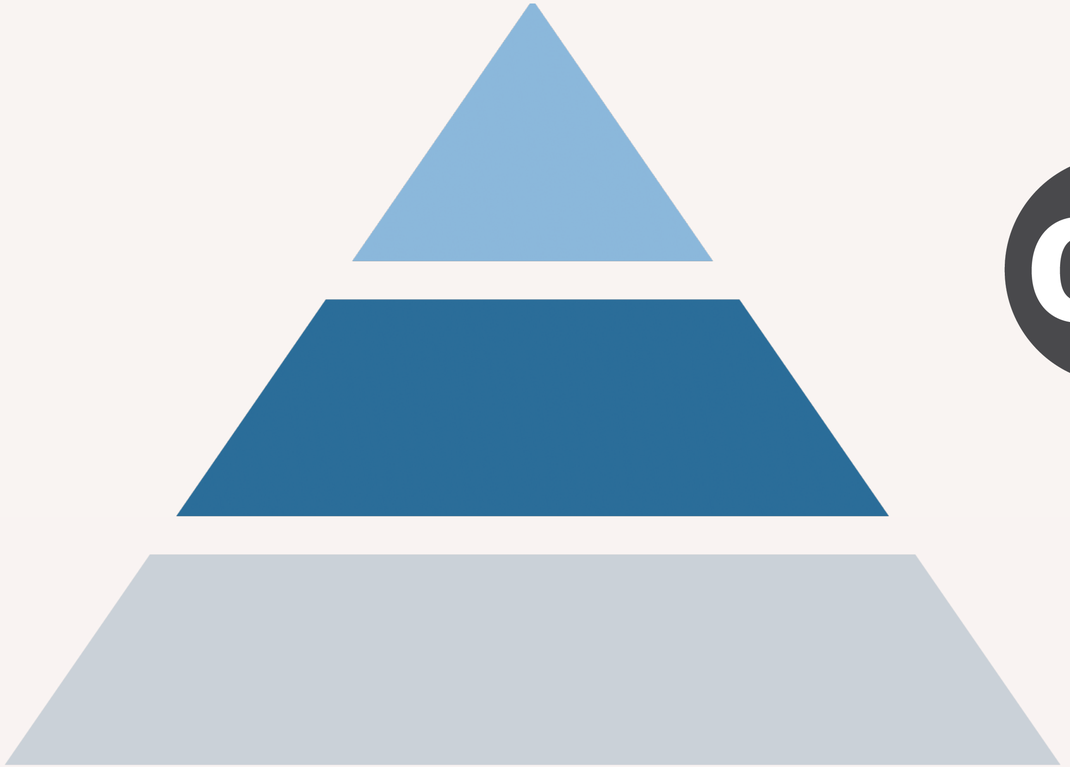
The Testing Pyramid

A Practical Guide for Investing in Tests


Unit Tests


Integration Tests


E2E Tests

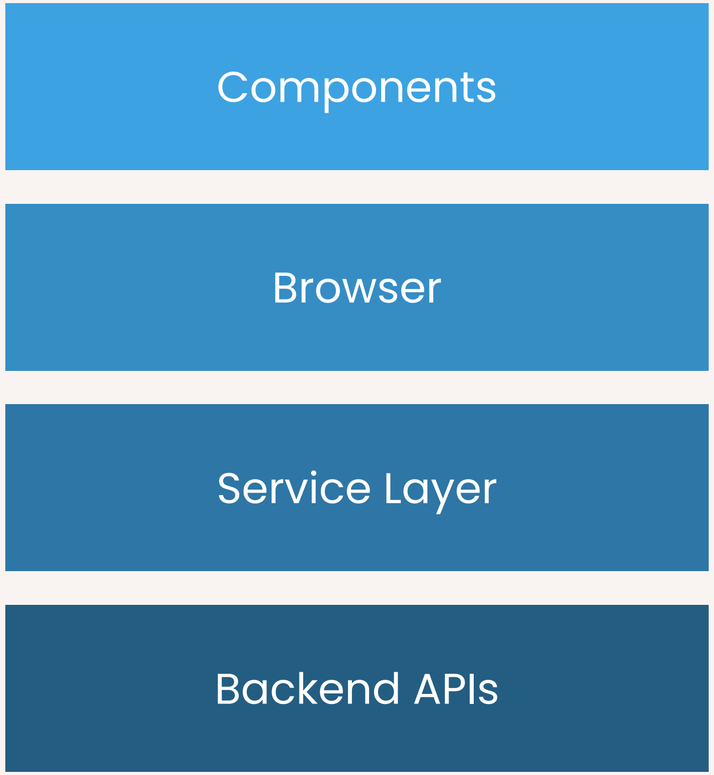


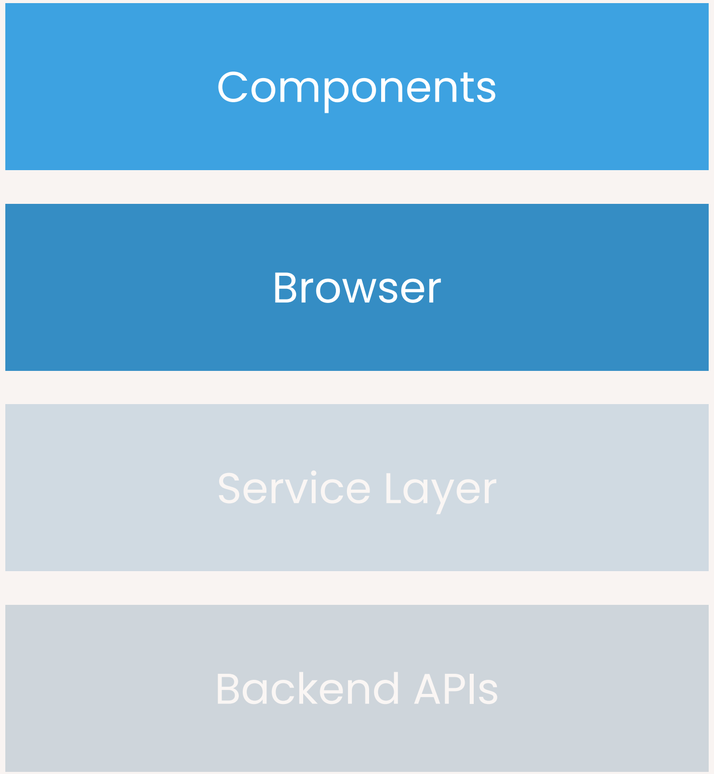
cypress.io

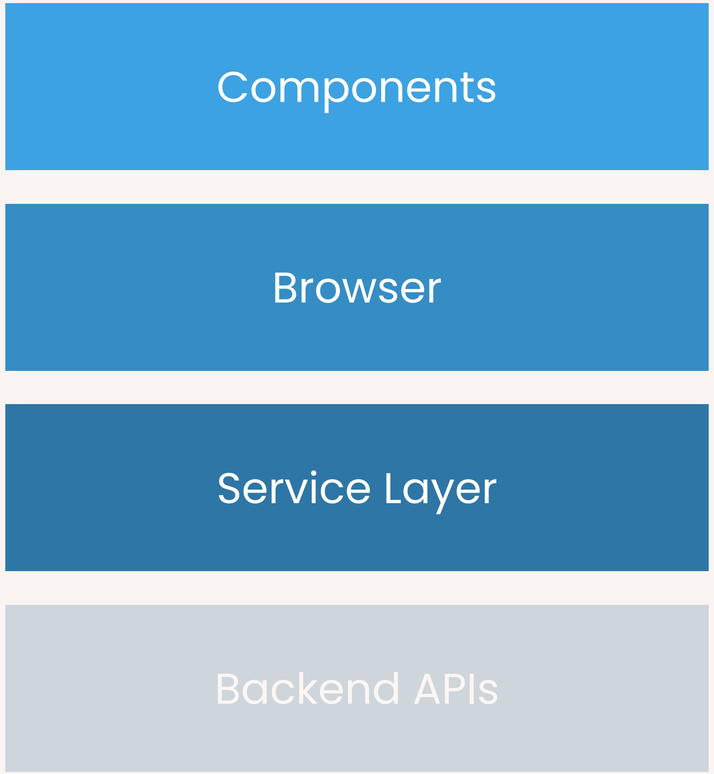
In this workshop, we will be using Cypress to write E2E tests for a Books Earnings application

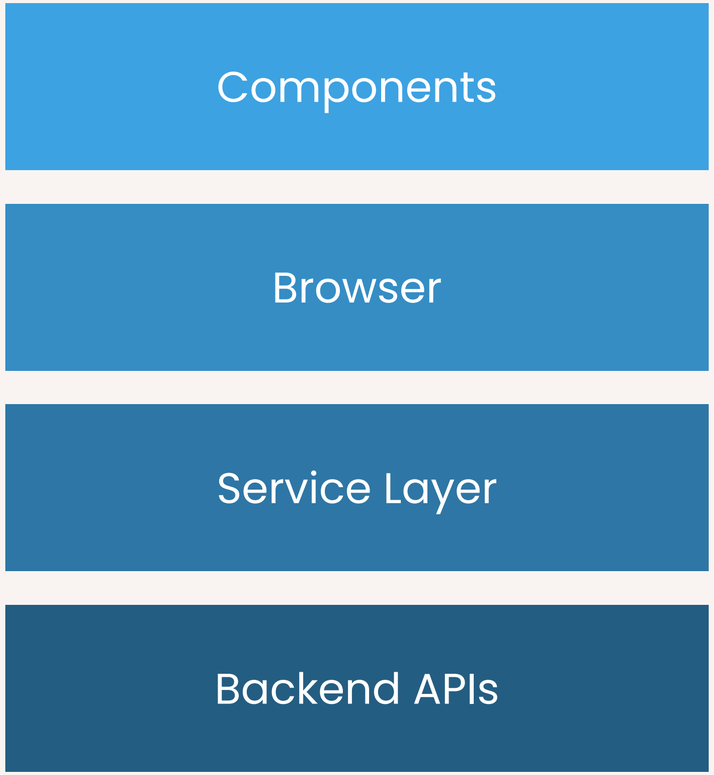
Demo

Write tests that exercise the critical paths of your application


Components
Browser
Service Layer
Backend APIs


Components
Browser
Service Layer
Backend APIs


Components
Browser
Service Layer
Backend APIs


Components
Browser
Service Layer
Backend APIs

Write a few E2E tests, mostly critical path, that test integration between components, the browser, your services, and backend APIs

The best E2E test is a test that is reliable, readable, and fast

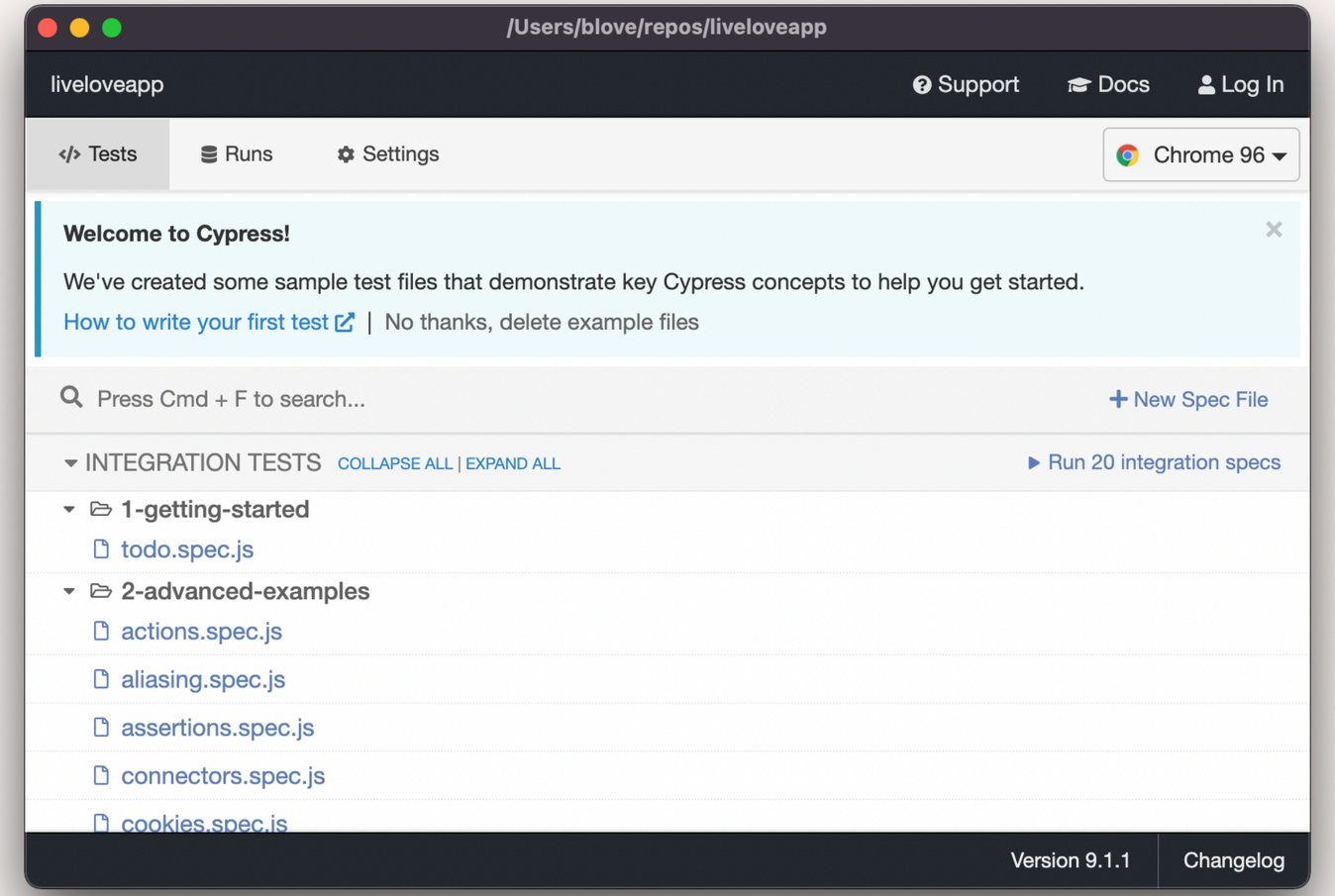
Getting Started with Cypress

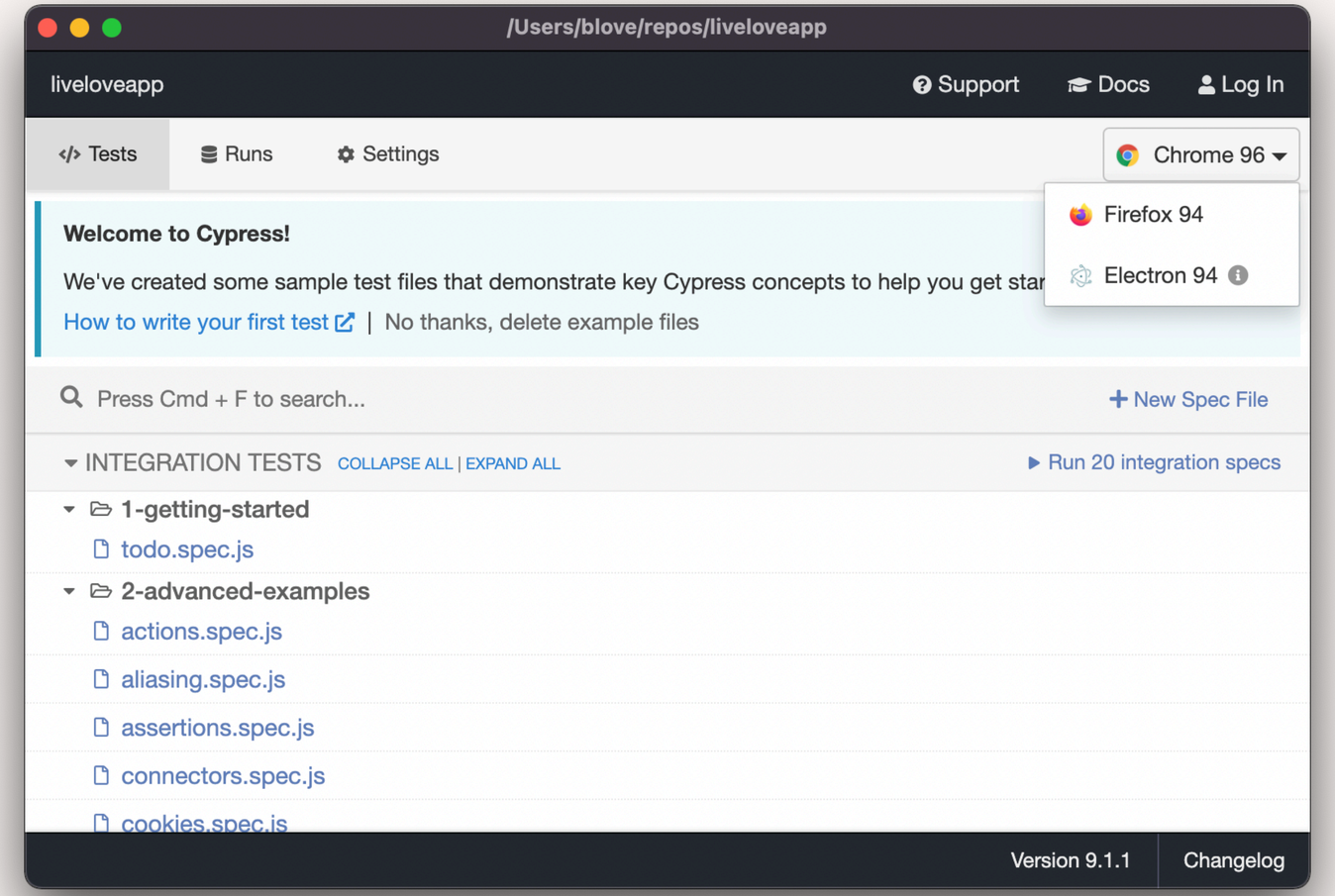
Getting Started with Cypress

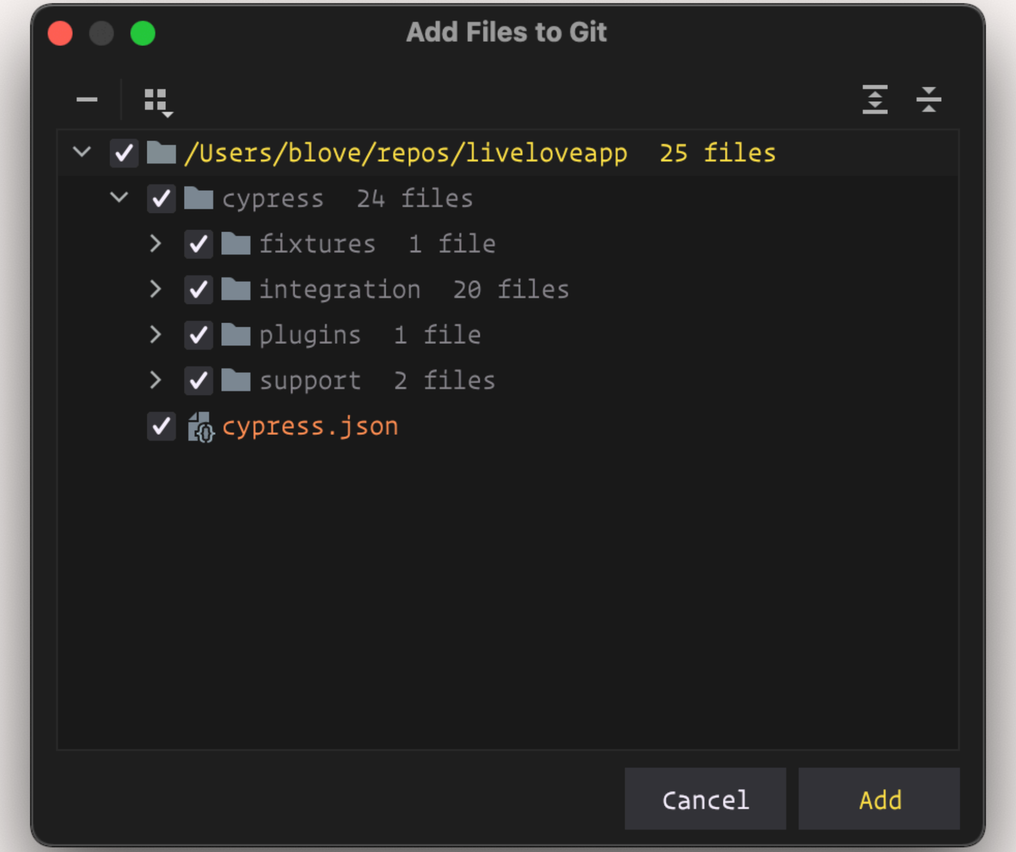
- [x] Installing Cypress
- [x] Launching the test runner
- [x] Using `cy.visit()`
- [x] Writing your first test

```
➜ npm install cypress --save-dev
```

```
➜ npx cypress open
```


/Users/blove/repos/liveloveapp
liveloveapp
Support
Docs
Log In
Tests
Runs
Settings
Chrome 96
Welcome to Cypress!
We've created some sample test files that demonstrate key Cypress concepts to help you get started.
How to write your first test
No thanks, delete example files
Press Cmd + F to search...
New Spec File
INTEGRATION TESTS
COLLAPSE ALL | EXPAND ALL
Run 20 integration specs
1-getting-started
todo.spec.js
2-advanced-examples
actions.spec.js
aliasing.spec.js
assertions.spec.js
connectors.spec.js
cookies.spec.js
Version 9.1.1
Changelog


/Users/blove/repos/liveloveapp
liveloveapp
Support
Docs
Log In
Tests
Runs
Settings
Chrome 96
Firefox 94
Electron 94
Welcome to Cypress!
We've created some sample test files that demonstrate key Cypress concepts to help you get star
How to write your first test
No thanks, delete example files
Press Cmd + F to search...
New Spec File
INTEGRATION TESTS
COLLAPSE ALL | EXPAND ALL
Run 20 integration specs
1-getting-started
todo.spec.js
2-advanced-examples
actions.spec.js
aliasing.spec.js
assertions.spec.js
connectors.spec.js
cookies.spec.js
Version 9.1.1
Changelog


Add Files to Git
/Users/blove/repos/liveloveapp 25 files
cypress 24 files
fixtures 1 file
integration 20 files
plugins 1 file
support 2 files
cypress.json
Cancel
Add

Scaffolding

- [x] `/fixtures`
- [x] `/integrations`
- [x] `/plugins`
- [x] `/support`

Fixtures

Scaffolding

- Fixed data
- We can use this to stub network request
- Ensure reliability of testing environment

Integrations

Scaffolding

- Test files
- By default we can use JS and JSX
- We can (and will) use TypeScript for our tests

Plugins

Scaffolding

- Extend the functionality of Cypress
- Tap into the Node process
- Wide array of plugins available in the ecosystem
- Plugins are curated by the Cypress team at: https://docs.cypress.io/plugins/director

Support

Scaffolding

- Executed prior to each test
- Executed once prior to all tests when using the “Run integration specs together” in UI
- Custom commands
- Global overrides

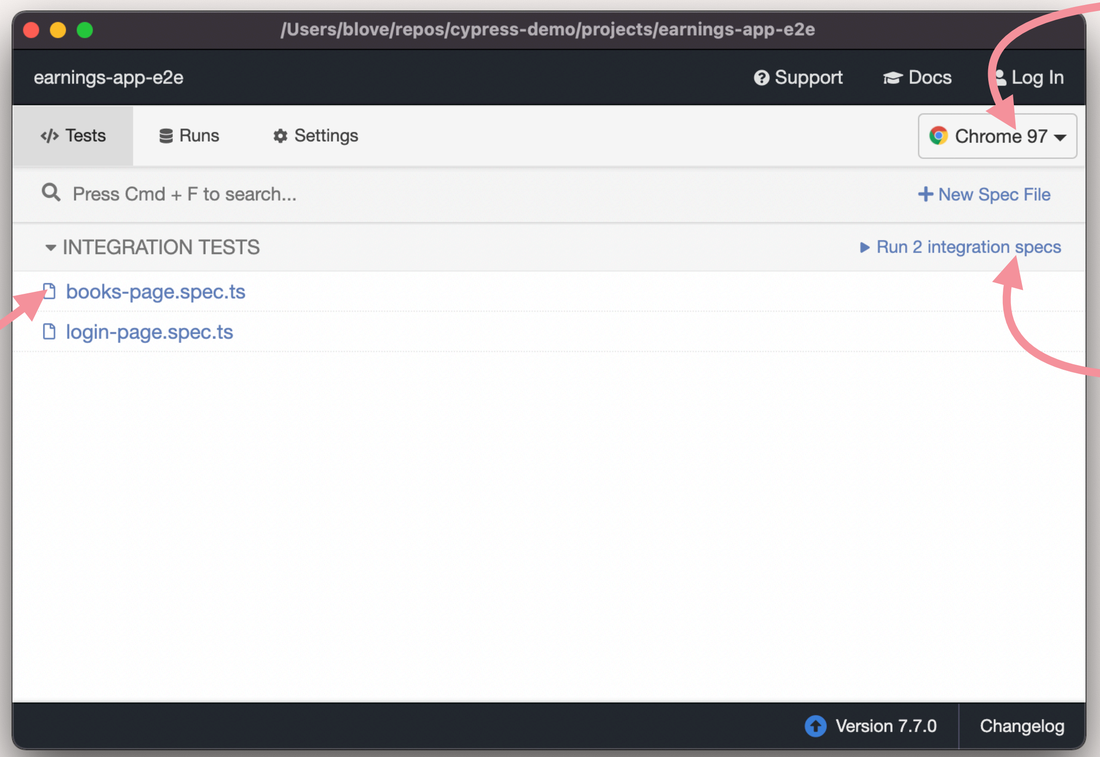
```
{
  "compilerOptions": {
    "target": "es5",
    "lib": ["es5", "dom"],
    "types": ["cypress"]
  },
  "include": ["**/*.ts"]
}
```

Running Cypress tests

1. Clone the repository: https://github.com/liveloveapp/cypress-demo
2. Follow the instructions in the README file
3. Checkout the complete branch
4. Using your terminal run **`npm run backend`**
5. Open another terminal and run **`npm run test`**
6. Verify that all of the tests pass

```
➜ npm run test --watch
```

Choose test environment



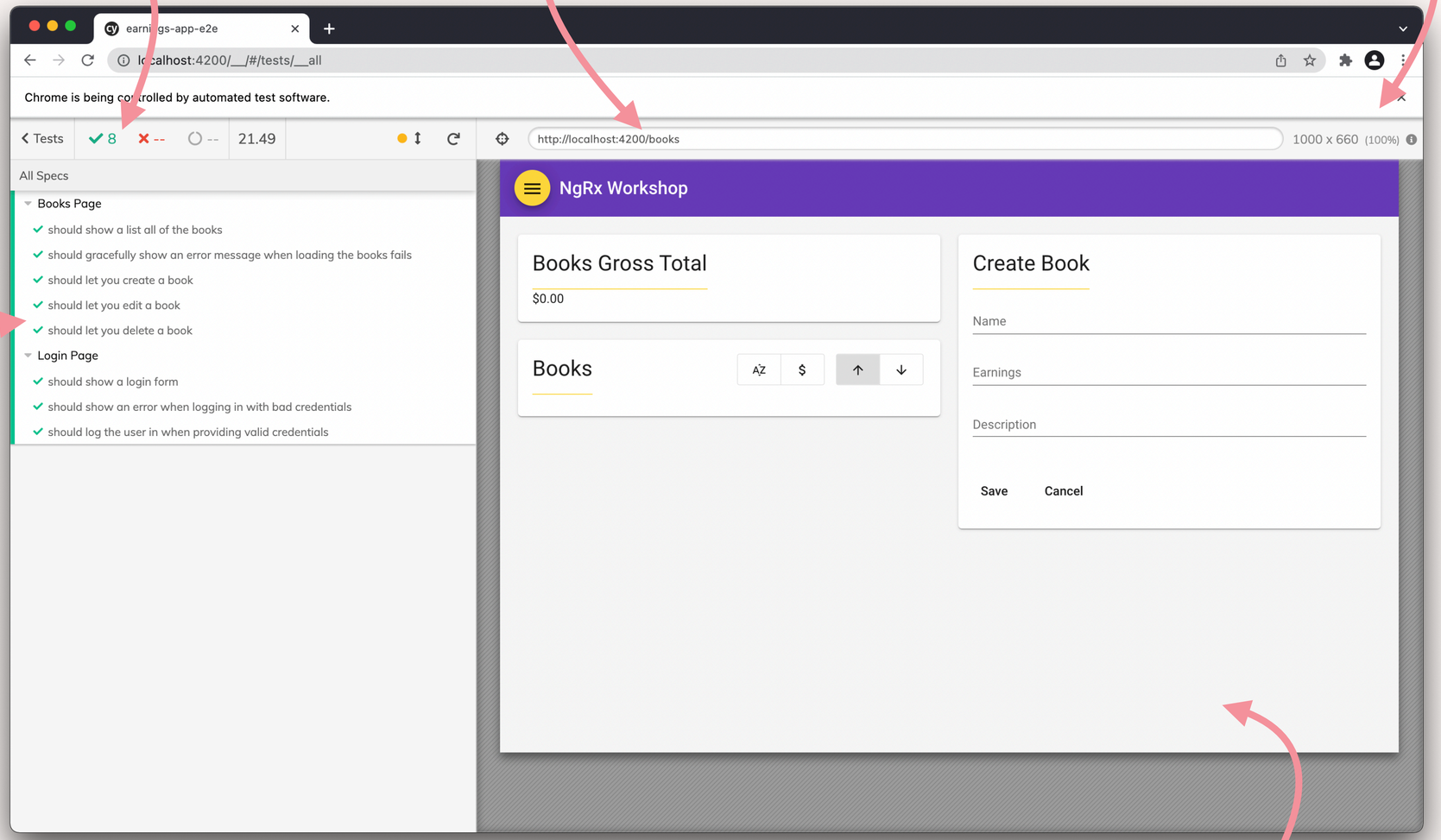
Run a specific test

Run all tests

Test status menu

URL preview

Viewport sizing

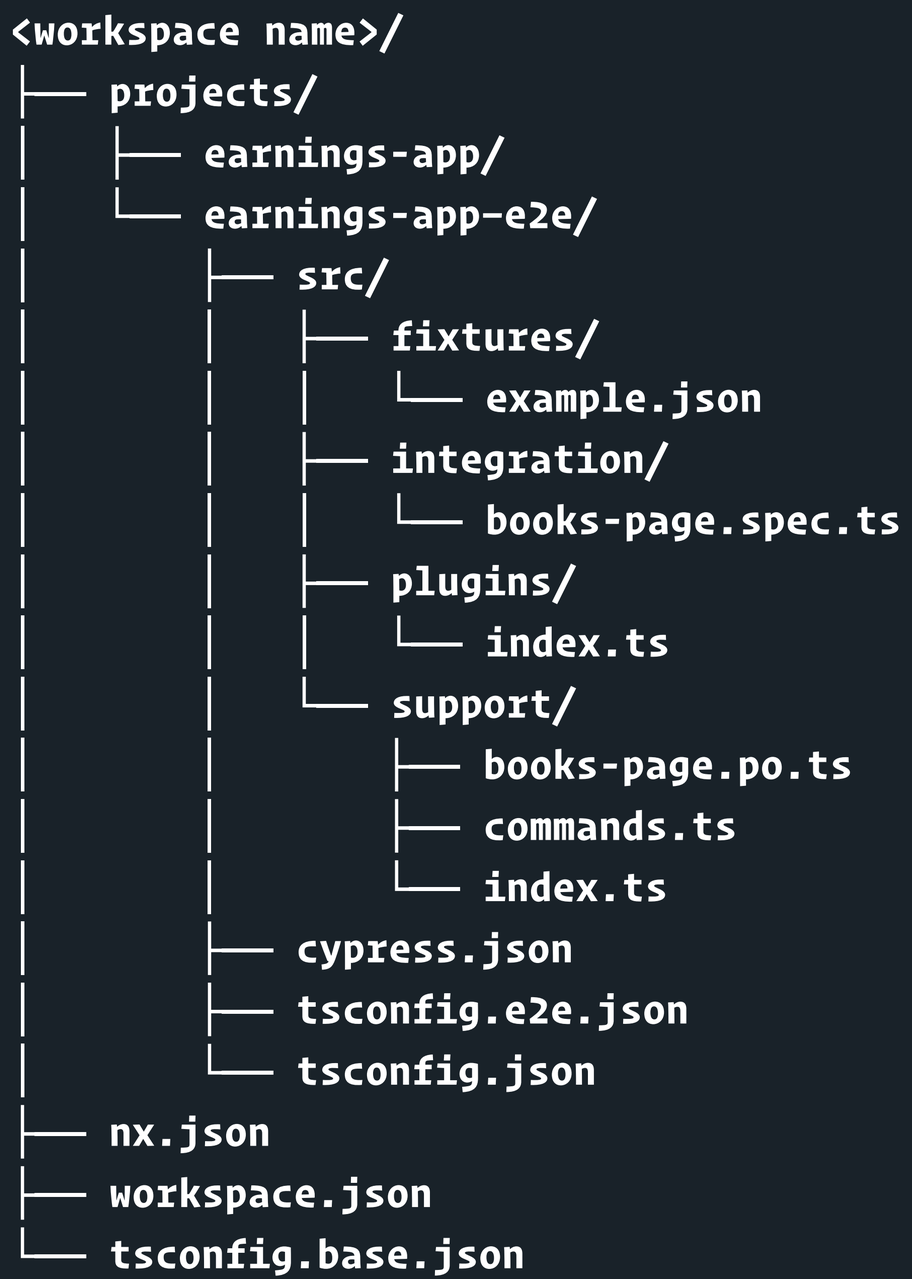


Command log

App preview

Running Cypress tests with watch mode

1. Ensure the backend is still running in another terminal using **`npm run backend`**
2. Open terminal and run **`npm run test --watch`**
3. Run all integration tests
4. Open Chrome's DevTools console
5. Select the "should show a list of all books" test and choose the **`get`** command in the command log
6. Note the information in the console log


<workspace name>/
projects/
earnings-app/
earnings-app-e2e/
src/
fixtures/
example.json
integration/
books-page.spec.ts
plugins/
index.ts
support/
books-page.po.ts
commands.ts
index.ts
cypress.json
tsconfig.e2e.json
tsconfig.json
nx.json
workspace.json
tsconfig.base.json

`cy.visit()`

- It is a best practice to set the **`baseUrl`** property in the Cypress configuration file
- With Nx, the baseUrl property is configured via the **`devServerTarget`** configuration for the project
- The **`cy.visit()`** command visits a remote URL

`cy.get()`

- Get one or more elements by a selector (or alias)
- Uses jQuery's **`$()`** selector syntax

```
describe('Login Page', () => {
  it('should display the login page', () => {
    // arrange
    setup();

    // act
    cy.visit('/');

    // assert
    cy.get('h3').contains('Login');
  });
});
```

Assertions

- Cypress bundles the Chai assertion library
- Assertions are retried (we'll talk more about this throughout the day)
- Assertions describe the desired state of the application

Writing your first test

`01-first-test`

1. Check out the **`challenge`** branch
2. Open **`/projects/earnings-app-e2e/src/integrations/login-page.spec.ts`**
3. Create a new test to assert the login page is displayed and that the heading contains the text “Login”
4. First, visit the application login page using **`cy.visit(‘/‘)`** command
5. Then, use the **`cy.get()`** command to get the **`h3`** element, and use the **`.contains()`** command to assert that the heading contains the text “Login”
6. Execute the test runner using the watch command: **`npm run test --watch`**

Finding Elements

Finding Elements

- It’s “jQuery”
- Using cy.get(), .within(), and .find()
- Best Practices

It's "jQuery"

- Cypress bundles jQuery
- Commands do not return a jQuery object
- The `.get()` querying behavior is the same as jQuery's `$()`

Using `cy.get()`

- Specify a jQuery selector to get an element in the DOM
- Uses a promise-like API to get the jQuery object asynchronously
- We can now use a chai assertion within the **`.then()`** callback function

```
cy.get(selector)
cy.get(alias)
cy.get(selector, options)
cy.get(alias, options)
```

```
it('should find the books list', () => {
  cy.get('.books').should('be.visible');
});
```

Using `cy.get()`

`02-cy-get`

1. Open **`/projects/earnings-app-e2e/src/integrations/login-page.spec.ts`**
2. Create a new test to assert that the login form is visible.
3. Refer to the visibility documentation at: https://docs.cypress.io/guides/references/assertions#Visibility
4. First, visit the application login page using the **`cy.visit('/')`** command
5. Then, use **`cy.get()`** command to get the **`form`** element, then use the **`.should('be.visible')`** assertion to ensure the element is visible
6. Execute the test runner using the watch command: **`npm run test --watch`**

Using `cy.get()` and `.within()`

- Scope all subsequent tests within the element queried for using **`cy.get()`**
- Useful for testing forms

```
it('should display the login form fields', () => {
  cy.visit('/');
  cy.get('form').within(($form) => {
    cy.get(‘input[formcontrolname="username"]')
      .should('be.visible');
    cy.get(‘input[formcontrolname="password"]')
      .should('be.visible');
  });
});
```

Using `cy.get()` and `.within()`

`03-cy-get-and-within`

1. Open **`/projects/earnings-app-e2e/src/integrations/login-page.spec.ts`**
2. Create a new test to assert that the username and password fields are visible
3. First, visit the application login page using **`cy.visit('/')`**
4. Then, use **`cy.get()`** command to get the **`form`** element, and then use the **`.within()`** command with a callback function to get the username and password elements using the **`cy.get()`** command and assert that the elements are visible
5. Execute the test runner using the watch command: **`npm run test --watch`**

Using `cy.get()` and `.find()`

- Find a descendent element using a jQuery selector
- Useful to assert specific classes on an element (such as active) within a parent element
- The **`find()`** command *cannot* be chained off of **`cy`**

```
it('should display the login form fields', () => {
  cy.visit('/');
  cy.get('form').find('input').should('have.length', 2);
});
```

Using `cy.get()` and `.find()`

`04-cy-get-and-find`

1. Open **`/projects/earnings-app-e2e/src/integrations/login-page.spec.ts`**
2. Create a new test to assert that the form fields exist and that there are 2 inputs
3. First, visit the application login page using **`cy.visit('/')`**
4. Then, use **`cy.get()`** command to get the **`form`** element, and then use the **`.find()`** command to query for **`input`** elements and assert the length of the elements found is **`2`**
5. Execute the test runner using the watch command: **`npm run test --watch`**

```
// ⛔ very brittle
cy.get('button')

// ⛔ coupled to styling
cy.get('.button.large')

// ⚠️ coupled to class
cy.get('.books')

// ⚠️ coupled to ID
cy.get('#books')

// ✅ isolated and not brittle
cy.get('[data-test-id="bookEditButton"]')
```

```
it('should display the login form fields', () => {
  cy.visit('/');
  cy.get('form').within(($form) => {
    cy.get(‘[data-test-id="usernameInput"]')
      .should('be.visible');
    cy.get(‘[data-test-id="passwordInput"]')
      .should(‘be.visible');
  });
});
```

Best Practices

`05-data-attr-selectors`

1. Open /projects/earnings-app-e2e/src/integrations/login-page.spec.ts

2. Refactor the previous test to find the elements using the data-test-id attributes:

    1. **`data-test-id="usernameInput"`**

    2. **`data-test-id="passwordInput"`**

3. Execute the test runner using the watch command: **`npm run test --watch`**

Interacting with Elements

Interacting with Elements

- Simulate user interactions
- Cypress triggers appropriate event
- Ensure element is actionable
- Error if element is not visible within timeout

Actionability

- Is the element visible?
- Is the element disabled?
- Is the element detached from the DOM?
- Is the **`readonly`** attribute set on the element when attempting to **`type()`**?

Actionability

- Is the element animating? If so, wait until the animation has stopped.
  - Determined by the **`animationDistanceThreshold`** configuration
  - Disabled using the **`waitForAnimations`** configuration
- Is the element covered by a parent element?
- Before interacting, element is scrolled into view
- Events are triggered at the center of the element

```
it('should log the user in when providing valid credentials', () => {
  cy.visit(‘/');

  cy.get('bco-login-form')
    .find('[data-test-id="usernameInput"]')
    .type(‘Admin');
});
```

```
it('should log the user in when providing valid credentials', () => {
  // code omitted for brevity

  cy.get('bco-login-form')
    .find('[data-test-id="passwordInput"]')
    .type(‘password');
});
```

```
it('should log the user in when providing valid credentials', () => {
  // code omitted for brevity

  cy.get('bco-login-form')
    .find('[data-test-id="loginButton"]')
    .click()
});
```

```
it('should log the user in when providing valid credentials', () => {
  cy.visit('/');

  cy.get('bco-login-form')
    .find('[data-test-id="usernameInput"]')
    .type('Admin');

  cy.get('bco-login-form')
    .find('[data-test-id="passwordInput"]')
    .type('password');

  cy.get('bco-login-form')
    .find('[data-test-id="loginButton"]')
    .click()
});
```

Special Character Sequences

- {enter}
- {leftarrow}
- {rightarrow}
- {backspace}
- {{}
- https://docs.cypress.io/api/commands/type#Arguments

Interacting with Elements

`06-interacting-with-elements`

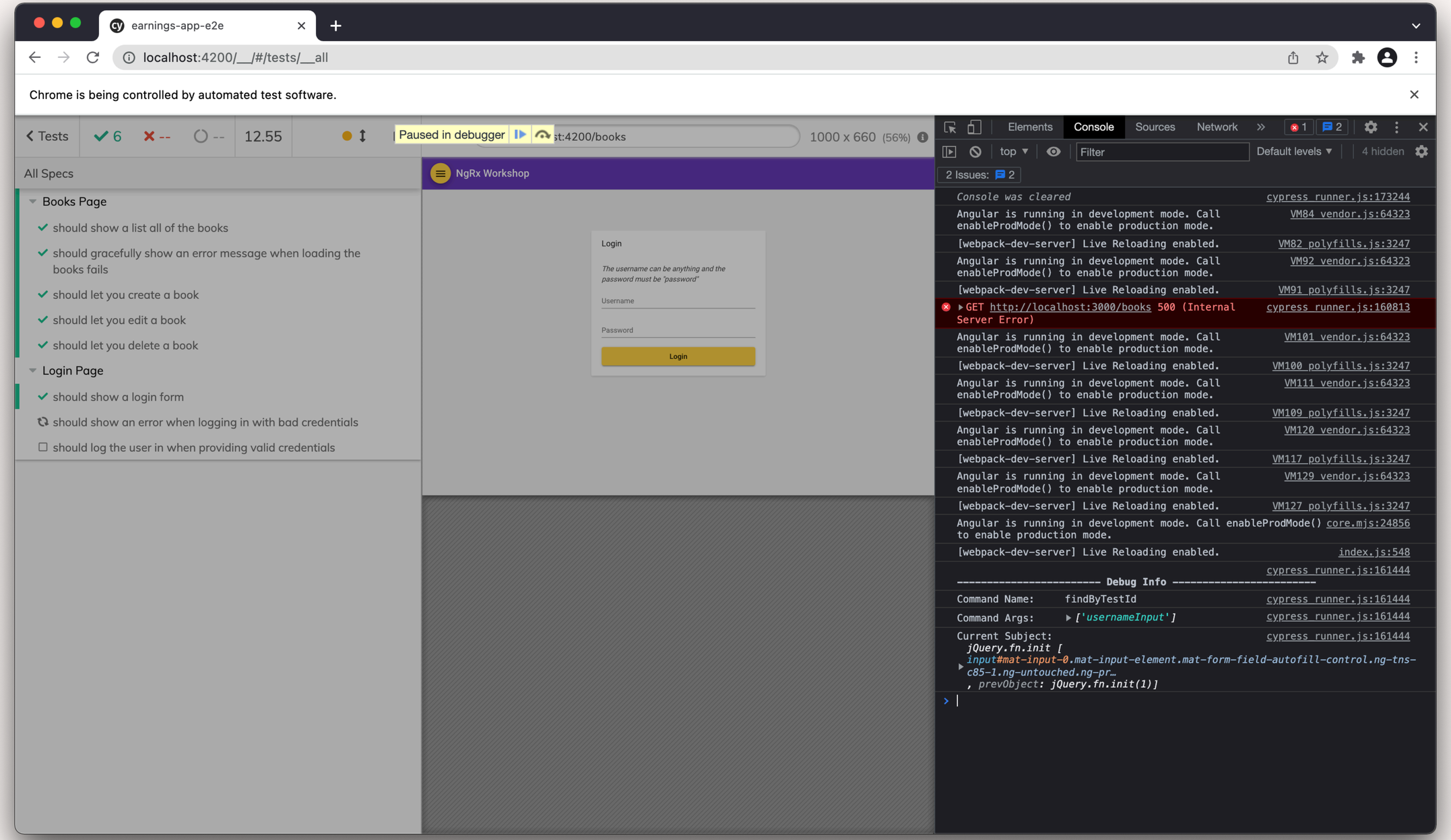
1. Open **`/projects/earnings-app-e2e/src/integrations/login-page.spec.ts`**
2. Create a new test that asserts the login page “should log the user in when providing valid credentials”
3. Use **`cy.get()`** to get the **`<bco-login-form>`** element
4. Then, use **`.find()`** to type the username “Admin” into the username input, and the password “password” into the password input
5. Finally, use **`.click()`** to trigger the click event on the login button
6. Execute the test runner using the watch command: **`npm run test --watch`**

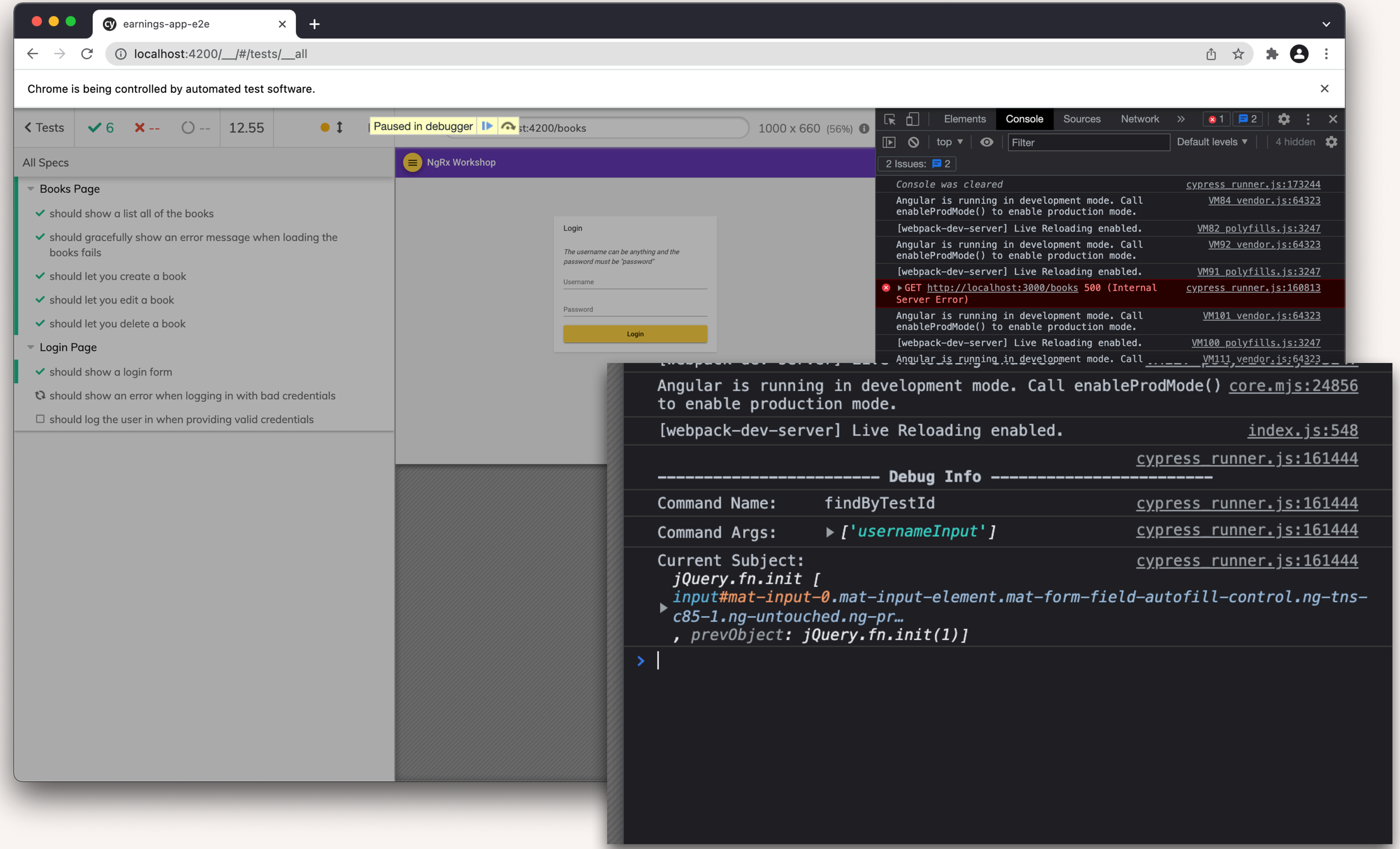
More Interactions

- `.dblclick()`
- `.rightclick()`
- `.clear()`
- `.check()`
- `.uncheck()`
- `.select()`
- `.trigger()`

```
cy.get('a').trigger('mousedown');
```

```
cy.get('a').debug().click();
```


earnings-app-e2e
localhost:4200/__/#/tests/__all
Chrome is being controlled by automated test software.
Tests
6
--
--
12.55
Paused in debugger
st:4200/books
1000 x 660 (56%)
All Specs
Books Page
should show a list all of the books
should gracefully show an error message when loading the books fails
should let you create a book
should let you edit a book
should let you delete a book
Login Page
should show a login form
should show an error when logging in with bad credentials
should log the user in when providing valid credentials
NgRx Workshop
Login
The username can be anything and the password must be "password"
Username
Password
Login
Elements
Console
Sources
Network
top
Filter
Default levels
4 hidden
2 Issues: 2
Console was cleared
cypress_runner.js:173244
Angular is running in development mode. Call enableProdMode() to enable production mode.
VM84 vendor.js:64323
[webpack-dev-server] Live Reloading enabled.
VM82 polyfills.js:3247
Angular is running in development mode. Call enableProdMode() to enable production mode.
VM92 vendor.js:64323
[webpack-dev-server] Live Reloading enabled.
VM91 polyfills.js:3247
GET http://localhost:3000/books 500 (Internal Server Error)
cypress_runner.js:160813
Angular is running in development mode. Call enableProdMode() to enable production mode.
VM101 vendor.js:64323
[webpack-dev-server] Live Reloading enabled.
VM100 polyfills.js:3247
Angular is running in development mode. Call enableProdMode() to enable production mode.
VM111 vendor.js:64323
[webpack-dev-server] Live Reloading enabled.
VM109 polyfills.js:3247
Angular is running in development mode. Call enableProdMode() to enable production mode.
VM120 vendor.js:64323
[webpack-dev-server] Live Reloading enabled.
VM117 polyfills.js:3247
Angular is running in development mode. Call enableProdMode() to enable production mode.
VM129 vendor.js:64323
[webpack-dev-server] Live Reloading enabled.
VM127 polyfills.js:3247
Angular is running in development mode. Call enableProdMode() to enable production mode.
core.mjs:24856
[webpack-dev-server] Live Reloading enabled.
index.js:548
cypress_runner.js:161444
---------- Debug Info ----------
Command Name: findByTestId
cypress_runner.js:161444
Command Args: ['usernameInput']
cypress_runner.js:161444
Current Subject:
cypress_runner.js:161444
jQuery.fn.init [
input#mat-input-0.mat-input-element.mat-form-field-autofill-control.ng-tns-c85-1.ng-untouched.ng-pr…
, prevObject: jQuery.fn.init(1)]


earnings-app-e2e
localhost:4200/__/#/tests/__all
Chrome is being controlled by automated test software.
Tests
6
--
--
12.55
Paused in debugger
1000 x 660 (56%)
All Specs
Books Page
should show a list all of the books
should gracefully show an error message when loading the books fails
should let you create a book
should let you edit a book
should let you delete a book
Login Page
should show a login form
should show an error when logging in with bad credentials
should log the user in when providing valid credentials
NgRx Workshop
Login
The username can be anything and the password must be "password"
Username
Password
Login
Elements
Console
Sources
Network
top
Filter
Default levels
4 hidden
2 Issues: 2
Console was cleared
cypress_runner.js:173244
Angular is running in development mode. Call enableProdMode() to enable production mode.
VM84 vendor.js:64323
[webpack-dev-server] Live Reloading enabled.
VM82 polyfills.js:3247
Angular is running in development mode. Call enableProdMode() to enable production mode.
VM92 vendor.js:64323
[webpack-dev-server] Live Reloading enabled.
VM91 polyfills.js:3247
GET http://localhost:3000/books 500 (Internal Server Error)
cypress_runner.js:160813
Angular is running in development mode. Call enableProdMode() to enable production mode.
VM101 vendor.js:64323
[webpack-dev-server] Live Reloading enabled.
VM100 polyfills.js:3247
Angular is running in development mode. Call enableProdMode() to enable production mode.
core.mjs:24856
[webpack-dev-server] Live Reloading enabled.
index.js:548
cypress_runner.js:161444
---------- Debug Info ----------
Command Name: findByTestId
cypress_runner.js:161444
Command Args: ['usernameInput']
cypress_runner.js:161444
Current Subject:
cypress_runner.js:161444
jQuery.fn.init [
input#mat-input-0.mat-input-element.mat-form-field-autofill-control.ng-tns-c85-1.ng-untouched.ng-pr…
, prevObject: jQuery.fn.init(1)]

Debugging Interactions

`07-debug-command`

1. Open **`/projects/earnings-app-e2e/src/integrations/login-page.spec.ts`**
2. Modify the “should log the user in when providing valid credentials” test using the **`.debug()`** command prior to the **`.click()`** command on the login button
3. Execute the test runner using the watch command: **`npm run test --watch`**
4. In Chrome DevTools note the value of **`subject`** in the Sources tab
5. In Chrome DevTools note the information in the Console tab

Assertions

Assertions

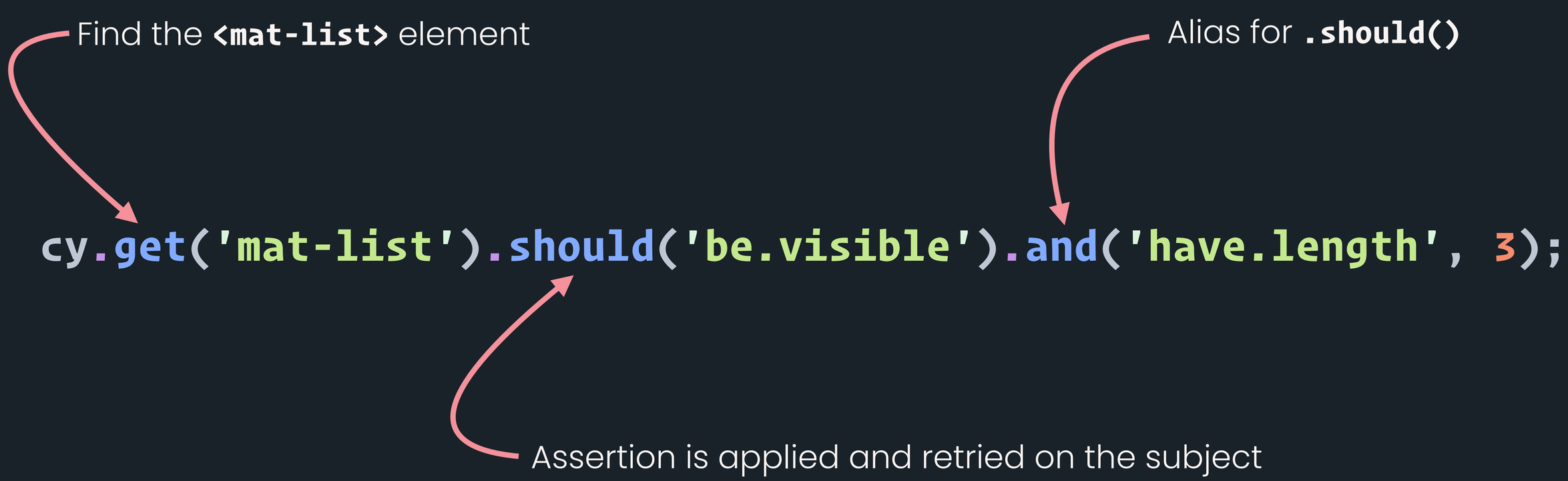
- [x] Default assertions
- [x] Writing assertions

How many assertions do you see?

```
it('should log the user in when providing valid credentials', () => {
  cy.visit('/');

  cy.get('bco-login-form')
    .find('[data-test-id="usernameInput"]')
    .type('Admin');

  cy.get('bco-login-form')
    .find('[data-test-id="passwordInput"]')
    .type('password');

  cy.get('bco-login-form')
    .find('[data-test-id="loginButton"]')
    .click()

  cy.get('bco-books-page')
    .should('be.visible')
});
```

Default Assertions

- The **`cy.visit()`** command asserts that the navigation results in a 200 success
- Each **`cy.get()`** command asserts that the element exists in the DOM
- Each **`.find()`** command asserts that the element exists in the DOM
- Each **`.type()`** command asserts the input elements are available and are not disabled
- The **`.click()`** command assert the button is available
- Finally, the **`.should()`** command asserts that the books page is visible


Find the <mat-list> element
Alias for .should()
cy.get('mat-list').should('be.visible').and('have.length', 3);
Assertion is applied and retried on the subject

```
.should(chainers)
.should(chainers, value)
.should(chainers, method, value)
.should(callbackFn)
```

https://github.com/chaijs/chai

Subjects and Promises

Subjects and Promises

- Commands yield a subject
- Consistent non-flaky tests
- Commands are asynchronous
- Commands are *not* Promises
- Subjects are chained together
- Tests are queued and executed when the test function exits

All Cypress command chains start with **`cy.`**

The **`.should()`** command yields a subject

```
cy.get('bco-book-list').should('be.visible');
```

All commands yield a subject

The **`.get()`** command yields a subject of the DOM element

Commands are asynchronous

```
it('should find the books list', () => {
  const books = cy.get('.books');
  expect(books).to.have.length(3);
});
```

```
it('should find the books list', () => {
  const books = cy.get('.books');
  ⛔ expect(books).to.have.length(3);
});
```

```
it('should find the books list', () => {
  cy.get('.books').then(($books) => {
    const el = $books.get(0);
    expect(el.children).to.have.length(3);
  });
});
```

```
it('should log the user in when providing valid credentials', () => {
  cy.visit('/');

  cy.get('bco-login-form')
    .find('[data-test-id="usernameInput"]')
    .type('Admin');

  cy.get('bco-login-form')
    .find('[data-test-id="passwordInput"]')
    .type('password');

  cy.get('bco-login-form')
    .find('[data-test-id="loginButton"]')
    .click()

  cy.get('bco-books-page')
    .should('be.visible')
});
```

All commands are queued and executed serially when the test returns

Cypress commands are *not* Promises

```
it('should find the books list', () => {
  ⛔ const books = await cy.get('.books');
});
```

Retry-ability is a core tenant of Cypress

`.get()` is retried until all assertions pass

```
cy.get('bco-book-list').should('be.visible');
```

Retry-ability

- Commands that query the DOM — e.g. **`.get()`**, **`.find()`**, **`.contains()`** — are retried until all assertions in the chain pass
- Some commands, such as **`.click()`**, are not retried as they can change the state of the application under test
- All assertions are retried in order until all assertions pass

Timeouts

- Retry-ability is limited to a timeout
- The default command timeout is 4000 ms
- We can override **`defaultCommandTimeout`** globally in the cypress.json configuration
- You can also disable retries by setting the **`timeout`** option to **`0`** for a specific command

Aliases

Aliases

- [x] Use aliases for fixtures
- [x] Use aliases for intercepts

```
describe('Login Page', () => {
  beforeEach(() => {
    // alias fixture as 'db'
    cy.fixture('example.json').as('db');
  });

  it('should have a login button', () => {
    cy.get<{books: BookModel[]}>('@db').then((db) => {
      cy.get(‘bco-book-list')
        .should('have.length', db.books.length);
    });
  });
});
```

```
describe('Login Page', () => {
  beforeEach(() => {
    // alias fixture as 'db'
    cy.fixture('example.json').as('db');
  });

  it('should have a login button', () => {
    cy.get<{books: BookModel[]}>(‘@db')
      .its(‘books')
      .then((books) => {
        cy.get(‘bco-book-list').should('have.length', books.length);
    });
  });
});
```

FAQs

Can I use async/await?

No

How do I get a native DOM element?

Use `.then()` command

How do I get the native `Document` element?

Use `cy.document()` command

How do I query for dynamic classes?

Use `data-*` attribute selectors

Will Cypress catch runtime exceptions?

Yes

Will Cypress catch rejected Promises?

No

Testing the Books Page

Books Page Critical Path

- [x] Shows a list of books
- [x] Gracefully handles errors when loading the books
- [x] Allows the user to create a book
- [x] Allows the user to edit an existing book
- [x] Allows the user to delete an existing book

Data Consistency

Use your backend's APIs to prepare data before running your tests

```
book = {
  id: uuid.v4(),
  name: 'The Lord of the Rings',
  earnings: '100',
  description:
    'The Lord of the Rings is an epic high fantasy novel...',
};

BooksApi.deleteAllBooks();
BooksApi.createBook(book);
```

```
cy.get(`[data-test-id="book-${book.id}"]`).should('contain', book.name);
```

Verifying the Books are Loaded

`08-list-of-books`

1. Open **`/projects/earnings-app-e2e/src/integration/books-page.spec.ts`**
2. Update the **`beforeEach`** code to delete all books using the **`BooksApi`** object and then create a book using the provided mock book
3. Copy and paste the login code you wrote in a previous section to log into the application in **`beforeEach`**
4. Write a test that verifies the book you created in **`beforeEach`** exists in the books list
    1. **`` `[data-test-id="book-${book.id}"]` ``**

We can improve the performance, reliability, and readability of this test

Intercepting Requests

Demo

```
cy.intercept('GET', 'http://localhost:3000/books').as('getBooks');

cy.visit('/');

cy.wait('@getBooks');
```

```
cy.intercept('GET', 'http://localhost:3000/books').as('getBooks');

cy.visit('/');

cy.wait('@getBooks');
```

```
cy.intercept('GET', 'http://localhost:3000/books').as('getBooks');

cy.visit('/');

cy.wait('@getBooks');
```

```
cy.intercept('GET', 'http://localhost:3000/books').as('getBooks');

cy.visit('/');

cy.wait('@getBooks');
```

```
cy.intercept('GET', 'http://localhost:3000/books').as('getBooks');

cy.visit('/');

cy.wait('@getBooks');
```

Intercept requests to intelligently wait for your UI to finish loading data from your APIs

Component Harnesses

Component Harnesses group common logic for interacting with components in your user interface

```ts
export const getBook = (bookId: string) =>
  cy.get(`[data-test-id="book-${bookId}"]`);
```

```
cy.get(`[data-test-id="book-${book.id}"]`).should('contain', book.name);
```

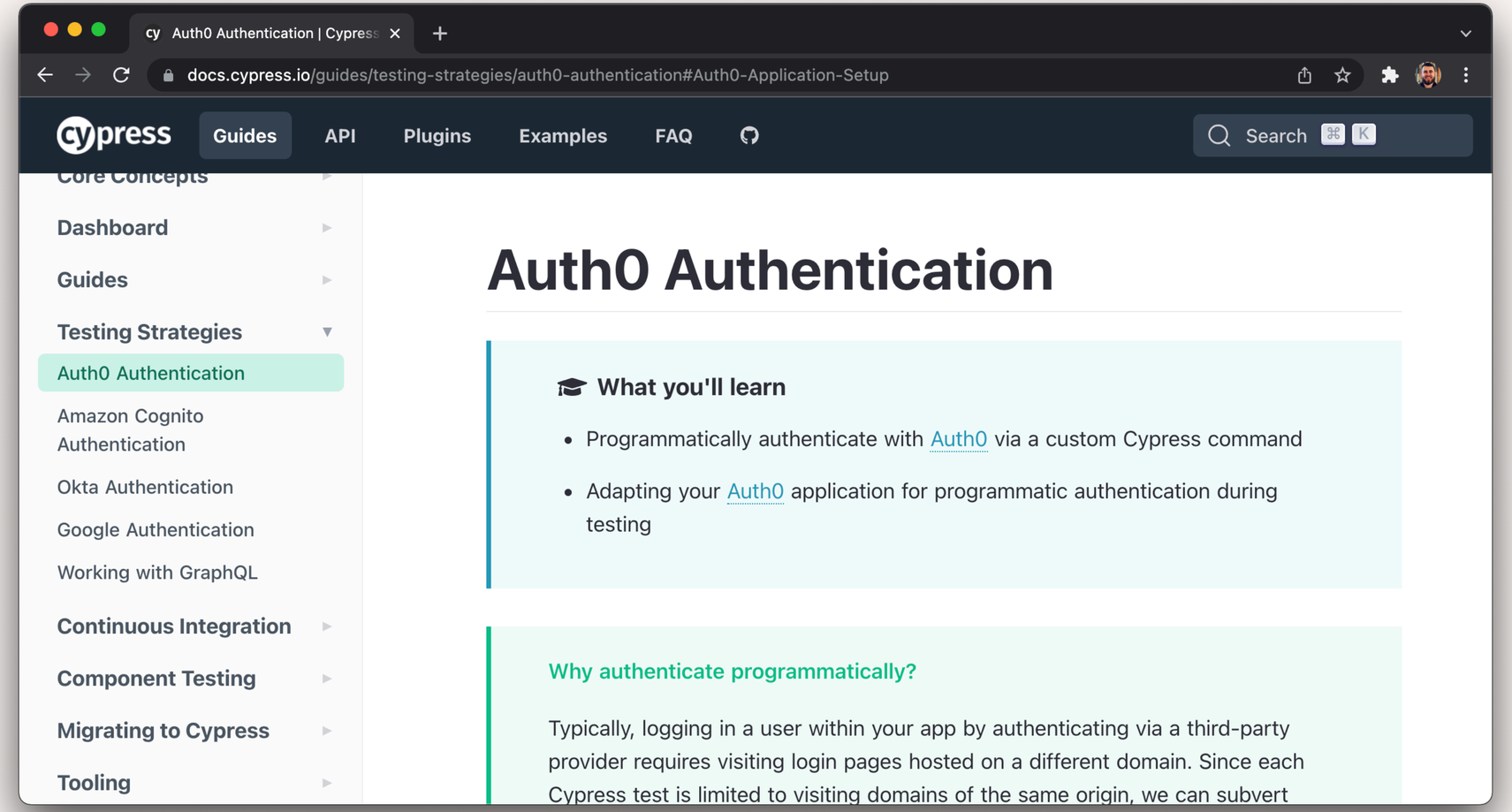
```
BookListComponent.getBook(book.id).should('contain', book.name);
```

```
const getLoginForm = () => cy.get('bco-login-form');
const getUsernameInput = () =>
  getLoginForm().find('[data-test-id="usernameInput"]');
```

```
const getLoginForm = () => cy.get('bco-login-form');
const getUsernameInput = () =>
  getLoginForm().find('[data-test-id="usernameInput"]');
```

Programmatic Authentication

Use your Authentication API to authenticate users instead of your user interface


cy Auth0 Authentication | Cypress
docs.cypress.io/guides/testing-strategies/auth0-authentication#Auth0-Application-Setup
cypress
Guides
API
Plugins
Examples
FAQ
Search ⌘ K
Core Concepts
Dashboard
Guides
Testing Strategies
Auth0 Authentication
Amazon Cognito Authentication
Okta Authentication
Google Authentication
Working with GraphQL
Continuous Integration
Component Testing
Migrating to Cypress
Tooling
Auth0 Authentication
What you'll learn
Programmatically authenticate with Auth0 via a custom Cypress command
Adapting your Auth0 application for programmatic authentication during testing
Why authenticate programmatically?
Typically, logging in a user within your app by authenticating via a third-party provider requires visiting login pages hosted on a different domain. Since each Cypress test is limited to visiting domains of the same origin, we can subvert

Refactor the Books List Test

`09-refactor-list-of-books`

1. Intercept requests to get all books and then wait for the request to finish after visiting the app

```
1.cy.intercept('GET', 'http://localhost:3000/books').as('getBooks');

2.cy.wait('@getBooks');
```

2. Replace the manual authentication code with a call to **`AuthApi.login(...)`** before **`cy.visit(...)`**
3. Add an element finder function that gets a book by ID to **`../support/book-list-component.harness.ts`**
4. Update the test in **`../integration/books-page.spec.ts`** to use your new element finder via the **`BookListComponent`** object

Interceptors with Mock Responses

Sparingly use interceptors to mock out the results of your user interface making HTTP requests

```
cy.intercept('GET', 'http://localhost:3000/books', {
  statusCode: 500,
  body: {
    error: 'Internal Server Error',
  },
});
```

Verifying the Books Page Handles Errors Gracefully

`10-error-loading-books`

1. In the **`"...gracefully show an error message..."`** test, re-intercept the request to get all books only this time mock out the response to throw a **`500 Internal Server Error`** response
2. Re-visit the page using **`cy.visit('/')`**
3. Wait for the intercepted request to complete
4. Add element finders to **`../support/books-page.harness.ts`** that retrieve the Books Page element and the contained error element

    `1.<books-page />`

    `2.<span class=".error">...</span>`

5. Assert that the Books Page contains the Error element after the intercepted request completes

De-Duplicating Setup Code using SIFERS

```ts
describe('Books Page', () => {
  let book: BookModel;

  beforeEach(() => {
    book = {
      id: uuid.v4(),
      name: 'The Lord of the Rings',
    };

    cy.intercept('GET', 'http://localhost:3000/books').as('getBooks');
    cy.visit('/');
    cy.wait('@getBooks');
  });

  it('should gracefully show an error message when loading the books fails', () => {
    cy.intercept('GET', 'http://localhost:3000/books', {
      statusCode: 500,
      body: {
        error: 'Internal Server Error',
      },
    }).as('getBooks');
    cy.visit('/');
    cy.wait('@getBooks');
  });
});
```

```
describe('Books Page', () => {
  let book: BookModel;

  beforeEach(() => {
    book = {
      id: uuid.v4(),
      name: 'The Lord of the Rings',
    };

    cy.intercept('GET', 'http://localhost:3000/books').as('getBooks');
    cy.visit('/');
    cy.wait('@getBooks');
  });

  it('should gracefully show an error message when loading the books fails', () => {
    cy.intercept('GET', 'http://localhost:3000/books', {
      statusCode: 500,
      body: {
        error: 'Internal Server Error',
      },
    }).as('getBooks');
    cy.visit('/');
    cy.wait('@getBooks');
  });
});
```

```ts
describe('Books Page', () => {
  let book: BookModel;

  beforeEach(() => {
    book = {
      id: uuid.v4(),
      name: 'The Lord of the Rings',
    };

    cy.intercept('GET', 'http://localhost:3000/books').as('getBooks');
    cy.visit('/');
    cy.wait('@getBooks');
  });

  it('should gracefully show an error message when loading the books fails', () => {
    cy.intercept('GET', 'http://localhost:3000/books', {
      statusCode: 500,
      body: {
        error: 'Internal Server Error',
      },
    }).as('getBooks');
    cy.visit('/');
    cy.wait('@getBooks');
  });
});
```

Simple **I**njectable **F**unctions **E**xplicitly **R**eturning **S**tate

```
let book: BookModel;

beforeEach(() => {
  book = {
    id: uuid.v4(),
    name: 'The Lord of the Rings',
  };

  cy.intercept('GET', 'http://localhost:3000/books').as('getBooks');
  cy.visit('/');
  cy.wait('@getBooks');
});

it('should gracefully show an error message when loading the books fails', () => {
  cy.intercept('GET', 'http://localhost:3000/books', {
    statusCode: 500,
    body: {
      error: 'Internal Server Error',
    },
  }).as('getBooks');
  cy.visit('/');
  cy.wait('@getBooks');
});
```

```
function setup(options: { throwErrorWhenLoadingBooks?: boolean } = {}) {
  const book = {
    id: uuid.v4(),
    name: 'The Lord of the Rings',
  };

  BooksApi.deleteAllBooks();
  BooksApi.createBook(book);

  if (options.throwErrorWhenLoadingBooks) {
    cy.intercept('GET', 'http://localhost:3000/books', {
      statusCode: 500,
      body: {
        error: 'Internal Server Error',
      },
    }).as('getBooks');
  } else {
    cy.intercept('GET', 'http://localhost:3000/books').as('getBooks');
  }

  AuthApi.login('Admin', 'password');
  cy.visit('/');
  cy.wait('@getBooks');

  return book;
}
```

```
function setup(options: { throwErrorWhenLoadingBooks?: boolean } = {}) {
  const book = {
    id: uuid.v4(),
    name: 'The Lord of the Rings',
  };

  BooksApi.deleteAllBooks();
  BooksApi.createBook(book);

  if (options.throwErrorWhenLoadingBooks) {
    cy.intercept('GET', 'http://localhost:3000/books', {
      statusCode: 500,
      body: {
        error: 'Internal Server Error',
      },
    }).as('getBooks');
```

```
};

BooksApi.deleteAllBooks();
BooksApi.createBook(book);

if (options.throwErrorWhenLoadingBooks) {
  cy.intercept('GET', 'http://localhost:3000/books', {
    statusCode: 500,
    body: {
      error: 'Internal Server Error',
    },
  }).as('getBooks');
} else {
  cy.intercept('GET', 'http://localhost:3000/books').as('getBooks');
}

AuthApi.login('Admin', 'password');
cy.visit('/');
```

```
function setup(options: { throwErrorWhenLoadingBooks?: boolean } = {}) {
  const book = {
    id: uuid.v4(),
    name: 'The Lord of the Rings',
  };

  BooksApi.deleteAllBooks();
  BooksApi.createBook(book);

  if (options.throwErrorWhenLoadingBooks) {
    cy.intercept('GET', 'http://localhost:3000/books', {
      statusCode: 500,
```

```
      statusCode: 500,
      body: {
        error: 'Internal Server Error',
      },
    }).as('getBooks');
  } else {
    cy.intercept('GET', 'http://localhost:3000/books').as('getBooks');
  }

  AuthApi.login('Admin', 'password');
  cy.visit('/');
  cy.wait('@getBooks');

  return book;
}
```

```
function setup(options: { throwErrorWhenLoadingBooks?: boolean } = {}) {
  const book = {
    id: uuid.v4(),
    name: 'The Lord of the Rings',
  };

  BooksApi.deleteAllBooks();
  BooksApi.createBook(book);

  if (options.throwErrorWhenLoadingBooks) {
    cy.intercept('GET', 'http://localhost:3000/books', {
      statusCode: 500,
      body: {
        error: 'Internal Server Error',
      },
    }).as('getBooks');
  } else {
    cy.intercept('GET', 'http://localhost:3000/books').as('getBooks');
  }

  AuthApi.login('Admin', 'password');
  cy.visit('/');
  cy.wait('@getBooks');

  return book;
}
```

Prefer using SIFERS over using beforeEach/beforeAll

```
it('should show a list all of the books', () => {
  const book = setup();

  BookListComponent.getBook(book.id).should('contain', book.name);
});

it('should gracefully show an error message when loading the books fails', () => {
  setup({ throwErrorWhenLoadingBooks: true });

  BooksPage.getError().should('contain', 'Error');
});
```

Using SIFERS to Deduplicate Setup Code

`11-sifers`

1. Create a SIFERS that accepts a parameter object with an option to throw an error when loading books

```
1.function setup(options: { throwErrorWhenLoadingBooks?: boolean } = {}) {}
```

2. Move setup code inside of **`beforeEach`** into your new SIFERS
3. Use the **`throwErrorWhenLoadingBooks`** flag to conditionally change the intercepted **`GET`** request to throw a **`500 Internal Server`** error
4. Update both tests to use the new **`setup`** SIFERS

Expanded Component Harnesses

Component Harnesses may also include functions that interact with the elements comprising the component

```
const getLoginForm = () => cy.get('bco-login-form');
const getUsernameInput = () =>
  getLoginForm().find('[data-test-id="usernameInput"]');
const getPasswordInput = () =>
  getLoginForm().find('[data-test-id="passwordInput"]');
const getLoginButton = () =>
  getLoginForm().find('[data-test-id="loginButton"]');

const fillLoginForm = (username: string, password: string) => {
  getUsernameInput().type(username);
  getPasswordInput().type(password);
};
const clickLoginButton = () => getLoginButton().click();
```

```
const getLoginForm = () => cy.get('bco-login-form');
const getUsernameInput = () =>
  getLoginForm().find('[data-test-id="usernameInput"]');
const getPasswordInput = () =>
  getLoginForm().find('[data-test-id="passwordInput"]');
const getLoginButton = () =>
  getLoginForm().find('[data-test-id="loginButton"]');

const fillLoginForm = (username: string, password: string) => {
  getUsernameInput().type(username);
  getPasswordInput().type(password);
};
const clickLoginButton = () => getLoginButton().click();
```

```
const getLoginForm = () => cy.get('bco-login-form');
const getUsernameInput = () =>
  getLoginForm().find('[data-test-id="usernameInput"]');
const getPasswordInput = () =>
  getLoginForm().find('[data-test-id="passwordInput"]');
const getLoginButton = () =>
  getLoginForm().find('[data-test-id="loginButton"]');

const fillLoginForm = (username: string, password: string) => {
  getUsernameInput().type(username);
  getPasswordInput().type(password);
};
const clickLoginButton = () => getLoginButton().click();
```

```
it('should log the user in when providing valid credentials', () => {
  cy.visit('/');

  cy.get('bco-login-form')
    .find('[data-test-id="usernameInput"]')
    .type('Admin');

  cy.get('bco-login-form')
    .find('[data-test-id="passwordInput"]')
    .type('password');

  cy.get('bco-login-form').find('[data-test-id="loginButton"]').click();
});
```

```
it('should log the user in when providing valid credentials', () => {
  cy.visit('/');

  LoginFormComponent.fillLoginForm('Admin', 'password');
  LoginFormComponent.clickLoginButton();
});
```

Test that Users Can Create a Book

`12-form-components`

1. Add element finders to **`../support/book-form-component.harness.ts`** that find the form element, name input, earnings input, description input, and save button

   `1.bco-book-detail`

   `2.[data-test-id="nameInput"]`

   `3.[data-test-id="earningsInput"]`

   `4.[data-test-id="descriptionInput"]`

   `5.[data-test-id="saveButton"]`

2. Add a function to **`../support/book-form-component.harness.ts`** called **`fillForm()`** that fill the inputs using a provided parameter of type **`BookRequiredProps`** and a function called **`saveForm()`** that clicks on the save button

3. Implement the **`"...create a book..."`** test using the **`BookFormComponent`** object

Use your backend's API to verify that your user interface interacted with the API in an expected way

```
BooksApi.getBooks()
  .its('body')
  .should((books) => {
    expect(books.find((b) => b.name === book.name)).to.exist;
  });
```

Assert the Book Was Created

`13-assert-book-was-created`

1. Add an interceptor to the **`setup`** SIFERS that intercepts requests to create books aliased as **`"createBook"`**

   ```
   1.POST http://localhost:3000/books
   ```

2. Update the **`"...create a book..."`** test to wait for the **`"createBook"`** request to complete after saving the form

3. Use the **`BooksApi`** object to get all books and use the **`body`** of the result to assert that the book was actually created

Test Editing a Book

```
getNameInput().type(book.name);
```

Demo

Editing a Book

`14-editing-book`

1. Add an element finder to **`../support/book-list-component.harness.ts`** that finds the edit button on a book

    `1.[data-test-id="bookEditButton"]`

2. Add a function to **`../support/book-list-component.harness.ts`** called **`clickEditButtonOnBook`** that clicks the edit button on the book

3. Update the **`fillForm`** function in **`../support/book-form-component.harness.ts`** to clear inputs before filling them in

4. Using these new functions, implement the first part of the **`"...edit a book..."`** test to click the edit button on the book created by the SIFERS and then use the **`fillForm`** function to update the name of the book

```
BooksApi.getBook(book.id).its('body').should('deep.include', {
  name: newName,
});
```

Assert the Book Was Edited

`15-assert-book-was-edited`

1. Add an interceptor to the **`setup`** SIFERS that intercepts update requests for books

    ```
    1.PATCH http://localhost:3000/books/*
    ```

2. Update the **`"...update a book..."`** test to save the form and wait for the intercepted **`PATCH`** request to complete

3. Assert that the book was actually updated by using the **`BooksApi`** object to get the book back from the API

    ```
    1.BooksApi.getBook(book.id)
    ```

Writing a Complete Test

Test That a User Can Delete a Book

`16-delete-book`

1. Write a complete implementation for the **`"...delete a book..."`** test

    1. **`[data-test-id="bookDeleteButton"]`**

2. As you write the test, consider:

    1. What network requests need to be intercepted?
    2. What changes need to be made to the **`setup`** SIFERS?
    3. What changes need to be made to the **`BookListComponent`**?
    4. How do you prove the book was deleted using **`BooksApi`**?

Bonus Challenges

Creating a Command

17-get-by-test-id

1. Open **`../support/commands.ts`**

2. Update the **`Chainable`** interface in the **`Cypress`** namespace to add a method called **`getByTestId`**

   ```
   1. getByTestId(id: string): Chainable<Element>;
   ```

3. Use **`Cypress.Commands.add(...)`** to implement the **`getByTestId`** command

4. Update **`getBook`** in **`../support/book-list-component.harness.ts`** to use **`cy.getByTestId(...)`**

Creating a Chainable Command

18-find-by-test-id

1. Open **`../support/commands.ts`**

2. Update the **`Chainable`** interface in the **`Cypress`** namespace to add a method called **`findByTestId`**

   ```
   1. findByTestId(id: string): Chainable<Element>;
   ```

3. Use **`Cypress.Commands.add(...)`** to implement the **`findByTestId`** command that operates on an element as the previous subject

   ```
   1. { prevSubject: 'element' }
   ```

4. Refactor all of the harnesses to use a mixture of **`getByTestId`** and **`findByTestId`**

Writing a Plugin and Using Environment Variables

`19-cypress-env`

1. Open **`../plugins/index.ts`**

2. Add code to the **`plugin`** function that modifies config.env to add environment variables for the app's **`API_BASE_URL`**, **`USERNAME`**, and **`PASSWORD`**

    1. **`http://localhost:3000/books`**

    2. **`Admin`**

    3. **`password`**

3. Refactor **`../integration/books-page.spec.ts`** to use **`Cypress.env(...)`**

Final Takeaways

Write a few E2E tests, mostly critical path, that test integration between components, the browser, your services, and backend APIs

The best E2E test is a test that is reliable, readable, and fast

Cypress test runner is open source

Cypress commands are executed asynchronously and are promise-like to ensure consistency and avoid flaky tests

Component harnesses enable shorter and more readable tests while also encouraging code reusability

Simple Injectable Functions Explicitly Returning State (SIFERS)

move setup code into reusable functions that allow tests to intelligently control setup behavior

liveloveapp

Find Absolute Joy in Shipping Apps