End-to-End (E2E) Testing.

Overview

- Testing the entire system as a whole.
  (UI + Server-side + Database)

- Concerns:
  - Functionality. ****
    - From the USER interface perspective.
  - Performance.
  - Load/Stress.

Overview

- Blackbox approach – we are not focused on the internals of the system; the only concern is expected response for specific inputs.
- Preceded by Unit and subsystem (e.g. web API) testing to resolve ‘internal’ errors.
- May also be interested in side-effects, e.g. database changes.
- The Asynchronous nature (for web/mobile apps).

Web App Target

- Web apps - Targeting the browser interface.
  - Functionality.
  - Form submits.
  - Navigation.
  - Flows e.g. shopping cart checkout.
  - Visual testing (CSS).

Automation Tool Suite

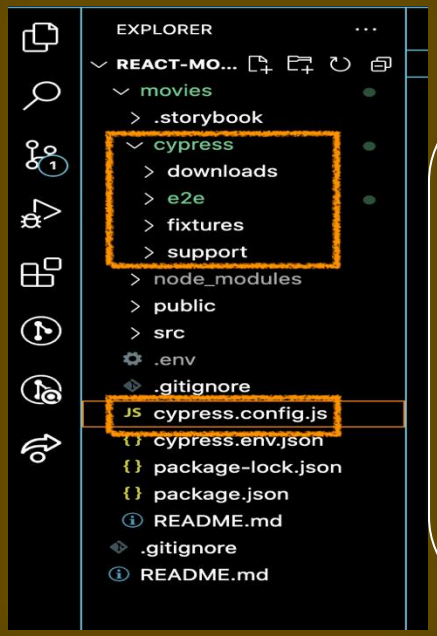
- Traditional tool suite: Mocha + Chai + Selenium.

- Modern tool suite: Cypress.
  - Uses Mocha and Chai internally.

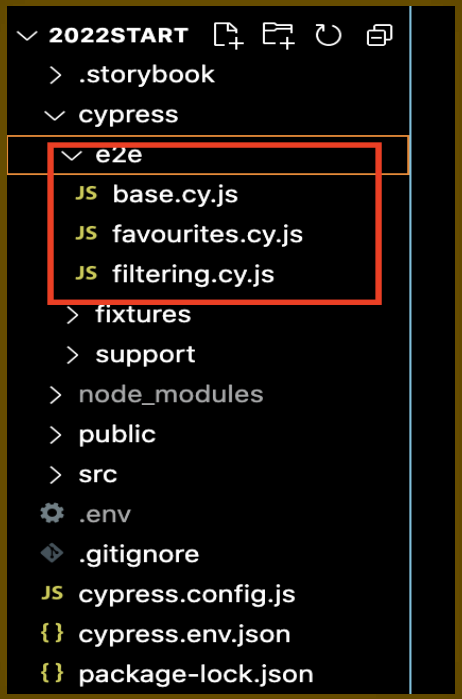
- Cypress.
  - Win / Mac / Linux.
  - MIT License.
  - Open Source.
  - Not suitable for Visual testing; Use Percy instead.

Cypress - Overview

- Getting started: $ npm install –save-dev cypress



- fixtures – sample data.
- e2e– test code, termed specs.
- support – utility code.
- **cypress.config.js** – config file.

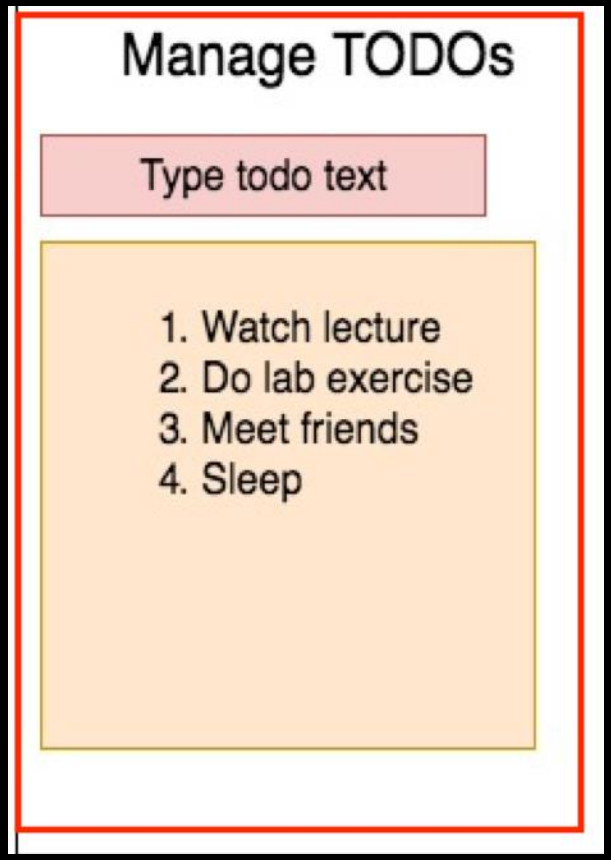


- CLI has two main commands:

  $ npx cypress open # GUI interactive mode

  $ npx cypress run # headless mode.

Sample Test Code.




```
describe(“TODO app", () => {
    it(‘should add 2 todos', () => {
      cy.visit('http://localhost:3000')
      cy.get(‘input')
          .type(‘Watch lecture{enter}')
          .type(’Do lab exercise{enter}')
      cy.get('li')
          .should('have.length', 2)
    })
})
```

- **Declarative style.**
- **Method Chaining style e.g.**
  cy.get(…).type(…)

Cypress statements.

- cy.get(..selector..).should(..expectation..)

  Command Expectations (Optional)

- Commands – All about accessing and interacting with browser DOM elements.
  - get(selector) - Get the DOM elements that match the selector.
  - contains(text) - Get the DOM element that contains the text, e.g. cy.contains('Add')
  - find(selector) - Get the child DOM element(s).

    e.g. Fund the dropdown menu inside the web form and select the Medium option.

    cy.get('form').find('select').select('Medium')

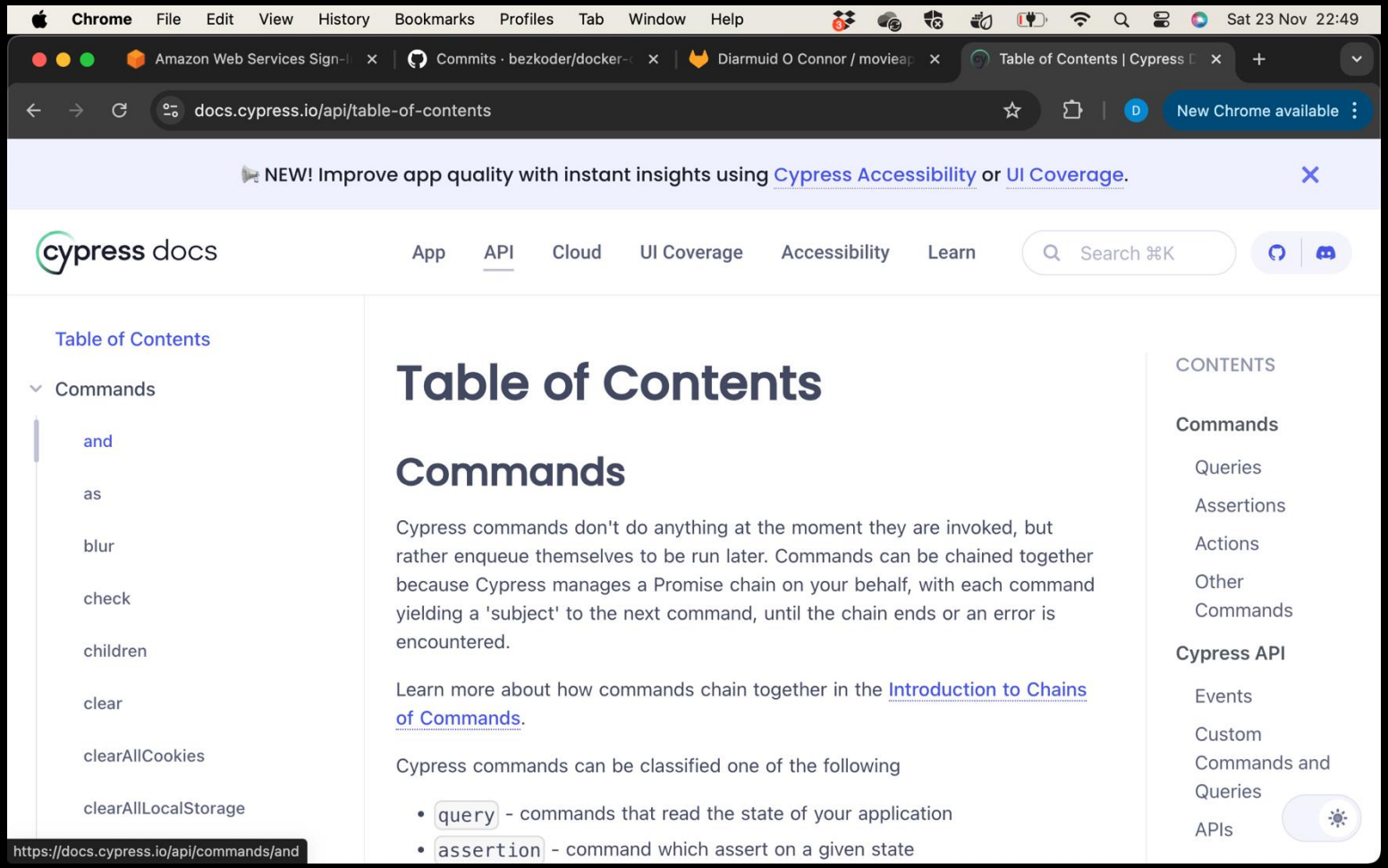
Cypress statements.

- Commands (Contd.) – All about accessing and interacting with browser DOM elements.
  - click() – click a DOM element, e.g. cy.get('button').click()
  - eq(n) – Get the nth DOM element in an array of elements, e.g.

    cy.get(‘input’).eq(2).type(‘1 Main Street’);

    cy.get('li’).eq(4).should(‘equal’, ‘Agile’)

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



Cypress statements - Selectors.

cy.get(selector).should(expectation)

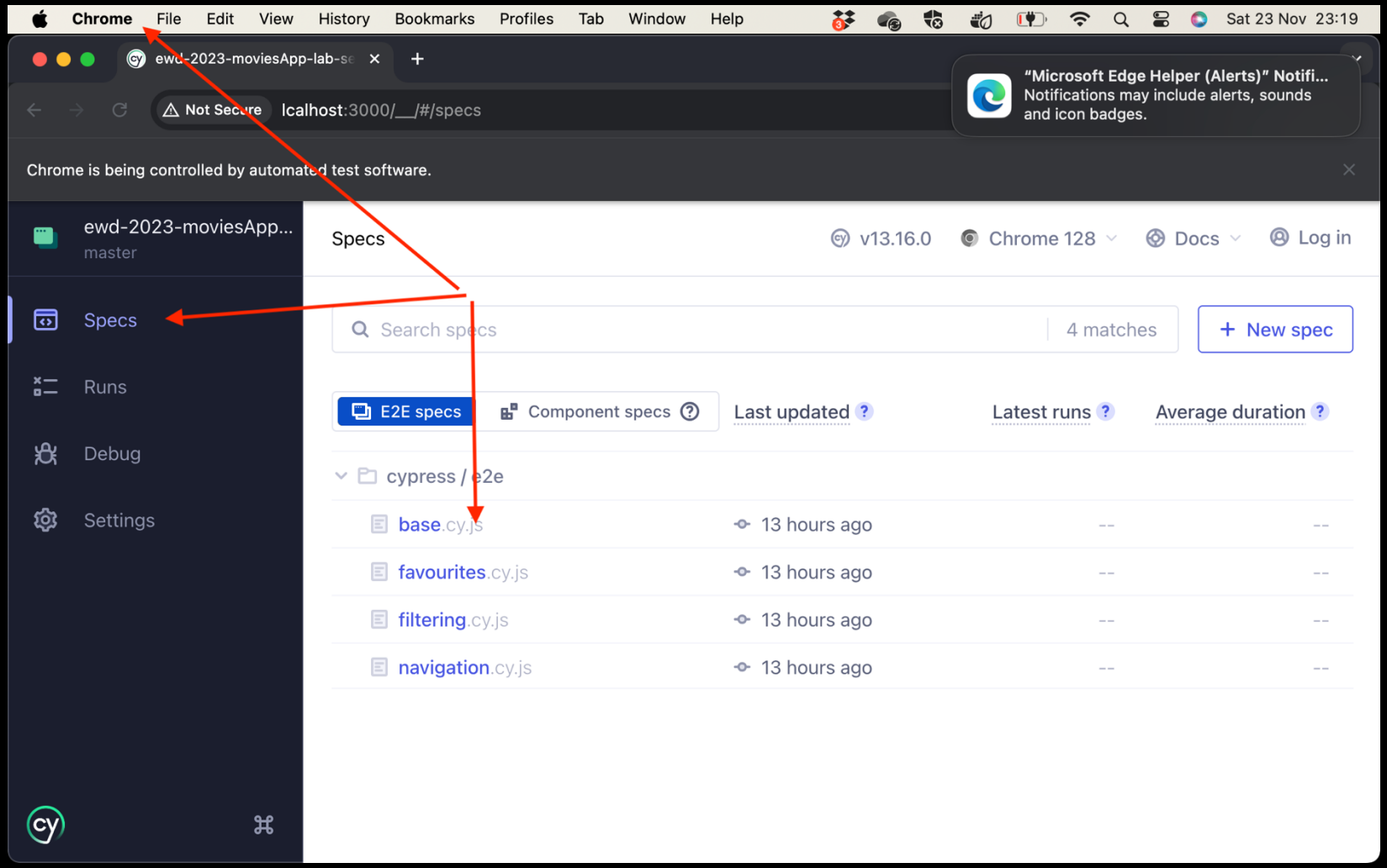
- Selector: Based on JQuery syntax.
  - HTML Tag type: e.g. cy.get(‘button’)
  - Elemen Id: e.g. cy.get(’#heading’)
  - CSS Class: e.g. cy.get(’.info-message’)
  - Attributes, e.g. cy.get('button[type=submit]').click()
  - nth-child, e.g. get the 8th column of the 3rd row in a table

  cy.get('tbody').find(‘tr’).eq(2).find(‘td’).eq(7)
  - Selectors can be combined, e.g. cy.get(‘div.container’) (the div tag with CSS class .container)

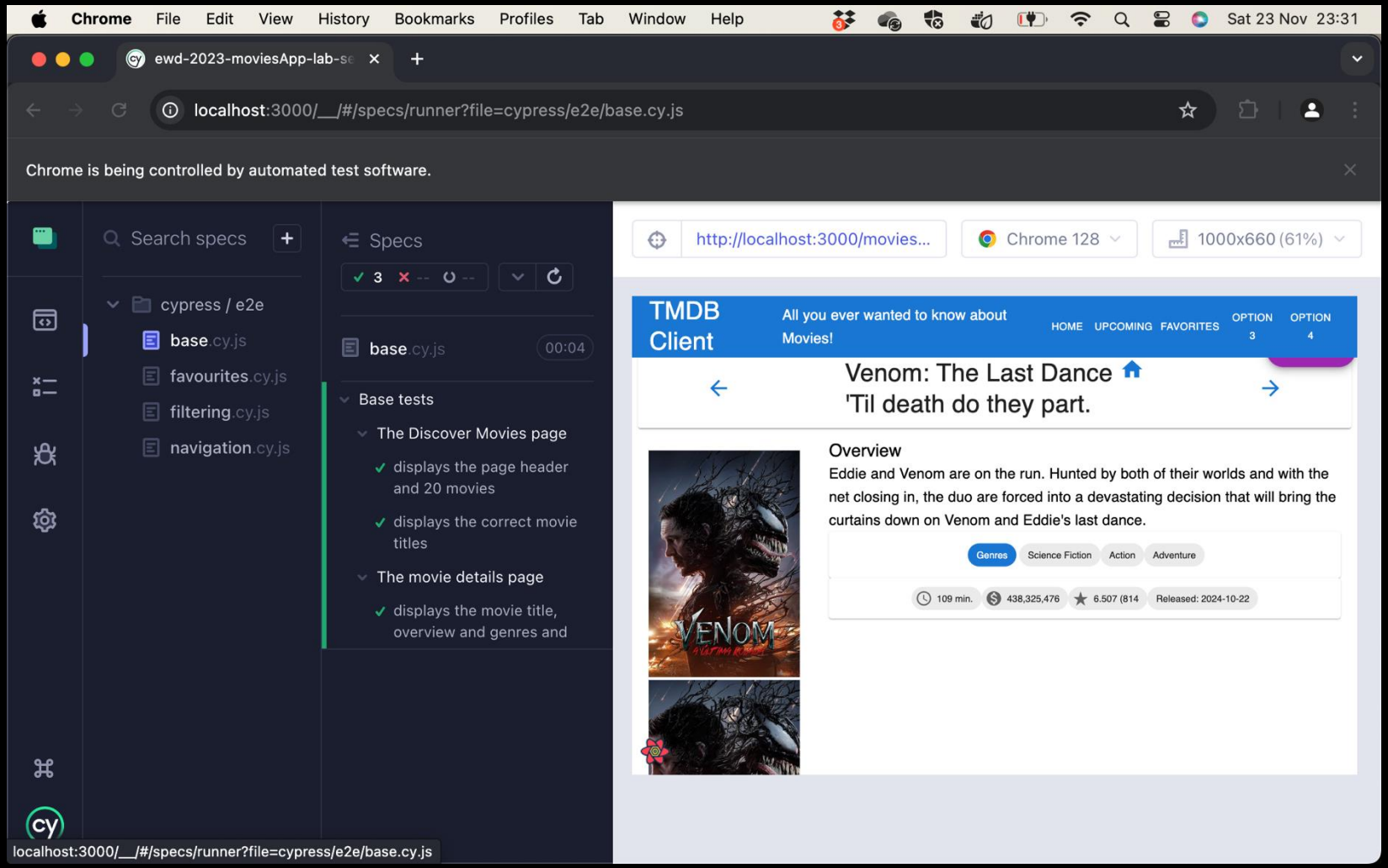
Test Runner

- Main features:
  - Tests run inside the browser.
  - Test code has Full access to browser's resources, e.g. DOM, cookies, local storage.
  - Web-App-Framework-agnostic.
  - Flake-free test execution.
  - Deterministic, repeatable, consistent execution flow.
  - Auto retries commands to accommodate slow DOM construction.
  - Deals with asynchronous nature of the web.

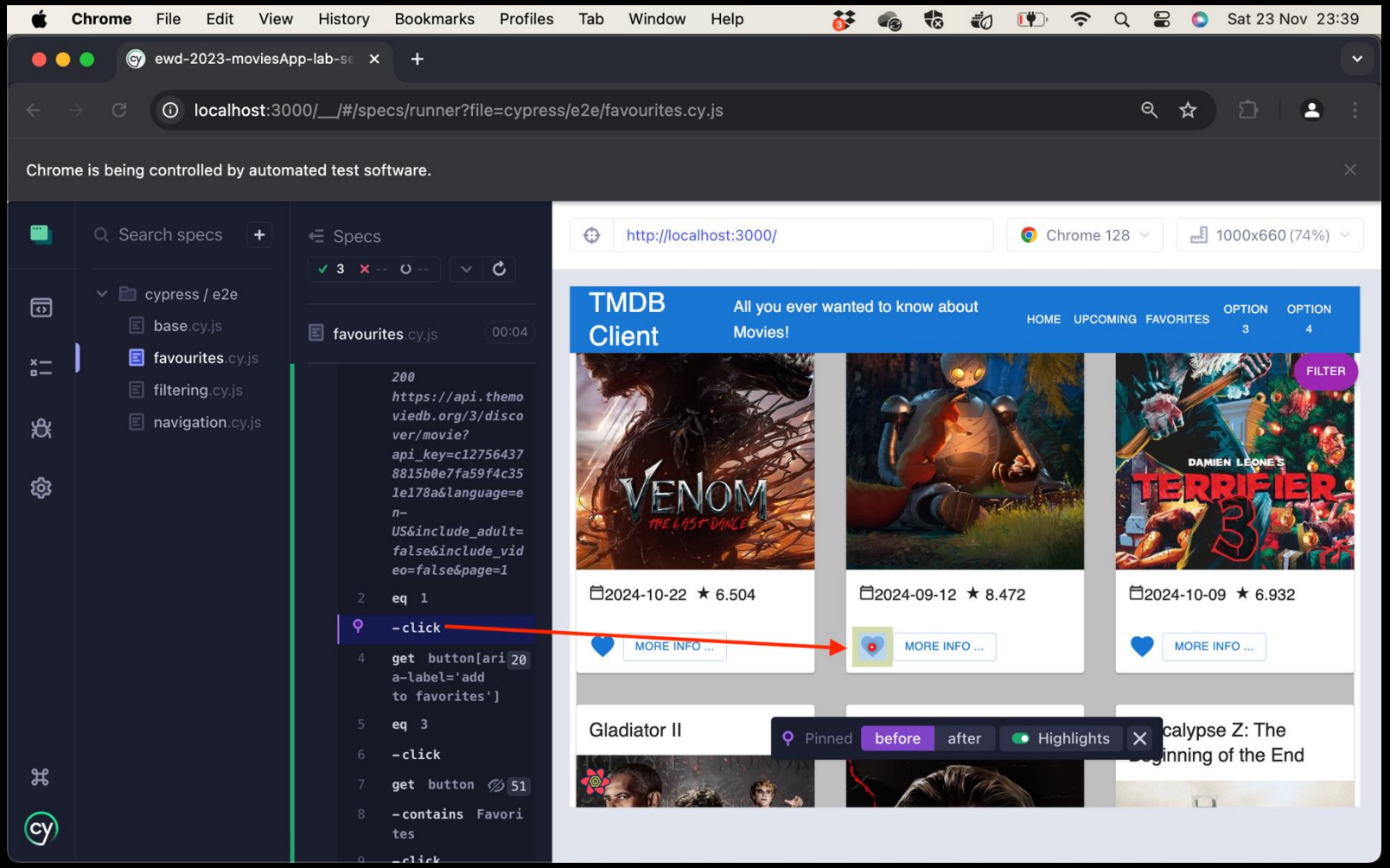
$ npx cypress open



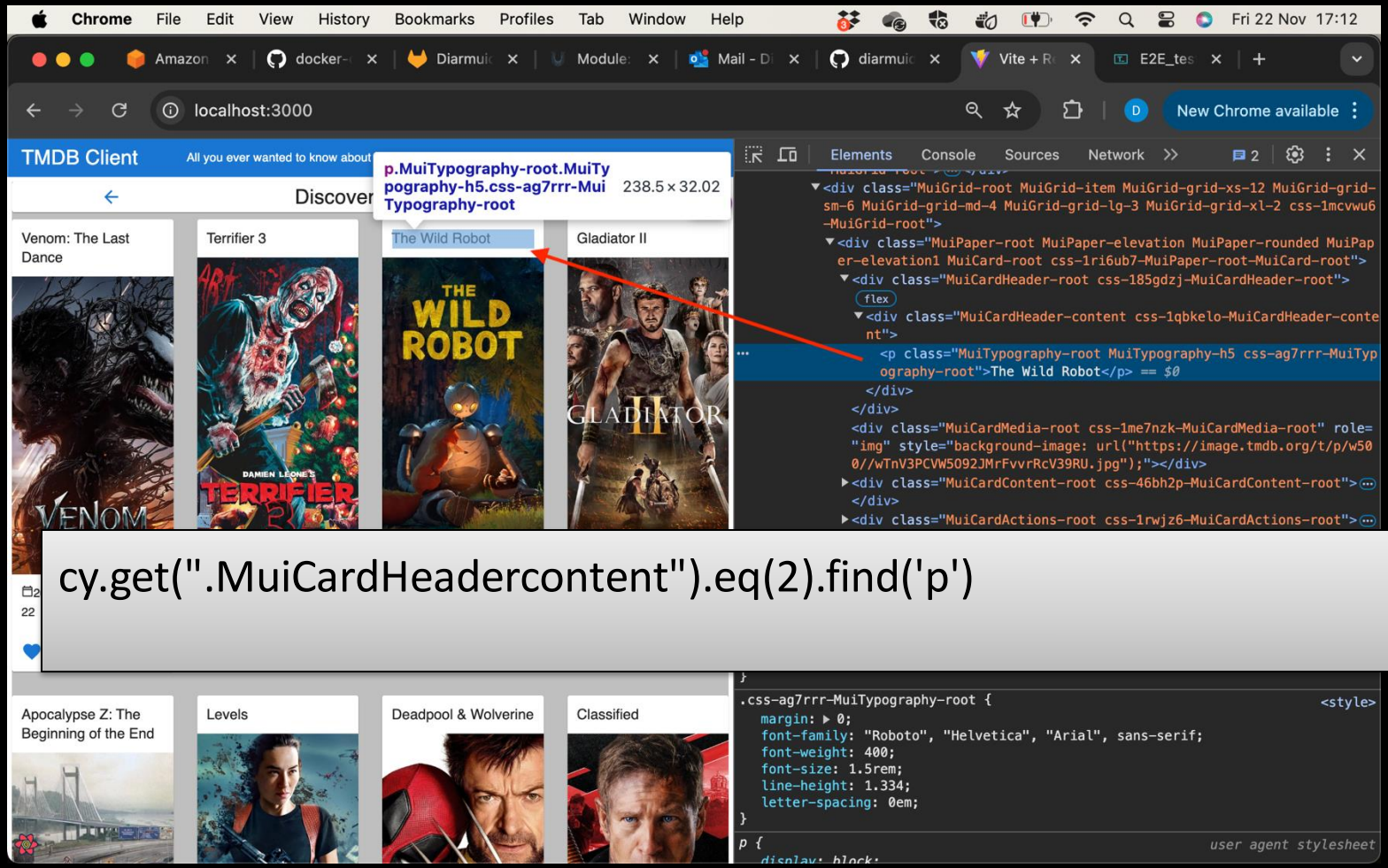
Test Execution
(Green bar / Red Bar)



- Time-travel **– Step through test code to track the app's UI state. Great for debugging.**



- Use Chrome's dev tools to help with choosing a selector for a command.



Headless Test Runner

- Running Cypress tests without a UI.
- Headless mode:

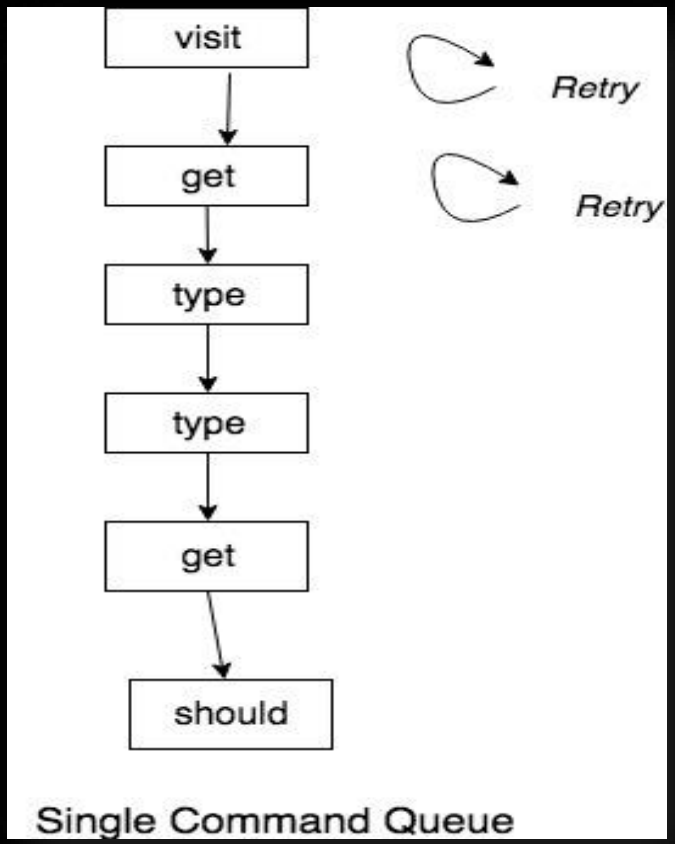
  $ npx cypress run
- Ideal for CI (Continuous Integration) environment.

- Can generates video recordings (disabled by default) and stored in *cypress/e2e/videos*.
  - .mp4 file type.
  - Facilitates sharing and project visibility.
- Lots of command line options
  - --spec file_name_pattern
  - --config – override settings in cy.config.js
  - --record, --browser etc,

Commands are enqueued.

```
describe(“TODO app", () => {
  it(‘should add 2 todos', () => {
    cy.visit('http://localhost’)
    cy.get(‘input')
      .type(‘Watch lecture{enter}')
      .type(’Submit lab {enter}')
    cy.get('li')
      .should('have.length', 2)
  })
})
```

- Commands are first enqueued and then run serially in a controlled manner, i.e. retries, delays.
- Guarantees a deterministic or flake-free test behaviour.



Headless Test Runner

- Chain of commands.

```
cy.get("h4").next().contains(movie.overview);
cy.get("li").eq(2).find('h3')
            .should('contain','Waterford')
```

- A chains always begins with cy.
- Each command yields a subject to the next one in the chain.
- We can act on a subject directly with .then()

```
cy.get('div').eq(2).find('button')
  .then ( (buttonElement) => {
    onst cls = buttonElement.class()
    . . . . . .
  })
```

Headless Test Runner

- E2E testing – aka System testing.

- Black-box mindset – does app produce expected output for a given input.

- Cypress – deterministic, repeatable, consistent test execution.
- Spec (specification) files.
- Test code structured according to Mocha framework.
- Commands – mainly concerned with querying the DOM and interacting with elements.
- Assertions/Expectations - built on Mocha and Chai libraries.