**Cypress Installation & Project Setup**

Step 1 - Install Node.js and Set NODE\_HOME Environment Variable (****Pre-requisites)****

Step 2 - Install Visual Studio Code (****Pre-requisites)****

Step 3 - Create a new folder for Cypress project

Step 4 - Open the folder in VS Code

Step 5 - Open VS Code terminal & run command >>>>>> npm init -y

Step 6 - npm init -y >>>>> i will create package.json

Step 7 - Install Cypress >>>>> npm install cypress --save-dev

Package.json contains

|  |  |
| --- | --- |
| 3  4  5  6  7  8  9  10  11 | {    "name": "automation",    "version": "1.0.0",    "description": "",    "main": "index.js",    "scripts": {      "test": "echo \"Error: no test specified\" && exit 1"    },    "author": "",    "license": "ISC"  } |

Where,

**package name**: Name of the package created. The user needs to enter this details

**version**: a version of your application/package. It can help in creating different versions of your package.

**description**: additional detail that one needs to provide for the package. Users can leave this blank as well.

**entry point**: What is the entry point for your application. It is pre-populated with index.js, so this needs no change.

**test command**: command that needs to run for testing of the application. If you have created any command which runs your test, you can set it there.

**git repository**: the path to git repository

**keywords**: any keywords to uniquely identify your package.

**author**: author of the repository which is generally the username

**Cross Browser Testing**

To make launching of Cypress with a specific browser even more convenient, npm scripts can be used as a shortcut:

"scripts": {

"cy:run:chrome": "cypress run --browser chrome",

"cy:run:firefox": "cypress run --browser firefox"}

The desired browser can also specified via the [--browser](https://docs.cypress.io/guides/guides/command-line" \l "Options) flag when using [run](https://docs.cypress.io/guides/guides/command-line" \l "cypress-run) command to launch Cypress. For example, to run Cypress tests in Firefox:

cypress run --browser firefox

Page Object Pattern in Cypress

Suppose we have to create a ****Page class****for the Home Page

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | class HomePage {  getUserName() {      return cy.get('#reg\_username');  }  }  export default HomePage |

and we have set it to export default. So for using these methods in our test, we just have to import them and use them.

// type definitions for Cypress object "cy"

// <reference types="cypress" />

import HomePage from '../../support/PageObjects/HomePage';

**Method 1**

1. Installing Allure Reporter is very straightforward with Cypress. After creating the project run the following npm command

npm i -D @shelex/cypress-allure-plugin

1. Add Allure plugin to cypress plugins. To do that add the following code snippet of allurewriter inside the plugins/index.js file.

const allureWriter = require('@shelex/cypress-allure-plugin/writer');

module.exports = (on, config) => {

allureWriter(on, config);

return config;};

1. Then register the command of allure plugin with cypress/support/index.js file

import '@shelex/cypress-allure-plugin';

1. To enable intelliSense in your IDE use following reference and add in to plugins/index.js

/// <reference types="@shelex/cypress-allure-plugin" />

**Method 2**

1. Once setting up the project, run the following command to obtain test results as Allure Report

npx cypress run --env allure=true

1. First create a custom command to run the test script in Package.json file

  "scripts": {

    "cy:run": "cypress run -- env allure=true",

    "allure:report": "allure generate allure-results --clean -o allure-report",

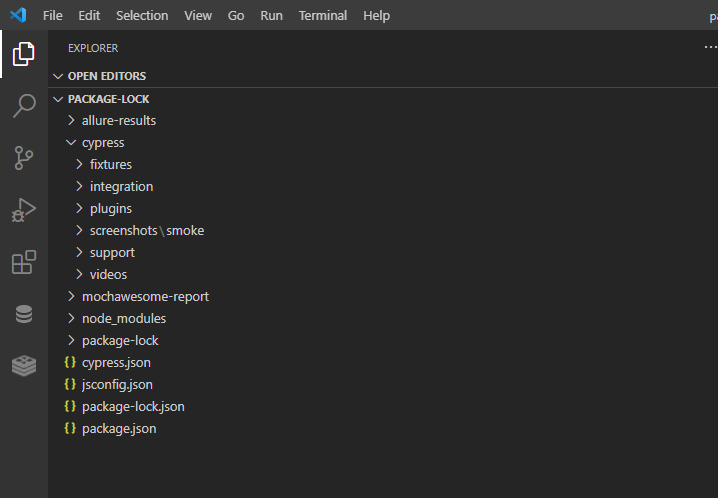
    "cy:clientLogin": "cypress run --spec \"cypress/integration/smoke/\*.spec.js\" --reporter mocha-allure-reporter",

    "report:allure": "allure generate allure-results --clean -o allurereport && allure open allure-report",

    "test:allure": "npm run cy:clientLogin && npm run report:allure"

  },

**Cypress has created a default folder hierarchy when it opens the first time. Below are the details for each of these folders/files created by Cypress:**



1. **fixtures**are external pieces of static data that can be used by your tests. We should not hard code data in the test case. It should drive from an external source like CSV, HTML, or JSON. They will be majorly used with the cy.fixture() command when you need to [stub](https://www.toolsqa.com/blogs/differences-between-stubbing-and-mocking/" \t "https://www.toolsqa.com/cypress/cypress-test/_blank) the network calls.

Cypress provides us with a feature to drive the data from external sources. We can draw a comparison to the Data-Driven Frameworks, which are one of the popular automation testing framework types in the current market. As we all know that, in Data-Driven Frameworks, we generally keep the data in excel files and use external libraries such as Apache POI to read the data and use it in our test with an automation framework. Similarly, Cypress has provided the same functionality in its architecture, where we can read the data from external files known as fixture files.

Cypress provides a directory named as fixtures, which stores various “JSON” files, and these JSON files can store the test data which can be read by multiple tests. We store test data in the form of key-values, which we can access in the test scripts.

1. **Integration folder** provides a place that writes out test cases. It also provides an “examples” directory, which contains the default test cases provided by Cypress and can be used to add new test cases also. We can also create our folder under the integration directory and add out test cases under that.
2. **Plugins** contain the plugins or listeners. By default, Cypress will automatically include the plugins file “cypress/plugins/index.js” before every test it runs. You can programmatically alter the resolved configuration and environment variables using plugins.
3. **Support**writes customized commands or reusable methods that are available for usage in all of your spec/test files. This file runs before every single spec file. That’s why you don’t have to import this file in every single one of your spec files.  The “support” file is a great place to put reusable behavior such as Custom Commands or global overrides that you want to be applied and available to all of your spec files.
4. **Node\_Modules** in the default project structure is the heart of the cypress project. All the node packages will be installed in the node\_modules directory and will be available in all the test files. So, in a nutshell, this is the folder where NPM installs all the project dependencies.
5. **Cypress.json** is used to store different configurations. E.g., timeout, base URL, test files, or any other configuration that we want to override for tweaking the behavior of Cypress. We can also manage the customized folder structure because it is part of by default Cypress Configurations.
6. All NPM packages contain a file; usually, in the project root, called **package.json,** this file holds various metadata and libraries relevant to the project. This file gives information to NPM that allows it to identify the project as well as handle the project’s dependencies. It is similar to ****pom.xml from Maven****

For generating the package.json file for our project, we will start with npm init either in Terminal of Visual Code or command prompt. It will initiate the npm and will ask us some details that need to be provided by the user before it generates the package.json file

Type below command on the terminal under your project directory:

**npm init -y**

1. **Screenshots** Cypress as a tool comes with the in-build capability to take screenshots irrespective of whether you are running in interactive mode using cypress open or cypress run and even in headless mode on the command line or CI as part of your pipeline. The Cypress framework is smart enough to take screenshots on the failure of a test case on its own; we don’t have to write any code for this. But if we want to take some explicit screenshots during the test runs.