



Fast, easy and reliable testing for  
anything that runs in a browser



# Content

Overview of the benefits of using Cypress, comparison with Selenium-based solutions and a short demo / live-coding session.

What is Cypress

Main Features

Comparison

Structure &  
Performance

Demo

# Terminology

- **End-to-end testing** = testing general user flows
- **Assertions** = expression at a specific point in a program which will be true unless there is a bug in the program
- **Spy / Mock / Stub** = fake methods / objects used for testing purposes

# Context

- Focus on end-to-end testing
  - End-to-end testing coverage is perceived as being a slow and time consuming process
  - Expensive to maintain tests suite
- Sync is often required between development / testing
  - E2E requires complex infrastructure to be added in CI




# What is Cypress?

Functional automation test runner  
for web applications

Launched in 2017, has a growing  
community and reached a mature state



# What is Cypress?

- Made for developers & QA engineers (easy to write, test, debug)
- Not based on Selenium (new architecture)
- All-in-one testing framework  
- Open source 

# Main Features

- GUI Test Runner

The screenshot shows the GUI Test Runner interface with several key features highlighted by orange arrows and labels:

- Test Status Menu**: Located at the top left, it displays test results: 4 passed (green checkmarks), 2 failed (red Xs), 0 pending (circles), and a total time of 3.94 seconds.
- Url Preview**: Located at the top center, it shows the current URL being tested: `http://localhost:8888/#/`.
- Viewport Sizing**: Located at the top right, it shows the current viewport size: 625 x 750 (97%).
- Command Log**: Located on the left side, it shows a list of test commands and their results. The log includes a 'BEFORE EACH' section with a 'VISIT' command, a 'TEST' section with 'GET' and 'ASSERT' commands, and a summary of test results at the bottom.
- App preview**: Located on the right side, it shows a real-time preview of the application being tested. The preview displays a 'todos' app with a list of items: 'buy some cheese' and 'feed the cat'. The app also shows a 'What needs to be done?' section and a 'Clear completed' button.

Annotations with orange arrows point from the labels to the corresponding features in the interface:

- An arrow points from **Test Status Menu** to the top left status bar.
- An arrow points from **Url Preview** to the URL input field.
- An arrow points from **Viewport Sizing** to the viewport size indicator.
- An arrow points from **Command Log** to the left sidebar log area.
- An arrow points from **App preview** to the right sidebar app preview area.

# Main Features

- GUI Test Runner
  - Real-time reload of test changes
  - Speeds up development
  - Easy to debug
  - Saves snapshots of each command / assert

## Test Status Menu

See how many tests passed or failed and how fast the tests took to run.

## Url Preview

The url of your app updates as you have more visibility on test

The screenshot shows a web browser window with a dark header. On the left, a sidebar displays test results for 'TodoMVC'. The main content area shows a 'New Todo' form with a list of items: 'buy some cheese' and 'feed the cat'. The URL bar shows 'http://localhost:8888/#/'.

**Test Status Menu** (indicated by an orange arrow pointing to the top left of the browser window):

- 4 green checkmarks (passed)
- 2 red X's (failed)
- 0 white circles (pending)
- 3.94 (time taken)

**Url Preview** (indicated by an orange arrow pointing to the address bar):

http://localhost:8888/#/

**Test Results** (indicated by an orange arrow pointing to the sidebar):

- ✓ should allow me to add a todo item
  - BEFORE EACH
    - 1 VISIT http://localhost:8888
    - (NEW URL) http://localhost:8888/#/
  - TEST
    - 1 GET .new-todo
    - 2 - TYPE buy some cheese
    - 3 - TYPE {enter}
    - 4 GET .todo-list li
    - 5 - FIRST
    - 6 - FIND label
    - 7 - ASSERT expected: <label> to contain: buy some cheese
- ✓ should clear text input field when an item i...
- ✓ should append new items to the bottom of...

## Command Log

Every command logs as it executes. Hover to highlight the affected element, or click to log more info to the console.

## App p

See your  
Use your



# Main Features

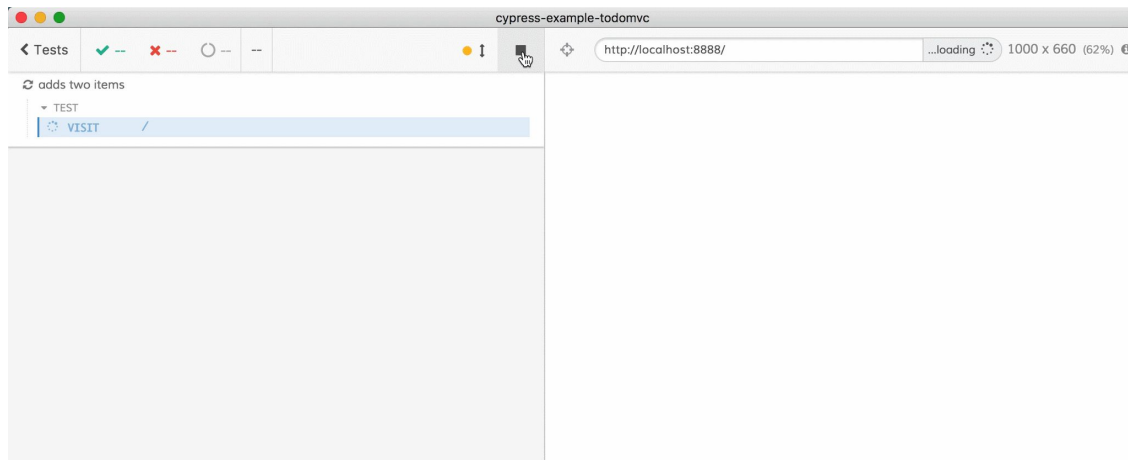
- CLI test runner
  - Useful for easy CI setup
  - Runs test in headless browser (e.g. Electron, Chrome)
  - View screenshots taken automatically on failure, or videos of your entire test suite when run from the CLI
- Test parallelization

(Run Finished)

Spec	Tests	Passing	Failing	Pending	Skipped
✓ examples\actions.spec.js	00:16	14	14	-	-
✓ examples\aliasing.spec.js	00:02	2	2	-	-
✓ examples\app.spec.js	00:02	1	1	-	-
✓ examples\assertions.spec.js	00:04	9	9	-	-
✓ examples\connectors.spec.js	00:02	8	8	-	-
✓ examples\cookies.spec.js	00:02	5	5	-	-
✓ examples\cypress_api.spec.js	00:03	12	12	-	-
✓ examples\files.spec.js	00:03	4	4	-	-
✓ examples\form.spec.js	00:05	15	15	-	-
✓ examples\local_storage.spec.js	00:01	1	1	-	-
✓ examples\location.spec.js	00:01	3	3	-	-
✓ examples\misc.spec.js	00:04	6	6	-	-
✓ examples\navigation.spec.js	00:03	3	3	-	-
✓ examples\network_requests.spec.js	00:08	6	6	-	-
✓ examples\querying.spec.js	00:03	5	5	-	-
✓ examples\spies_stubs_clocks.spec.js	00:04	7	7	-	-
✓ examples\traversal.spec.js	00:04	18	18	-	-
✓ examples\utilities.spec.js	00:03	5	5	-	-
✓ examples\viewport.spec.js	00:04	1	1	-	-
✓ examples\waiting.spec.js	00:06	2	2	-	-
✓ examples>window.spec.js	00:01	3	3	-	-
✓ All specs passed!	01:33	130	130	-	-

# Main Features

- Automatic waiting
  - Automatically waits for commands and assertions before moving on
  - Doesn't need specific timeouts / wait statements in tests
  - Built internally by using a retry-ability mechanism (4s default)



# Main Features

- Time travel & Debugging
  - Cypress takes snapshots as your tests run
  - Can navigate at any command / assertion from any test from the suite
  - Readable errors and stack traces make debugging easier

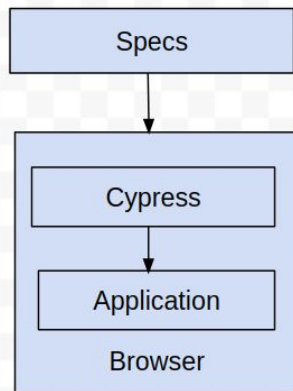
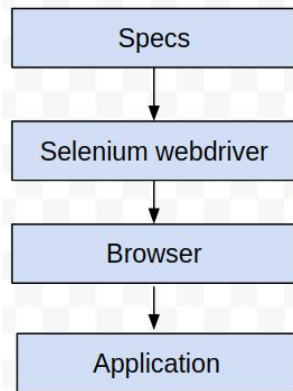
# Main Features

- **Native access to application and network traffic control**
  - Can mock network requests
  - Verify and control the behavior of functions, server responses, or timers using spies, stubs or clocks
  - **Can take shortcuts programmatically** (e.g. login) while maintaining tests sandboxing
  - Emulates same-origin policies to meet web security checks (e.g. CORS)

# Comparison



# Architecture



# Cypress

- Easy setup / easy to write / easy debug
- Native access to application (Node.js process)
- All-in-one framework
- Growing Community

# Selenium

- Complex setup / more difficult to write & maintain
- Application access through WebDriver
- Requires testing framework, assertion library
- Established community

# Cypress

- Supports only Javascript language
- No support for multiple browser testing
- Limited browser support (Chrome, Firefox, Edge, Electron)

# Selenium

- Supports all popular languages (e.g. Java, Python)
- Supports multiple browser testing
- Supports all browsers
- Supports multiple testing frameworks

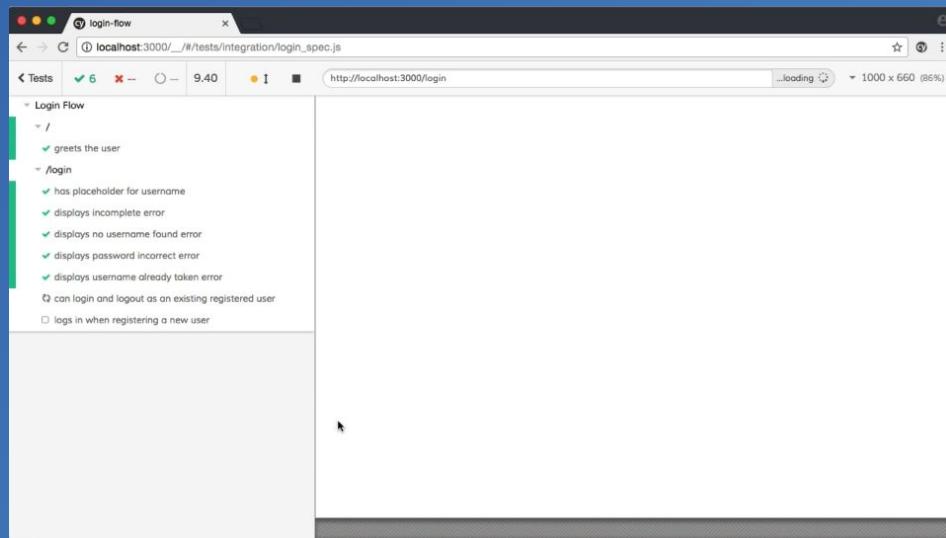


# To summarize

- Cypress has a completely different architecture
- Has native access to everything (runs inside the browser), including network requests & can interact directly with application state
- Can take shortcuts programmatically
- Not flaky (automatic waiting)
- It's fast (since it runs natively)
- Very easy to debug

# Structure & Performance

A few words about how to write tests



# Project Structure

- Default folder structure
  - **Fixtures:** test demo data
  - **Screenshots / Video:** test recordings
  - **Integration:** test suites
  - **Plugin:** custom scripts to be run on the Node.js process level
  - **Support:** common logic reusable by all test suites

```
/cypress
  /fixtures
    - example.json

  /integration
    /examples
      - actions.spec.js
      - aliasing.spec.js
      - assertions.spec.js
      - connectors.spec.js
      - cookies.spec.js
      - cypress_api.spec.js
      - files.spec.js
      - local_storage.spec.js
      - location.spec.js
      - misc.spec.js
      - navigation.spec.js
      - network_requests.spec.js
      - querying.spec.js
      - spies_stubs_clocks.spec.js
      - traversal.spec.js
      - utilities.spec.js
      - viewport.spec.js
      - waiting.spec.js
      - window.spec.js

  /plugins
    - index.js

  /support
    - commands.js
    - index.js
```

# Test Structure

- Flexible syntax, built on top of Mocha and Chai
- The test interface
  - `describe()` or `context()`
  - `it()` or `specify()`
  - `beforeEach()`, `beforeAll()`
  - `afterEach()`, `afterAll()`

```
// -- Start: Our Application Code --  
function add(a, b) {  
  return a + b  
}  
  
function subtract(a, b) {  
  return a - b  
}  
  
function divide(a, b) {  
  return a / b  
}  
  
function multiply(a, b) {  
  return a * b  
}  
// -- End: Our Application Code --  
  
// -- Start: Our Cypress Tests --  
describe('Unit test our math functions', () => {  
  context('math', () => {  
    it('can add numbers', () => {  
      expect(add(1, 2)).to.eq(3)  
    })  
  
    it('can subtract numbers', () => {  
      expect(subtract(5, 12)).to.eq(-7)  
    })  
  
    specify('can divide numbers', () => {  
      expect(divide(27, 9)).to.eq(3)  
    })  
  
    specify('can multiply numbers', () => {  
      expect(multiply(5, 4)).to.eq(20)  
    })  
  })  
})  
// -- End: Our Cypress Tests --
```

# Performance & Best practices

- Programmatic login to drastically reduce tests time
- Tests should run independently
- Re-use generic tests
- Avoid writing 'tiny' tests

```
describe('my form', () => {  
  before(() => {  
    cy.visit('/users/new')  
    cy.get('#first').type('johnny')  
  })  
  
  it('has validation attr', () => {  
    cy.get('#first').should('have.attr', 'data-validation', 'required')  
  })  
  
  it('has active class', () => {  
    cy.get('#first').should('have.class', 'active')  
  })  
  
  it('has formatted first name', () => {  
    cy.get('#first').should('have.value', 'Johnny') // capitalized first  
  })  
})
```

```
describe('my form', () => {  
  before(() => {  
    cy.visit('/users/new')  
  })  
  
  it('validates and formats first name', () => {  
    cy.get('#first')  
      .type('johnny')  
      .should('have.attr', 'data-validation', 'required')  
      .and('have.class', 'active')  
      .and('have.value', 'Johnny')  
  })  
})
```

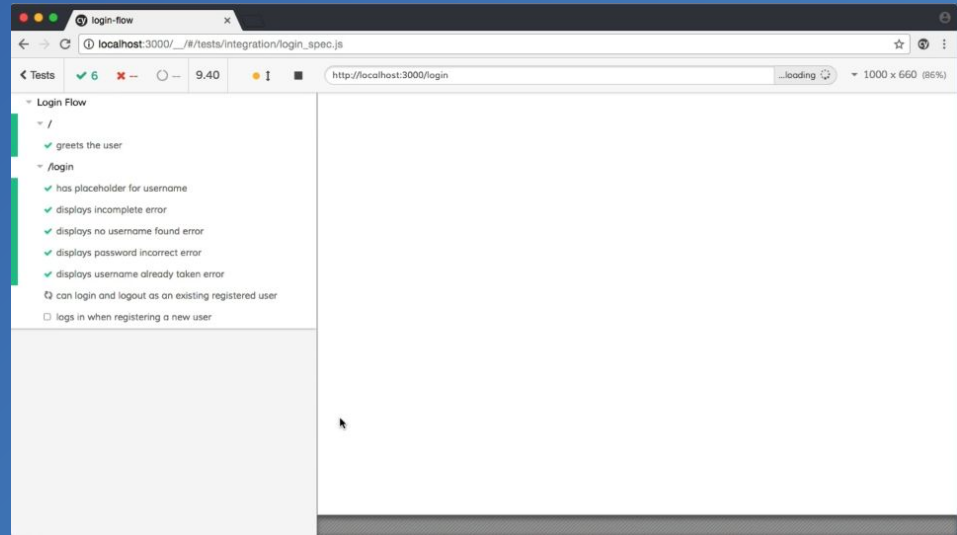
# Performance & Best practices

- Use Cypress specific selectors to avoid flaky tests
- Cypress wraps all DOM queries with robust retry-and-timeout logic

Selector	Recommended	Notes
<code>cy.get('button').click()</code>	⚠ Never	Worst - too generic, no context.
<code>cy.get('.btn.btn-large').click()</code>	⚠ Never	Bad. Coupled to styling. Highly subject to change.
<code>cy.get('#main').click()</code>	⚠ Sparingly	Better. But still coupled to styling or JS event listeners.
<code>cy.get('[name=submission]').click()</code>	⚠ Sparingly	Coupled to the name attribute which has HTML semantics.
<code>cy.contains('Submit').click()</code>	✅ Depends	Much better. But still coupled to text content that may change.
<code>cy.get('[data-cy=submit]').click()</code>	✅ Always	Best. Isolated from all changes.

# Demo

Live test run



**Thank you!**



# Bibliography

- <https://dzone.com/articles/why-should-you-switch-to-cypress-for-modern-web-te>
- <https://blog.logrocket.com/cypress-io-the-selenium-killer/>
- <https://www.valentinog.com/blog/cypress/>
- <https://www.cypress.io/how-it-works>
- <https://medium.com/@klmlfl/cypress-io-a-hands-on-overview-ad4d498944ee>
- <https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-selenium-test-automation-tool/>
- <https://www.browserstack.com/guide/cypress-vs-selenium>
- <https://www.cuelogic.com/blog/cypress-vs-selenium>
- <https://medium.com/@v.sivaa/why-cypress-and-why-is-it-so-famous-4bf98e61608>
- <https://docs.cypress.io/guides/getting-started/installing-cypress>