



# 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

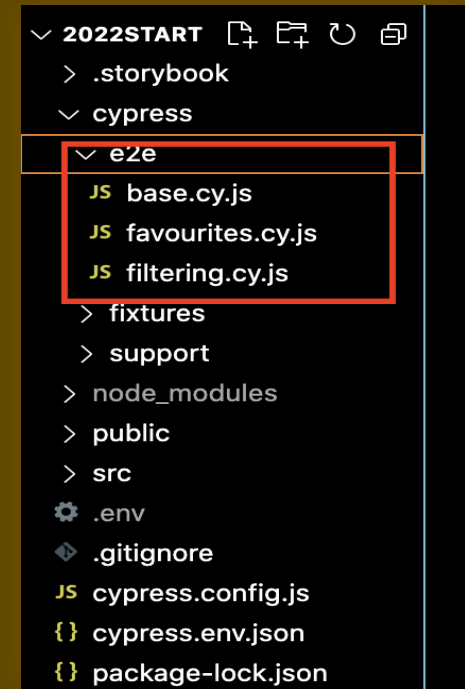
- Traditional tool suite: Mocha + Chai + Selenium.
- Modern tool suite: Cypress.
  - Uses Mocha and Chai internally.
- Cypress.
  - Win / Mac / Linux.
  - MIT License.
  - Open Source.
  - Not suitable for Visual testing; Use Percy instead.

# Cypress - Overview

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

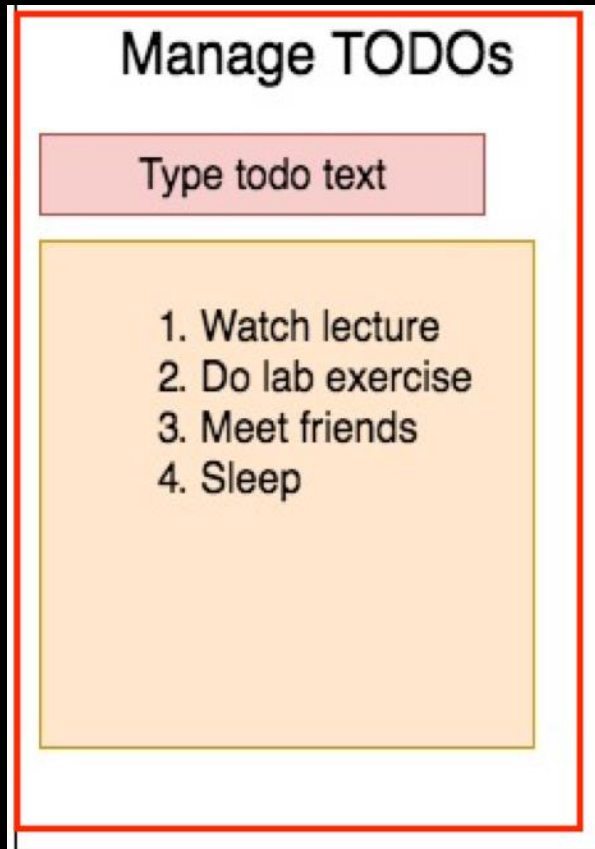


- fixtures – sample data.
- e2e – test code, termed specs.
- support – utility code.
- cypress.config.js** – config file.



- 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>

The screenshot shows a web browser window with the Cypress documentation website. The browser's address bar displays `docs.cypress.io/api/table-of-contents`. The website's header includes the Cypress logo, navigation links for App, API, Cloud, UI Coverage, Accessibility, and Learn, and a search bar. A sidebar on the left lists various command categories, with 'Commands' expanded to show a list including 'and', 'as', 'blur', 'check', 'children', 'clear', 'clearAllCookies', and 'clearAllLocalStorage'. The main content area features a large heading 'Table of Contents' followed by a sub-heading 'Commands'. The text explains that Cypress commands are asynchronous and chainable, yielding a 'subject' to the next command. It also provides a link to the 'Introduction to Chains of Commands' and lists the types of commands: 'query' (for reading state) and 'assertion' (for asserting state). A right sidebar titled 'CONTENTS' lists the site's structure: Commands, Queries, Assertions, Actions, Other, Cypress API, Events, Custom Commands and Queries, and APIs.

NEW! Improve app quality with instant insights using [Cypress Accessibility](#) or [UI Coverage](#).

**cypress docs** App API Cloud UI Coverage Accessibility Learn Search 🔍

**Table of Contents**

Commands

- and
- as
- blur
- check
- children
- clear
- clearAllCookies
- clearAllLocalStorage

## Table of Contents

### Commands

Cypress commands don't do anything at the moment they are invoked, but rather enqueue themselves to be run later. Commands can be chained together because Cypress manages a Promise chain on your behalf, with each command yielding a 'subject' to the next command, until the chain ends or an error is encountered.

Learn more about how commands chain together in the [Introduction to Chains of Commands](#).

Cypress commands can be classified one of the following

- `query` - commands that read the state of your application
- `assertion` - command which assert on a given state

**CONTENTS**

- Commands
  - Queries
  - Assertions
  - Actions
  - Other
  - Commands
- Cypress API
  - Events
  - Custom
  - Commands and Queries
  - APIs

# Cypress statements - Selectors.

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

- Selector: Based on JQuery syntax.
  - HTML Tag type: e.g. `cy.get('button')`
  - Element 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

The screenshot shows a Chrome browser window with the Cypress web interface. The browser tab is titled "ewd-2023-moviesApp-lab-se" and the address bar shows "localhost:3000/\_\_\_/#/specs". The interface includes a left sidebar with a dark blue background and a main content area. The sidebar has a "Specs" button highlighted with a red arrow. The main content area shows a list of specs under the "cyress / e2e" folder, including "base.cy.js", "favourites.cy.js", "filtering.cy.js", and "navigation.cy.js". Each spec has a timestamp of "13 hours ago".

Chrome is being controlled by automated test software.

ewd-2023-moviesApp... master

Specs

Search specs | 4 matches | + New spec

E2E specs | Component specs ?

Last updated ? | Latest runs ? | Average duration ?

cyress / e2e

base.cy.js	13 hours ago	--	--
favourites.cy.js	13 hours ago	--	--
filtering.cy.js	13 hours ago	--	--
navigation.cy.js	13 hours ago	--	--

# Test Execution (Green bar / Red Bar)

The screenshot shows a web browser window with the address bar displaying `localhost:3000/_/#/specs/runner?file=cypress/e2e/base.cy.js`. The browser's status bar at the bottom indicates `localhost:3000/_/#/specs/runner?file=cypress/e2e/base.cy.js`.

On the left side of the browser window, the Cypress test runner interface is visible. It includes a sidebar with a search bar labeled "Search specs" and a list of spec files under the folder `cypress / e2e`: `base.cy.js`, `favourites.cy.js`, `filtering.cy.js`, and `navigation.cy.js`. The `base.cy.js` file is selected, showing a list of tests under the heading "Base tests". The tests are:

- ✓ The Discover Movies page
  - ✓ displays the page header and 20 movies
  - ✓ displays the correct movie titles
- ✓ The movie details page
  - ✓ displays the movie title, overview and genres and

The right side of the browser window displays a movie page titled "Venom: The Last Dance" with the tagline "'Til death do they part." The page includes an overview section with the text: "Eddie and Venom are on the run. Hunted by both of their worlds and with the net closing in, the duo are forced into a devastating decision that will bring the curtains down on Venom and Eddie's last dance." Below the overview, there are tabs for "Genres" (Science Fiction, Action, Adventure) and a row of statistics: "109 min.", "438,325,476", "6.507 (814)", and "Released: 2024-10-22".

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

The screenshot shows a web browser window with the address bar displaying `localhost:3000/_/#/specs/runner?file=cypress/e2e/favourites.cy.js`. The browser interface includes a menu bar (File, Edit, View, History, Bookmarks, Profiles, Tab, Window, Help) and a status bar (Sat 23 Nov 23:39). A notification bar states "Chrome is being controlled by automated test software."

On the left, the Cypress test runner interface is visible. It shows a search bar, a list of specs, and a detailed view of the `favourites.cy.js` spec. The spec contains the following code:

```
200
https://api.themoviedb.org/3/discover/movie?
api_key=c12756437
8815b0e7fa59f4c35
1e178a&language=en-
US&include_adult=
false&include_video=
false&page=1

2 eq 1
3 -click
4 get button[aria-label='add to favorites']
5 eq 3
6 -click
7 get button
8 -contains Favorites
9 -click
```

On the right, the TMDB Client application is displayed. It features a header with the title "TMDB Client" and navigation links: HOME, UPCOMING, FAVORITES, OPTION 3, and OPTION 4. The main content area shows a grid of movie cards. The first card is for "VENOM THE LAST DANCE" (2024-10-22, 6.504 stars). The second card is for "Tomb Raider" (2024-09-12, 8.472 stars). The third card is for "TERRIFIER 3" (2024-10-09, 6.932 stars). Each card has a "MORE INFO ..." button. A red arrow points from the `-click` command in the test runner to the "MORE INFO ..." button on the "Tomb Raider" card.



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

TMDB Client All you ever wanted to know about

Discover

Venom: The Last Dance

Terrifier 3

The Wild Robot

Gladiator II

Elements

```
<div class="MuiGrid-root MuiGrid-item MuiGrid-grid-xs-12 MuiGrid-grid-sm-6 MuiGrid-grid-md-4 MuiGrid-grid-lg-3 MuiGrid-grid-xl-2 css-1mcvwu6-MuiGrid-root">
  <div class="MuiPaper-root MuiPaper-elevation MuiPaper-rounded MuiPaper-elevation1 MuiCard-root css-1ri6ub7-MuiPaper-root-MuiCard-root">
    <div class="MuiCardHeader-root css-185gdzj-MuiCardHeader-root">
      flex
      <div class="MuiCardHeader-content css-1qbke1o-MuiCardHeader-content">
        <p class="MuiTypography-root MuiTypography-h5 css-ag7rrr-MuiTypography-root">The Wild Robot</p> == $0
      </div>
    <div class="MuiCardMedia-root css-1me7nzk-MuiCardMedia-root" role="img" style="background-image: url('https://image.tmdb.org/t/p/w500/0//wTnV3PCVW5092JMrFvvrRcV39RU.jpg');"></div>
    <div class="MuiCardContent-root css-46bh2p-MuiCardContent-root">
    <div class="MuiCardActions-root css-1rwjz6-MuiCardActions-root">
```

cy.get(".MuiCardHeadercontent").eq(2).find('p')

```
.css-ag7rrr-MuiTypography-root {
  margin: 0;
  font-family: "Roboto", "Helvetica", "Arial", sans-serif;
  font-weight: 400;
  font-size: 1.5rem;
  line-height: 1.334;
  letter-spacing: 0em;
}
```

user agent stylesheet

p {
 display: block;



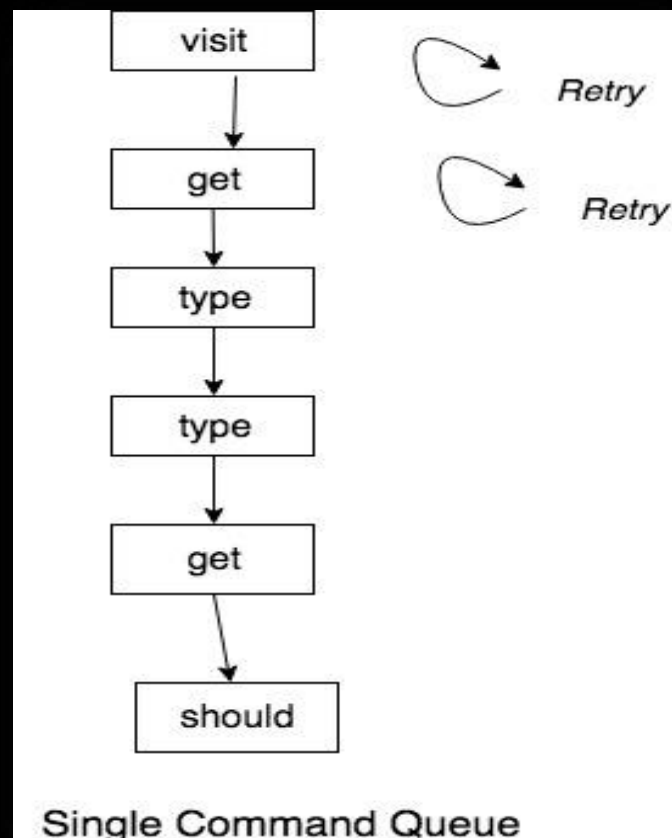
# Headless Test Runner

- Running Cypress tests without a UI.
- Headless mode:
  - \$ npx cypress run
- Ideal for CI (Continuous Integration) environment.
- Can generate 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.