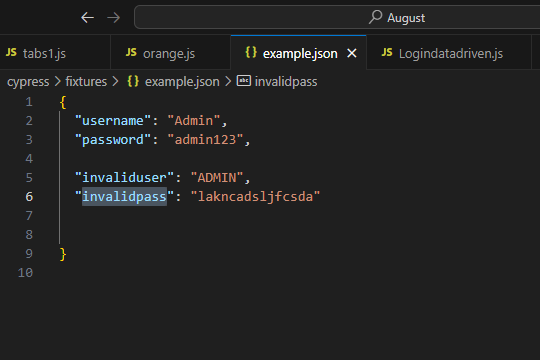
### **Data Driven Testing -**

### **Seperating the data, from the test scripts, so that the data is not hardcoded in the form of external files like json file, text file, excel file .**

The data with which we are testing is called as Test Data.

In cypress , we store the data in a json file.



Using fixture() , we can mention the filename and then use then() to make it a global variable.

describe('using data driven approach', () => {

// define before hook -

before(() => {

// code to connect with the external file - example.json

cy.fixture('example').then((data)=>{

// initialize my data , i have to make it a global instance

globalThis.data = data; //

})

});

it('test1 ', () => {

cy.visit('https://opensource-demo.orangehrmlive.com/web/index.php/auth/login')

// using data variable to fetch the values of username and password from the external json file

cy.get('[name="username"]').type(data.username)

cy.get('[placeholder="Password"').type(data.password)

cy.get('.orangehrm-login-button').click();

});

it('test2', () => {

cy.visit('https://opensource-demo.orangehrmlive.com/web/index.php/auth/login')

// using data variable to fetch the values of username and password from the external json file

cy.get('[name="username"]').type(data.invaliduser)

cy.get('[placeholder="Password"').type(data.invalidpass)

cy.get('.orangehrm-login-button').click();

});

});

### **6.1 Handling Frames**

### Frames (or iFrames) are used to embed another HTML document within the current document. Since Cypress commands operate within the context of the current document, interacting with elements inside an iFrame requires a bit of extra work.

#### **Example of an iFrame:**

#### **Cypress Commands:**

### **Accessing iFrame Content:**

### To access and interact with content inside an iFrame, you need to use the cy.frameLoaded and cy.iframe commands from the cypress-iframe plugin.

### **Installation:**

### First, install the cypress-iframe plugin:

### npm install -D cypress-iframe

### Then, include it in your cypress/support/commands.js file:

### import 'cypress-iframe'

### Usage :

// Load the iFrame and interact with its content

cy.frameLoaded('#myFrame')

cy.iframe().find('button#insideButton').click()

cy.iframe().find('input#insideInput').type('Hello, world!')

describe('tests', () => {

it('test1', () => {

cy.visit('https://rahulshettyacademy.com/AutomationPractice/')

// We are laoding the required i frame.

cy.frameLoaded('#courses-iframe')

// Using iframe() i am checking for a Blog element and then clicking on it

cy.iframe().contains('Blog').click();

// cy.iframe().find('a[href="mentorship"]').click({multiple: true});

});

it('frame2 ', () => {

cy.visit('https://v1.training-support.net/selenium/iframes')

// cy.frameLoaded('iframe[src="/selenium/frame1"]')

// using find() - providing locator and then click on the forst matching element

cy.iframe('iframe[src="/selenium/frame1"]').find('#actionButton').first().click();

});

});

### **6.2. Handling Date Pickers**

Date pickers are common UI components used for selecting dates. They can be implemented in various ways, including custom components or HTML5 date inputs.

#### **Example of an HTML5 Date Input:**

#### **Cypress Commands:**

**Interacting with Date Inputs:**

For HTML5 date inputs, you can set the value directly using the .type() command:

// Set the date input value

cy.get('#datePicker').type('2023-12-31')

// Assert the value

cy.get('#datePicker').should('have.value', '2023-12-31')

#### **Example of a Custom Date Picker:**

Custom date pickers often involve clicking through a calendar UI to select a date. Here's how you might interact with such a component:

<div class="date-picker">

<input type="text" id="customDatePicker" />

<div class="calendar">

<!-- Calendar structure here -->

<button class="day" data-day="31">31</button>

<button class="day" data-day="1">1</button>

<!-- More days... -->

</div>

</div>

#### **Cypress Commands:**

**Interacting with Custom Date Pickers:**

// Open the date picker

cy.get('#customDatePicker').click()

// Select a date

cy.get('.calendar').contains('31').click()

// Assert the input value (assuming the date picker updates the input value)

cy.get('#customDatePicker').should('have.value', '2023-12-31')

### 

### **Comprehensive Example**

Here’s a comprehensive example demonstrating how to handle frames and date pickers in Cypress:

describe('Handling Frames and Date Pickers', () => {

beforeEach(() => {

// Visit the page containing the iFrame and date picker

cy.visit('https://example.com')

})

it('Interacts with an iFrame', () => {

// Load the iFrame and interact with its content

cy.frameLoaded('#myFrame')

cy.iframe().find('button#insideButton').click()

cy.iframe().find('input#insideInput').type('Hello, world!')

})

it('Interacts with an HTML5 date picker', () => {

// Set the date input value

cy.get('#datePicker').type('2023-12-31')

cy.get('#datePicker').should('have.value', '2023-12-31')

})

it('Interacts with a custom date picker', () => {

// Open the date picker and select a date

cy.get('#customDatePicker').click()

cy.get('.calendar').contains('31').click()

cy.get('#customDatePicker').should('have.value', '2023-12-31')

})

})

### **Data Driven Testing -**

Testing, writing tests by giving the test data from an external data source like json,excel, properties is called as Data driven testing.

cy.type(‘akshay’); // Hard coding - Directly providing values inside the script.

cy.type(‘koulgi’);

We are not supposed to hard code our test scripts.

The data should be entered from an external file. (This is the Rule of Automation.)

**7.1 Page Object model**

* Page Object Model (POM) is a design pattern used to create an object repository for web elements.
* It helps in reducing code duplication and improves test maintenance by separating the page-specific code (like locators and methods) from test scripts.

#### **Key Concepts**

1. Page Classes:
   * Each web page in the application has a corresponding Java class.
   * The class contains web elements and methods to interact with those elements.

2. Methods:

* + Methods in the page class represent actions that can be performed on the web page (e.g., login, click, enter text).

3 Test Blocks like describe and it blocks:

* + Separate from the page classes, they contain test scripts that use the methods from the page classes to perform tests.

#### **7.1aBenefits**

* Improved Code Maintenance: Changes in the UI require updates only in the page classes, not in the test scripts.
* Reusability: Common actions are written once in the page class and reused in multiple test scripts.
* Readability: Test scripts are cleaner and more readable, focusing only on test logic.

#### **7.1b Best Practices**

* Single Responsibility: Each page class should represent a single page, and methods should perform only actions related to that page.
* Naming Conventions: Use meaningful names for classes, methods, and variables to improve readability.

To define Individual elements :