End-to-End (E2E) Testing

System testing.

System Testing

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

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

API Testing (aka Integration testing)

- **Many similarities between E2E amd API testing:**
  - **Blackbox – not looking at internals; only concern is expected output for specific inputs.**
    - **May also be interested in side-effects, e.g. database changes.**
  - **The Asynchronuous nature (for web/mobile apps).**

- **Unit and Integration test should have ironed out (most) 'low level' errors.**

E2E Testing

- **Web apps - Targeting the browser interface.**
  - **Functionality.**
  - **Form submits.**
  - **Navigation.**
  - **Flows e.g. shopping cart checkout.**

Automation Tools

- **Traditional tool suite: Mocha + Chai +** Selenium**.**
- **(Very) modern tool suite:** Cypress
  - **Uses Mocha and Chai internally.**
- **Cypress.**
  - **Win / Mac / Linux.**
  - **MIT License.**
  - **Open Source.**

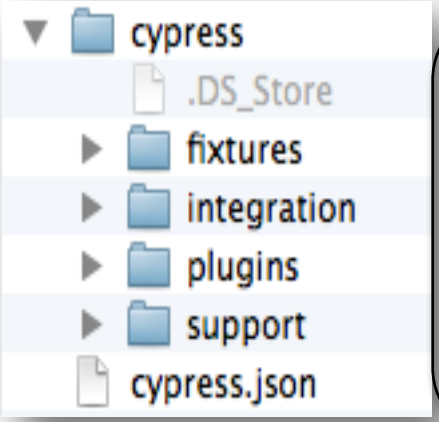
Cypress

- **Getting started:**

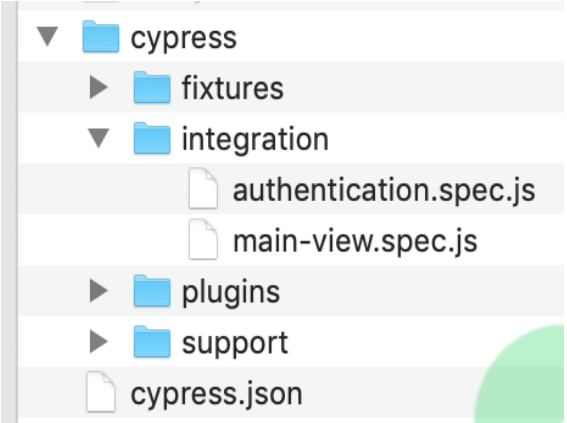
  $ npm install –save-dev cypress

  (already included in lab apps)

- **(Default) Test code folder structure:**



- **fixtures** – sample data.
- **Integration** – test code, termed specs.
- **cypress.json** – config file.

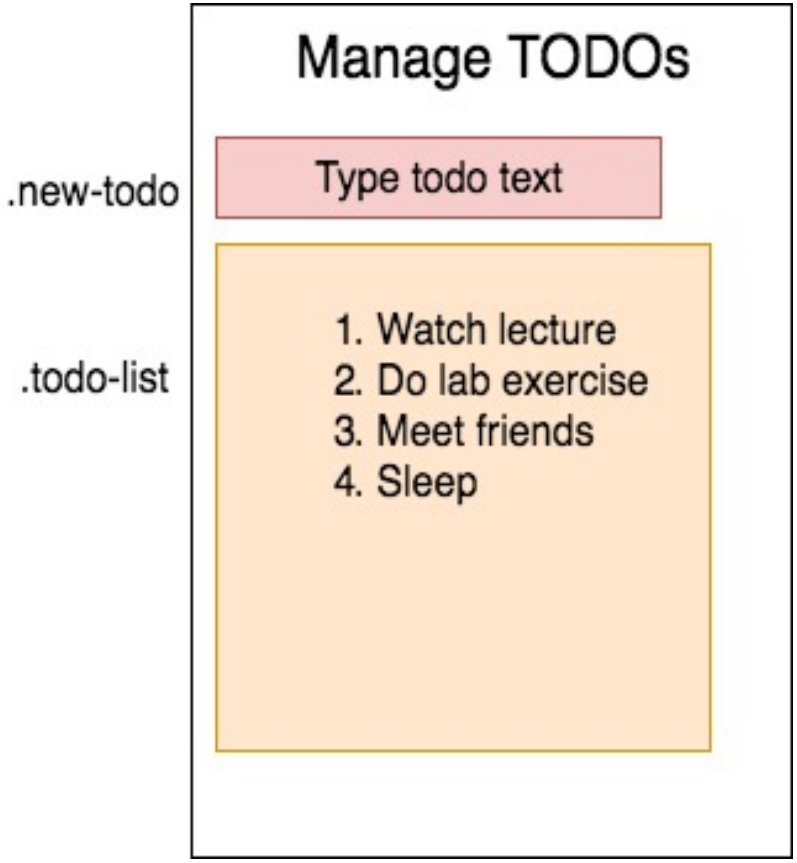


- **CLI (Command Line Interface) has 2 main commands:**

  $ npx cypress open **- GUI interactive mode (Dev)**

  $ npx cypress run **- headless mode (System Test / Continuous Integration).**

Sample test code




```
describe("TODO app", () => {
    it('adds 2 todos', () => {
      cy.visit('http://localhost:3000')
      cy.get('.new-todo')
          .type('learn React{enter}')
          .type('complete lab{enter}')
      cy.get('.todo-list li')
          .should('have.length', 2)
    })
})
```

- **Declarative style.**
- **Method Chaining style e.g.**

  cy.get(…).type(…)

Cypress statements

cy.get(. . .selector …).should(…assertion/expectation)

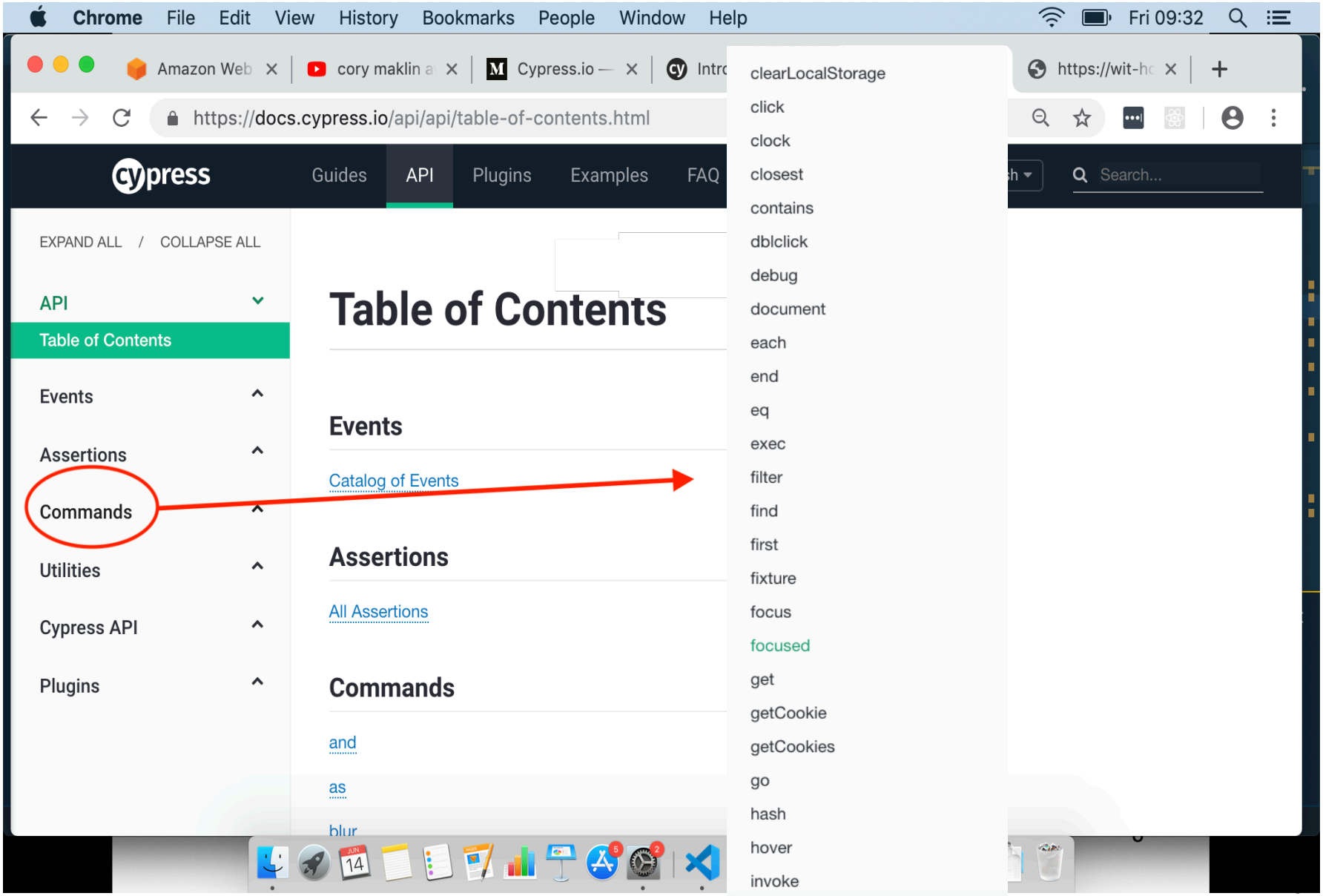
Command **Assertion** (Expectations)

- **Commands:**
  - **get() - Get one or more DOM elements by selector.**
  - **contains(text) - Get the DOM element containing the text, e.g**. cy.contains(‘Welcome’)
  - **find() - Get the descendent DOM elements.**

    e.g. cy.get('.article').find(button') // Yield the ’button' within '.article’
  - **click() – click a DOM element,** e.g. cy.get('button').click()
  - **select() - Select an <option> within a <select>.**

    e.g. cy.get('#paymenttype').select('Visa')
  - **eq() – Get a DOM element at a specific index in an array of elements,** e.g. cy.get(‘input’).eq(2).type(‘1 Main Street’)

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



Cypress statements

cy.get(. . .selector …).should(…assertion/expectation)

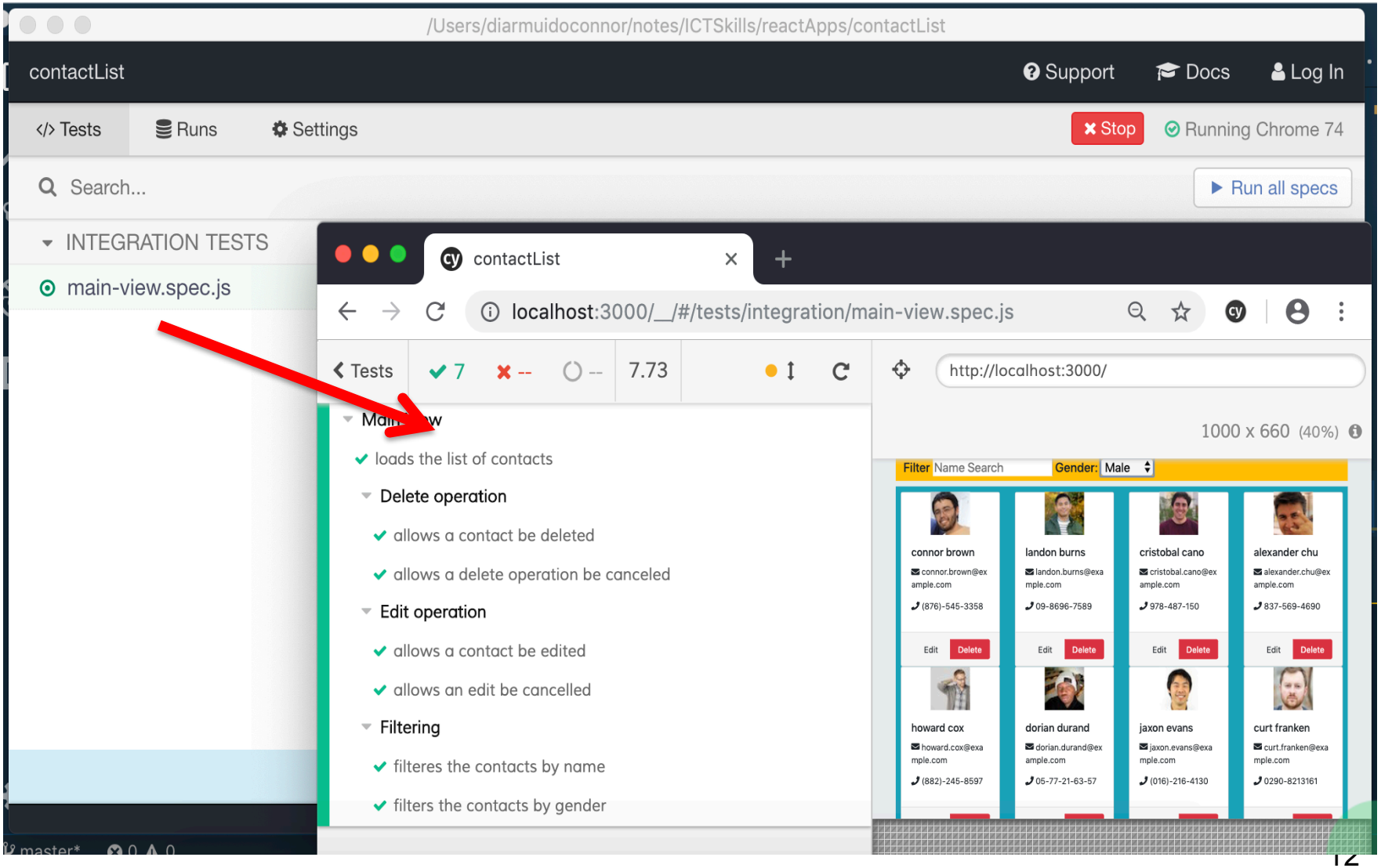
- **Selector: Based on CSS/JQuery style.**
  - **Id,** e.g. cy.get(’#heading’)
  - **CSS Class,** e.g. cy.get(’.info-message’)
  - **Tag,** e.g. cy.get(p’)
  - **Attributes,** cy.get('button[type=submit]').click()
    - **The** data-test **attribute.**
  - **nth-child, e.g. get the 8th column of the 3rd row in a table**

    cy.get('tbody').find('tr:nth-child(3)').find('td:nth-child(8)’)
  - **These can be combined, e.g.** cy.get(‘div.container’) (the div tag with CSS class .container)

Cypress Test Runner

- **Main features:**
  - **Tests run** inside the browser.
    - **Full access to browser resources – DOM, cookies, local storage.**
    - **Framework-agnostic.**
  - Flake-free **test execution.**
    - **Deterministic, repeatable, consistent execution flow.**
    - **Auto retries commands (e.g. get()) to cope with slow DOM construction.**
    - **Deals with unpredictable nature of the web.**
  - **Supports** time-travel **for convenient debugging.**

$ npx cypress open (Interactive runner)



- **Time-travel – Step through test code to track UI state.**

contactList

localhost:3000/__/#/tests/integration/main-view.spec.js

Tests ✓ 7 ✗ -- -- 7.73 http://localhost:3000/ 1000 x 660 (57%)

```
GET @cardbuttons
 -CONTAINS Edit
 -CLICK
GET @targetcard
 -FIND .card-body
 -ASSERT expected <div.card-body> to have CSS property background-color
 -ASSERT expected rgba(0, 0, 0, 0) to equal rgba(0, 0, 0, 0)
GET @targetcard
 -FIND input
 -FIRST
 -CLEAR
 -TYPE test@example.com
GET @cardbuttons
 -CONTAINS Save
 -CLICK
GET @targetcard
 -FIND [data-icon=envelope]
 -NEXT
 -ASSERT expected <span> to contain test@example.com
```

Contact List 50

Filter Name Search Gender: All

milton alexander — milton.alexander@example.com — 013873 42871 — Edit Delete

luana altun — (591)-899-5841 — Save Cancel

dwight black — dwight.black@example.com — 01180 338929 — Edit Delete

sharlene brooks — sharlene.brooks@example.com — 07-0343-7357 — Edit Delete

tammy burton — tammy.burton@ex

rosalyn coleman — rosalyn.coleman@

roméo dubois — roméo.dubois@ex

magdalena dupuis

DOM Snapshot (pinned) before after

Selectors

cy.get(. . .selector …).should(…assertion/expectation)

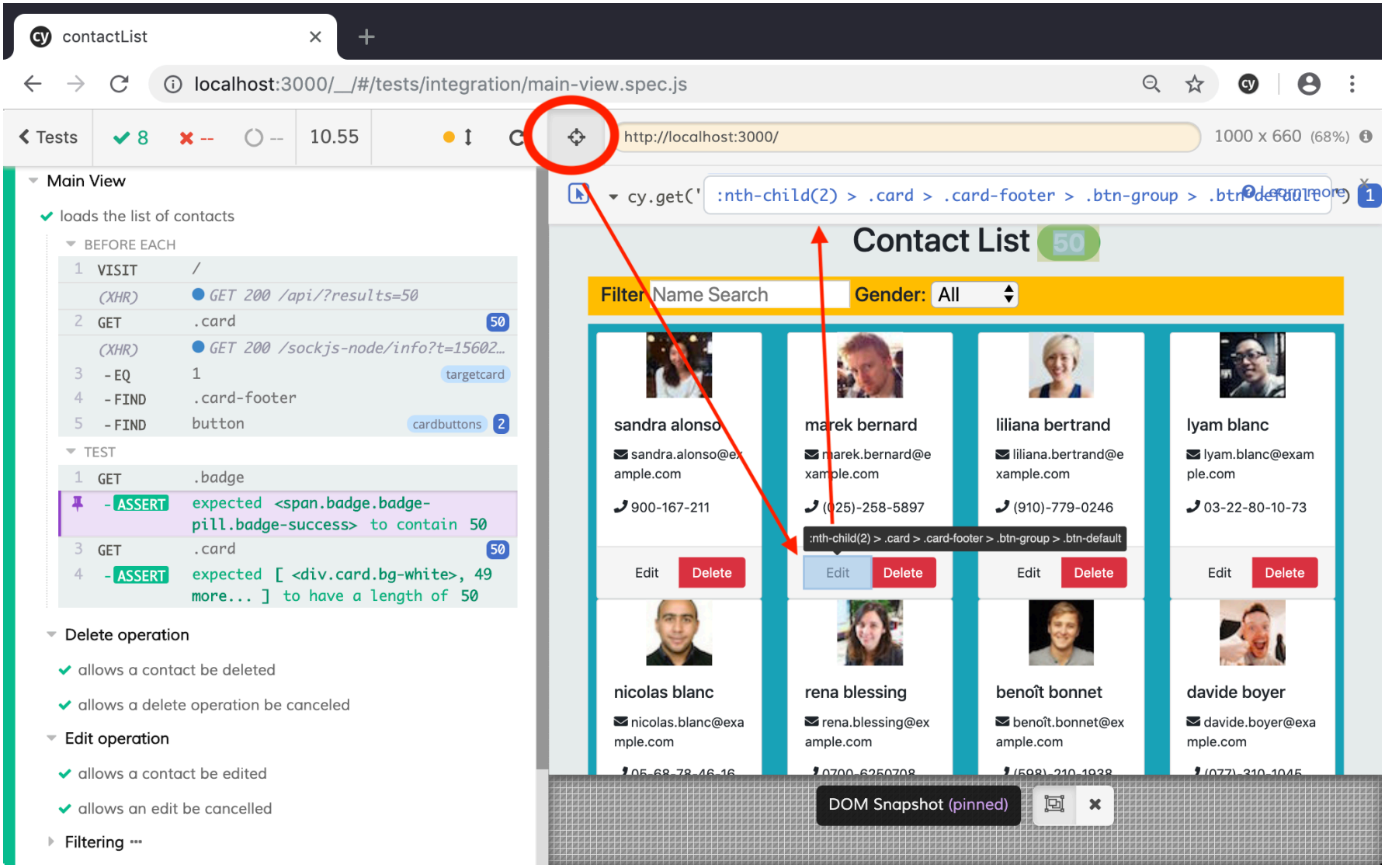
- **Have impact on test** brittleness.
  - **Small changes to CSS or HTNL structure can cause test failure.**

- **Use** data-test **attribute, where possible, to avoid brittle tests, e.g.**

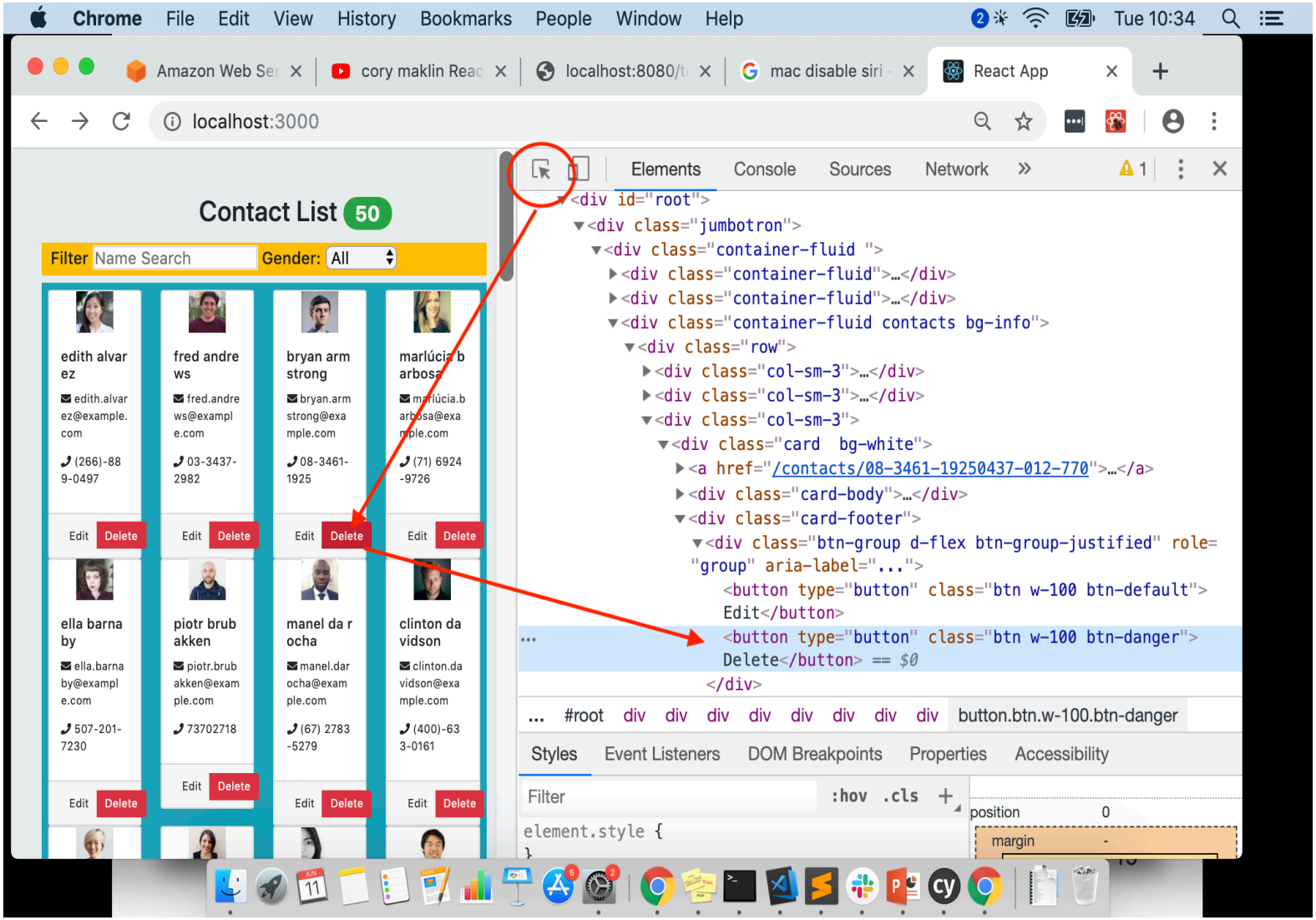
```
<input data-test=”name" type="text" className="form-control” onChange={….} />
```

```
cy.get('input[data-test=name]').type(“Joe Bloggs”)
```

- Selector Playground **– good learning aid.**



- **Use** Chrome dev tools **to also assist with choosing a selector.**



Cypress Test Runner

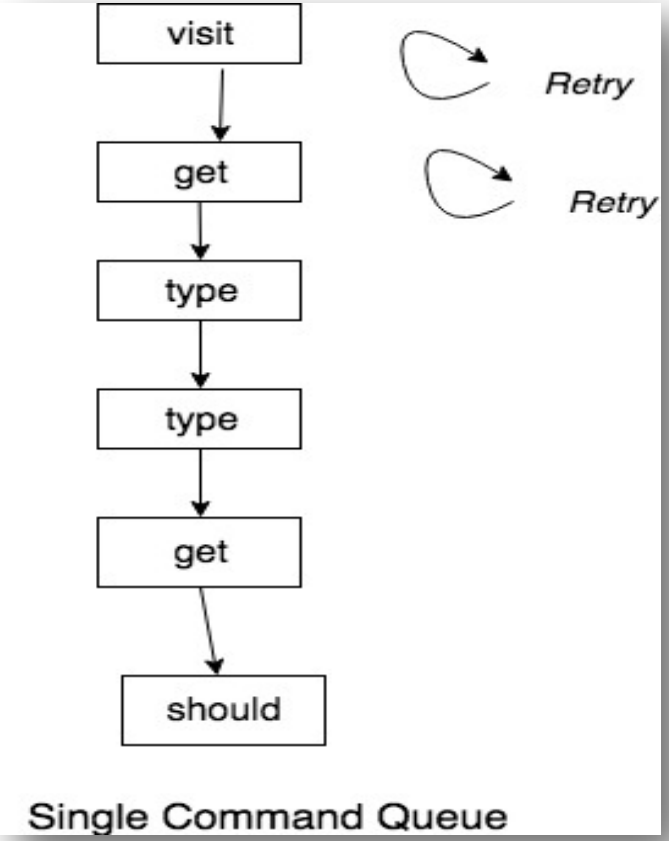
- **Headless mode:**

  $ npx cypress run

- **Runs all tests.**
- **Ideal for CI (Continuous Integration) environment.**
- **Generates video recordings (default; configurable).**
  - **.mp4 file type.**
  - **Facilitates sharing and project visibility.**
  - **Dashboard service (Publish recordings)**

Commands are enqueued

```
it('adds 2 todos', () => {
  cy.visit('http://localhost:3000')
  cy.get('.new-todo')
    .type('learn testing{enter}')
    .type(’complete lab{enter}')
  cy.get('.todo-list li')
    .should('have.length', 2)
})
```



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

Deterministic test execution

- **Cypress strives for flake-free test execution.**
- **EX.: Execution trail for above test case.**

  Visit a URL

  and wait for the page load event to fire after all external resources have loaded

  Find an element by its selector

  and retry until it is found in the DOM

  Perform a typing action on that element

  after we wait for the element to reach an actionable state.

  Grab the DOM elements by the selector and place in an array

  Assert the number of elements in the array

  and retry until it does

Command chains

- **Chain of commands.**

  cy.get('.article').eq(3).find(‘h3.title’').should(‘contain’,’Waterford’)

  – **Chain begins with** cy**.**
- **Each command yields a** subject **to the next one in the chain**

- **We can act on a command’s** subject **directly with** .then()

```
cy.get('.article').eq(2).find(‘button')
    .then ( (buttonElement) => {
        const cls = buttonElement.class()
            . . . . . . . .
    })
```

Cypress is asynchronous (kinda)

- **Cypress cannot yield you primitive values isolated away from other commands.**

```
let numBoxes = 0
// Get all the boxes. Assume their are 20
cy.get('.box').its('length’).then(len => numBoxes = len) // ****
cy.('button').click() // Add a new box
cy.get('.box').its('length').should('eq',numBoxes+1)
```

- **Instead imbed dependent command chain(s) inside then()**

```
cy.get('.box').its('length’)
    .then( numBoxes => {
        cy.('button').click()
        cy.get('.box').its('length').should('eq',numBoxes+1)
})
```

Summary

- **E2E testing – aka System testing**
- **Black-box mindset – does app produce expected output (UI) for given inputs.**
- **Cypress – deterministic, repeatable, consistent test execution.**
  - **Test code structured according to Mocha framework**
  - **Commands – mainly concern querying the DOM and interacting with elements**
  - **Assertions - built on Mocha and Chai libraries**