### **3.2 Handling Invisible Elements**

#### **3.2a Identifying Invisible Elements**

Handling invisible elements in Cypress can often be tricky because you might run into situations where elements are not interactable or visible on the page. Cypress provides robust debugging tools to understand why an element is not visible or interactable. Here’s a step-by-step guide and code example to handle such scenarios.

### **Understanding Logs in Cypress**

Cypress automatically logs detailed information about each command it executes. You can use these logs to understand why an element might not be visible or interactable. Common reasons include:

* The element is hidden by CSS.
* The element is off the screen.
* The element is covered by another element.
* The element is not yet rendered.

### **Step-by-Step Guide**

1. **Check the Visibility**: Use Cypress commands to check if the element is visible or hidden.
2. **Debug Logs**: Use Cypress logs to understand the state of the element.
3. **Force Actions**: If necessary, force the action to be performed on the element even if it is not visible.

### **Code Example**

#### **1. Checking Visibility**

cy.get('#invisible-element').should('be.visible');

#### **3.2c Understanding Cypress Logs**

#### **Using Debug Logs**

Cypress logs every command to the Command Log, which you can see in the Cypress Test Runner. To get more details, you can use the .debug() command:

cy.get('#invisible-element').debug().should('be.visible');

#### **3. Forcing Actions**

If you need to interact with an element that is not visible, you can use the { force: true } option:

cy.get('#invisible-element').click({ force: true });

#### **3.2b Interacting with Invisible Elements**

● Learn techniques to interact with elements that are not immediately visible, such as scrolling into view or making elements visible before interacting.

● Gain knowledge of how to use Cypress logs to debug issues related to invisible elements and other test failures.

#### **3.2d Using .should('be.visible')**

● Learn to use the .should('be.visible') assertion to ensure that elements are visible before interacting with them.

**Hooks in Cypress**

Hooks in Cypress are functions that run at specific points in your test lifecycle, allowing you to perform setup, teardown, and common actions that need to be repeated across multiple tests. Cypress supports the following hooks:

1. before: Runs once before all tests in the block.
2. beforeEach: Runs before each test in the block.
3. afterEach: Runs after each test in the block.
4. after: Runs once after all tests in the block.

**Sample Code :**

describe('Hooks in Cypress', () => {

// This runs once before all tests in the block

before(() => {

cy.log('This runs once before all tests');

// Setup code like database seeding or setting up initial application state

});

// This runs before each test in the block

beforeEach(() => {

cy.log('This runs before each test');

// Code to reset application state before each test

cy.visit('https://example.cypress.io/commands/actions');

});

// This runs after each test in the block

afterEach(() => {

cy.log('This runs after each test');

// Code to clean up after each test

});

// This runs once after all tests in the block

after(() => {

cy.log('This runs once after all tests');

// Cleanup code like deleting test data

});

it('should click a button', () => {

// Your test code here

cy.get('.action-btn').click();

cy.get('.action-btn').should('have.class', 'active');

});

it('should type into an input field', () => {

// Your test code here

cy.get('.action-email')

.type('test@example.com')

.should('have.value', 'test@example.com');

});

it('should submit a form', () => {

// Your test code here

cy.get('.action-form')

.find('[type="text"]').type('Cypress')

.get('[type="submit"]').click();

cy.get('.action-form').should('contain', 'Your form has been submitted!');

});

});

### 

### **3.3 Cypress Inbuilt Plugin in Test Runner to Generate Locators**

#### **3.3a Using Cypress Inbuilt Plugin**

● Learn how to use the Cypress inbuilt plugin in the test runner to generate locators for web elements.

#### **3.3b Inspecting Elements**

● Understand how to inspect web elements using the Cypress Test Runner to identify locators.

#### **3.3c Generating Locators**

● Gain proficiency in generating and copying locators directly from the Cypress Test Runner for use in test scripts.

### **3.4 Cypress Commands**

#### **3.3a Command: cy.get(),cy.find(), cy.then()**

● Learn to use the cy.get(),cy.find() and cy.then() command to select elements based on various selectors such as CSS, ID, class, etc.

find() - is used to find the child selector. From whatever selector u have given in get()

// Select parent element and then find child elements within it

cy.get('.parent-class').find('input[name="username"]').type('myusername')

// Select parent element by class and then find child elements within it

cy.get('.form-class').find('.login-button').click()

// Perform operations after getting the element

cy.get('input[name="username"]').then(($input) => {

// Perform operations on $input

expect($input).to.have.value('')

$input.type('myusername')

})

// Perform operations after getting the element

cy.get('#password').then(($input) => {

// Perform operations on $input

expect($input).to.have.attr('type', 'password')

$input.type('mypassword')

})

**Program to Demo Expect using then() – asserting the text.**

describe('Assertion Example using expect', () => {

it('should have the correct header text', () => {

// Visit the HTML page

cy.visit('path/to/your/test/page.html');

// Get the header element

cy.get('#main-header').then(($header) => {

// Use expect to assert the text content

expect($header.text()).to.equal('Welcome to Cypress Testing');

});

});

});

### **3.4 Grabbing Text from Edit Box for Validations**

#### **3.4a Using cy.get().invoke()**

● Learn to use cy.get().invoke('val') to grab the text from an edit box or input field for validation purposes.

#### **3.4b Validating Text Content**

● Understand how to perform assertions on the text content retrieved from edit boxes to validate form inputs and other text-based elements