Introduction

Agenda

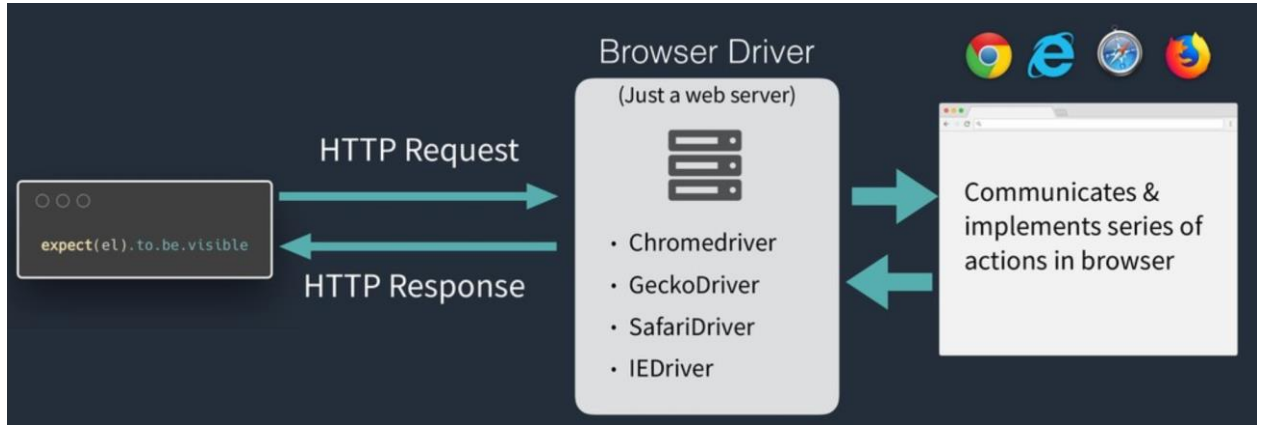
- What is Cypress?
- Selenium Vs Cypress Architectural Differences
- Cypress Ecosystem
- How Cypress is different than Other tools
- Features of Cypress
- Limitations

What is Cypress?

- Cypress is a next generation front end testing tool built for the modern web applications.
- Cypress uses JavaScript to write automated tests.
- Cypress addresses the key pain points from other automation tools.
- Cypress built on Node.js and comes packaged as an npm module.
- As it is built on Node.js, It uses JavaScript for writing tests. But 90% of coding can be done using Cypress inbuilt commands which are easy to understand.
- Cypress makes our tests very simple when we compared with other tools.
- Cypress is having different architecture when you compare with selenium.
- We can write faster, easier and more reliable tests using Cypress.

Selenium Vs Cypress Architectures

- Most testing tools (like Selenium) operate by running outside of the browser and executing remote commands across the network.
- But Cypress engine directly operates inside the browser. In other words, It is the browser that is executing your test code.
- This enables Cypress to listen and modify the browser behavior at run time by manipulating DOM and altering Network requests and responses on the fly.



Cypress ecosystem

- Cypress is an Open source tool and consists of…..

1. Test Runner (Open Source Component. Locally Installed) helps you set up and start writing tests.
2. Dashboard Service(Recording tests).

   The Dashboard provides you insight into what happened when your tests ran.

Install the Cypress Test Runner and write tests locally.

Set up tests

Write tests

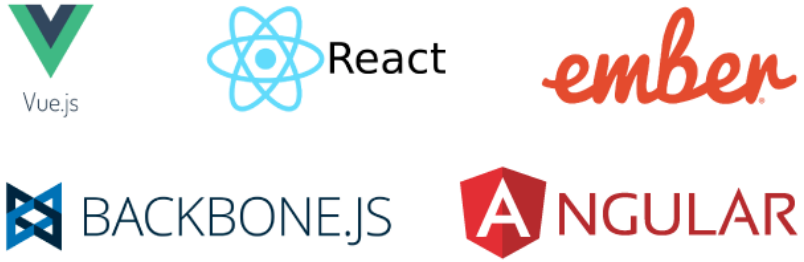
Build up a suite of CI tests, record them and gain powerful insights

Run tests

Record tests

7 ways Cypress is different

- Cypress does not use Selenium.
- Cypress focuses on doing end-to-end testing well.
- Cypress works on any front-end framework or website.
- Cypress tests are only written in JavaScript.
- Cypress is all in one.
- Cypress is for developers and QA engineers.
- Cypress runs much, much faster.



Before Cypress

Choose a framework

Mocha

Jasmine QUnit

Karma

Choose an assertion library

Chai

Expect.js

Install

Selenium

Choose a Selenium wrapper

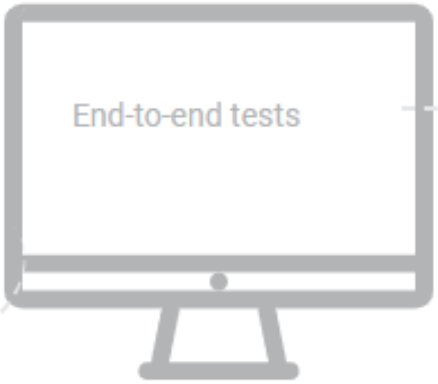
Protractor

Nightwatch Webdriver

Add additional libraries

Sinon

TestDouble



vs

✨ With Cypress ✨

cypress

All-in-one testing framework, assertion library, with mocking and stubbing, all without Selenium.

Features

Time travel

Debuggability

Real time reloads

Automatic waiting

Spies, stubs, and clocks

Network traffic control

Consistent results

Screenshots and videos

Limitations

- Support Limited set of browsers – Chrome, Canary, Electron
- Page Object Model is not supported
- Tough to read data from files
- Third Party Reporting tool integration is limited.

Environment Setup

Cypress Installation & Project Setup

- 1) Download Node & NPM
  https://nodejs.org/en/download/
- 2) Set NODE_HOME Environment Variable
- 3) Create Cypress Working Folder
- 4) Generate package.json
  npm init
- 5) Install Cypress
  npm install cypress --save-dev
- 6) Download Visual Studio Code Editor
  https://code.visualstudio.com/download

Environment Setup

macOS®

Cypress Installation & Project Setup

- Download Node & NPM
- Create Cypress Working Folder
- Generate package.json
- Install Cypress
- Download Visual Studio Code Editor

Test Runner

Agenda

- How to Launch Test Runner in Cypress
- Explore sample Tests in Cypress

Launching Test Runner

Opening Cypress

If you used `npm` to install, Cypress has now been installed to your `./node_modules` directory, with its binary executable accessible from `./node_modules/.bin`.

Now you can open Cypress from your project root one of the following ways:

The long way with the full path

```
$ ./node_modules/.bin/cypress open
```

Or with the shortcut using `npm bin`

```
$ $(npm bin)/cypress open
```

Writing First Test Case

Test Suite & Test Case Structure in Cypress (Mocha)

```
describe('Test Suite', function() {

    it('Test Case1', function() {
          Steps
     })

    it('Test Case2', function() {
     Steps
     })

})
```

Run Tests in Cypress

Cypress Test Runner

Cypress Terminal

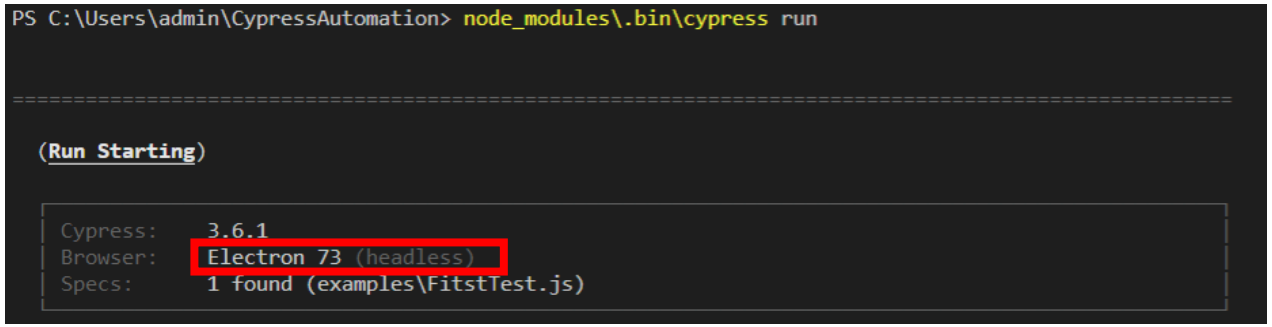
- To open **Test Runner**

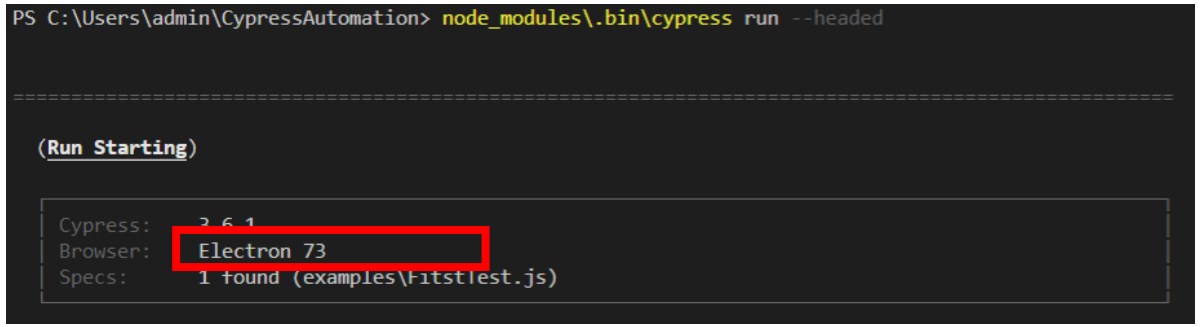
```
node_modules\.bin\cypress open
```

- To Run **All the tests** under **examples** directory

```
node_modules\.bin\cypress run
```

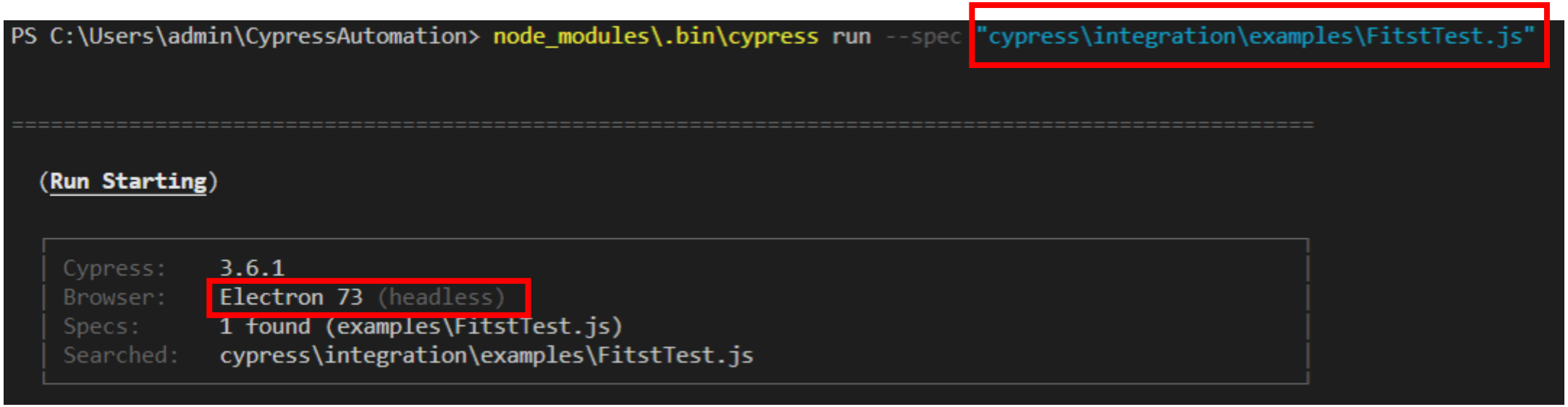
```
node_modules\.bin\cypress run --headed
```





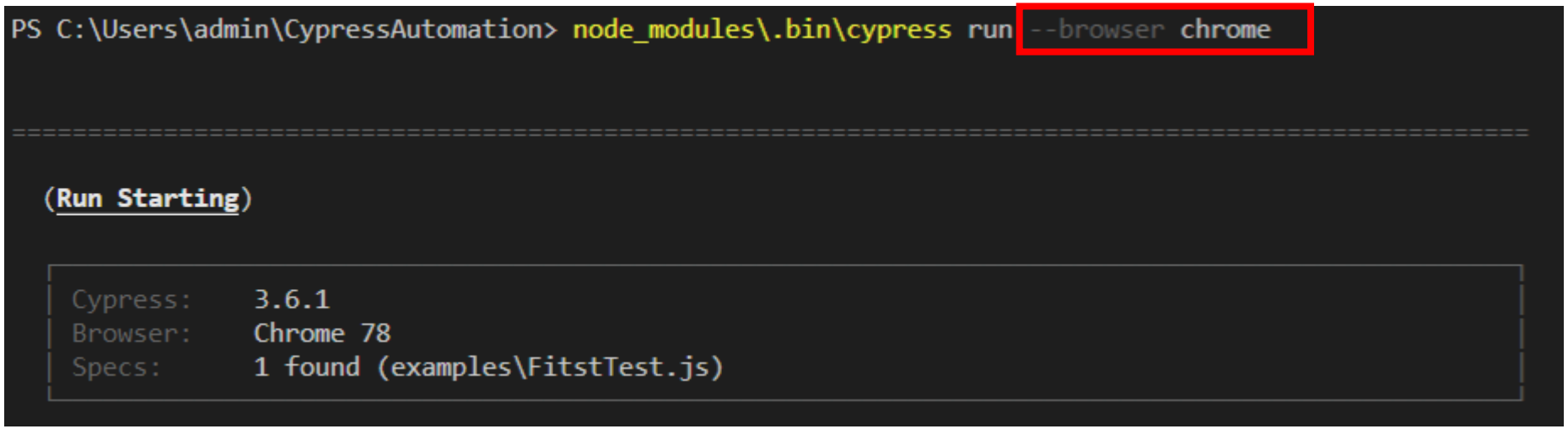
- To **Run Single test** under **examples** directory

```
node_modules\.bin\cypress run –spec "cypress\integration\examples\FirstTest.js"
```



- To Run **All the tests** under **examples** directory using **Chrome**

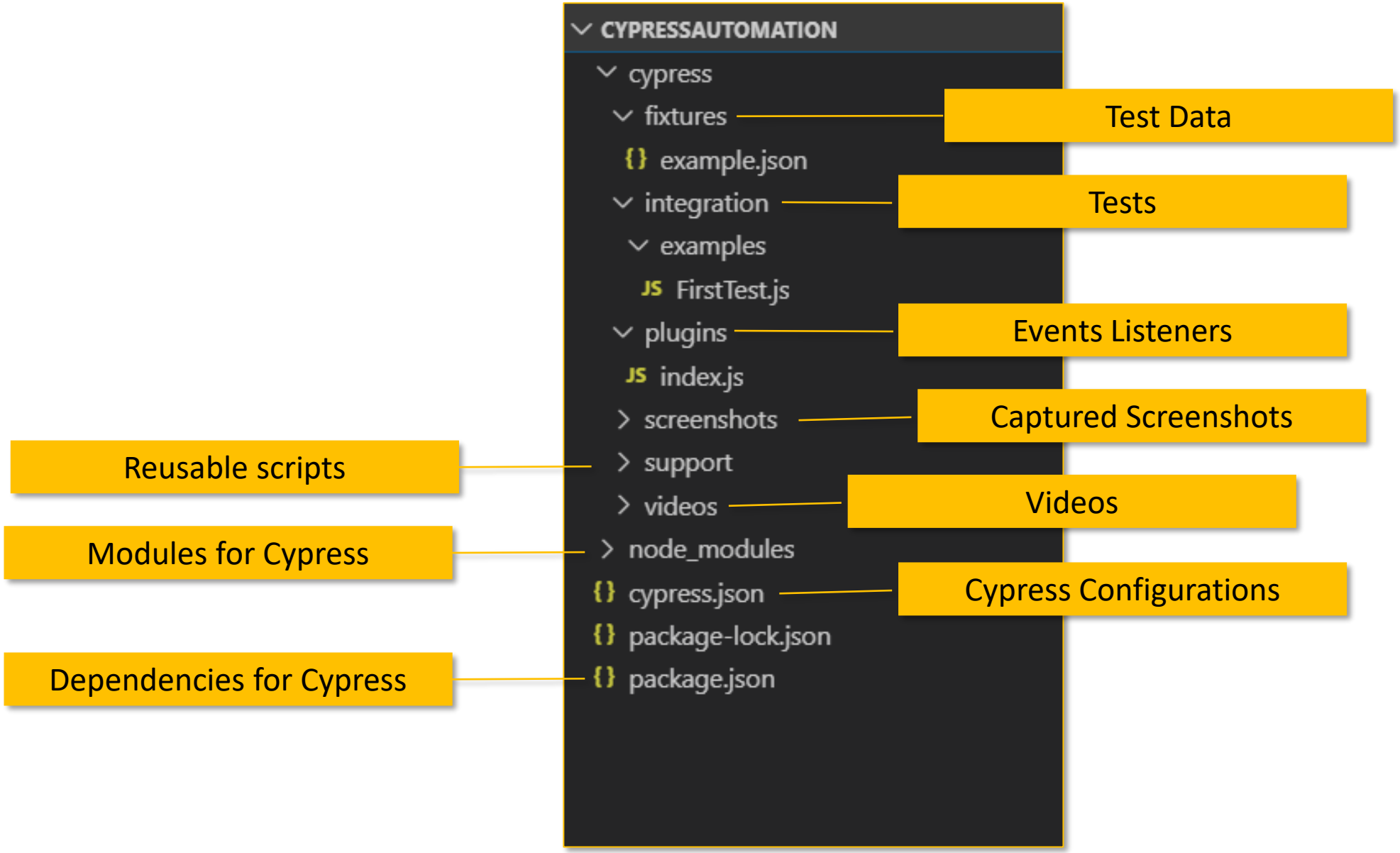
```
node_modules\.bin\cypress run
```



Command Line (Documentation)

- https://docs.cypress.io/guides/guides/command-line.html#Installation

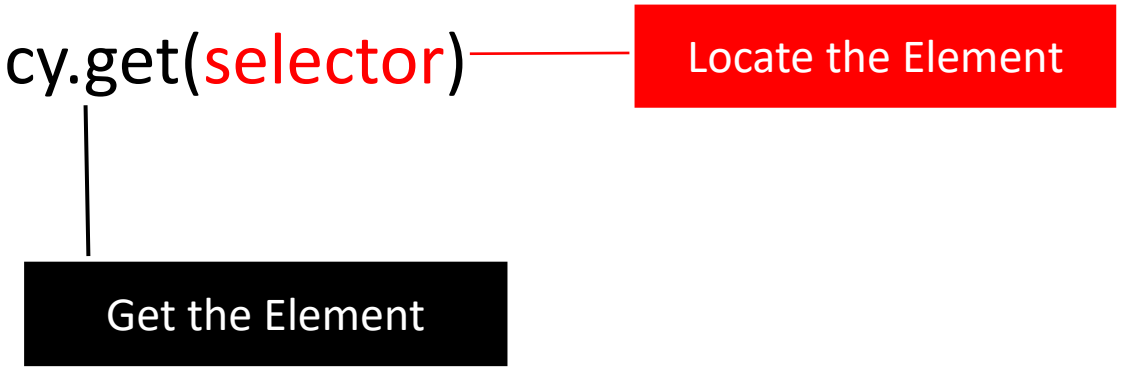
Folder Structure


CYPRESSAUTOMATION
cypress
fixtures
example.json
integration
examples
FirstTest.js
plugins
index.js
screenshots
support
videos
node_modules
cypress.json
package-lock.json
package.json
Test Data
Tests
Events Listeners
Captured Screenshots
Reusable scripts
Videos
Modules for Cypress
Cypress Configurations
Dependencies for Cypress

Locating Elements

get()

- **get**
- Get one or more DOM elements by selector.



CSS Selector

- .class
- #id
- [attribute=value]
- .class[attribute=value]

- Ref: https://www.w3schools.com/cssref/css_selectors.asp

Examples

- **Basic CSS Selectors**
- .class
- #id
- [attribute=value]
- .class[attribute=value]

https://demo.nopcommerce.com/

- **Examples:**

- **1) Locating Log-in ink (using class)**

  .ico-login

- **2) Locating inputbox (using ID)**

  #small-searchterms

  ```
  cy.get("#small-searchterms").type("Apple MacBook Pro 13-inch")
  ```

- **3) Locating search button (Using [attribute=value])**

  [type=submit]

  ```
  cy.get("[type=submit]").click()
  ```

- **4) Locating add to cart button (using .class[attribute=value])**

  .product-box-add-to-cart-button[value="Add to cart"]

  ```
  cy.get(".product-box-add-to-cart-button[value='Add to cart']").click()
  ```

Capturing CSS Selectors

- Right Click on Element→ Inspect → Copy → Copy Selector
- ChroPath Extension for Chrome
- Selector Playground from Test Runner

Locating Elements - Demo

1) Launch Browser & Open URL

https://demo.nopcommerce.com/

2) Enter Text in Search box **"Apple MacBook Pro 13-inch"**

3) Click on **Search** Button

4) Click on **Add to cart**

5) Provide Quantity **2**

6) Click on **Add to cart**

7) Click on **Shopping Cart** Link at the top of the page

8) Verify the total amount.

Code Snippet

```
/// <reference types="cypress" />
describe('Locating Elements', function()
{
    it('Verify types of locators', function()
    {
    cy.visit('https://demo.nopcommerce.com/')
    cy.get("#small-searchterms").type("Apple MacBook Pro 13-inch")//Search box
    cy.get("[type=submit]").click() //Search button
    cy.get(".product-box-add-to-cart-button[value='Add to cart']").click() //add to cart
    cy.get("#addtocart_4_EnteredQuantity").clear()   //Clear Number of products
    cy.get("#addtocart_4_EnteredQuantity").type('2')    //Number of products
    cy.wait(2000)
    cy.get("#add-to-cart-button-4").click()  //add to cart
    cy.wait(2000)
    cy.get("#topcartlink > a > span.cart-label").click() //Shopping cart link
    cy.wait(2000)
    cy.get(".product-unit-price").contains('$1,800.00') //Validating total amount
    })
  })
```

Interacting With UI Elements

- UI Elements
  - Input box
  - Radio Buttons
- Commands
  - visit()
  - url()
  - get()
  - title()

Interacting With UI Elements

- UI Elements
  - Check Boxes
  - Drop Downs

Interacting With UI Elements

- UI Elements
  - Alerts

Navigating Pages

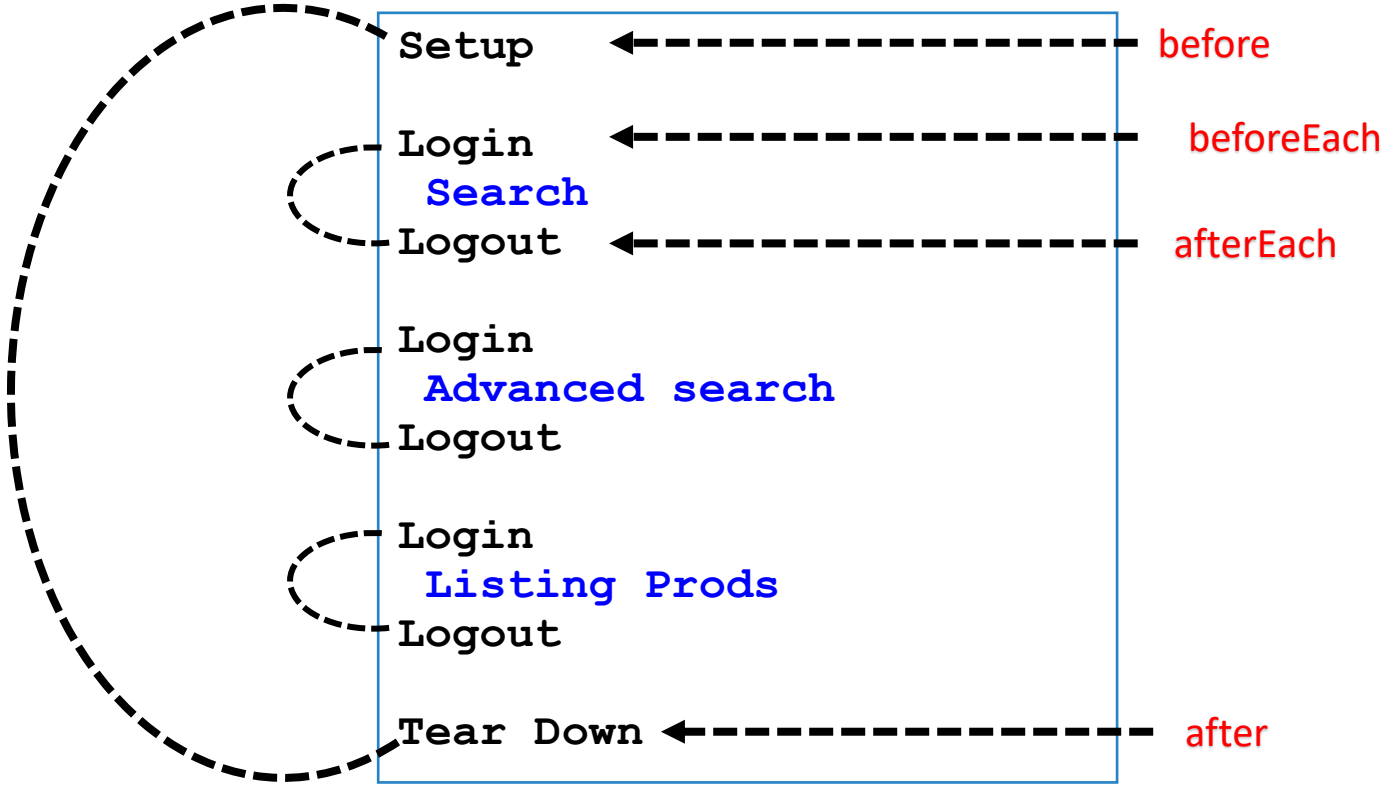
- Go()
- Navigate back or forward to the previous or next URL in the browser's history

Handling Web Table

- Check Value presence anywhere in the table
- Check Value presence in specific row & column
- Check Value presence based on condition by iterating rows.
  - *Check the book name "Master In Java" whose author is Amod*

Cypress Hooks

Cypress hooks borrowed from Mocha used to organizing tests.



- beforeEach
- afterEach
- before
- after

Cypress Fixture

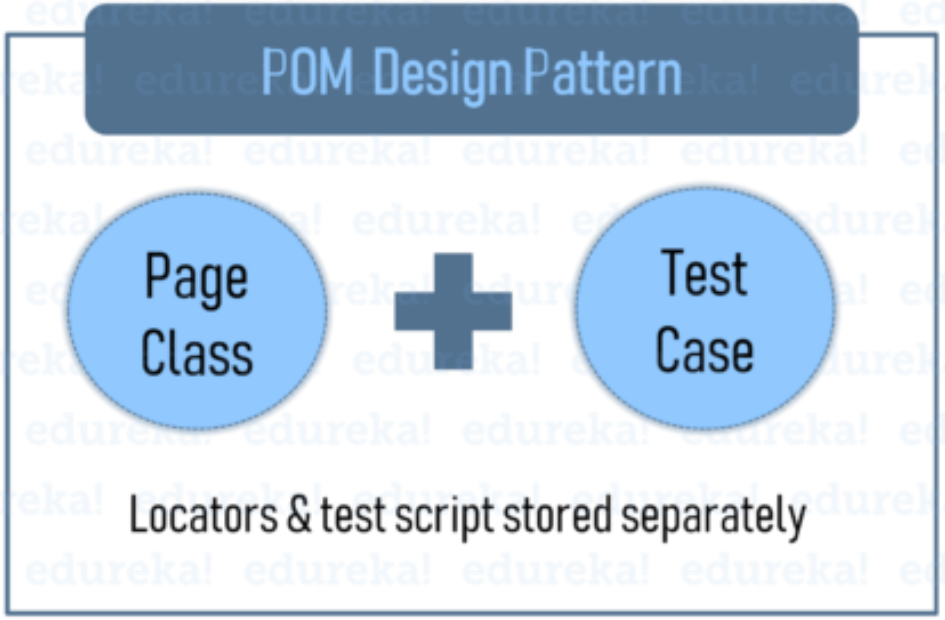
- Load a fixed set of data located in a file.

Custom Commands

- How to Create Custom commands in Cypress

Page Object Model Pattern

- Page Object Model is a design Patten where Page objects are separated from Automation test scripts.
- Advantage:
  - Reusability
  - Maintainability



Cypress Command Line & Dashboard Services

- How to run Cypress from the command line
- How to specify which spec files to run
- Working with Dashboard features
  - Capture screenshots
  - Recording

- cd C:\Users\admin\CypressAutomation>
- To Run all the specs in command line
  - node_modules\.bin\cypress run
- To Run specific specs the tests in command line
  - node_modules\.bin\cypress run --spec "cypress\integration\examples\FirstTest.spec.js"
- **Dashboard**
- https://dashboard.cypress.io/login
- Screenshots & Recording feature in Dashboard
  - cypress run --record --key d63f3548-892b-41b9-bcb7-07ae3cbf9f9b

Continuous Integration

- Cypress Integration with Jenkins

- "scripts": {
- "test": "node_modules\\.bin\\cypress run --config pageLoadTimeout=100000",
- "runtests":"npm run test --"