CYDEO

# Cypress Training

By OSCAR

Lesson 1

## LESSON 1: Introduction and installation

- What is Cypress?
- Differences between Cypress and Selenium or other UI testing tools
- Installation
  - Node .js
  - IDE- Visual Studio Code
  - Cypress CLI
  - Creating a new project with cypress
  - Git for Version Control
- Understanding Framework structure
- Test structure and test execution

# What is Cypress?

- Cypress is front end testing tool, with functions that can also do requests for back end (to API end points)
- Cypress vs Selenium: Cypress is both fundamentally and architecturally different.
- While Selenium operates by running outside of the browser, Cypress is **executed in the same loop as the application**.
- Cypress is not constrained by the same restrictions as Selenium.
- Cypress can test anything that runs in a browser.

## What is Cypress?

- Cypress enables you to write all types of tests:
  - End-to-end tests
  - Component tests
  - Integration tests
  - Unit tests
- Features
  - Time Travel: Cypress takes snapshots as your tests run. Hover over commands in the Command Log to see exactly what happened at each step.
  - Automatic Waiting: Never add waits or sleeps to your tests.
  - Screenshots and Videos: View screenshots taken automatically on failure, or videos of your entire test suite when run from the CLI.
  - Cross browser Testing

## Major Advantages Of Cypress

- Selenium requires the installation of browser drivers so that the script can interact with the web elements on the page. However, installing Cypress does not have any additional dependencies, extra downloads since the test cases run directly inside the browser.
- Selenium is purely a test automation tool, whereas both developers and QA engineers use Cypress. It is built on JavaScript that is widely used for front-end development.
- Cypress processes respond to the application's events and processes command in real-time. With real-time reloads in Cypress, tests are reloaded automatically as and when changes are made in the app.

## Major Advantages Of Cypress

- Compared to Selenium, the Cypress framework is more capable of delivering consistent results. This is because Cypress has tighter control over the entire automation process (from top to bottom), due to which it has a better understanding of things happening in and out of the browser.
- There is no necessity for adding implicit and explicit wait statements in Cypress since Cypress automatically waits for the element to exist in the DOM. It also waits for commands and assertions before the execution moves to the next statement. As far as stale elements are concerned, Cypress never yields stale elements that are no longer a part of the DOM.

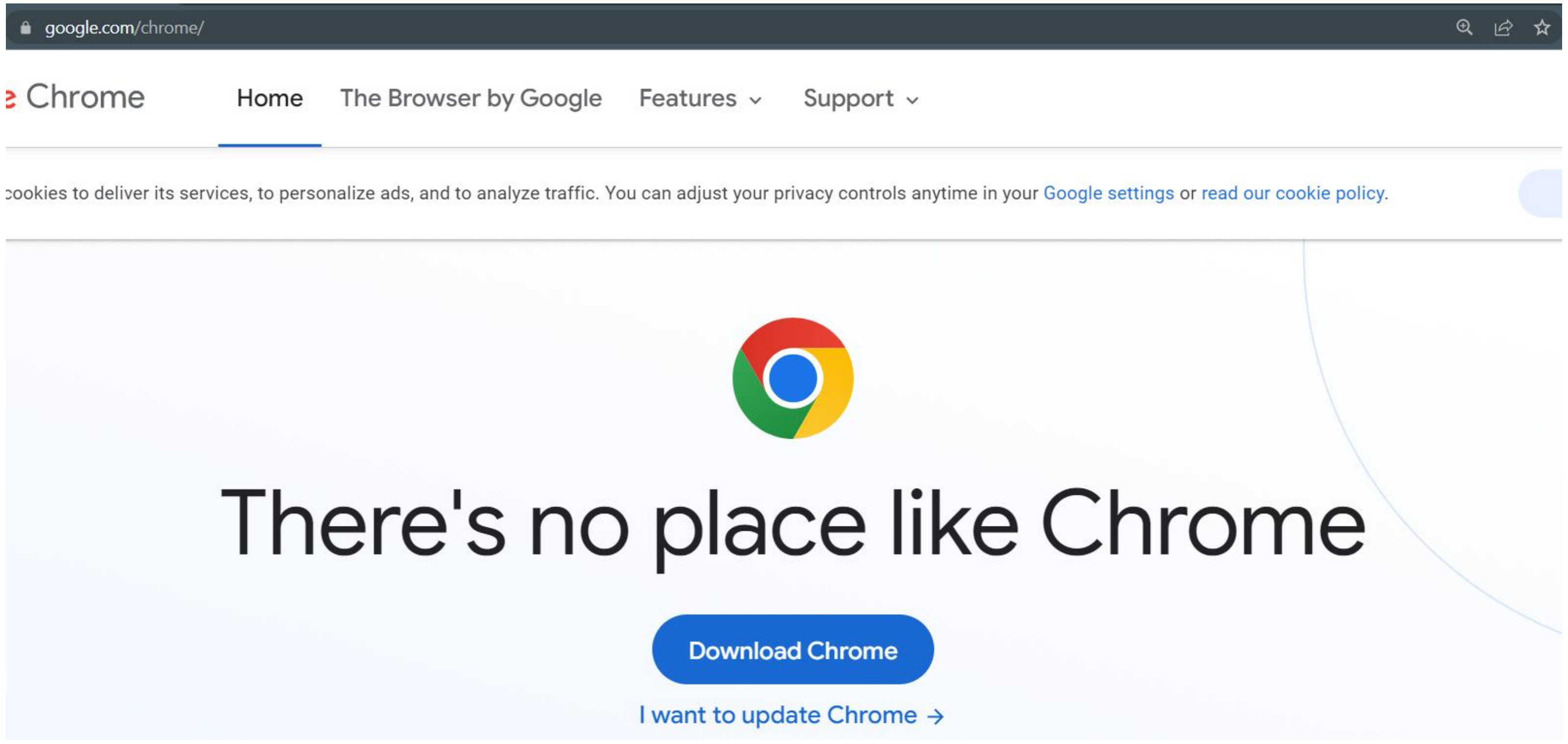
CYDEO

## Major Advantages Of Cypress

- There is no network lag and flakiness in tests executed with Cypress as tests are executed inside the browser and have complete visibility of everything happening in the application synchronously. It even knows when an element is animating and waits for it to stop animating. This makes Cypress tests more flake resistant compared to tests executed with other test automation tools.
- Cypress takes a snapshot at every test step. This enables the developer to check the state and activity at any particular step in the test script.

## Setting up tests

- Download Google Chrome (if you already do not have it)
- https://www.google.com/chrome/

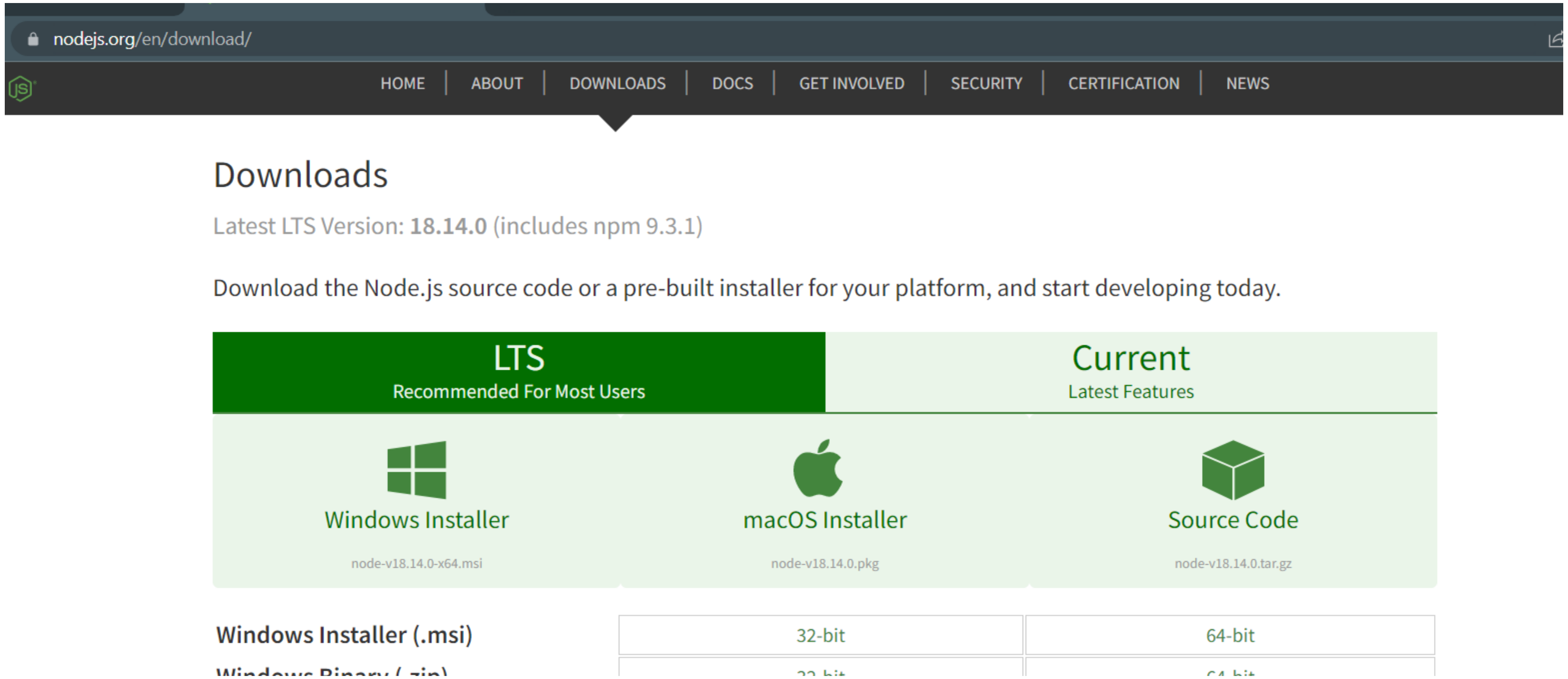


cookies to deliver its services, to personalize ads, and to analyze traffic. You can adjust your privacy controls anytime in your Google settings or read our cookie policy.

There's no place like Chrome

# Setting up tests

- Download and install Nodejs - https://nodejs.org/en/download/
- Make sure to install **LTS Version** for macOS or Windows



Downloads

Latest LTS Version: **18.14.0** (includes npm 9.3.1)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

| LTS Recommended For Most Users | Current Latest Features |
|---|---|

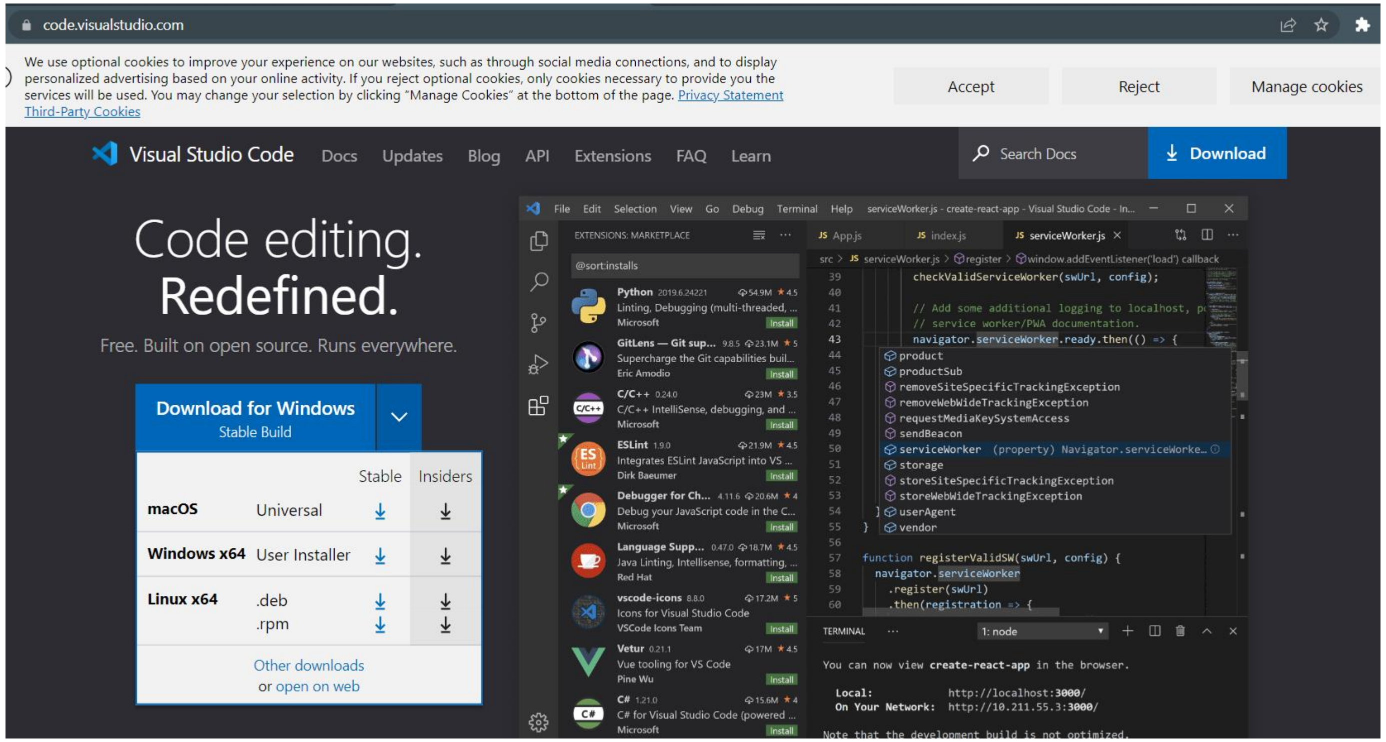
| Windows Installer | macOS Installer | Source Code |
|---|---|---|
| node-v18.14.0-x64.msi | node-v18.14.0.pkg | node-v18.14.0.tar.gz |

| | | |
|---|---|---|
| Windows Installer (.msi) | 32-bit | 64-bit |

CYDEO

# Setting up tests

- Download and install Visual Studio Code (VSC) Editor - [https://code.visualstudio.com/](https://code.visualstudio.com/)
- Make sure to install Stable Version for macOS or Windows



code.visualstudio.com

We use optional cookies to improve your experience on our websites, such as through social media connections, and to display personalized advertising based on your online activity. If you reject optional cookies, only cookies necessary to provide you the services will be used. You may change your selection by clicking "Manage Cookies" at the bottom of the page. Privacy Statement Third-Party Cookies

Accept Reject Manage cookies

Visual Studio Code Docs Updates Blog API Extensions FAQ Learn Search Docs Download

Code editing.
Redefined.

Free. Built on open source. Runs everywhere.

Download for Windows
Stable Build

| | | Stable | Insiders |
|---|---|---|---|
| macOS | Universal | ↓ | ↓ |
| Windows x64 | User Installer | ↓ | ↓ |
| Linux x64 | .deb<br>.rpm | ↓<br>↓ | ↓<br>↓ |

Other downloads
or open on web

File Edit Selection View Go Debug Terminal Help serviceWorker.js - create-react-app - Visual Studio Code - In...

EXTENSIONS: MARKETPLACE

@sort:installs

| | | | |
|---|---|---|---|
| Python 2019.6.24221 | 54.9M | ★4.5 | Linting, Debugging (multi-threaded, ... Microsoft |
| GitLens — Git sup... 9.8.5 | 23.1M | ★5 | Supercharge the Git capabilities buil... Eric Amodio |
| C/C++ 0.24.0 | 23M | ★3.5 | C/C++ IntelliSense, debugging, and ... Microsoft |
| ESLint 1.9.0 | 21.9M | ★4.5 | Integrates ESLint JavaScript into VS ... Dirk Baeumer |
| Debugger for Ch... 4.11.6 | 20.6M | ★4 | Debug your JavaScript code in the C... Microsoft |
| Language Supp... 0.47.0 | 18.7M | ★4.5 | Java Linting, Intellisense, formatting, ... Red Hat |
| vscode-icons 8.8.0 | 17.2M | ★5 | Icons for Visual Studio Code VSCode Icons Team |
| Vetur 0.21.1 | 17M | ★4.5 | Vue tooling for VS Code Pine Wu |
| C# 1.21.0 | 15.6M | ★4 | C# for Visual Studio Code (powered ... Microsoft |

App.js index.js serviceWorker.js

src > serviceWorker.js > register > window.addEventListener('load') callback

```
        checkValidServiceWorker(swUrl, config);

        // Add some additional logging to localhost, p
        // service worker/PWA documentation.
        navigator.serviceWorker.ready.then(() => {
          product
          productSub
          removeSiteSpecificTrackingException
          removeWebWideTrackingException
          requestMediaKeySystemAccess
          sendBeacon
          serviceWorker  (property) Navigator.serviceWorke...
          storage
          storeSiteSpecificTrackingException
          storeWebWideTrackingException
      } userAgent
  }     vendor

function registerValidSW(swUrl, config) {
  navigator.serviceWorker
    .register(swUrl)
    .then(registration => {
```

TERMINAL 1: node

```
You can now view create-react-app in the browser.

  Local:            http://localhost:3000/
  On Your Network:  http://10.211.55.3:3000/

Note that the development build is not optimized.
```

CYDEO

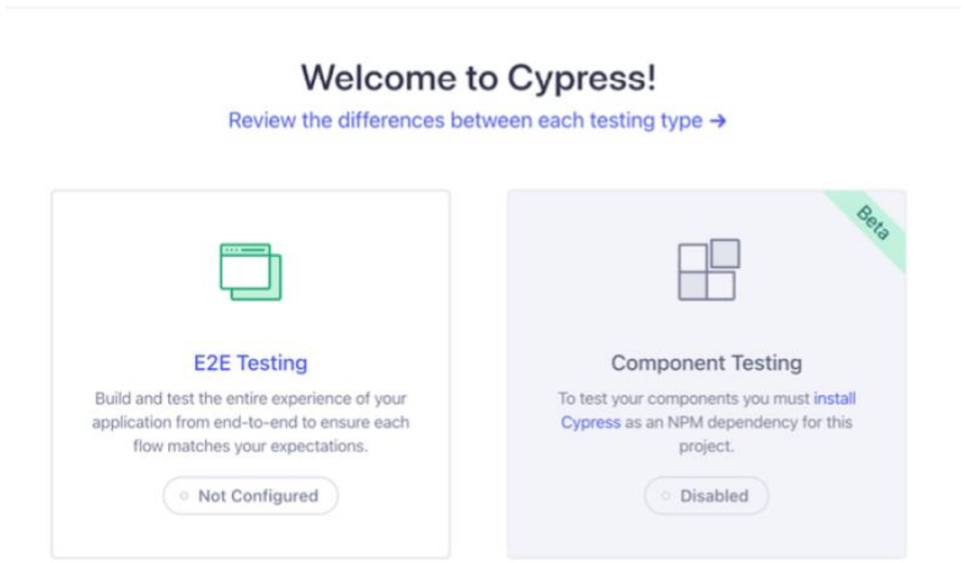
# Setting up tests - Installing Cypress

- Open Terminal  (FOR MACOS)
- Change Directory to Desktop
  - cd ~/Desktop
- Make a new Folder
  - mkdir CypressAutomation
- To see what is inside the directory you can run ‘ls’ command
- Go inside CypressAutomation directory
  - cd CypressAutomation
- Initialize necessary Node.js Modules inside our project
  - npm init
- Install Cypress Tools inside your project
  - Npm install cypress
- Open Cypress CLI tool for the rest of the installation process
  - npx cypress open

- Open Command Prompt or PowerShell (FOR WINDOWS)
- Change Directory to Desktop
  - cd Desktop
- Make a new Folder
  - mkdir CypressAutomation
- To see what is inside the directory you can run ‘dir’ command
- Go inside CypressAutomation directory
  - cd CypressAutomation
- Initialize necessary Node.js Modules inside our project
  - npm init
- Install Cypress Tools inside your project
  - Npm install cypress
- Open Cypress CLI tool for the rest of the installation process
  - npx cypress open

# Setting up tests - Installing Cypress

- Open Cypress CLI application that you previously installed
- Chrome. You should see something like this and select E2E, then click next, this will install example framework structure to your project



Welcome to Cypress!

Review the differences between each testing type →

E2E Testing

Build and test the entire experience of your application from end-to-end to ensure each flow matches your expectations.

Not Configured

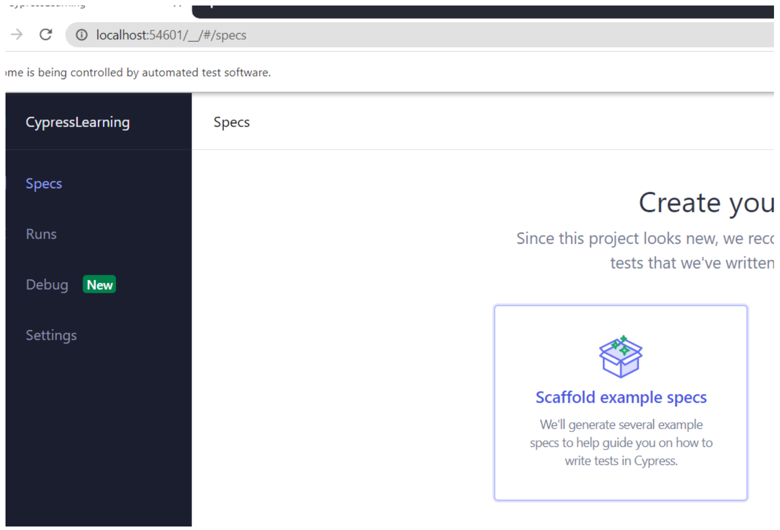
Component Testing

To test your components you must install Cypress as an NPM dependency for this project.

Disabled

# Setting up tests - Installing Cypress

- Select scaffold files. This will generate initial Cypress folder structure, and provide some example tests (which we don't need for this lesson, but we need the structure )



localhost:54601/__/#/specs

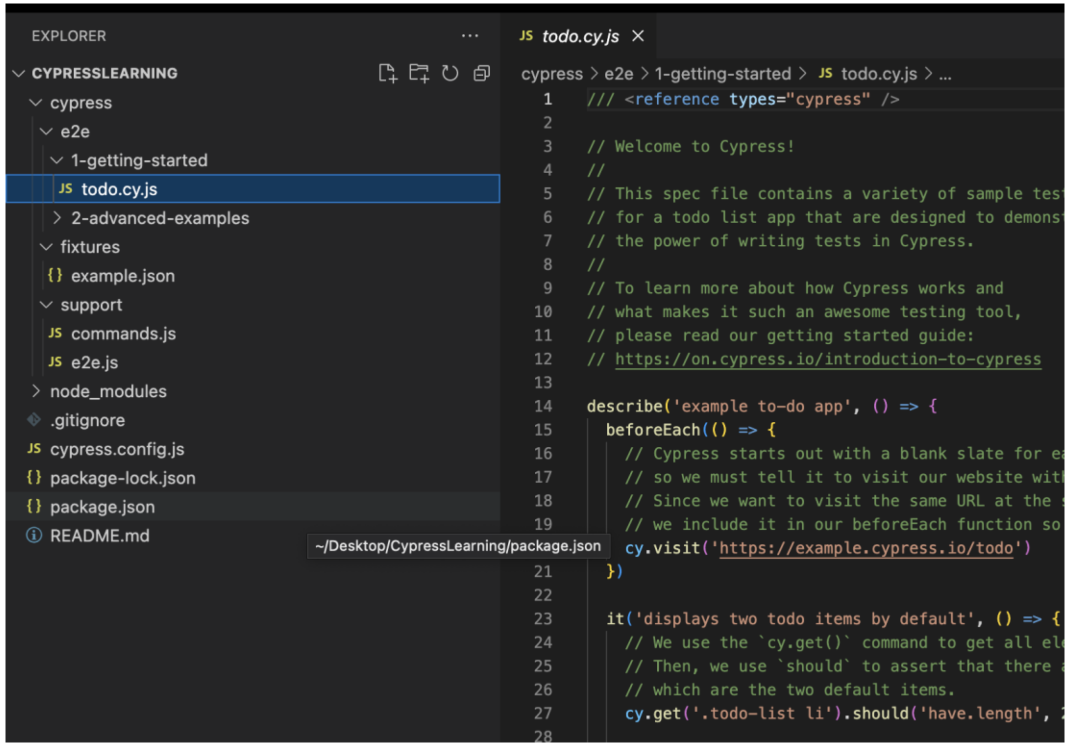
me is being controlled by automated test software.

Create you

Since this project looks new, we rec
tests that we've written

**Scaffold example specs**

We'll generate several example specs to help guide you on how to write tests in Cypress.



```
cypress
  e2e
    1-getting-started
      todo.cy.js
    2-advanced-examples
  fixtures
    example.json
  support
    commands.js
    e2e.js
node_modules
.gitignore
cypress.config.js
package-lock.json
package.json
README.md
```

~/Desktop/CypressLearning/package.json

todo.cy.js

cypress > e2e > 1-getting-started > todo.cy.js > ...

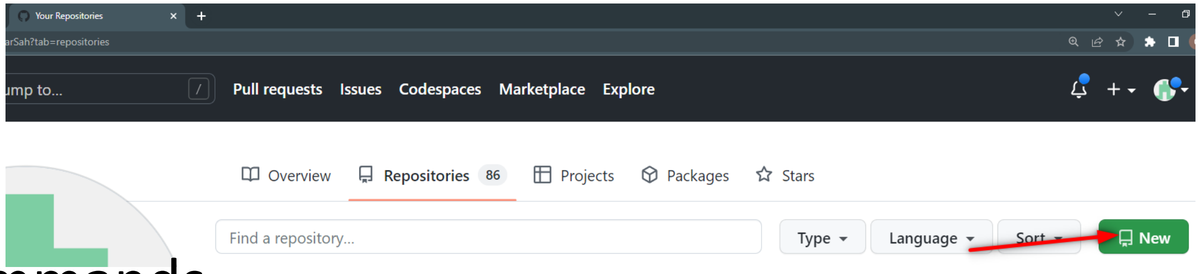
```
/// <reference types="cypress" />

// Welcome to Cypress!
//
// This spec file contains a variety of sample tes
// for a todo list app that are designed to demons
// the power of writing tests in Cypress.
//
// To learn more about how Cypress works and
// what makes it such an awesome testing tool,
// please read our getting started guide:
// https://on.cypress.io/introduction-to-cypress

describe('example to-do app', () => {
  beforeEach(() => {
    // Cypress starts out with a blank slate for ea
    // so we must tell it to visit our website wit
    // Since we want to visit the same URL at the
    // we include it in our beforeEach function so
    cy.visit('https://example.cypress.io/todo')
  })

  it('displays two todo items by default', () => {
    // We use the `cy.get()` command to get all el
    // Then, we use `should` to assert that there
    // which are the two default items.
    cy.get('.todo-list li').should('have.length',
```

## Git – For Version Control Process (optional)

- For these lessons I will also show how to use Git for version control and as a CI/CD (continuous integration and development) tool and apply it in the videos
- However this is totally optional and not necessary to understand Cypress, you can just follow Cypress steps if you want
- For this you need to install Git on your machine, generate SSH keys and upload your public SSH key to your GitHub account for authorization. I provided some instructions links for these steps:
- https://devqa.io/install-git-mac-generate-ssh-keys/
- https://inchoo.net/dev-talk/how-to-generate-ssh-keys-for-git-authorization/

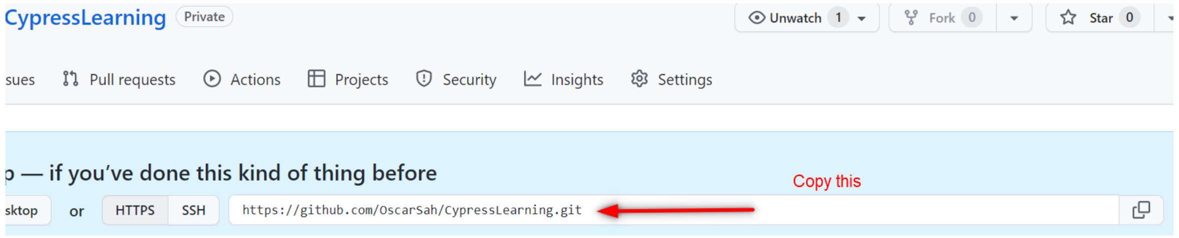
# Git – For Version Control Process (optional)

- Create repository on Github: go to your profile create a new repository (CypressLearning)



- In VSC terminal run following commands one by one

```
git init
git add .
git commit -m "add: first commit"
git branch -M main
git remote add origin <your_remote_origin>
git push -u origin main
```



https://github.com/OscarSah/CypressLearning.git

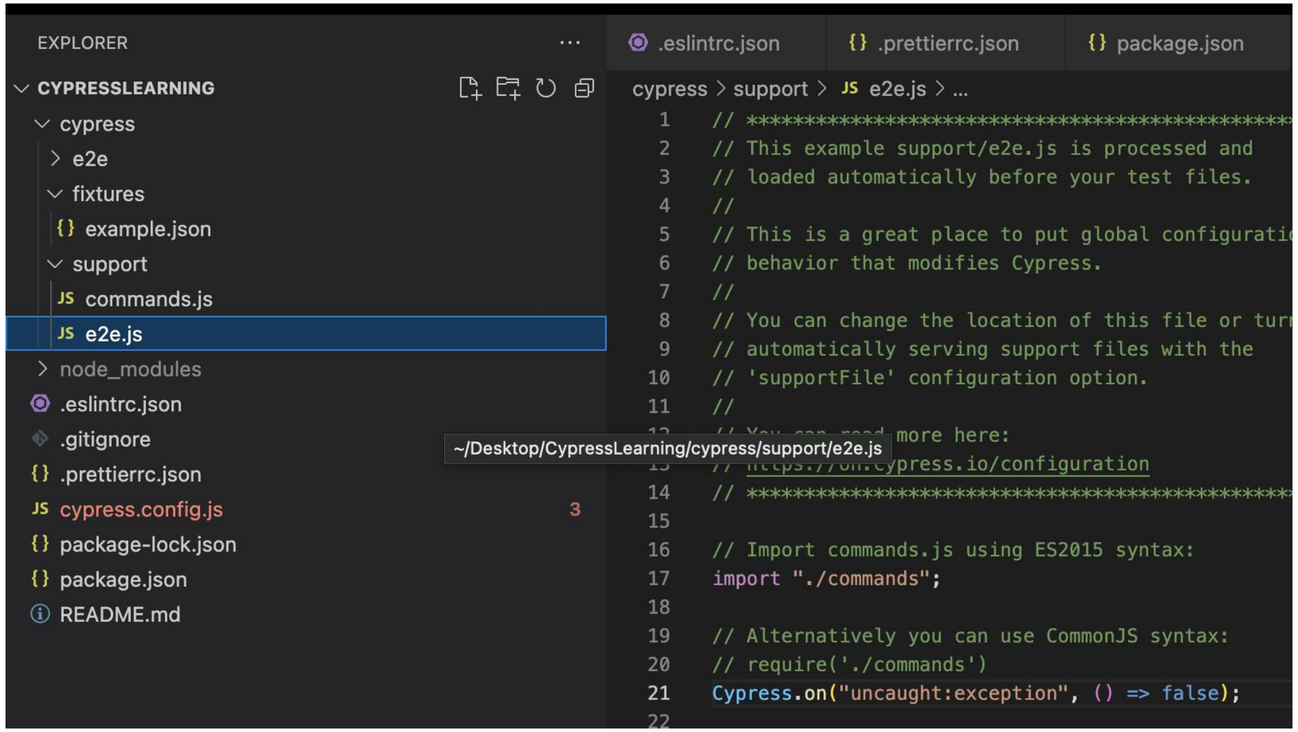
# Git – For Version Control Process (optional)

- Every time you finished your work, and you are sure everything is fine, push to Github repository. You will always run the same commands for purposes of this Lesson. No need to use anything more complex than this for now.
- In this case it is meaningful to run:

```
git add .
git commit -am "add: README"
git push
```

- So, every time you want to add something, lets say you added a new test - you will write commit message to be something like this add: name of the test . Or, if you fixed some test for example - fix: name of the test . That is a standard that many people follow in order to be able to easily search through commits when you are looking at git history.

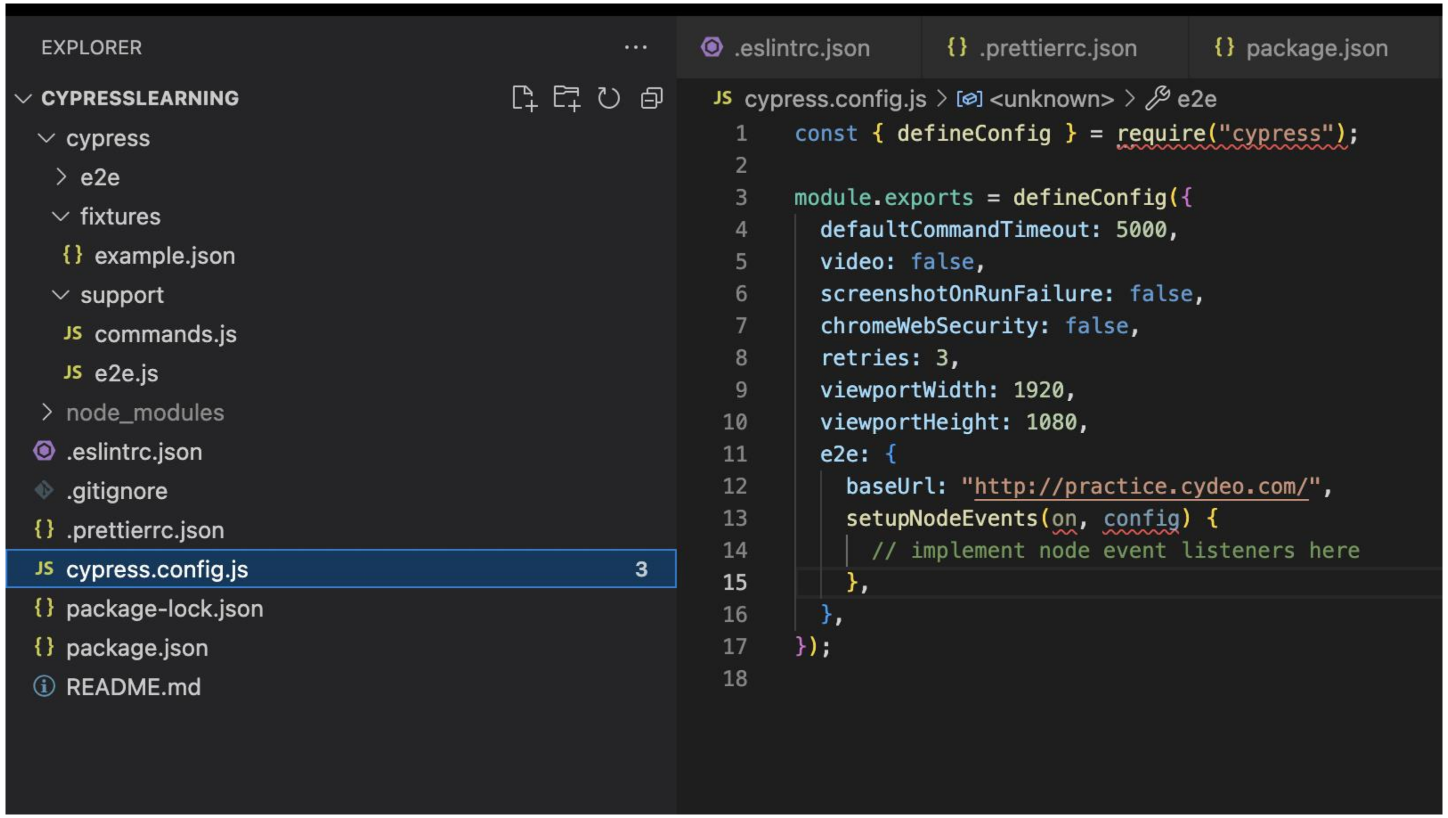
# General SetUp of Tests Framework



```
// ***********************************************
// This example support/e2e.js is processed and
// loaded automatically before your test files.
//
// This is a great place to put global configurati
// behavior that modifies Cypress.
//
// You can change the location of this file or tur
// automatically serving support files with the
// 'supportFile' configuration option.
//
// You can read more here:
// https://on.cypress.io/configuration
// ***********************************************

// Import commands.js using ES2015 syntax:
import "./commands";

// Alternatively you can use CommonJS syntax:
// require('./commands')
Cypress.on("uncaught:exception", () => false);
```

- Add error handler under support/e2e.js: The reason why we need this is because we don't want that our test execution fail because of console error events in the production application.

# General SetUp of Tests Framework



```javascript
const { defineConfig } = require("cypress");

module.exports = defineConfig({
  defaultCommandTimeout: 5000,
  video: false,
  screenshotOnRunFailure: false,
  chromeWebSecurity: false,
  retries: 3,
  viewportWidth: 1920,
  viewportHeight: 1080,
  e2e: {
    baseUrl: "http://practice.cydeo.com/",
    setupNodeEvents(on, config) {
      // implement node event listeners here
    },
  },
});
```

- Add Cypress configuration in cypress.config.js file

# Understanding Framework Folder structure

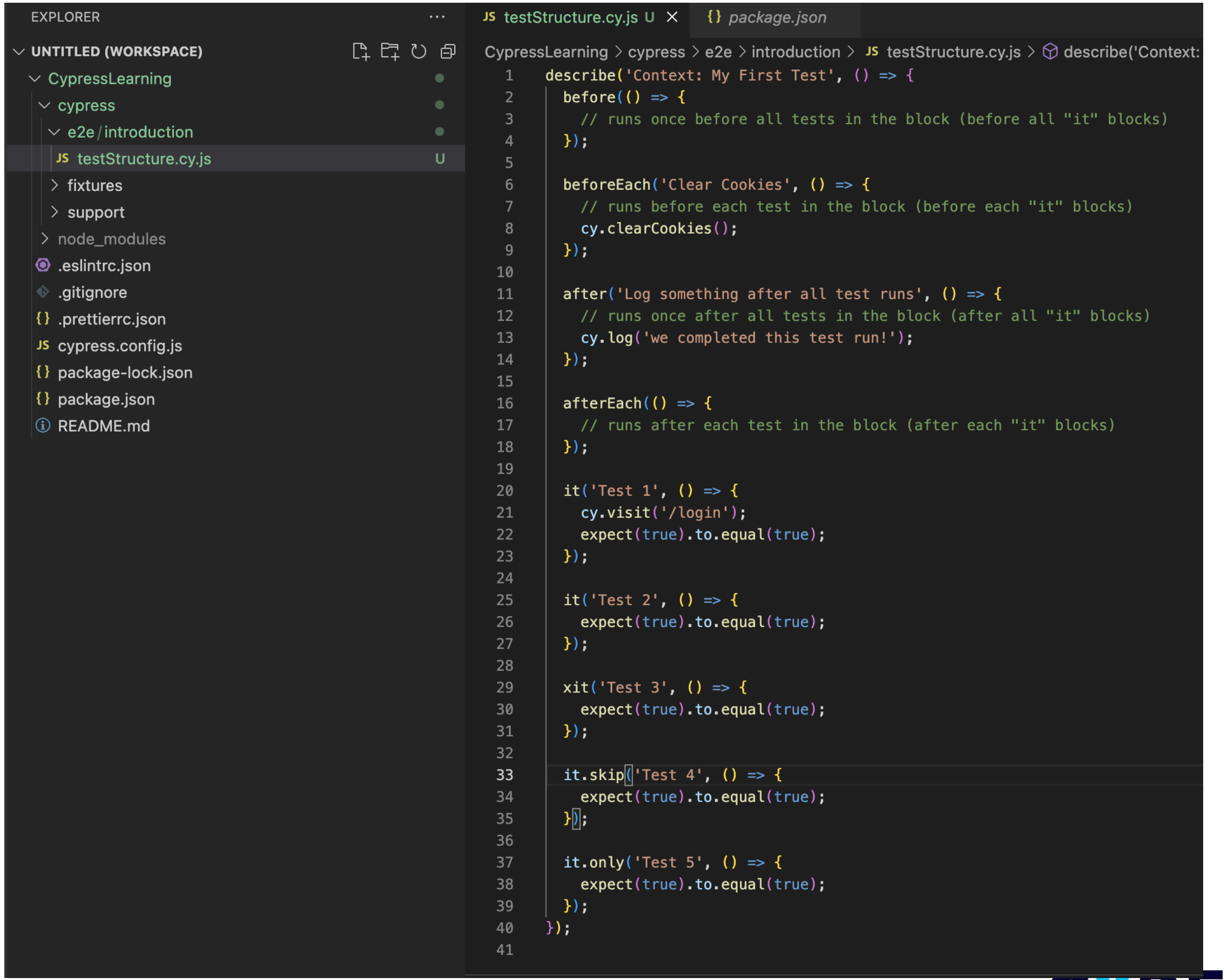
- After adding a new project, Cypress will automatically scaffold out a suggested folder structure. By default it will create:
  - /cypress.config.js
  - /cypress/fixtures/example.json
  - /cypress/support/commands.js
  - /cypress/support/e2e.js
- Test files are located in cypress/e2e by default.
- Fixtures are used as external pieces of static data that can be used by your tests.
- The support file is a great place to put reusable behavior such as custom commands or global overrides that you want applied and available to all of your spec files.

# Main constructs in Cypress test development

- describe(): It is is simply a way to group our tests.
- it(): We use it for an individual test case.
- before(): It runs once before all tests in the block.
- after(): It runs once after all tests in the block.
- beforeEach(): It runs before each test in the block.
- afterEach(): It runs after each test in the block.
- .only(): To run a specified suite or test, append ".only" to the function.
- .skip(): To skip a specified suite or test, append ".skip()" to the function. (you can also put “x” in front of it block to skip it)

# First Lesson
# Test Code

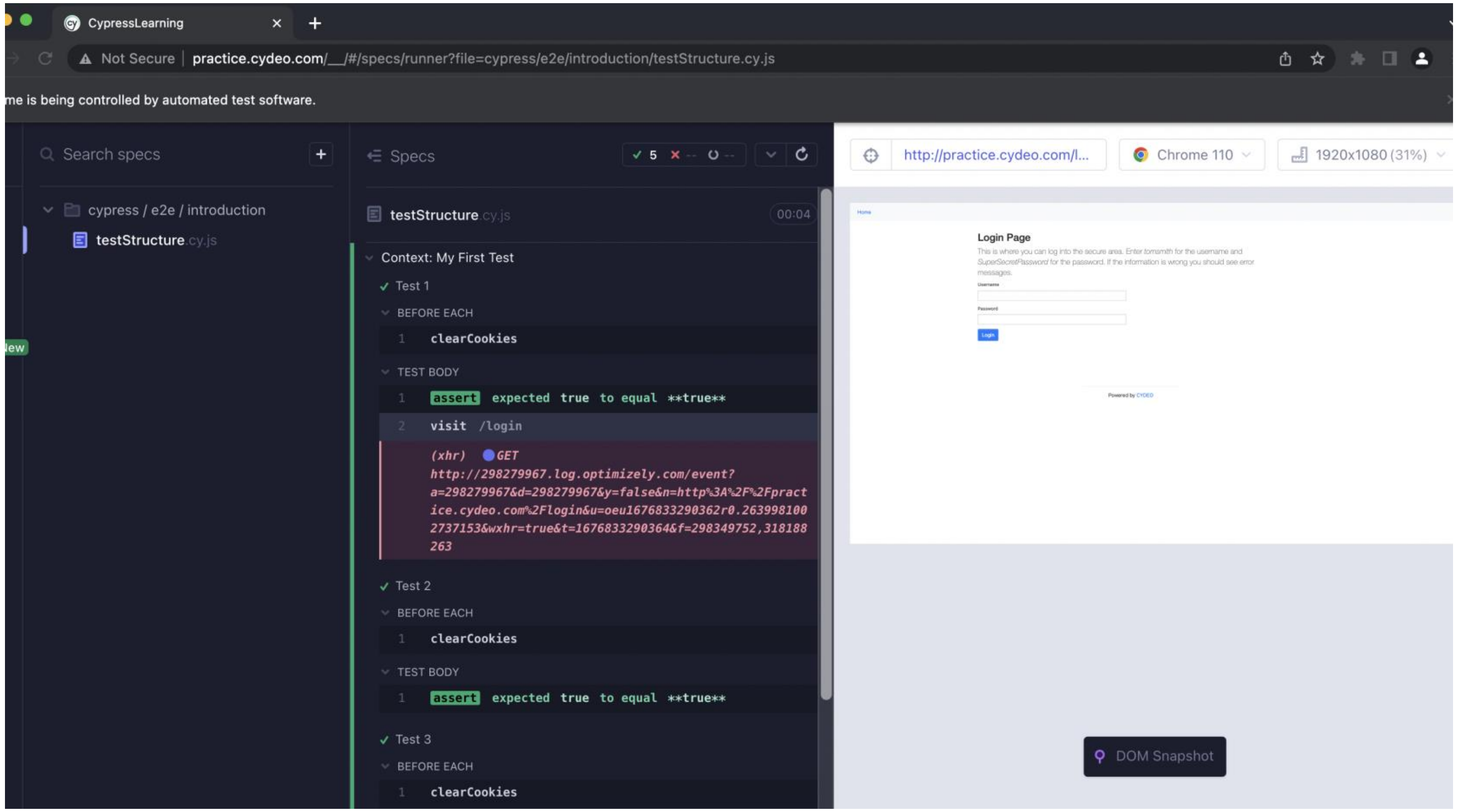
- Create a folder under e2e (introduction)
- Create a file inside introduction folder (testStructure.cy.js)
- Write following code



```javascript
describe('Context: My First Test', () => {
  before(() => {
    // runs once before all tests in the block (before all "it" blocks)
  });

  beforeEach('Clear Cookies', () => {
    // runs before each test in the block (before each "it" blocks)
    cy.clearCookies();
  });

  after('Log something after all test runs', () => {
    // runs once after all tests in the block (after all "it" blocks)
    cy.log('we completed this test run!');
  });

  afterEach(() => {
    // runs after each test in the block (after each "it" blocks)
  });

  it('Test 1', () => {
    cy.visit('/login');
    expect(true).to.equal(true);
  });

  it('Test 2', () => {
    expect(true).to.equal(true);
  });

  xit('Test 3', () => {
    expect(true).to.equal(true);
  });

  it.skip('Test 4', () => {
    expect(true).to.equal(true);
  });

  it.only('Test 5', () => {
    expect(true).to.equal(true);
  });
});
```

## Running your tests on the Cypress CLI

- At VSC Terminal:  npm run cypress-cli



```
CypressLearning
Not Secure | practice.cydeo.com/__/#/specs/runner?file=cypress/e2e/introduction/testStructure.cy.js
me is being controlled by automated test software.

Search specs
cypress / e2e / introduction
  testStructure.cy.js

Specs                         ✓ 5  ✗ --  ◌ --
testStructure.cy.js                                00:04
Context: My First Test
✓ Test 1
  BEFORE EACH
    clearCookies
  TEST BODY
    assert expected true to equal **true**
    visit /login
    (xhr) GET
    http://298279967.log.optimizely.com/event?
    a=298279967&d=298279967&y=false&n=http%3A%2F%2Fpract
    ice.cydeo.com%2Flogin&u=oeu1676833290362r0.263998100
    2737153&wxhr=true&t=1676833290364&f=298349752,318188
    263
✓ Test 2
  BEFORE EACH
    clearCookies
  TEST BODY
    assert expected true to equal **true**
✓ Test 3
  BEFORE EACH
    clearCookies

http://practice.cydeo.com/l...   Chrome 110   1920x1080 (31%)
Login Page
DOM Snapshot
```

CYDEO