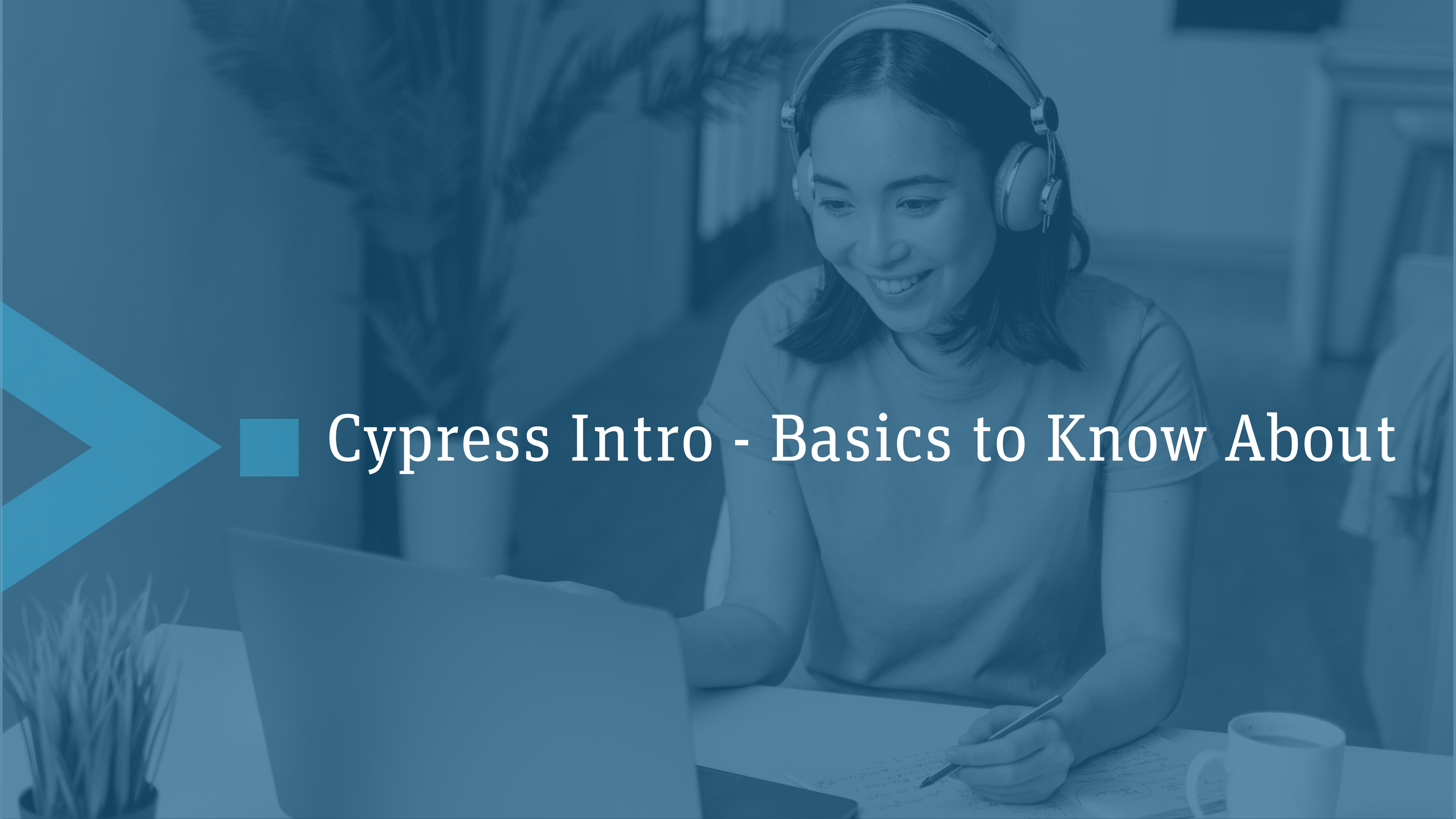
CYDEO

Cypress Training



Cypress Intro - Basics to Know About

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

What is Cypress? - 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 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.

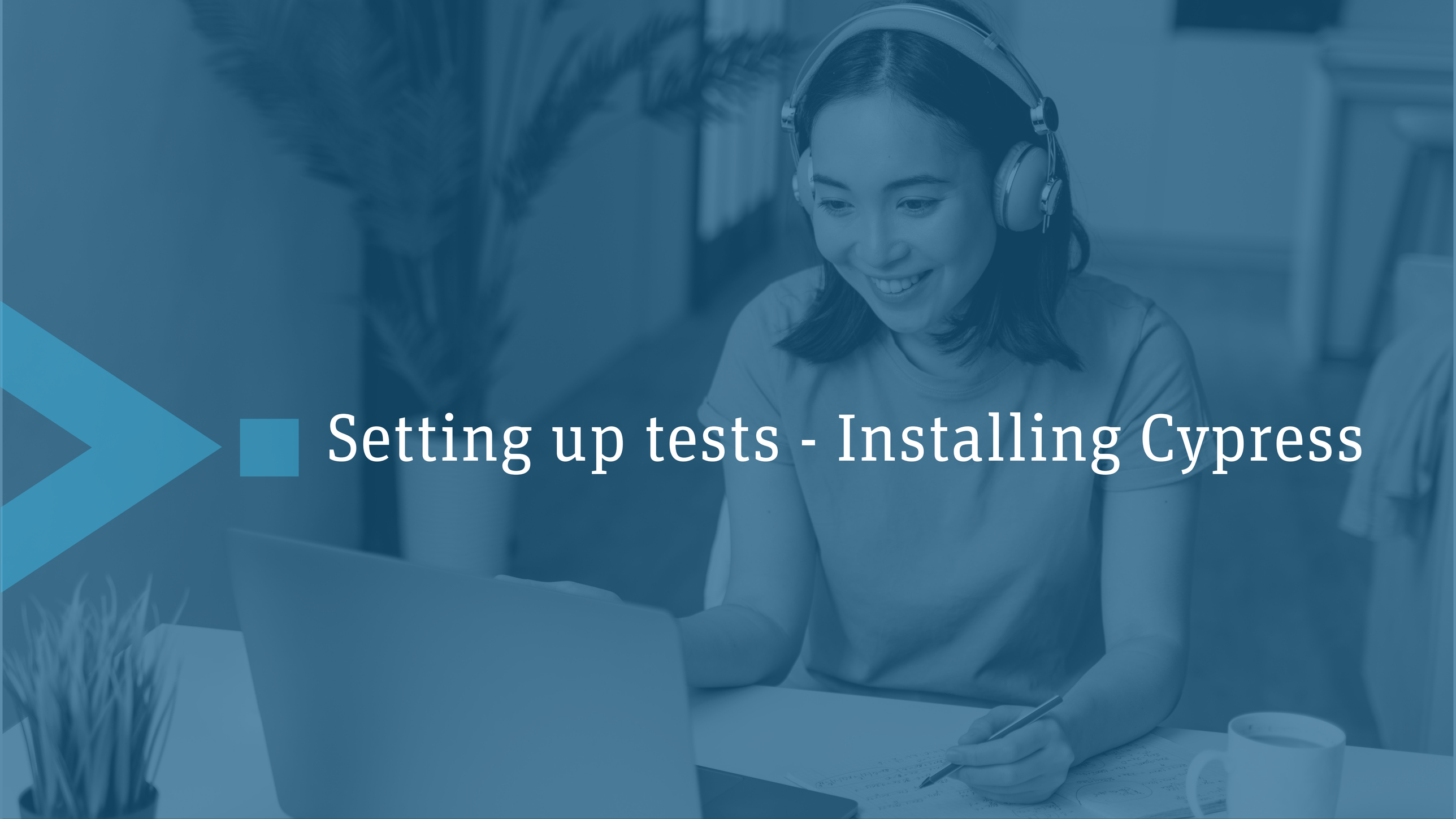
Cypress processes respond to the application's events and processes command in real-time. With real-time reloads in Cypress, tests are reloaded automatically as and when changes are made in the app.

Major Advantages Of Cypress

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.

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.

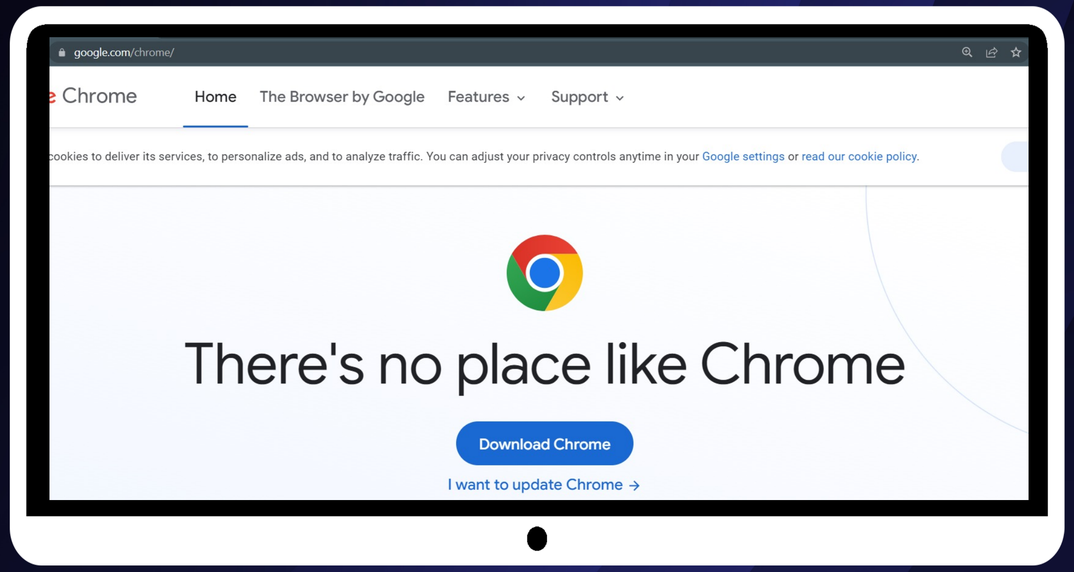


Setting up tests - Installing Cypress

Setting up tests

Download Google Chrome (if you already do not have it)

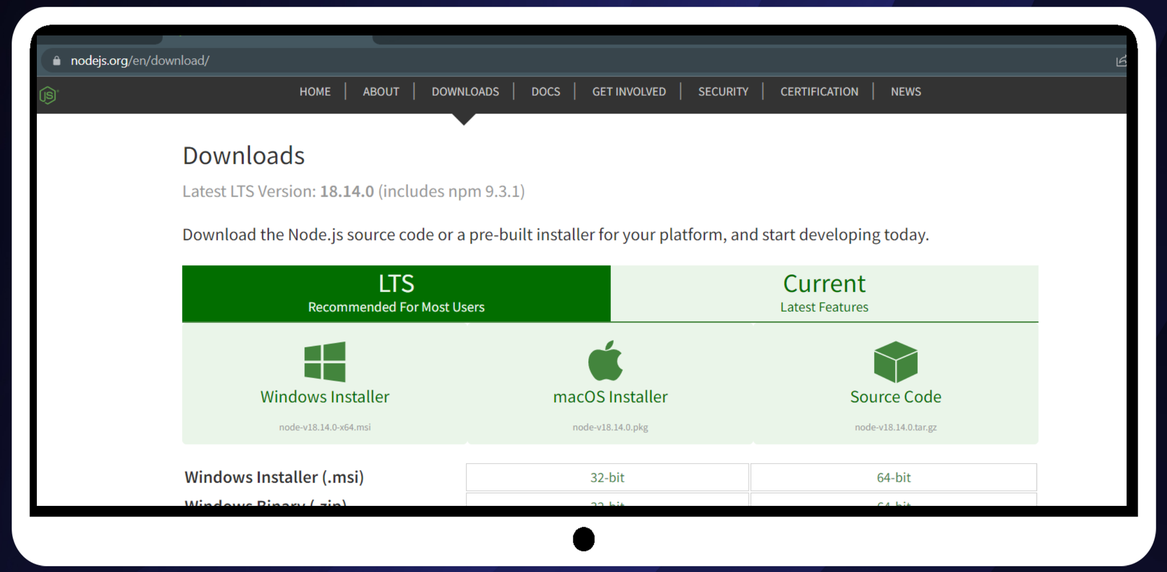
https://www.google.com/chrome/



Setting up tests

Download and install Nodejs - https://nodejs.org/en/download/

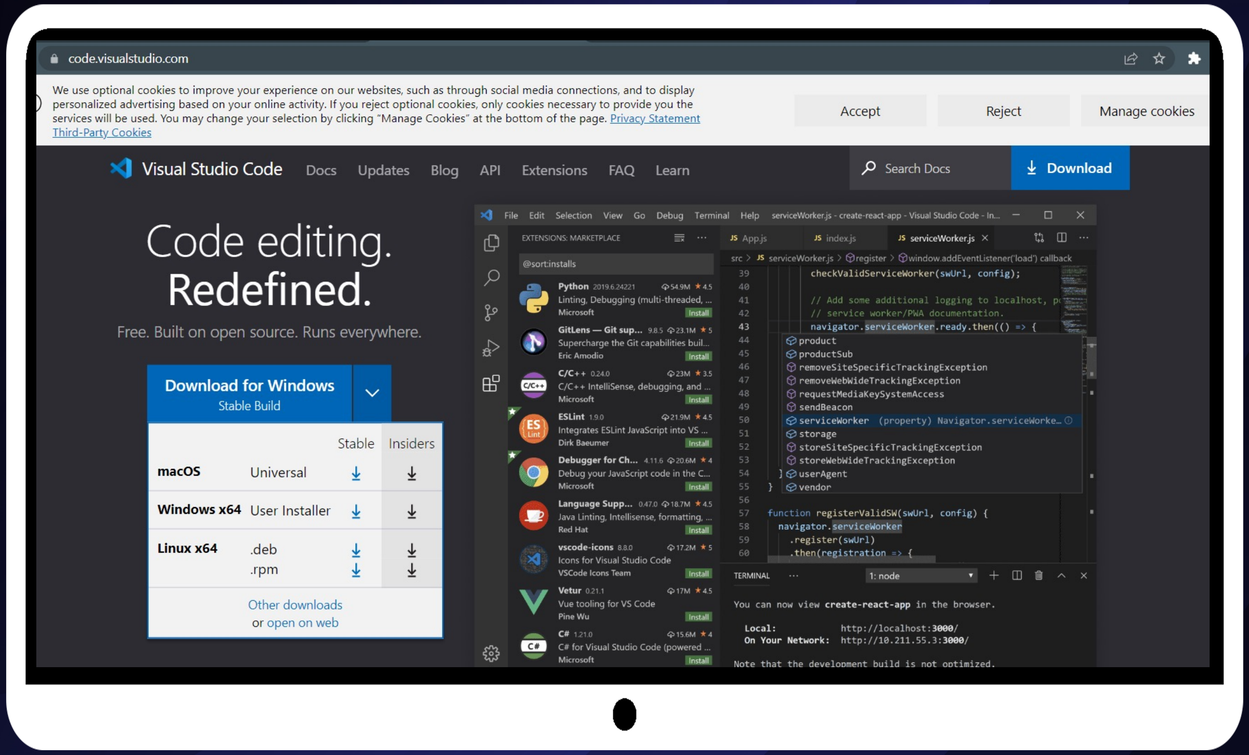
Make sure to install **LTS Version** for macOS or Windows



Setting up tests

Download and install Visual Studio Code (VSC) Editor - https://code.visualstudio.com/

Make sure to install Stable Version for macOS or Windows



Setting up tests - Installing Cypress

MacOS Installation

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

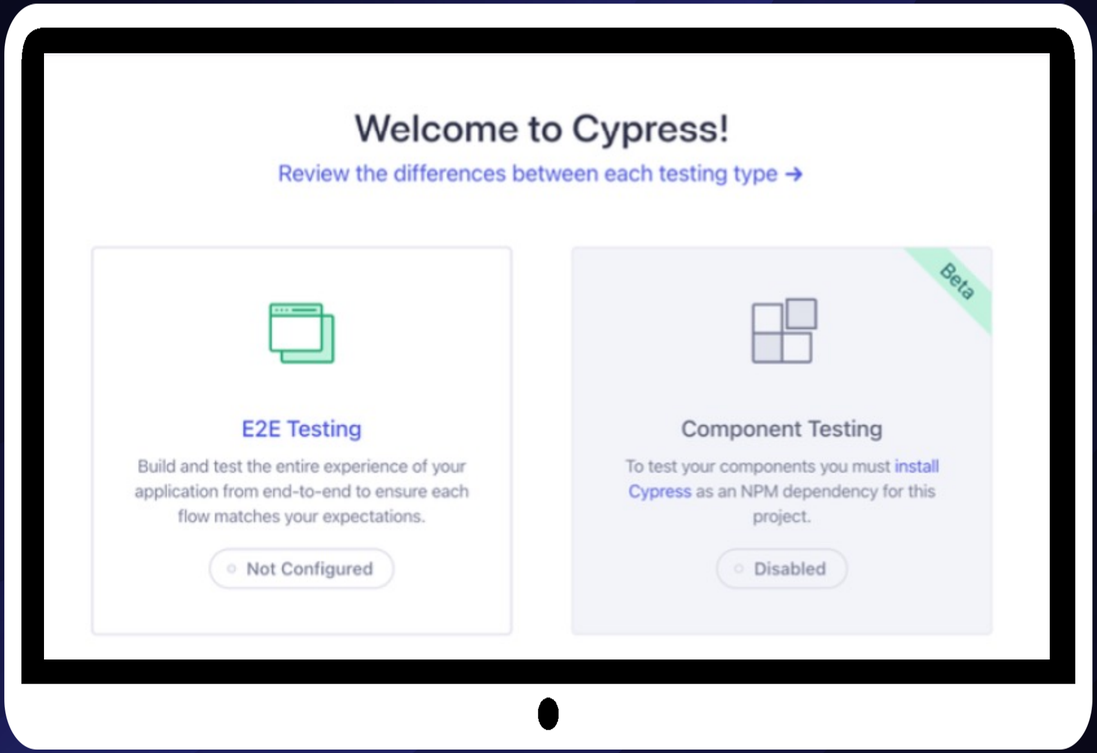
Windows Installation

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

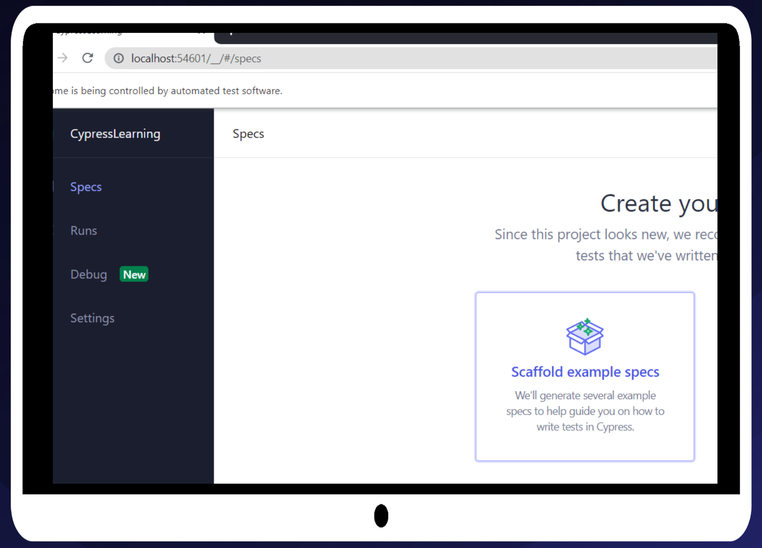
Open Cypress CLI application that you previously installed Chrome.

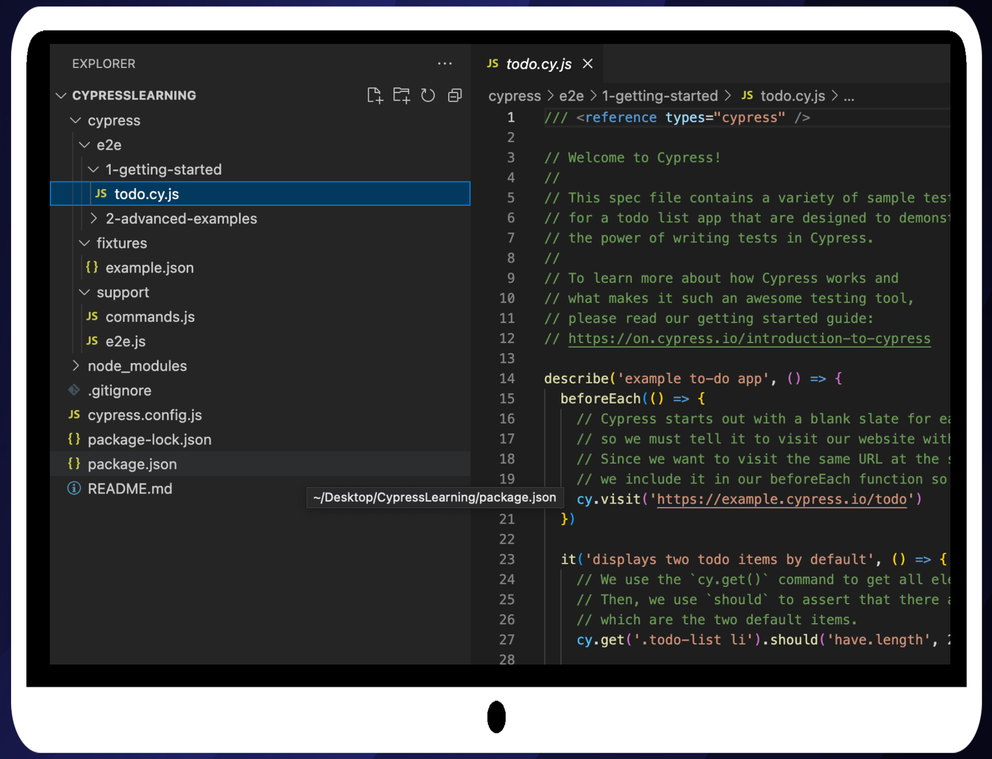
You should see something like this and select E2E, then click next, this will install example framework structure to your project



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)





Git – For Version Control (optional)

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/

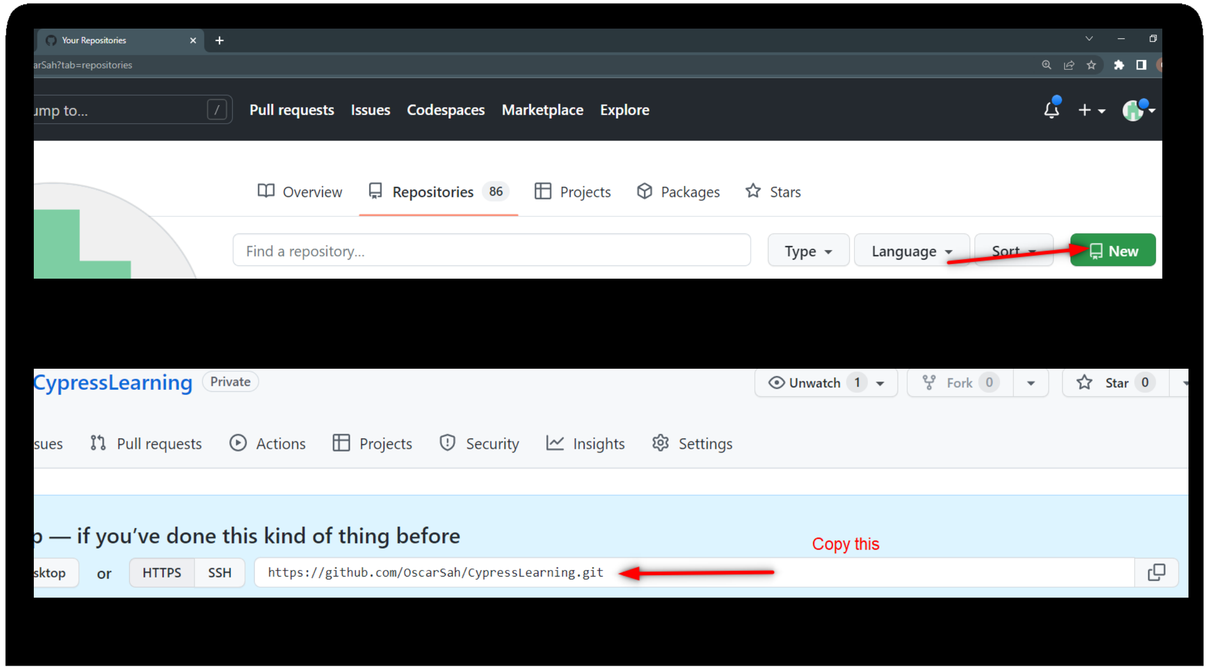
https://inchoo.net/dev-talk/how-to-generate-ssh-keys-for-git-authorization/

Git – For Version Control Process (optional)

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
```

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



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.

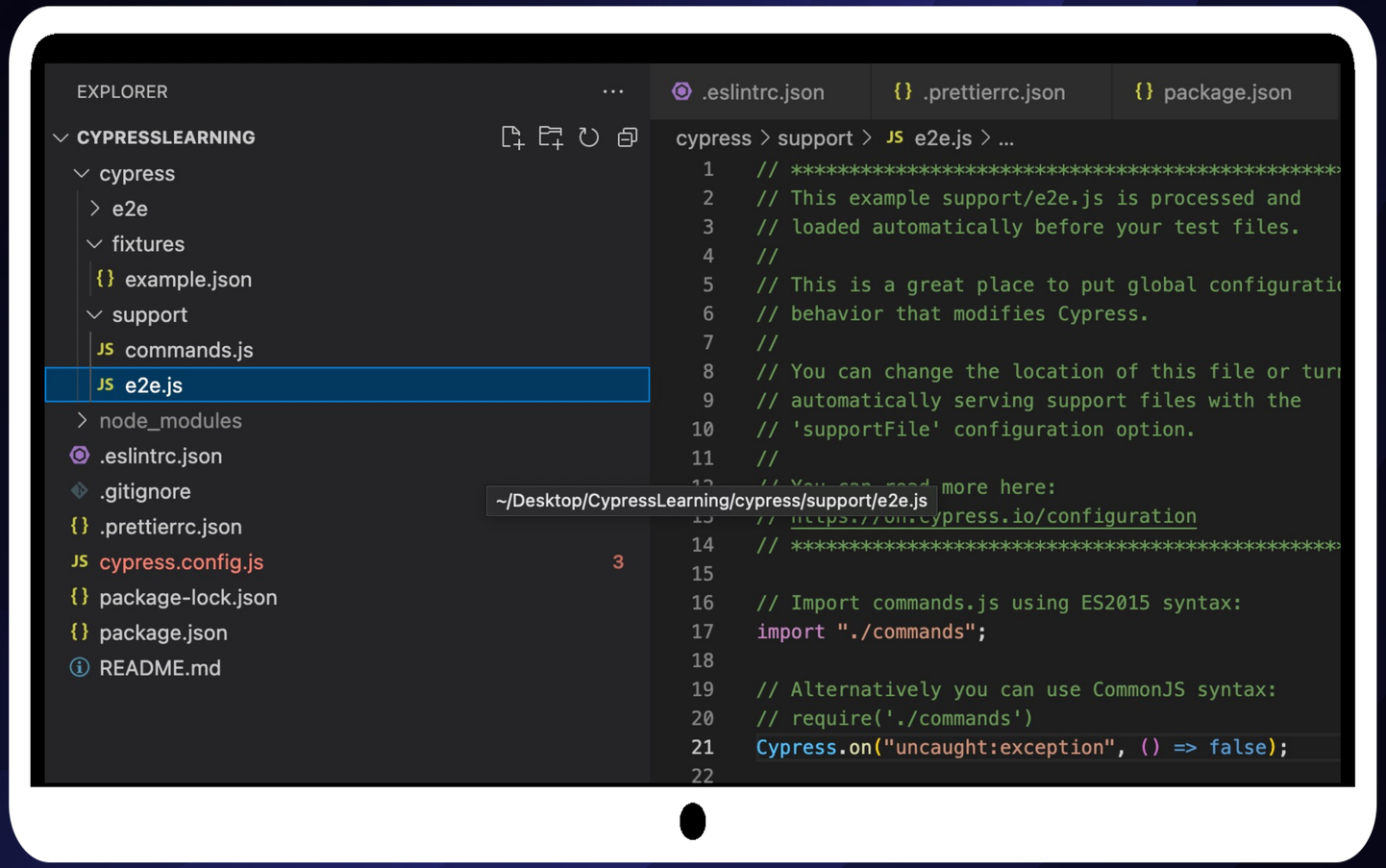
Cypress project and global configuration

General Set-Up of Test Framework

Add error handler under support/e2e.js:This is a great place to put global configuration and behavior that modifies Cypress.

```
Cypress.on('uncaught:exception', () => false);
```

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.

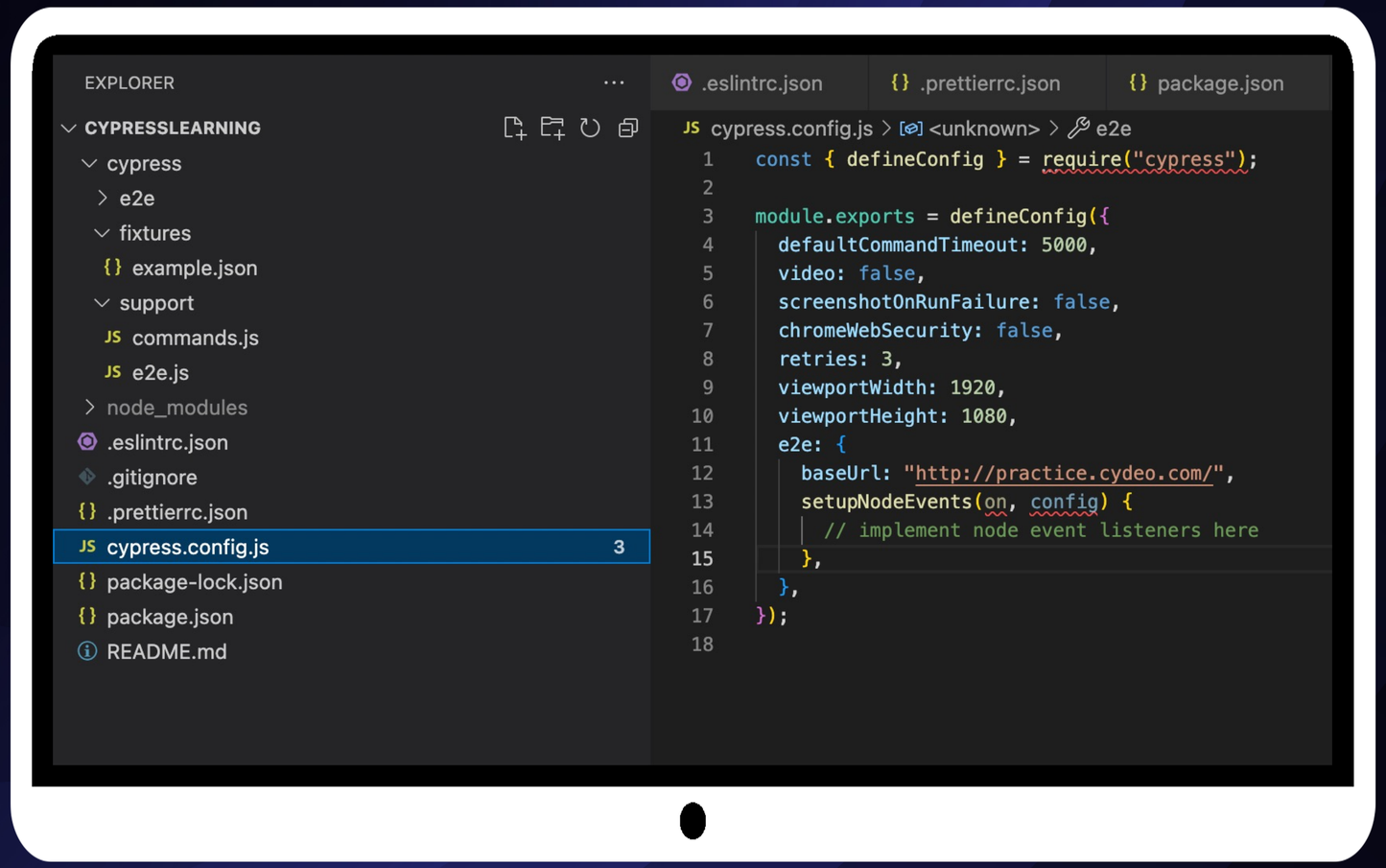


General Set-Up of Test Framework

```
baseUrl: 'https://practice.cydeo.com/',
```

Add Cypress configuration in cypress.config.js file

The reason why we need this is because we are defining a base URL that all our Project Tests will use globally.



Configuration in cypress.config.js file

```
module.exports = defineConfig({
e2e: {
baseUrl:
'https://practice.cydeo.com/',
env: {
login:"/login",
},
video: false,
retries: 1,
defaultCommandTimeout: 5000,
viewportHeight: 800,
viewportWidth: 1200,
setupNodeEvents(on, config)
```

Main Test Framework Construct

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)

Main constructs in Cypress test development

Specs

✓ 5 ✗ -- ↻ --

testStructure.cy.js 305ms

Context: My First Tests

✓ Opening a web application

BEFORE ALL

```
1  log  runs once before all test cases in this describe block
```

BEFORE EACH

```
1  log  run before each test case
2  clearCookies
```

TEST BODY

```
1  visit  /registration_form
```

AFTER EACH

```
1  log  run after each test case
```

```
(xhr)  GET https://298279967.log.optimizely.com/event?
a=298279967&d=298279967&y=false&n=https%3A%2F%2Fpractice.cydeo.com%2Fregistratio
n_form&u=oeu1694952596414r0.3929178285983246&wxhr=true&t=1694952596414&f=2983497
52,318188263
```

✓ Test 2

https://practice.cydeo.com/registration_form

Home

Registration form

First name

first name

Last name

last name

Username

username

Email address

email@email.com

Password

Phone number

571-000-0000

Gender

○ Male

○ Female

○ Other

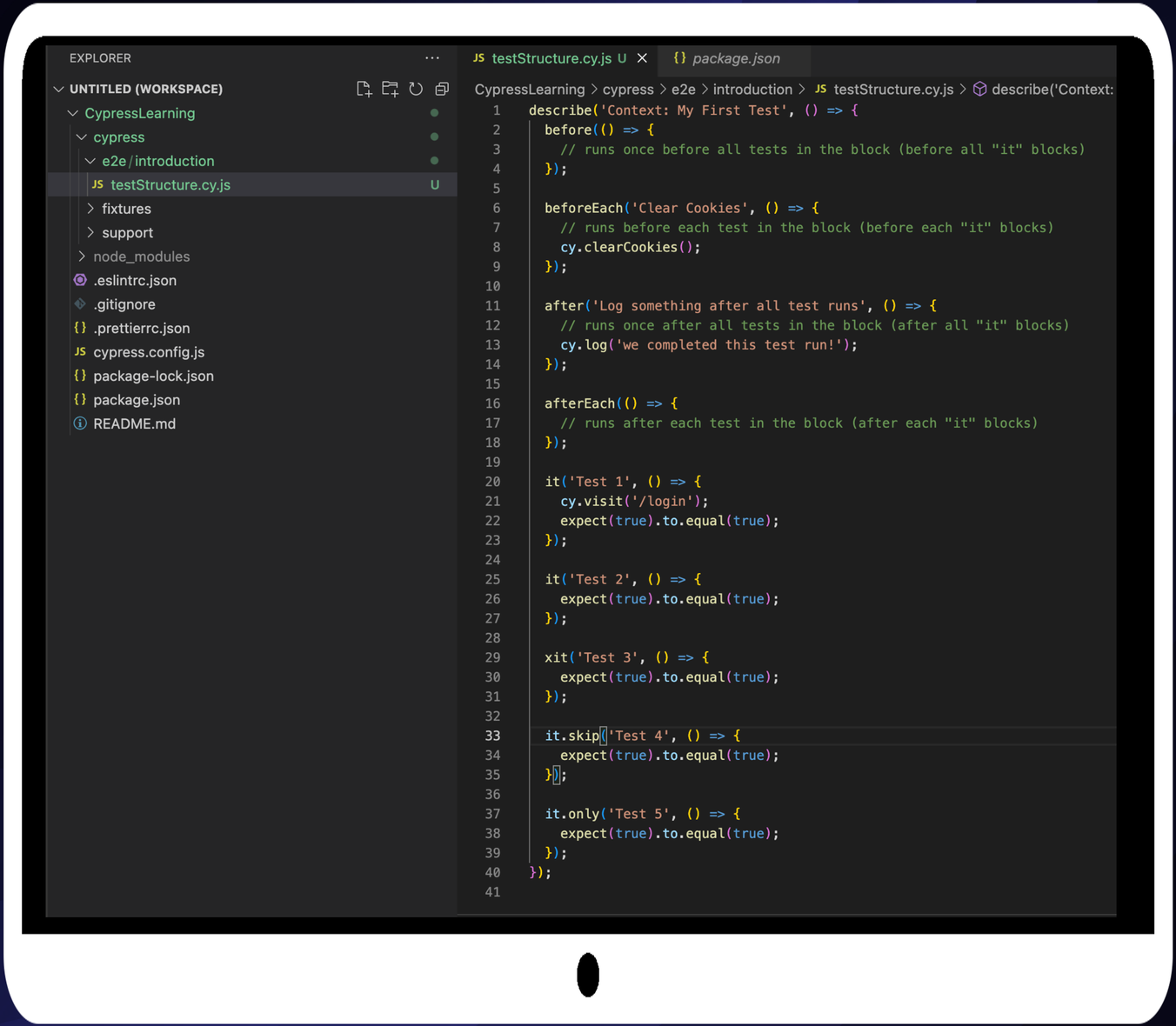
Date of birth

First Lesson Test Code

Create a folder under e2e (introduction)

Create a file inside introduction folder (testStructure.cy.js)

Write following code



Running your tests on the Cypress CLI

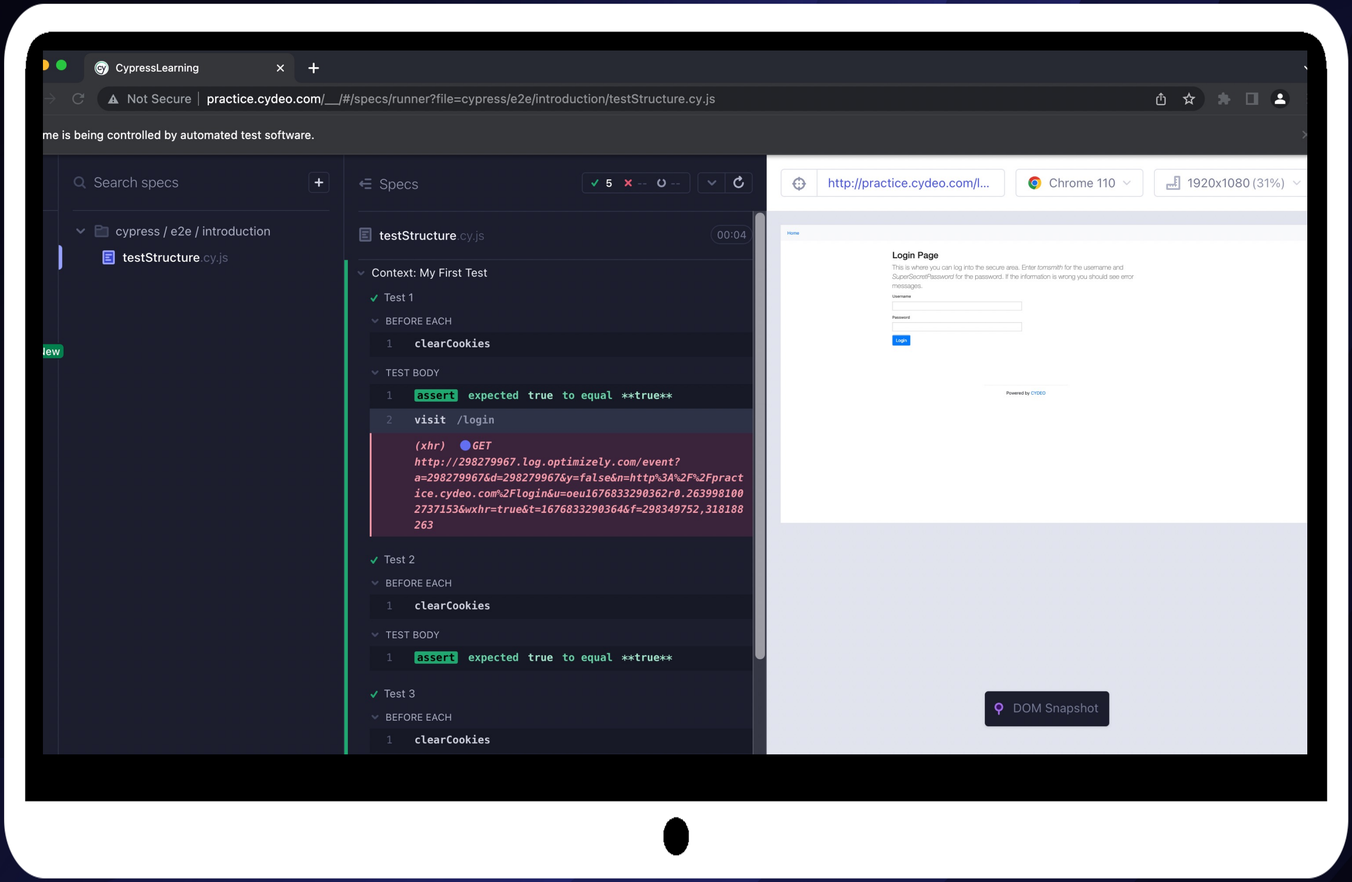
First Option

At VSC Terminal:
npm run cypress-cli

First Option

At VSC Terminal:

npx cypress open



DOM Elements and Locators

DOM Elements

In simple terms, any web page that you encounter is a document. Viewing the document as an HTML source or in a browser window is just an approach of presenting one thing in different ways.

The document object model (DOM) represents a webpage as objects to enable its manipulation. With the document-oriented model, a scripting language such as JavaScript may modify it by targeting the objects.

When reading HTML code, it is important to differentiate elements, such as tags, attribute names, attribute values etc.

What are Cypress Locators?

A Selector or Locator is an object that finds and returns web page items/elements on a page based on the query.

In Automation, to perform any operation on the web elements first, you need to locate an element and perform an action on that element.

The Locator or Selector helps to locate an element in the webpage. There are different types of locators, such as id, CSS, XPath, tag-based selectors, etc.

What are Cypress Locators?

Cypress supports various locators such as tags, id, class, attributes, text, etc.

Cypress also supports XPath Selectors; however, it needs installation of the third-party plugin cypress-xpath. Xpath has not been supported out of the box in the Cypress framework.

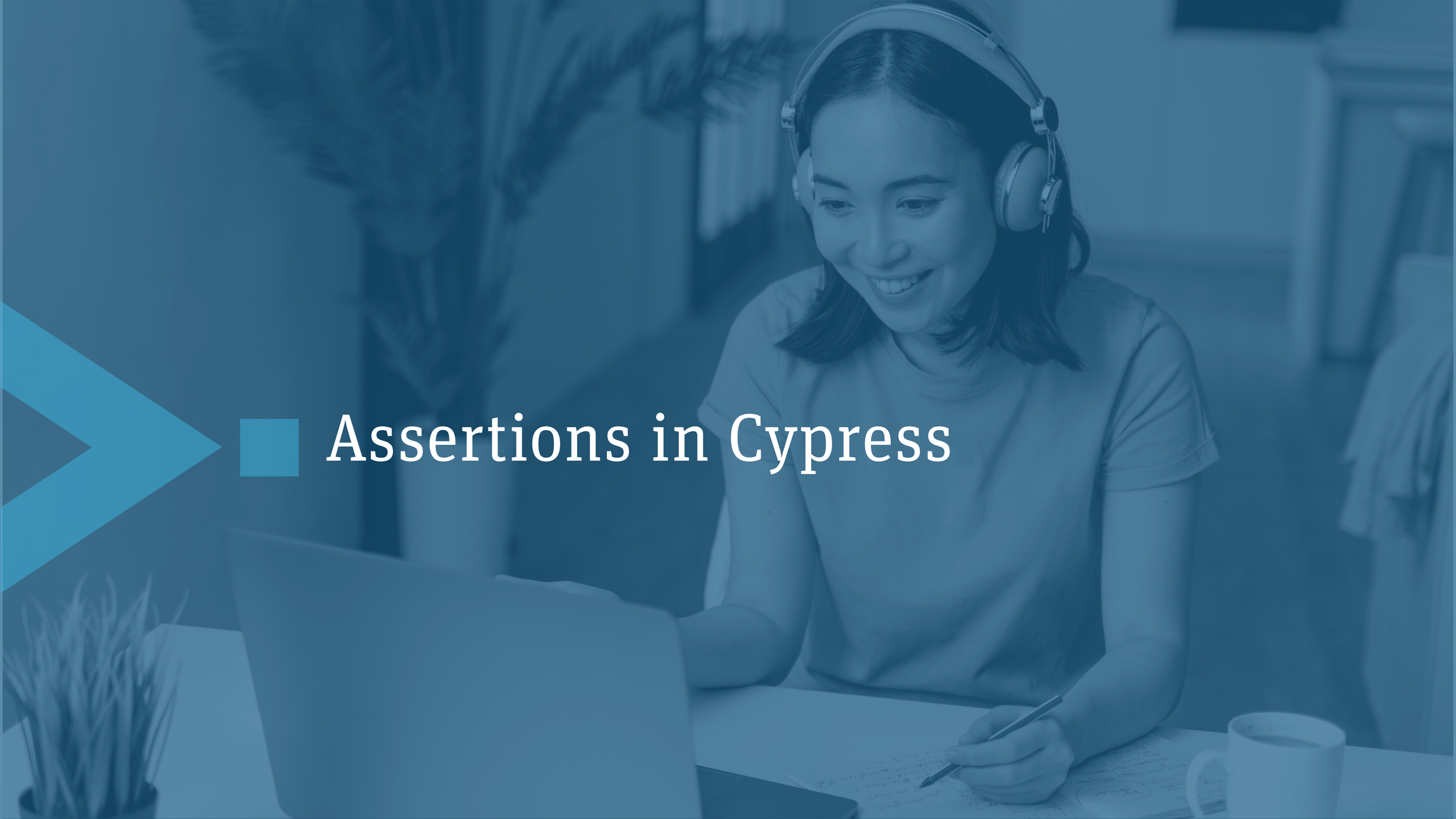
To fetch the HTML element in Cypress using different types of locators cy.get() method is used.

Why do we need Locators?

We need Locators in order to manipulate elements on the page, to interact with them as user would manually do.

For example: in order to click on certain element, or type a text, you have to first locate exactly which element that is on the page.

- Create another test file under /e2e/intro folder and call it for example locators.cy.js
- Locators: Get elements by different locator strategies
- Navigate to test page: /login



Assertions in Cypress

Types of Cypress Assertions

Cypress integrates multiple assertions from various JS assertion libraries such as Chai, jQuery, etc.

We can broadly classify all of these assertions into two segments based on the subject on which we can invoke them:

- Implicit Assertions: When the assertion applies to the object provided by the parent chained command, it's called an Implicit assertion.
- Explicit Assertions: When there is a need to pass an explicit subject for the assertion, it falls under the category of Explicit assertion.

Implicit Assertions

This category of assertions generally includes commands such as ".should()" and ".and()"

As these commands don't stand independently and always depends on the previously chained parent command, they automatically inherit and acts on the object yielded by the previous command.

Generally, we use Implicit assertions when we want to:

- Assert multiple validations about the same subject.
- Alter the subject before making the assertions on the subject.

Explicit Assertions

This category of assertions contains the commands such as "expect()" and "assert()", which allow you to pass an explicit subject/object.

Generally, you will be using "Explicit assertions" when you want to:

- Perform some custom logic before making the assertions on the given subject.
- Perform multiple assertions against the same subject after applying custom logic.

Elements Manipulation in Cypress

Input Fields

cypres works in a chainable functions structure

Navigate to registration page on Cydeo practice app

- Check different input box fields and verify
- Check different radio button actions
- Check selection of all select dropdowns options

Button Elements

When we locate more than one elements, we can loop through the elements with .each() method

Navigate to registration page on Cydeo practice app

- select a button with text
- locate multiple elements with class or tagName attribute and create a list then select 3rd element from the list
- Use cy.log() command to write the text at the test console

WebTables

If you need to navigate to a URL other than your baseUrl, you define it at describe block or in the it block

Change the BaseUrl that is defined in the config file with baseUrl: 'https://demoqa.com'

- Navigate to WebTables, locate table body - then update an info with a different one

- We use Cypress Fixtures files for Test Inputs

Handling Alerts with Cypress

Browser Commands, window:alert, window:confirm, window:on etc...

We create a stub function to handle Alerts

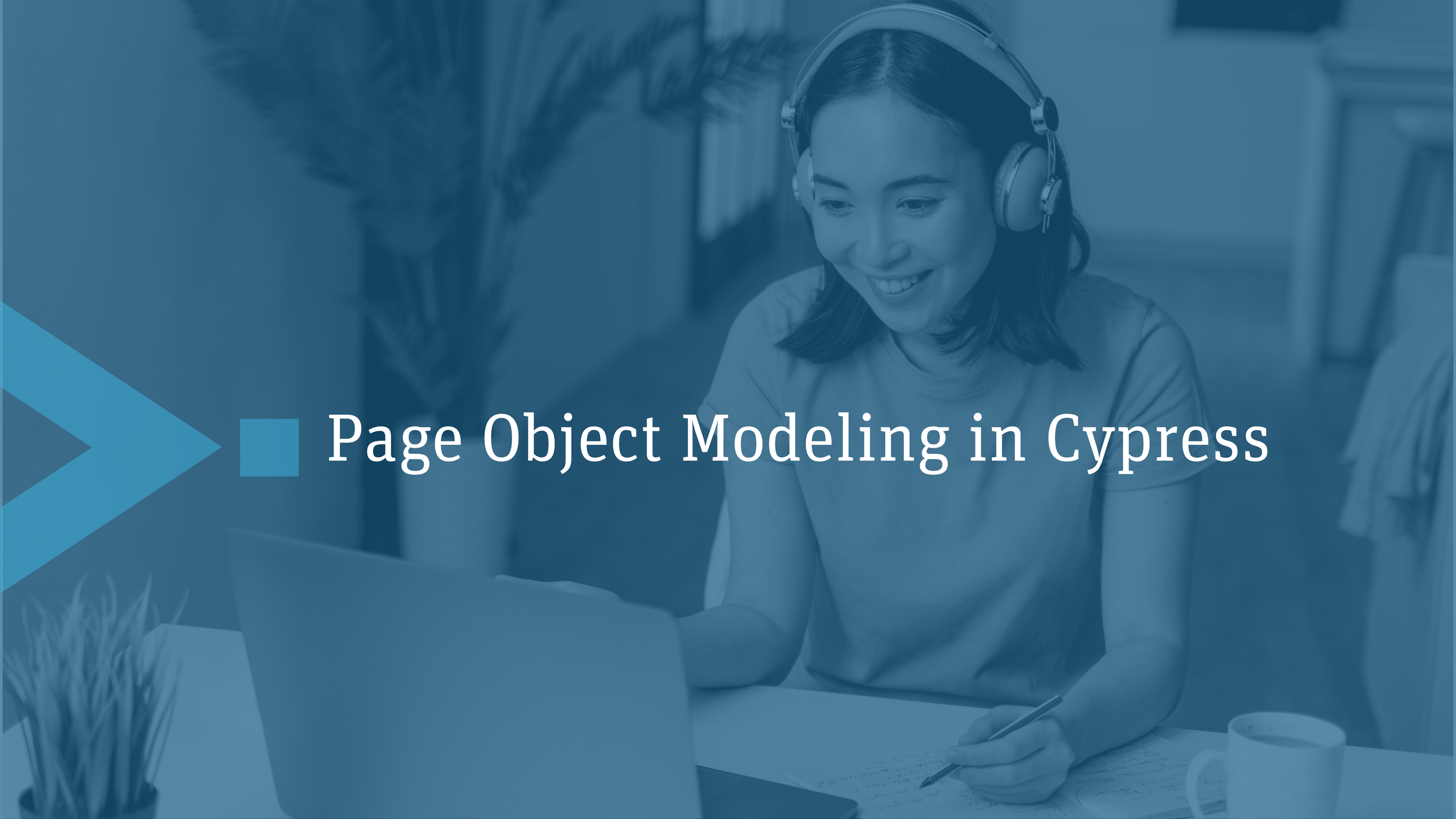
cy.on('window:confirm', stub); = > when this confirmation command initiated store and give the control to stub function

File Upload Action in Cypress

Follow these steps in order to upload a file and test result

In order to upload files in Cypress we need to install plugin, import necessary command to our project

- Run npm install -dev cypress-file-upload
- Inside support folder we have commands.js file, add import 'cypress-file-upload';
- The file that you want to upload should be in your fixture folder



Page Object Modeling in Cypress

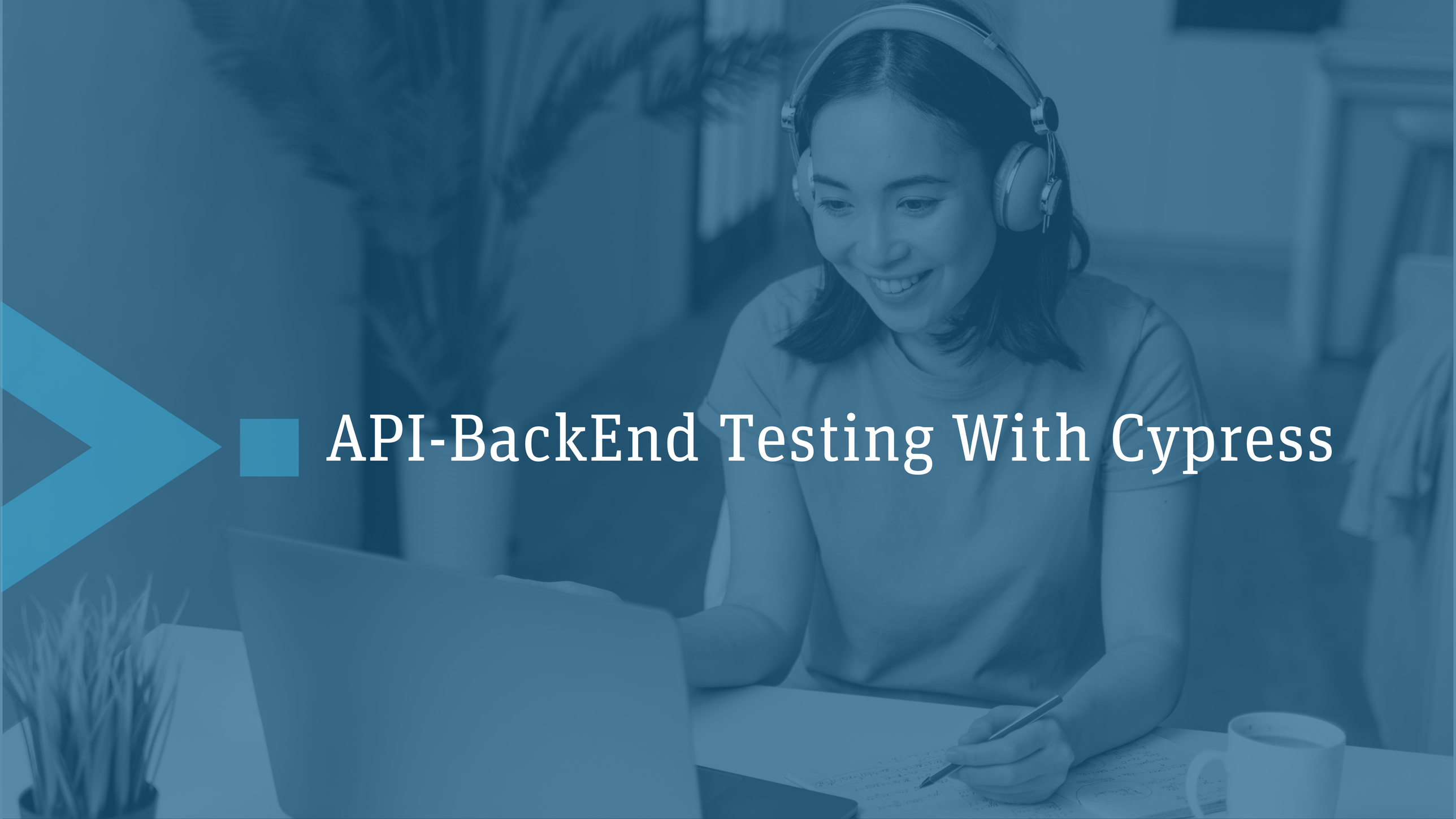
Creating and Using Page Objects in Cypress

In Cypress, page objects are created as Classes. And we can have more than one class in a file. And name does not need to be same with file name.

We will create locators as functions of javascript that finds elements and returns them.

We will export locators and utiltiy functions from page object classes to our tests with the Objects created. And tests will import these objects and use locators and functions

Custom timeouts can be used inside locator functions.



API-BackEnd Testing With Cypress

Simple Request-Response Structure

We use cy.request() method from Cypress to make request to an API end point

Cy.request() method takes a json object as function parameter which consists of our request body; url, path, HTTP method etc.

We can define URL and Path in the cypress config file instead of hard coding the parameters

```
e2e: {
  baseUrl: 'https://practice.cydeo.com/',
  env: {
    login:"/login",
    apiUrl:"https://demoqa.com",
    apiBooks:"/BookStore/v1/Books",
    generateUser:"/Account/v1/User",
    generateToken:"/Account/v1/GenerateToken",
    loginAPI:"/Account/v1/Login"
  },
```

Running API for Pre-Steps of UI Env

At the frontend testing we can integrate API request for our Pre-Test Conditions; such as Login.

This approach enables our tests to run faster with Cypress.

In the example we will create a random user, generate a token for it so we can log into book application, then we will verify the user and delete it from the system; all in a same test run. UNBELIEVABLE !!!

CI/CD Processes with Cypress OPTIONAL

Setting Up Git Environment For Project

For this step you need to watch Optional Git Installation for our Cypress Project

Creating YML File For CI/CD Process

In order to control CI/CD process with GitHub Actions capability we need to create a '.yml' file

We will trigger the actions with 'on' command and define which 'jobs' needs to be running

- The schedule event allows you to trigger a workflow at a scheduled time.
- 'push' command runs your workflow when you push a commit or tag.