← Advanced API Testing

Advanced Cypress & API Testing →

Cypress Fundamentals

Table of Contents

1. Cypress Architecture and Workflow
2. Cypress vs Playwright Comparison
3. Project Setup and Configuration
4. Core Cypress Commands
5. Element Selection and Locators
6. Actions and Interactions
7. Assertions and Expectations
8. Test Structure and Hooks
9. Fixtures and Data-Driven Testing
10. Debugging with Cypress Test Runner
11. Practice Exercises

Cypress Architecture and Workflow

What is Cypress?

Key Differentiators:

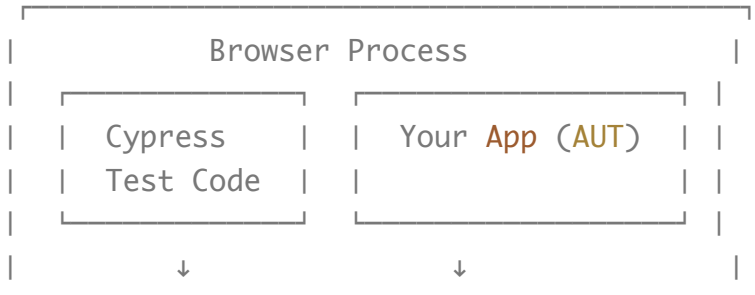
- Runs in the same run-loop as your application
- Real-time reloads during development
- Automatic waiting and retry mechanisms
- Network traffic control at the proxy level
- Time travel debugging with snapshots
- Automatic screenshots and video recording
- Intuitive, developer-friendly API
- No flaky tests due to async issues

Cypress Architecture Deep Dive

Traditional Selenium Architecture:


```
Problems:
- Network latency
- Synchronization issues
- Flaky tests
- Complex setup
```

Cypress Architecture:



```
Benefits:
- Direct DOM access
- No network lag
- Synchronous code understanding
- Reliable and fast
```

Cypress Node Process:

```
┌─────────────────────────────┐
│   Node.js Process           │
│  - File system access       │
│  - Network proxy            │
│  - Task execution           │
│  - Plugin system            │
└─────────────────────────────┘
          ↕ (IPC)
┌─────────────────────────────┐
│   Browser Process           │
│  - Test execution           │
│  - App under test           │
└─────────────────────────────┘
```

Command Queue System

Cypress commands are **not promises** - they are enqueued and executed asynchronously:

```
// ❌ This doesn't work like you think
const button = cy.get('button')  // Returns Cypress chainable, not element
button.click()  // Error!

// ✅ Correct - chain commands
cy.get('button').click()

// ❌ This won't work as expected
cy.get('button')
const text = cy.get('button').text()  // text is not a string!
console.log(text)  // Undefined

// ✅ Correct - use .then()
cy.get('button').then($btn => {
```

Understanding the Queue:

```
 console.log('1')        // Executes immediately

cy.visit('/page')       // Queued (1st)
console.log('2')        // Executes immediately

cy.get('button')        // Queued (2nd)
console.log('3')        // Executes immediately

// Output: 1, 2, 3, then visit, then get

// Execution order:
// 1. Synchronous JS (console.log)
// 2. Cypress commands (cy.visit, cy.get)
```

Command Chaining:

```
cy.get('button')
  .click()                      // Command
  .should('have.class', 'active')  // Assertion
  .and('be.visible')            // Assertion
  .then($button => {             // Access jQuery element
    // Work with element
  })
```

Automatic Waiting and Retry

Cypress automatically waits and retries commands until they pass or timeout:

What Cypress Waits For:

1. Element to exist in DOM
2. Element to be visible
3. Element not to be disabled
4. Element not to be covered

```
// Cypress automatically waits up to 4 seconds (default)
cy.get('.async-button').click()

// No need for explicit waits!
// ❌ Don't do this
cy.wait(5000)
cy.get('button').click()

// ✅ Do this - Cypress handles it
cy.get('button', { timeout: 10000 }).click()
```

Retry-ability:

```
// This will retry until button has text "Loaded" or timeout
cy.get('button').should('have.text', 'Loaded')

// Cypress keeps retrying:
// Attempt 1: button text = "Loading..."
// Attempt 2: button text = "Loading..."
// Attempt 3: button text = "Loaded" ✓ (passes!)
```

Interactive Debugging

Cypress Test Runner provides excellent debugging capabilities:

Features:

- Interactive test execution
- Command log with details
- DOM snapshots at each step
- Click commands to see state
- Console logs and errors
- Network requests visible
- Automatic screenshots on failure

Debug workflow:

3. Click any command
4. Inspect DOM state
5. See what happened
6. Fix and re-run

Network Layer Control

Cypress runs a **network proxy** to control all network traffic:

Capabilities:

- Intercept HTTP requests
- Stub responses
- Spy on API calls
- Modify requests/responses
- Simulate network conditions
- Control timing

```
 // Intercept and stub
cy.intercept('GET', '/api/users', { fixture: 'users.json' })

// Intercept and spy
cy.intercept('POST', '/api/login').as('loginRequest')
cy.get('button').click()
cy.wait('@loginRequest')
```

Cypress vs Playwright Comparison

Detailed Comparison

Created By	Cypress.io (Independent company)	Microsoft (Open Source)
Open Source	✅ Yes (MIT License)	✅ Yes (Apache 2.0)
Languages	JavaScript, TypeScript only	JS, TS, Python, Java, .NET
Browsers	Chrome, Firefox, Edge	Chrome, Firefox, Safari, Edge
Architecture	Runs in browser	Runs outside browser
Speed	Very fast (in-browser)	Very fast (protocol-based)
Multiple Tabs	Limited support (v12.4+)	✅ Full native support
Multiple Domains	Limited (cy.origin() v9.6+)	✅ Full native support
iFrames	Requires cy.frameLoaded()	✅ Simple (locator.frameLocator())
Shadow DOM	✅ Native (v5.2+)	✅ Native
Automatic Waiting	✅ Built-in	✅ Built-in
Test Runner UI	✅ Excellent interactive UI	✅ UI Mode (--ui flag) since v1.32
Video Recording	✅ Built-in	✅ Built-in
Screenshots	✅ Automatic on failure	✅ Automatic on failure
Parallel Execution	⚠️ Free locally, paid in cloud	✅ Free everywhere (--workers)

CI/CD	✅ Excellent	✅ Excellent
Learning Curve	Moderate	Moderate
Community	✅ Large, mature (since 2015)	✅ Large, mature (Microsoft-backed)
API Testing	✅ cy.request()	✅ Full APIRequestContext
Pricing Model	Free OSS + Paid Cloud features	✅ Completely free

Pricing Comparison

Cypress:

- ✅ **Free Open Source**: Test runner, all testing features
- 💰 **Cypress Cloud (Paid)**:
  - Parallel execution in cloud
  - Test analytics dashboard
  - Test recording & debugging
  - Flaky test detection
  - GitHub/GitLab integration
  - Starting at $75/month

Playwright:

- ✅ **Completely Free**: Everything included
- ✅ No paid tiers
- ✅ Parallel execution free
- ✅ All features free
- ✅ Microsoft-backed open source

When to Choose Cypress

- Team is JavaScript/TypeScript focused
- Need excellent interactive debugging
- Chrome, Firefox, Edge browsers are sufficient
- Want mature ecosystem with many plugins
- Prefer in-browser test execution

Cypress Strengths:

- ✅ Excellent interactive debugging
- ✅ Real-time test reloading during development
- ✅ Large ecosystem and community (since 2015)
- ✅ Better documentation and tutorials
- ✅ Great developer experience
- ✅ Easy to get started
- ✅ Strong for SPAs

Cypress Limitations:

- ⚠️ Limited multi-tab support
- ⚠️ Cross-domain requires workarounds (cy.origin)
- ⚠️ Paid cloud for parallel execution
- ⚠️ No Safari on Windows/Linux
- ⚠️ JavaScript/TypeScript only

When to Choose Playwright

Choose Playwright When:

- Need cross-browser including Safari
- Team uses multiple languages
- Need free parallel execution everywhere
- Working with multi-tab scenarios

- Want completely free solution

Playwright Strengths:

- ✅ True cross-browser (Safari on all OS)
- ✅ Completely free (all features)
- ✅ Multiple language support
- ✅ Native multi-tab and multi-domain
- ✅ Better cross-domain support
- ✅ Better mobile device emulation
- ✅ Microsoft backing

Playwright Limitations:

- ⚠️ Newer (less community content)
- ⚠️ Steeper initial setup for some

Feature Evolution

Recent Cypress Improvements:

- v12.4+: Better multi-tab support
- v9.6+: cy.origin() for cross-domain
- v5.2+: Native Shadow DOM support
- Continuous improvements to match Playwright

Recent Playwright Improvements:

- v1.32+: UI Mode for interactive testing
- Trace Viewer for debugging
- Component testing
- Better accessibility testing
- Continuous rapid development

```
// Selenium
driver.findElement(By.id('username')).sendKeys('admin')
driver.findElement(By.id('password')).sendKeys('pass123')
driver.findElement(By.id('login')).click()
WebDriverWait wait = new WebDriverWait(driver, 10)
wait.until(ExpectedConditions.urlContains('/dashboard'))

// Cypress
cy.get('#username').type('admin')
cy.get('#password').type('pass123')
cy.get('#login').click()
cy.url().should('include', '/dashboard')
```

From Playwright to Cypress:

```
// Playwright
await page.goto('https://example.com')
await page.locator('#username').fill('admin')
await page.locator('#password').fill('pass123')
await page.locator('button[type="submit"]').click()
await expect(page).toHaveURL(/.*dashboard/)

// Cypress
cy.visit('https://example.com')
cy.get('#username').type('admin')
cy.get('#password').type('pass123')
cy.get('button[type="submit"]').click()
cy.url().should('match', /.*dashboard/)
```

Project Setup and Configuration

Installation

Install Cypress:

```
# Using yarn
yarn add --dev cypress

# Using pnpm
pnpm add -D cypress
```

Open Cypress:

```
 # Interactive mode (Test Runner)
npx cypress open

# Headless mode
npx cypress run

# Specific browser
npx cypress run --browser chrome
npx cypress run --browser firefox
npx cypress run --browser edge

# Specific spec file
npx cypress run --spec "cypress/e2e/login.cy.js"
```

Project Structure

Default Structure (Generated):

```
 my-project/
├── cypress/
|   ├── e2e/                    # Test files (.cy.js)
|   |   ├── login.cy.js
|   |   ├── products.cy.js
|   |   └── checkout.cy.js
|   ├── fixtures/               # Test data (JSON)
|   |   ├── users.json
|   |   └── products.json
|   ├── support/                # Reusable code
|   |   ├── commands.js         # Custom commands
|   |   └── e2e.js              # Global setup
|   ├── downloads/              # Downloaded files
|   └── screenshots/            # Failure screenshots
```

Recommended Enhanced Structure:

```
my-project/
├── cypress/
|   ├── e2e/
|   |   ├── auth/
|   |   |   ├── login.cy.js
|   |   |   └── signup.cy.js
|   |   ├── products/
|   |   |   ├── search.cy.js
|   |   |   └── filters.cy.js
|   |   └── checkout/
|   |       └── payment.cy.js
|   ├── fixtures/
|   |   ├── auth/
|   |   |   └── users.json
|   |   └── products/
|   |       └── products.json
|   ├── support/
|   |   ├── commands/
|   |   |   ├── auth.js
|   |   |   └── ui.js
|   |   ├── pages/              # Page objects
|   |   |   ├── LoginPage.js
|   |   |   └── ProductPage.js
|   |   ├── commands.js
|   |   └── e2e.js
|   └── plugins/
|       └── index.js
├── cypress.config.js
└── package.json
```

Configuration File

cypress.config.js (Complete):

```
const { defineConfig } = require('cypress')

module.exports = defineConfig({
  e2e: {
```

```
  // Viewport
  viewportWidth: 1280,
  viewportHeight: 720,

  // Timeouts
  defaultCommandTimeout: 8000,     // Command timeout
  pageLoadTimeout: 60000,          // Page load timeout
  requestTimeout: 10000,           // API request timeout
  responseTimeout: 30000,          // API response timeout

  // Video and Screenshots
  video: true,                     // Record video
  videoCompression: 32,            // Compression level
  screenshotOnRunFailure: true,    // Auto screenshot on fail
  trashAssetsBeforeRuns: true,     // Clean before run

  // Retry
  retries: {
    runMode: 2,                    // CI retry count
    openMode: 0,                   // Interactive retry count
  },

  // Browser
  chromeWebSecurity: false,        // Disable web security

  // Spec patterns
  specPattern: 'cypress/e2e/**/*.cy.{js,jsx,ts,tsx}',
  excludeSpecPattern: '*.hot-update.js',

  // Support file
  supportFile: 'cypress/support/e2e.js',

  // Fixtures
  fixturesFolder: 'cypress/fixtures',

  // Setup node events
  setupNodeEvents(on, config) {
    // Implement node event listeners here
    return config
  },
},

// Environment variables
env: {
  apiUrl: 'http://localhost:8080/api',
```

```
})
```

Environment-Specific Configuration

Development:

```
// cypress.config.dev.js
module.exports = defineConfig({
  e2e: {
    baseUrl: 'http://localhost:3000',
    env: {
      apiUrl: 'http://localhost:8080/api',
    },
  },
})
```

Staging:

```
// cypress.config.staging.js
module.exports = defineConfig({
  e2e: {
    baseUrl: 'https://staging.example.com',
    env: {
      apiUrl: 'https://api-staging.example.com',
    },
  },
})
```

Usage:

```
# Use specific config
npx cypress run --config-file cypress.config.staging.js
```

Environment Variables

Three Ways to Set:

```
module.exports = defineConfig({
  env: {
    apiUrl: 'http://localhost:8080',
    username: 'testuser',
  },
})
```

2. In cypress.env.json:

```
{
  "apiUrl": "http://localhost:8080",
  "username": "testuser",
  "password": "secret"
}
```

3. Via CLI:

```
npx cypress run --env apiUrl=http://localhost:8080,username=admin
```

Access in Tests:

```
cy.visit(Cypress.env('apiUrl'))
const username = Cypress.env('username')
```

Core Cypress Commands

Navigation Commands

cy.visit():

```
// Simple visit
cy.visit('/')
cy.visit('/login')
```

```
cy.visit('/login', {
  timeout: 30000,
  onBeforeLoad(win) {
    // Runs before page loads
    win.localStorage.setItem('token', 'abc123')
  },
  onLoad(win) {
    // Runs after page loads
    console.log('Page loaded!')
  },
})

// With query params
cy.visit('/products?category=electronics&sort=price')

// With authentication
cy.visit('/', {
  auth: {
    username: 'admin',
    password: 'secret',
  },
})
```

cy.go():

```
cy.go('back')       // Browser back
cy.go('forward')    // Browser forward
cy.go(-1)           // Back one page
cy.go(1)            // Forward one page
```

cy.reload():

```
cy.reload()         // Reload page
cy.reload(true)     // Force reload (bypass cache)
```

cy.url():

```
cy.url().should('include', '/dashboard')
cy.url().should('eq', 'https://example.com/login')
```

cy.title():

```
 cy.title().should('eq', 'Login Page')
cy.title().should('contain', 'Welcome')
```

Query Commands

cy.get():

```
 // By CSS selector
cy.get('button')
cy.get('.submit-button')
cy.get('#login-form')
cy.get('[data-cy="submit"]')

// Complex selectors
cy.get('form input[type="text"]')
cy.get('.parent > .child')
cy.get('ul li:first-child')

// With options
cy.get('button', { timeout: 10000 })
cy.get('.loading', { timeout: 30000 })
```

cy.contains():

```
 // Find by text content
cy.contains('Submit')
cy.contains('Click here to continue')

// With selector
cy.contains('button', 'Submit')
cy.contains('a', 'Learn More')
cy.contains('.nav-item', 'Home')

// Case insensitive (with regex)
cy.contains(/submit/i)
```

cy.within():

```
 // Scope queries to a specific element
cy.get('form').within(() => {
  cy.get('input[name="email"]').type('user@example.com')
  cy.get('input[name="password"]').type('password123')
  cy.get('button[type="submit"]').click()
})

// Nested within
cy.get('.modal').within(() => {
  cy.get('.modal-header').within(() => {
    cy.get('h2').should('have.text', 'Confirm')
  })
})
```

cy.find():

```
 // Find children
cy.get('.parent').find('.child')
cy.get('form').find('input')

// Chaining
cy.get('ul').find('li').first()
```

cy.filter():

```
 // Filter elements
cy.get('li').filter('.active')
cy.get('button').filter(':visible')
cy.get('input').filter('[required]')
```

Traversal Commands

cy.children():

cy.parent():

```
 cy.get('button').parent()
cy.get('input').parent('form')
```

cy.parents():

```
 cy.get('button').parents()
cy.get('input').parents('form')
```

cy.closest():

```
 cy.get('button').closest('form')
cy.get('td').closest('tr')
```

cy.siblings():

```
 cy.get('.active').siblings()
cy.get('li:first').siblings('li')
```

cy.next():

```
 cy.get('li').next()
cy.get('label').next('input')
```

cy.prev():

```
 cy.get('input').prev()
cy.get('input').prev('label')
```

cy.first() / cy.last():

cy.eq():

```
 cy.get('li').eq(0)   // First (zero-indexed)
cy.get('li').eq(2)   // Third
cy.get('li').eq(-1)  // Last
```

Element Selection and Locators

CSS Selectors

By ID:

```
 cy.get('#username')
cy.get('#submit-button')
```

By Class:

```
 cy.get('.btn')
cy.get('.btn-primary')
cy.get('.card.active')  // Both classes
```

By Attribute:

```
 cy.get('[type="text"]')
cy.get('[name="email"]')
cy.get('[data-cy="login-form"]')  // Recommended!
cy.get('[placeholder="Search"]')
```

Attribute Operators:

```
cy.get('[id~="user"]')       // Word match
```

Pseudo-classes:

```
cy.get('input:visible')
cy.get('button:enabled')
cy.get('input:checked')
cy.get('li:first-child')
cy.get('li:last-child')
cy.get('li:nth-child(2)')
cy.get('input:not([type="hidden"])')
```

Combinators:

```
cy.get('form input')         // Descendant
cy.get('form > input')       // Direct child
cy.get('label + input')      // Adjacent sibling
cy.get('label ~ input')      // General sibling
```

Data Attributes (Best Practice)

HTML:

```
<button data-cy="submit-button">Submit</button>
<input data-cy="email-input" type="email">
<div data-cy="error-message" class="error">Error!</div>
```

Tests:

```
cy.get('[data-cy="submit-button"]').click()
cy.get('[data-cy="email-input"]').type('user@example.com')
cy.get('[data-cy="error-message"]').should('be.visible')
```

Why data-cy?

- Decoupled from styling

- Easy to search in codebase
- Cypress convention

Custom Command:

```
// cypress/support/commands.js
Cypress.Commands.add('getByCy', (selector) => {
  return cy.get(`[data-cy="${selector}"]`)
})

// Usage
cy.getByCy('submit-button').click()
```

jQuery Methods

Cypress uses jQuery under the hood:

```
cy.get('button').then($btn => {
  // $btn is a jQuery object
  expect($btn.text()).to.include('Submit')
  expect($btn).to.have.class('active')
  expect($btn).to.be.visible
})

// jQuery methods
cy.get('input').eq(0)        // First input
cy.get('div').first()        // First div
cy.get('div').last()         // Last div
cy.get('li').filter('.active') // Filter
cy.get('button').not(':disabled') // Exclude
```

Actions and Interactions

Click Actions

```
// Simple click
cy.get('button').click()

// Click with options
cy.get('button').click({ force: true })  // Force click even if covered
cy.get('button').click({ multiple: true }) // Click all matched elements
cy.get('button').click({ timeout: 10000 })

// Position clicks
cy.get('.canvas').click(100, 200)  // Click at coordinates
cy.get('button').click('topLeft')
cy.get('button').click('topRight')
cy.get('button').click('bottomLeft')
cy.get('button').click('bottomRight')
cy.get('button').click('center')

// Multiple clicks
cy.get('button').click().click().click()  // Triple click
cy.get('button').dblclick()  // Double click
cy.get('button').rightclick() // Right click
```

Type Actions

cy.type():

```
// Simple typing
cy.get('input').type('Hello World')

// Special characters
cy.get('input').type('user@example.com')
cy.get('input').type('Hello{enter}')  // Press Enter
cy.get('input').type('{selectall}{backspace}')  // Clear
cy.get('input').type('{ctrl}a')  // Ctrl+A

// Special keys
cy.get('input').type('{enter}')
cy.get('input').type('{esc}')
cy.get('input').type('{backspace}')
cy.get('input').type('{del}')
cy.get('input').type('{leftarrow}')
cy.get('input').type('{rightarrow}')
cy.get('input').type('{uparrow}')
cy.get('input').type('{downarrow}')
```

```
cy.get('input').type('{pagedown}')

// Modifiers
cy.get('input').type('{ctrl}c')  // Copy
cy.get('input').type('{ctrl}v')  // Paste
cy.get('input').type('{shift}{rightarrow}{rightarrow}')  // Select

// Options
cy.get('input').type('text', { delay: 100 })  // Slower typing
cy.get('input').type('text', { force: true })
```

cy.clear():

```
cy.get('input').clear()
cy.get('input').clear({ force: true })
```

Checkbox and Radio

cy.check():

```
// Check checkbox
cy.get('[type="checkbox"]').check()

// Check multiple
cy.get('[type="checkbox"]').check({ multiple: true })

// Check by value
cy.get('[type="radio"]').check('option1')

// Check specific checkboxes
cy.get('[name="colors"]').check(['red', 'blue'])
```

cy.uncheck():

```
cy.get('[type="checkbox"]').uncheck()
cy.get('[name="colors"]').uncheck(['red'])
```

```
 // Select by text
cy.get('select').select('United States')

// Select by value
cy.get('select').select('us')

// Select by index
cy.get('select').select(0)

// Select multiple
cy.get('select[multiple]').select(['option1', 'option2'])

// With force
cy.get('select').select('option1', { force: true })
```

File Upload

cy.selectFile():

```
 // Select file
cy.get('input[type="file"]').selectFile('path/to/file.pdf')

// Multiple files
cy.get('input[type="file"]').selectFile([
  'file1.pdf',
  'file2.pdf'
])

// From fixtures
cy.get('input[type="file"]').selectFile('cypress/fixtures/document.pdf')

// Drag and drop
cy.get('.dropzone').selectFile('file.pdf', { action: 'drag-drop' })
```

Focus and Blur

cy.focus() / cy.blur():

```
cy.get('input').blur()
cy.get('input').should('not.have.focus')
```

Scrolling

cy.scrollIntoView():

```
 cy.get('#footer').scrollIntoView()
cy.get('.element').scrollIntoView({ duration: 2000 })
```

cy.scrollTo():

```
 cy.scrollTo(0, 500)        // Scroll to position
cy.scrollTo('bottom')
cy.scrollTo('top')
cy.scrollTo('center')
cy.scrollTo('bottomRight')
```

Assertions and Expectations

Should Assertions

Basic Assertions:

```
 // Visibility
cy.get('button').should('be.visible')
cy.get('.error').should('not.be.visible')
cy.get('#hidden').should('be.hidden')

// Existence
cy.get('button').should('exist')
cy.get('.deleted').should('not.exist')

// State
```

```
cy.get('[type="checkbox"]').should('not.be.checked')

// Text
cy.get('h1').should('have.text', 'Welcome')
cy.get('p').should('contain', 'Hello')
cy.get('span').should('include.text', 'World')

// Attributes
cy.get('a').should('have.attr', 'href', '/login')
cy.get('input').should('have.value', 'admin')
cy.get('div').should('have.class', 'active')
cy.get('input').should('have.id', 'username')

// CSS
cy.get('button').should('have.css', 'background-color', 'rgb(0, 0, 255)')
cy.get('div').should('have.css', 'display', 'none')

// Length
cy.get('li').should('have.length', 5)
cy.get('.item').should('have.length.greaterThan', 0)
cy.get('.product').should('have.length.lessThan', 20)
```

Chaining Assertions:

```
cy.get('button')
  .should('be.visible')
  .and('be.enabled')
  .and('have.text', 'Submit')
  .and('have.class', 'btn-primary')
```

Custom Assertions with .should(callback):

```
cy.get('input').should($input => {
  expect($input).to.have.value('admin')
  expect($input).to.have.attr('type', 'text')
  expect($input).to.be.visible
})

cy.get('ul li').should($items => {
  expect($items).to.have.length(5)
  expect($items.eq(0)).to.contain('First')
  expect($items.eq(4)).to.contain('Last')
```

Expect Assertions

BDD Style (Chai):

```
 cy.get('button').then($btn => {
  expect($btn).to.be.visible
  expect($btn).to.have.class('active')
  expect($btn.text()).to.equal('Submit')
})

// Multiple expectations
cy.get('input').then($input => {
  expect($input).to.have.attr('type', 'text')
  expect($input).to.have.attr('name', 'username')
  expect($input).to.have.value('')
})
```

Common Chai Assertions:

```
 expect(true).to.be.true
expect('hello').to.equal('hello')
expect([1, 2, 3]).to.have.length(3)
expect({ name: 'John' }).to.have.property('name')
expect(5).to.be.greaterThan(3)
expect(5).to.be.lessThan(10)
expect('hello').to.include('ell')
expect([1, 2, 3]).to.include(2)
expect(null).to.be.null
expect(undefined).to.be.undefined
```

URL Assertions

```
 cy.url().should('eq', 'https://example.com/login')
cy.url().should('include', '/dashboard')
cy.url().should('contain', 'user=admin')
cy.url().should('match', /\/products\/\d+/)
```

```
// Wait for element
cy.get('.loading').should('not.exist')
cy.get('.data-loaded').should('be.visible')

// Wait for text
cy.contains('Data loaded successfully', { timeout: 10000 })

// Wait for attribute
cy.get('button').should('not.have.attr', 'disabled')

// Wait for count
cy.get('.item').should('have.length', 10)
```

Test Structure and Hooks

Test Organization

describe() and it():

```
describe('Login Feature', () => {
  it('should display login form', () => {
    cy.visit('/login')
    cy.get('form').should('be.visible')
  })

  it('should login successfully', () => {
    cy.visit('/login')
    cy.get('#username').type('admin')
    cy.get('#password').type('password123')
    cy.get('button[type="submit"]').click()
    cy.url().should('include', '/dashboard')
  })

  it('should show error for invalid credentials', () => {
    cy.visit('/login')
    cy.get('#username').type('invalid')
    cy.get('#password').type('wrong')
    cy.get('button[type="submit"]').click()
```

Nested describe:

```
describe('E-commerce Application', () => {
  describe('Authentication', () => {
    describe('Login', () => {
      it('should login with valid credentials', () => {
        // Test
      })

      it('should show error with invalid credentials', () => {
        // Test
      })
    })

    describe('Signup', () => {
      it('should register new user', () => {
        // Test
      })
    })
  })

  describe('Shopping Cart', () => {
    it('should add items to cart', () => {
      // Test
    })
  })
})
```

Hooks

beforeEach() and afterEach():

```
describe('Product Tests', () => {
  beforeEach(() => {
    // Runs before each test
    cy.visit('/products')
    cy.login('admin', 'password123')
  })

  afterEach(() => {
```

```
  })

  it('test 1', () => {
    // Test
  })

  it('test 2', () => {
    // Test
  })
})
```

before() and after():

```
describe('Suite', () => {
  before(() => {
    // Runs once before all tests
    cy.task('seedDatabase')
  })

  after(() => {
    // Runs once after all tests
    cy.task('cleanDatabase')
  })

  it('test 1', () => {})
  it('test 2', () => {})
})
```

Test Isolation

```
describe('Isolated Tests', () => {
  beforeEach(() => {
    // Reset state before each test
    cy.clearCookies()
    cy.clearLocalStorage()
    cy.visit('/')
  })

  it('test 1', () => {
    // Independent test
  })
```

```
  })
})
```

Fixtures and Data-Driven Testing

Creating Fixtures

cypress/fixtures/users.json:

```
{
  "admin": {
    "username": "admin",
    "password": "admin123",
    "email": "admin@example.com"
  },
  "standard": {
    "username": "user1",
    "password": "user123",
    "email": "user1@example.com"
  },
  "guest": {
    "username": "guest",
    "password": "guest123",
    "email": "guest@example.com"
  }
}
```

cypress/fixtures/products.json:

```
[
  {
    "id": 1,
    "name": "iPhone 15",
    "price": 999,
    "category": "Electronics"
  },
  {
```

```
    "category": "Computers"
  }
]
```

Using Fixtures

Method 1: cy.fixture() with alias:

```
describe('Login Tests', () => {
  beforeEach(() => {
    cy.fixture('users.json').as('users')
  })

  it('should login as admin', function() {
    cy.visit('/login')
    cy.get('#username').type(this.users.admin.username)
    cy.get('#password').type(this.users.admin.password)
    cy.get('button').click()
  })

  it('should login as standard user', function() {
    cy.visit('/login')
    cy.get('#username').type(this.users.standard.username)
    cy.get('#password').type(this.users.standard.password)
    cy.get('button').click()
  })
})
```

Method 2: cy.fixture() with .then():

```
it('should login with fixture data', () => {
  cy.fixture('users.json').then(users => {
    cy.visit('/login')
    cy.get('#username').type(users.admin.username)
    cy.get('#password').type(users.admin.password)
    cy.get('button').click()
  })
})
```

Method 3: Import fixture:

```
  cy.visit('/login')
  cy.get('#username').type(users.admin.username)
  cy.get('#password').type(users.admin.password)
  cy.get('button').click()
})
```

Data-Driven Tests

Iterate over fixture data:

```
describe('Data-Driven Login Tests', () => {
  beforeEach(() => {
    cy.fixture('users.json').as('users')
  })

  it('should test multiple users', function() {
    Object.keys(this.users).forEach(userType => {
      const user = this.users[userType]

      cy.visit('/login')
      cy.get('#username').type(user.username)
      cy.get('#password').type(user.password)
      cy.get('button').click()

      cy.url().should('include', '/dashboard')
      cy.get('.logout').click()
    })
  })
})
```

Dynamic test generation:

```
describe('Product Search Tests', () => {
  const searchTerms = ['iPhone', 'MacBook', 'iPad', 'AirPods']

  searchTerms.forEach(term => {
    it(`should search for ${term}`, () => {
      cy.visit('/')
      cy.get('[data-cy="search"]').type(term)
      cy.get('[data-cy="search-button"]').click()
```

```
})
```

Debugging with Cypress Test Runner

Cypress Test Runner Features

Interactive UI:

- Real-time test execution
- Command log with snapshots
- Time travel debugging
- DOM snapshots
- Before/after state
- Automatic screenshots

Opening Test Runner:

```
npx cypress open
```

Debug Commands

cy.pause():

```
it('should debug test', () => {
  cy.visit('/login')
  cy.pause()  // Pauses execution
  cy.get('#username').type('admin')
  cy.pause()  // Pause again
  cy.get('#password').type('password')
})
```

cy.debug():

```
    .debug() // opens debugger
    .click()
})
```

cy.log():

```
it('should log messages', () => {
  cy.log('Starting test')
  cy.visit('/')
  cy.log('Visited homepage')
  cy.get('button').click()
  cy.log('Clicked button')
})
```

Time Travel

Hover over commands:

- See DOM at that moment
- Inspect elements
- View console output
- Check network requests

Pin snapshots:

- Click command to pin
- Compare before/after
- Debug state changes

Screenshots

Automatic screenshots on failure:

```
// Configured in cypress.config.js
screenshotOnRunFailure: true
```

```
 cy.screenshot()
cy.screenshot('login-page')
cy.screenshot('full-page', { capture: 'fullPage' })

cy.get('.modal').screenshot()  // Screenshot specific element
```

Videos

Automatic video recording:

```
 // cypress.config.js
video: true,
videoCompression: 32,
```

Videos saved to:

```
 cypress/videos/
```

Browser DevTools

Open DevTools:

- Click "DevTools" in Test Runner
- Or press F12

Inspect elements:

```
 cy.get('button').then($btn => {
  debugger  // Opens browser debugger
  console.log($btn)
})
```

Practice Exercises

```
describe('Exercise 1: Login Flow', () => {
  it('should complete login successfully', () => {
    // 1. Visit login page
    // 2. Enter username
    // 3. Enter password
    // 4. Click submit
    // 5. Verify redirect to dashboard
    // 6. Verify welcome message

    // Your implementation
  })

  it('should show error for invalid credentials', () => {
    // 1. Visit login page
    // 2. Enter invalid credentials
    // 3. Click submit
    // 4. Verify error message

    // Your implementation
  })
})
```

Exercise 2: Fixtures Usage

Task: Use fixtures for data-driven testing

```
// Create users.json fixture
// Test login with different users

describe('Exercise 2: Data-Driven Tests', () => {
  // Your implementation
})
```

Exercise 3: Form Interactions

Task: Fill and submit complex form

```
    // 2. Select dropdown
    // 3. Check checkboxes
    // 4. Upload file
    // 5. Submit form
    // 6. Verify success

    // Your implementation
  })
})
```

Exercise 4: Navigation

Task: Test navigation menu

```
describe('Exercise 4: Navigation', () => {
  it('should navigate through all menu items', () => {
    // 1. Visit homepage
    // 2. Click each menu item
    // 3. Verify URL changes
    // 4. Verify page content

    // Your implementation
  })
})
```

Exercise 5: Assertions

Task: Practice different assertions

```
describe('Exercise 5: Assertions', () => {
  it('should test element states', () => {
    // Test visibility, text, attributes, CSS, etc.

    // Your implementation
  })
})
```

In Day 8, you mastered:

✅ Cypress Architecture

- In-browser execution
- Command queue system
- Automatic waiting
- Time travel debugging

✅ Cypress vs Playwright

- Feature comparison
- When to use each
- Migration strategies

✅ Project Setup

- Installation
- Configuration
- Directory structure
- Environment variables

✅ Core Commands

- Navigation
- Querying
- Traversal
- Filtering

✅ Locators

- CSS selectors
- Data attributes
- Best practices

- Form interactions
- File uploads
- Scrolling

✅ Assertions

- Should syntax
- Expect syntax
- Custom assertions

✅ Test Structure

- describe/it blocks
- Hooks
- Test isolation

✅ Fixtures

- Creating fixtures
- Using fixtures
- Data-driven testing

✅ Debugging

- Test Runner
- Debug commands
- Screenshots/videos
- DevTools

Key Takeaways

- **Cypress runs in browser** for speed and reliability
- **Automatic waiting** eliminates flaky tests
- **Time travel** makes debugging easy

- **Test isolation** is critical
- **Interactive debugging** is powerful

Best Practices

1. Use data-cy attributes
2. Keep tests independent
3. Use fixtures for test data
4. Leverage automatic waiting
5. Write descriptive test names
6. Use beforeEach for setup
7. Clean up after tests
8. Use custom commands for reusability
9. Debug with Test Runner
10. Follow Cypress conventions

Cypress Checklist

- ☐ Project structure created
- ☐ Configuration file setup
- ☐ Custom commands defined
- ☐ Fixtures created
- ☐ Tests organized
- ☐ Data attributes added
- ☐ Assertions comprehensive
- ☐ Debugging practiced
- ☐ Best practices followed
- ☐ Test isolation ensured

Next Steps

- Network interception (cy.intercept)
- API testing (cy.request)
- Environment variables
- Plugins and configuration

End of Day 8 Documentation

← Advanced API Testing

Advanced Cypress & API Testing →