

# **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 and 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 Asynchronous 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

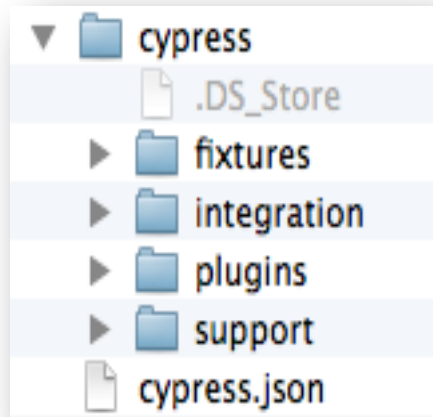
- **Traditional tool suite: Mocha + Chai + Selenium.**
- **(Very) modern tool suite: Cypress**
  - **Uses Mocha and Chai internally.**
- **Cypress.**
  - **Win / Mac / Linux.**
  - **MIT License.**
  - **Open Source.**

# Cypress

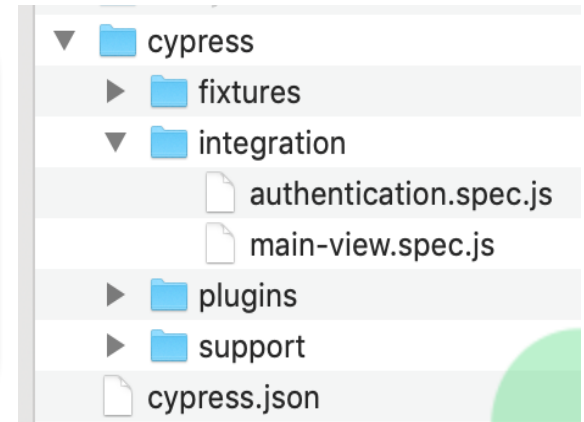
- **Getting started:**

\$ npm install --save-dev cypress  
(already included in lab apps)

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



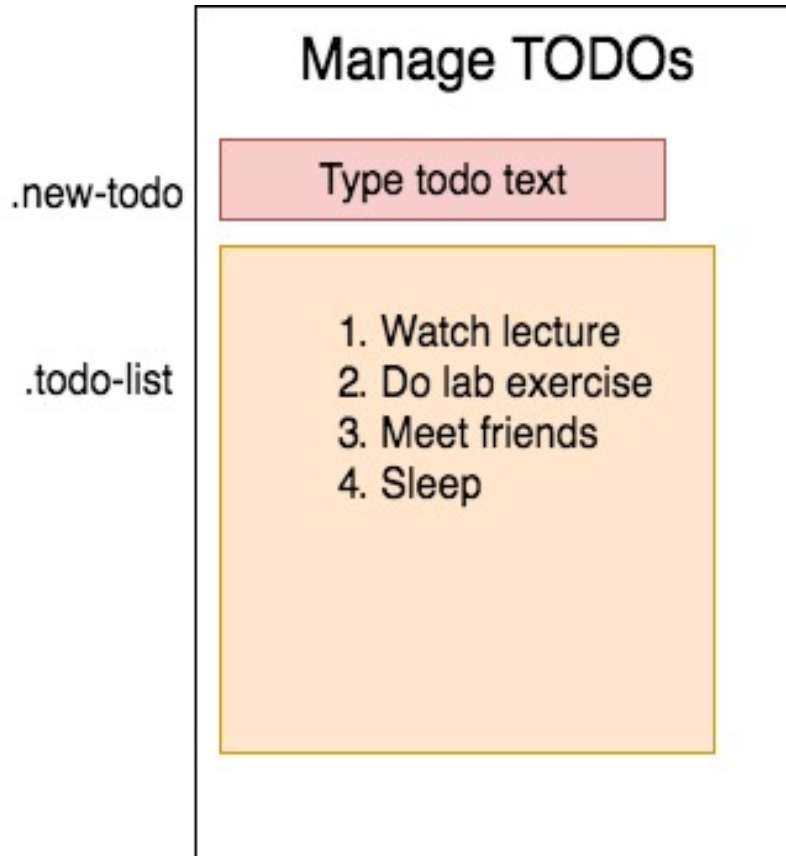
- **fixtures** – sample data.
- **Integration** – test code, termed specs.
- **cypress.json** – config file.



- **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>

The screenshot shows the Cypress.io API Table of Contents page. The left sidebar contains a list of navigation items: API, Table of Contents, Events, Assertions, **Commands** (circled in red), Utilities, Cypress API, and Plugins. The main content area displays the 'Table of Contents' with sections for Events, Assertions, and Commands. A red arrow points from the 'Commands' link in the sidebar to the 'Commands' section in the main content area. A dropdown menu is open over the 'Commands' section, listing various Cypress commands: clearLocalStorage, click, clock, closest, contains, dblclick, debug, document, each, end, eq, exec, filter, find, first, fixture, focus, **focused** (highlighted in green), get, getCookie, getCookies, go, hash, hover, and invoke.

# 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 8<sup>th</sup> column of the 3<sup>rd</sup> 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)

The screenshot displays the Cypress application interface. On the left, the file explorer shows 'main-view.spec.js' under 'INTEGRATION TESTS'. A red arrow points from this file to the 'Main view' section in the test runner. The test runner shows a list of tests with green checkmarks indicating they passed. The application under test is a contact list with a filter and search bar, and a grid of contact cards.

contactList

Support Docs Log In

Tests Runs Settings

Stop Running Chrome 74

Search...

Run all specs

main-view.spec.js

contactList

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

Tests 7 7.73

Main view

- loads the list of contacts
- Delete operation
  - allows a contact be deleted
  - allows a delete operation be canceled
- Edit operation
  - allows a contact be edited
  - allows an edit be cancelled
- Filtering
  - filters the contacts by name
  - filters the contacts by gender

1000 x 660 (40%)

Filter Name Search Gender: Male

 connor brown ✉ connor.brown@example.com ☎ (876)-545-3358	 landon burns ✉ landon.burns@example.com ☎ 09-8696-7589	 cristobal cano ✉ cristobal.cano@example.com ☎ 978-487-150	 alexander chu ✉ alexander.chu@example.com ☎ 837-569-4690
 howard cox ✉ howard.cox@example.com ☎ (882)-245-8597	 dorian durand ✉ dorian.durand@example.com ☎ 05-77-21-63-67	 jaxon evans ✉ jaxon.evans@example.com ☎ (016)-216-4130	 curt franken ✉ curt.franken@example.com ☎ 0290-8213161

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

The screenshot shows the Cypress test runner interface. The left pane displays the test code, and the right pane shows the application's UI. A red arrow points from a test step to a specific UI element.









**Test Code (Left Pane):**

```
1 GET @cardbuttons
2 -CONTAINS Edit
3 -CLICK
4 GET @targetcard
5 -FIND .card-body
6 -ASSERT expected <div.card-body> to have CSS property background-color
7 -ASSERT expected rgba(0, 0, 0, 0) to equal rgba(0, 0, 0, 0)
8 GET @targetcard
9 -FIND input
10 -FIRST
11 -CLEAR
12 -TYPE test@example.com
13 GET @cardbuttons
14 -CONTAINS Save
15 -CLICK
16 GET @targetcard
17 -FIND [data-icon=envelope]
18 -NEXT
19 -ASSERT expected <span> to contain test@example.com
```

**Application UI (Right Pane):**

**Contact List 50**

Filter: Name Search Gender: All

 milton alexander ✉ milton.alexander@example.com ☎ 013873 42871 Edit Delete	 luana altun ✉ [redacted] ☎ (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@example.com	 rosalyn coleman ✉ rosalyn.coleman@example.com	 roméo dubois ✉ roméo.dubois@example.com	 magdalena dupuis ✉ magdalena.dupuis@example.com

DOM Snapshot (pinned) before after

# Selectors

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

- **Have impact on test brittleness.**
  - **Small changes to CSS or HTML 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.

The screenshot displays the Cypress test runner interface. On the left, the 'Main View' test suite is shown with a list of tests. The first test, 'loads the list of contacts', is expanded, showing a 'BEFORE EACH' hook and a 'TEST' block. The 'TEST' block contains an assertion: `-ASSERT expected <span.badge.badge-pill.badge-success> to contain 50`. Below this, there are sections for 'Delete operation', 'Edit operation', and 'Filtering'.

On the right, a live DOM snapshot of the 'Contact List' application is shown. The application has a header with 'Contact List' and a badge with the number '50'. Below the header is a search bar and a 'Gender' dropdown. The main content area displays a grid of contact cards. Each card contains a profile picture, name, email, phone number, and 'Edit' and 'Delete' buttons. A red circle highlights the 'Selector Playground' icon in the top right corner of the browser window. A red arrow points from this icon to the 'nth-child(2)' selector in the 'cy.get()' command in the test runner. Another red arrow points from the 'nth-child(2)' selector to the second contact card in the grid, which belongs to 'marek bernard'.

The browser window shows the URL `http://localhost:3000/` and the page title 'Contact List'. The page content includes a search bar, a 'Gender' dropdown, and a grid of contact cards. The cards are for 'sandra alonso', 'marek bernard', 'liliana bertrand', 'lyam blanc', 'nicolas blanc', 'rena blessing', 'benoit bonnet', and 'davide boyer'. Each card has an 'Edit' button and a 'Delete' button.





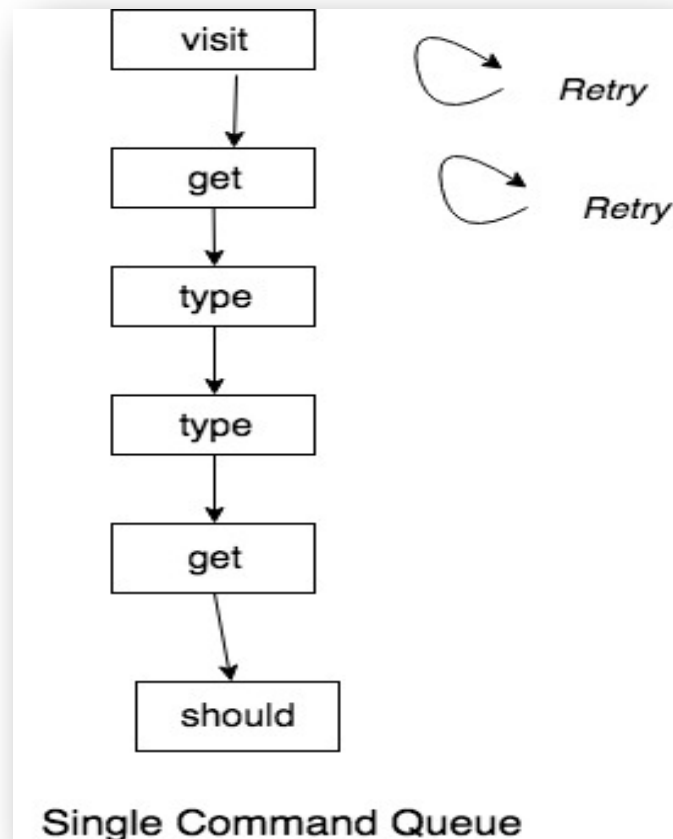
# Cypress Test Runner

- **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**

