**Cypress Requirements & Implementation**

1. **Pre-Requisistes:**

To install cypress npm command is :

**npm init** - This command will add the node\_modules folder and package.json file in the root of your project to ensure cypress is installed in the correct directory.

**npm install cypress --save-dev -** This will install Cypress locally as a dev dependency for the project.

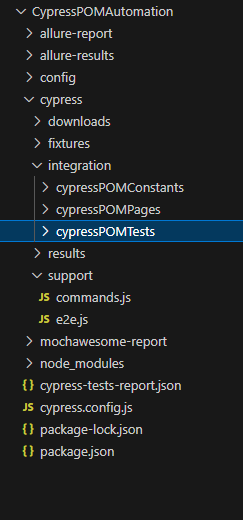
Command to open cypress:

<project root path> **npx cypress open**

1. **Javascript Testing Framework** - We have used **Mocha** as JavaScript testing framework that runs on Node.js and in the browser. It’s designed for testing both synchronous and asynchronous code, runs tests serially to deliver flexible and accurate reporting . Assertion Library used is “**Chai**”.
2. **Framework Structure:**

We have used **Page Object Model framework**.The project structure for the Page Object Model in Cypress:

A folder names “integration” is created under “cypress” folder which is used to store cypressPOMPages, cypressPOMconstants and cypressPOMTests folder.



**cypressPOMconstants** folder - is used to store the locators in a variable inside a class, so that you can reuse locators as well. These classes are javascript class with .js extension used to store locators page wise in web app.

These locators are created using css selectors. These classes are exported so that it can be accessible to all other classes.

**cypressPOMPages** folder - consists of all the methods that are related to the different pages of the website stored in different .js page classes.These classes access locators from constants classes which are imported through “require” keyword. These classes are also exported so that it can be accessible to all other classes using “export default” keyword.

**cypressPOMTests** folder - has test classes consisting of describe() block which is used to group our test cases & it() is used for an individual test case. It() block calls different page classes methods which are imported by using “import” statements and object of these classes are created using “const” and “new” keyword. Implementation of mocha based describe & it block in a test class is implemented by using below statement:

**const { describe, it } = require('mocha');**

**Default folder structure provided by Cypress Framework:**

**Cypress folder**- The folder named cypress is located in the project root folder which is main folder for cypress automation framework.

By default Cypress folder contains 4 subfolders namely fixtures, integration, downloads and support:

A)**.Fixtures folder** - can be used to store our external Json data files and we can use these files in our tests using the command cy.fixture().

B**).Integration folder-**  mainly consists of our actual spec/test files

C). **Support folder** - contains the special file index.js which will be run before each and every test. Support folder can also be used to create utility methods which will be helpful in through out our automation framework. This file is the perfect place to put all your reusable behavior such as Custom commands or global overrides that you want to be applied to all of your spec files.

D). **Downloads folder** – Contains any file files downloaded during a test run

**Default Files:**

**Cypress.config.js** - This file is used to specify any custom configuration which is located in the root of our project.

**Package.json file** - here we can enter our customized command in the scripts field. The goal of this JSON file is to make the complicated command easy with an alias command fixated to it.

Example: To clean the mochawsome report folder & remove mochawesome.json file before any execution, run a spec file and then generate a new mochawsome-report we have clubbed the commands as below:

**"specFileTest": "npx cypress run --spec cypress/integration/cypressPOMTests/Testcase1Test.js --headed --browser chrome",**

**"clean:reports": "rm -R -f cypress/results && rm -R -f mochawesome-report && rm -R -f mochawesome.json",**

**"pretest": "npm run clean:reports",**

**"combine-reports": "npx mochawesome-merge cypress/results/\*.json -o mochawesome.json",**

**"generate-report": "npx marge mochawesome.json",**

**"posttest": "npm run combine-reports && npm run generate-report",**

**"test": "npm run specFileTest || npm run posttest",**

**We just need to provide in the command line at Project root :**

**npm run test**

1. **Cross Browser Testing** - Cypress has the capability to run tests across multiple browsers. Currently, Cypress has support for Chrome, Edge , Electron & firefox browsers. The desired browser can also specified via the --browser flag when using run command to launch Cypress. For example, to run Cypress tests in Firefox:

**npm cypress run --browser firefox**

When considering to ignore or only run a particular test within a given browser

you can specify a browser to run or exclude by passing a matcher to the suite or test within the test configuration using the browser option. Ex:

// Run the test if Cypress is run via Firefox

it('Download extension in Firefox', { **browser: 'firefox'** }, () => {

cy.get('#dl-extension')

.should('contain', 'Download Firefox Extension')

})

1. **Rerun failed testcases with cypress Retries configuration:**

To define different retry attempts for cypress run versus cypress open. You can configure this in the Cypress configuration by passing the retries option an object with the following options:

**runMode** allows you to define the number of test retries when running cypress run

**openMode** allows you to define the number of test retries when running cypress open

{

retries: {

// Configure retry attempts for `cypress run`

// Default is 0

runMode: 2,

// Configure retry attempts for `cypress open`

// Default is 0

openMode: 0

}

}

We can configure retry attempts on a specific test by using the test's configuration inside it() block. We can also configure try attempts for a suite of tests, by setting the suite's configuration inside describe() block.

1. **Cypress Cloud -** To Integrate local code to the cypress cloud ,we need 2 things :
2. **project id**
3. **record key**

To run a test from spec file , command is :

**npx cypress run --record --key <project key> --spec <test.js> --headed --browser chrome**

**Cloud url -** [**https://cloud.cypress.io/**](https://cloud.cypress.io/)

**Login Credentials for cloud used in project:**

**email - kajol.singh@optimusunfo.com**

**pwd - TestAutomation**

**Record key - cd35a028-2fd1-408c-b21f-20e69740d58b**

**project id - 4dkwwk**

**Npm command to run a specfile on cloud in headed mode on chrome browser :**

**npx cypress run --record --key cd35a028-2fd1-408c-b21f-20e69740d58b --spec cypress/integration/cypressPOMTests/Testcase1Test.js --headed --browser chrome**

Tests will run in **local** only but results of test execution will be reported in your account that we created in cypress cloud.

For failed test cypress cloud report under latest runs shows logs and failed messages, screenshots & test replay feature to watch video of entire execution.

1. **Custom commands** - Cypress comes with its own API for creating custom commands and overwriting existing commands. This is **Cypress.Commands.add()** - used to add a custom command to use when writing tests. It is recommended to define queries in your cypress/support/commands.js file, since it is loaded before any test files are evaluated via an import statement in the supportFile.

Example:

// Click link containing text

Cypress.Commands.add(**'clickLink'**, (label) => {

cy.get('a').contains(label).click()

})

In test code we can use this custom command as:

**cy.clickLink('Buy Now')**

1. **Exception handling -** Unlike other Javascript-Based Frameworks, Cypress doesn’t allow you to use the try and catch block to handle the exception. Cypress provides a unique mechanism for handling exceptions in your code.

**A). Exceptions Handling in Cypress due to Unexpected Status Codes :**

We have Handled Exceptions in Cypress due to Unexpected Status Codes as Cypress throws an error if the web page returns any status code other than 2xx and 3xx. To avoid the test case from failing due to the status codes, we have use the failOnStatusCode:false option when opening a URL as:

**cy.visit(Cypress.env('url'),**

**{**

**failOnStatusCode: false,**

**});**

**B). Handling Uncaught Exceptions in Cypress**

An uncaught exception can cause tests to fail unexpectedly, leading to unclear error messages and a lack of understanding of the root cause of the failure.

To avoid your test case from failing due to uncaught exceptions in Cypress, we have use **cy.on/Cypress.on command** to listen for the uncaught:exception event. We have put this configuration in the [**supportFile**](https://docs.cypress.io/guides/core-concepts/writing-and-organizing-tests#Support-file) **“commands.js”**, since it is loaded before any test files are evaluated.

**Cypress.on('uncaught:exception', (err, runnable) => {**

**// returning false here prevents Cypress from failing the test**

**cy.log(err)**

**cy.log("error", err.message);**

**cy.log("runnable", runnable);**

**return false**

**})**

1. **Cypress Mocha Allure Report integration using mocha-allure-reporter -**

For Allure integration with Mocha framework npm command is -

**npm install mocha-allure-reporter**

After installation of this reporter, npm command to generate allure report is :

**npm cypress run --spec <specfilePath.js> --headed --browser chrome --reporter mocha-allure-reporter**

After running tests you will get raw tests result into allure-results directory.

1. **Logging - cy.log()** command is used to Print a message to the Cypress Command Log.
2. **ViewPort Testing -** Control the size and orientation of the screen for your application.

To Resize the viewport to iPhone 6 width and height

**cy.viewport('iphone-6')** // viewport will change to 414px x 736px

To Change the orientation to landscape

**cy.viewport('iphone-6', 'landscape')**

We can change these default dimensions by adding “viewportWidth” & “viewportHeight” to your Cypress configuration in cypress.config.json andwe can set viewport in the test configuration.

We can also Organize desktop vs mobile tests separately by providing specific viewport dimension for all the tests in that spec within beforeEach() hook.

We can also use npm cypress run command to launch Cypress with mobile dimensions:

**npm run specFileTest --config viewportWidth=375,viewportHeight=667**

1. **Test data management using fixtures** - Fixtures are used to load a fixed set of data located in a file. A path to a file within the [fixturesFolder](https://docs.cypress.io/guides/references/configuration#Folders-Files) , which defaults to ”cypress/fixtures”. Cypress command to load data from a fixture .json file: **cy.fixture('users.json').as('usersData')**
2. **Excel File System Access through Code :** File conversion is one method we could use to read data from excel files. Due to Cypress’s lack of support.xlsx file. The only format supported by Cypress is JSON, thus we would need to convert it. In other words, we will convert the Excel.xlsx file to a JSON file. And using that JSON file, we’ll be able to extract the info we require.

For this we need to follow below steps:

**A).Install “convert-excel-to-json" plugin:**

**npm install convert-excel-to-json**

**B).Create Cypress task:** Cy.task() - is used to create some set of actions outside the scope of cypress. We can execute any node commands using the task. Cypress.config.js:

e2e: {

experimentalRunAllSpecs: true,

setupNodeEvents(on, config) {

on('task', {

excelToJsonConvertor(filePath)

{

const result = excelToJson({

source: fs.readFileSync(filePath)

})

return result

}

})

return config;

},

**C). Execute tasks into tests:** In Below code, cy.task() will call “excelToJsonConvertor” task defined in cypress.config.js and passes the excel file path. The data from excel file will be extracted in the Json format and cy.writeFile() method will be used to store this extracted data in json format.

const filePath = 'cypress/fixtures/excelData.xlsx'

//const filePath = Cypress.config("fileServerFolder")+("/cypress/fixtures/excelData.xlsx")

cy.task('excelToJsonConvertor', filePath).then(function(result)

{

cy.log(result)

cy.log(result.Sheet1[0].A)

// WRITE JSON FILE

cy.writeFile('cypress/fixtures/productData.json',

{groceryProductName1: result.Sheet1[0].A, groceryProductName2: result.Sheet1[1].A})

})

// READ DATA FROM JSON FILE

cy.log("Read datafrom JSON files")

// cy.fixture('productData.json').as('productDataJson').then(function(data)

/// READ JSON FILE

cy.readFile('cypress/fixtures/productData.json').then((data) =>

{

cy.get(homePageConstants.searchBox).click().type(data.groceryProductName2)