**Cypress documentation:**

[**https://www.bigbinary.com/blog/cypress-environment-config**](https://www.bigbinary.com/blog/cypress-environment-config)

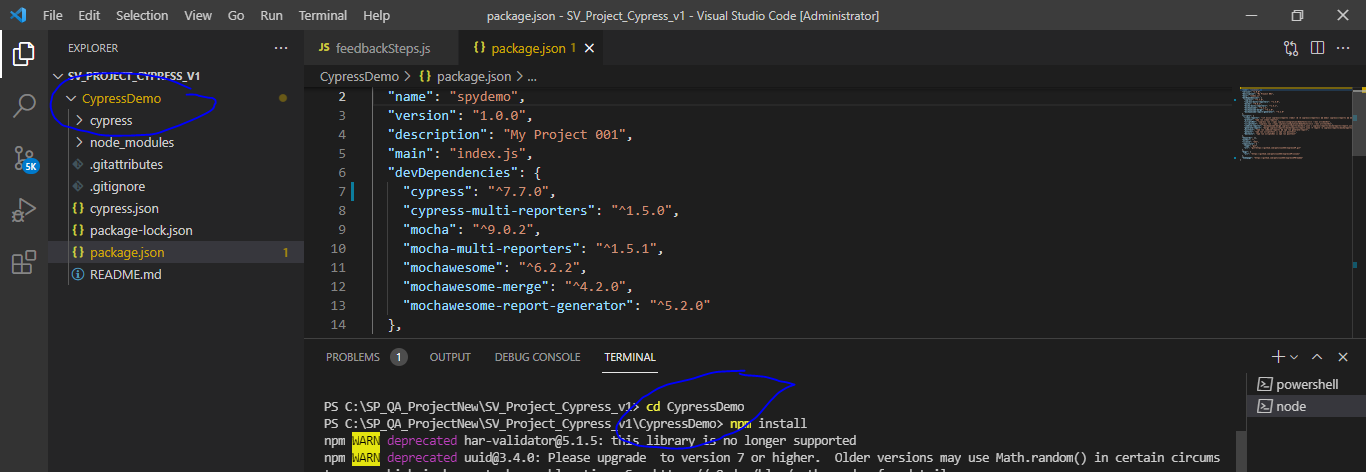
**https://dzone.com/articles/configure-cypress-tests-to-run-on-multiple-environments**

Source: Udemy courses

# [Cypress -Modern Automation Testing from Scratch + Framework](https://cognizant.udemy.com/course/cypress-tutorial/)

It supports chrome/FF/MS edge/electron

1. Install node.js and verify node and npm versions
2. **Npm install** all the packages mentioned in package.json
3. **npm install cypress --save-dev**
4. **node\_modules/.bin/cypress open**

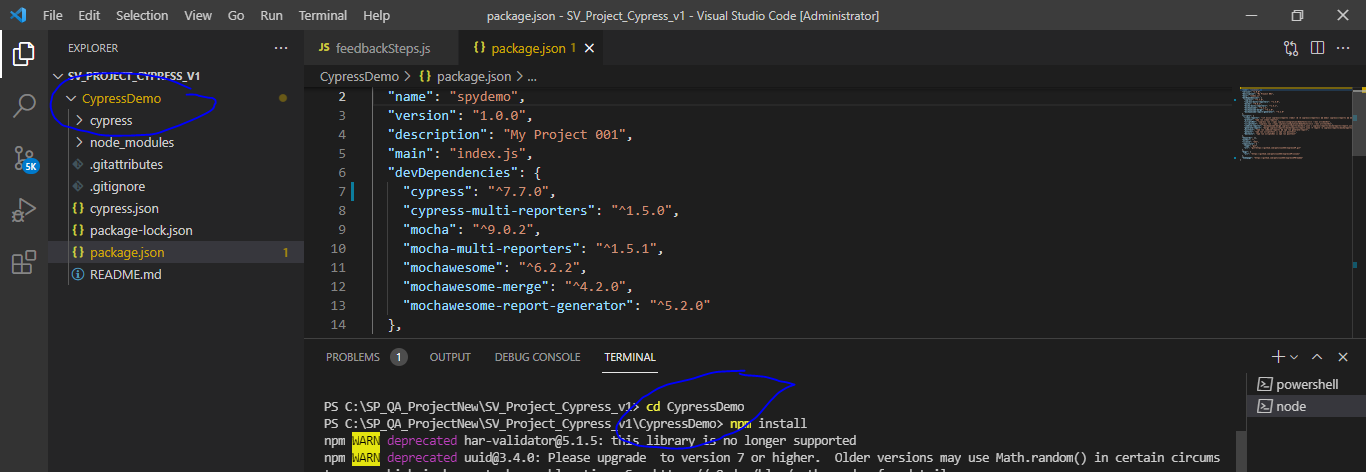


1. To create package.json🡪 npm init , to auto generate package.json
2. If you haven’t specified cypress library(module) in package.json, then you can use

npm install - - save-dev

1. Create folder ‘SP\_CyprescsFolder’
2. Using cmd at this folder level 🡪 install cypress using command : npm –install cypress
3. Now use the command: code **.** to open folder in VS code
4. Create a package.json inside the folder and add dependency for cypress like below using vscode intel sense Within VSCODE..

Mention the latest version and install using terminal



1. Verify cypress module installed in node\_modules subfolder inside folder ‘SP\_CypressFolder’
2. Run command ‘Npx cypress open’ from termina**l**
3. **C:\Sp\_others\SPCyDemo\cypress\integration\DemoTests\javascriptexamples> node basics.js**



**Working with UI and console based Test runner in Cypress:**

* **Cypress UI Runner leverage:**

You can open UI Cypress runner using from terminal : npx cypress open

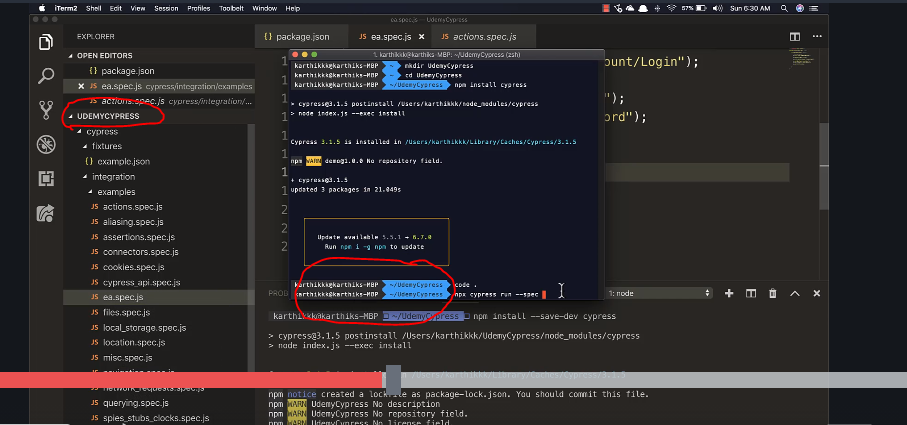
Or ./node\_modules/.bin/cypress run

* **Command line run:**

Cypress when runs spec files through command line**, it runs in headless mode**

Default browser is electron

While executing we must specify the browser name in case of ‘Chrome’

**We have to be in main parent folder of cypress folder**

* npx cypress run –spec /users/Sushant/pathofSpecFile/ea.spec.js

Headless run using available browser, electron 59

still stores the execution evidence video.

* npx cypress run –spec /users/Sushant/pathofSpecFile/ea.spec.js –browser chrome

Chrome browser run

* npx cypress run --spec cypress/integration/DemoTests/\*\*/\*

You should be able to comma separate globs as well so you can run multiple separate directories or files, like:

* npx cypress run --spec 'cypress/integration/examples/\*\*/\*,cypress/integration/other-folder/\*\*/\*,cypress/integration/other-file.js'

cypress run --spec cypress/integration/DemoTests/\*\*/\*firstYY.spec.js

* npx run test 🡪 this will run all the specified specs inside package.json🡪scripts json obj

**Interacting with Cypress UI Elements**

*Default timeout to find elements is 4000 miliseconds*

Will discuss most common cypress commands

* Click ()

Options:

1. Co-ordinates: Check at cypress site
2. Force: true

When in click options once we have {force : true}, it will skip the below checks

* + Type ()
  + Clear ()
  + Check()
  + Select()

**Locators:**

1. Id -> tagName#IdName
2. className 🡪 .className
3. Customized with Any attributes 🡪 tagname[attribute=’value’]
4. Navigating from parent to child 🡪 form input

(in xpath 🡪 form/input , where as in CSS it’s space required instead of ‘/’)

1. Nth child -🡪 table/tbody/tr/td[2] -- xpath

In CSS 🡪 table tbody tr td:nth-child(2) -- css

**How to protect sensitive data with Cypress**

That is, sensitive data is protected, and we can run the same tests in different environments (local, staging, production, etc.)

Finally, if you want to protect your password from “leaking” when running tests in interactive mode in your local environment, pass the option { log: false } as the second argument of .type() and that command will not be listed in the list Cypress runner commands (see below.)

describe('Login', () => {

it('successfully', () => {

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

cy.get('#user')

.type(Cypress.env('user\_name'))

cy.get('#password')

.type(Cypress.env('user\_password'), { log: false })

cy.contains('Login').click()

cy.get('.navbar-top .avatar')

.should('be.visible')

})

})

**Should** is the assertion type of **CHAI**

<div class=’prodcuts’ style css=’1’>

<div class=’prodcut’ style css=’…’>

<div class=’prodcut’ style css=’…’>

<div class=’prodcut’ style css=’…’>

<div class=’prodcut’ style css=’…’>

</div>

Ex:

Cy.get(‘.searchKeyWord’).type(‘carrot’)

Cy.get(‘.product’).should(‘have.length’, 4) 🡺 to validate number of products equal to 4

If we have 5 elements, out of which one is invisible –then the above line fails

So now

Cy.get(‘.prodcut:visible’).should(‘have.length’,4) -> will pass now, To retrieve only visible elements

**Parent-child pairing:**

Cy.get(‘.products’).find(‘.product’).should(‘have.length’, 4)

**cy.get(‘.products’).find(‘,product’).eq(2)** 🡪A number indicating the index to find the element at within an array of elements. A negative number counts index from the end of the list

**cy.get(‘.products’).find(‘,product’).eq(2).contains(‘Add to cart’) .click() 🡪** Here clicking on 2 nd product to move it to add to card

**Dynamically clicking on particular product: using each() method**

Iterating over the array of elements using each() method

<div class=’prodcuts’ style css=’1’>

<div class=’prodcut’ style css=’…’>

<div class=”product-image”>..</div>

<h4> class=’product-name’> capsicum </h4>

<div class=’prodcut’ style css=’…’>

<div class=’prodcut’ style css=’…’>

<div class=’prodcut’ style css=’…’>

</div>

Get each product, check is it the required product, then do expected action on it

cy.get(‘.products’).find(‘.product’).each(($element, index, $list) =>{

const reqVeg =$element.find(‘h4.product-name’).text()

if (reqVeg.includes(‘capsicum’) { 🡪 in java substring, javscript it’s includes

$ element.contains(‘Add to cart’).click()

}

})

**Cypress : it’s asynchronous nature and promise handling**

Sequence of execution is not guarantee in Cypress and Protractor since they use node.js. Whereas selenium is sequentially executes as it uses Java

But cypress handles sequence execution by default [Cypress developers handles it using wrappers by default]

**Promise**: Every asynchronous step returns a **promise** and state will be one of these stage.

Comes in 3 stages

Resolve🡪

Pending🡪

Reject🡪

We have to wait until promise is resolved, how to come to know whether promise is resolved: It’s done by using **then()** method.

**Understanding between Jquery methods and Cypress commands**

Non-cypress commands cannot resolve promise by themselves, we need to manually resolve them

**Const logo=cy.get(‘.brand’)**

**Cy.log(logo.text()) // won’t work**

Text() is not a cypress command 🡪 needs to resolve using promise

Text() is jquery method .Solving it using promise i.e use of then() method

**Cy.get(‘.brand’).then(function(logoelelment){**

**Cy.log(logelement.text())**

**})**

**Assertion on element text():**

**cy.get(‘.brand’).should(‘have.text’, ‘FACEBOOK’) // using chai keyword**

**Aliasing :**

**Cy.get(‘.product’).as(‘myalias’)**

**Cy.get(@myalias).find(..)**

**a)Console.log(‘Mycomment’) 🡪 you can see it on console of browser, asynchronous, at the topmost line**

**b) Cy.log(‘Mycomments’)🡪 printed in cypress logs , synchronous**

**How to make statement as synchronous**

**cy.get(‘.products’).find(‘.product’).eq(2)** .then(function(){

**console.log(‘Mycomment’)** // Now we will get this comment after execution of this line

**})**

**Checkboxes:**

Cy.get(‘#checkbox1’).check().should(‘be.checked’).and(‘have.value’, ‘ Cricket’)

Cy.get(‘#checkbox1’).uncheck().should(‘not.be.checked’)

**Multiple checkboxes**

Cy.get(‘input[type=”checkbox”]’).check([‘option1’,’option2’]) **-> out of 5 options we can check 1, & 2**

**Radio Buttons:**

Cy.get(‘[value=”radio2”]’).check().should(‘be.checked’)

**Dropdowns:**

**Static dropdown:**

**Cy.get(‘select’).select(‘option2’).should(‘have.value’,’value1’)** // if dropdown options specified in a select tag

**Dynamic dropdown:**

Cy.get(‘#autoSearch’).type(‘Ind’) 🡪 Enter a text in autosearch field(ex: google search)

Cy.get(‘.ui-menu-item div’).each(($element1,index,$list){

If($element.text()===”India”) {

$element.click()

}

})

cy.get(‘#autoSeacrh’).should(‘have.value’, ‘India’) 🡪 Assertion

**Handling visible and non visible elements using assertions**

Cy.get(‘.displayedText).should(‘not.be.visible’)

🡪Perofrm action to hide the text

Cy.get(‘.displayedText).should(‘be.visible’)

**Auto handles Alerts [Pop-ups]**

Cypress by default accepts pop-ups.

But still there is way to verify the pop-up text using Cypress

Simple Alert handling:

Cypress has browser events to which it listens –> **windlow:Alert** event . Through cypress we need to trigger this event

On() 🡪 this method is used to trigger the events.. it takes two arguments

1—event name, 2 –output of the event text

Cy.on(‘window:alert’, (str)=>

{

expect(str).to.equal(‘Provide the text of Alert’) // Mocha assertion

})

confirm alert: **Using the event -> window:confirm**

cy.on(‘window:confirm’, (str)=>

{

expect(str).to.equal(‘Confirmation alert text to be verified’)

})

**Handling child tab/window using cypress and jquery commands:**

1. Using removal of target attribute using jquery function removeAttr():

To open child tab in the tab of parent itself we need to delete ‘target’ attribute through cypress. Cypress supports all jquery functions, invoke() is method through which we can use Jquery functions.

There is the function ‘**removeattr()’** in jquery to remove the particular attribute from the DOM

Cy.get(‘#openTab’).invoke(‘removeAttr’, ‘target’).click()

1. By grabbing href attribute value and then visit that URL:

Grab the href attribute value and visit that href url

Using jquery method

Cy.get(‘.opentab’).then(function(urlElement){

Const reqUrl= urlElement.prop(‘href’)

Cy.visit(reqUrl) // [this required url should be of same origin policy, then only it will work[domain should be same]

}

**Navigating browser controls**

Cy.get(‘#openTab’).invoke(‘removeAttr’, ‘target’).click()

**Cy.go(‘back’)**

Cy.url().should(‘include’, ‘urlsubstring’) **🡪 to get current url and verify**

-- cy.go('forward')

To reload the page, use the [cy.reload()](https://on.cypress.io/reload) command.

To visit a remote page, use the [cy.visit()](https://on.cypress.io/visit) command.

**How to handle dynamic web tables:**

# [Cypress -Modern Automation Testing from Scratch + Framework](https://cognizant.udemy.com/course/cypress-tutorial/)

|  |  |  |
| --- | --- | --- |
| Instructor | Course | Price |
| ---- | ---- | --- |
| Rahul Shetty | Master selenium Automation | 25 |
| ---- | ---- | --- |
| --- | --- | -- |

<table id=’product’

<tbody>

<tr>

<tr>

<td>Rahul Shetty</td>

**<td> Master selenium Automation</td>**

<td> 25</td>

</tr>

|  |
| --- |
| Cy.get(‘tr td:**nth-child(2)’).**each(($element, index, $list) =>{ |
| Const reText=$element.text() |
| If(reText.includes(“Master selenium Automation”)){ |
| cy.get(‘tr td:nth-child(2)’**).eq(index).next().**then(function(price){ |
| const reqPrice=price.text() |
| expect(reqPrice).to.equal(‘25’) |
| }) |
| // index will give you the index of required dynamic course element |
| // next() cypress method to move to sibling element |
| } |
| }) |

**Web table:**

**Nth child :** [**https://www.youtube.com/watch?v=XrpvzUr8esY**](https://www.youtube.com/watch?v=XrpvzUr8esY)

Check value presence in specific row [tr:nth-child(2) ] and column [td:nth-child(3)]

Cy.get(‘#table > tbody >tr:nth-child(2) > td:nth-child(3)’).conatins(‘Selenium’).should(‘be.visible’)

Tags: <table>, <tbody>, <tr>, <td>,<th>

Cypress commands: Each, parent, wrap, within

Get whole table data:

Cy.get(‘#table->tbody->tr’).should(‘have.length’, 4) 🡪 Validate number of rows

Cy.get(‘#table->tbody->tr:eq(0)->td’).should(‘have.length’, 6) 🡪 Get number of columns from the first row

Get the whole table data

**Cy.get(‘#table->tbody->tr’).each($row, index, $rows){**

**Cy.wrap($row).within(function(){**

->warp, to convert jquery into cy element, then to use within with call back function

**Cy.get(‘td’).each($celldata, index, $cells){**

**->** using within we fixed dom tag to particular row

**Cy.log($celldata.text())**

})

Get the single row data :

**Cy.get(‘#table->tbody->tr’).first().within(function(){**

**Cy.get(‘td’).eq(3).should(‘contain.text’, ‘textof3rdElementOnfirstRow’)**

**})**

**Or**

**Cy.get(‘#table->tbody->tr’).eq(0).within(function(){**

**Cy.get(‘td’).eq(3).should(‘contain.text’, ‘textof3rdElementOnfirstRow’)**

**})**

Or

**Cy.get(‘#table->tbody->tr’).eq(1).within(function(){**

**Cy.get(‘td’).eq(3).should(‘contain.text’, ‘textof3rdElementOnfirstRow’)**

**})**

Get the specific cell value based on another cell

**Cy.egt(‘#table’).contains(‘cellValue’).parent().within(function(){**

**Cy.get(‘td’).eq(4).then(function(websitelink){**

**Cy.log(websitelink.text())**

**}**

[**https://www.youtube.com/watch?v=6UQUEgfGSH0**](https://www.youtube.com/watch?v=6UQUEgfGSH0)

**Mouse hover using Cypress:**

Jquery provides show() method using which we can achieve mouse hover actions in Cypress

Which function help us to use of JQuery? : Ans : .invoke()

Cy.get(‘.mousehover’).invoke(‘show’)

Cy.contains(‘Top’).click()

Cy.url().should(‘include’, ‘top’)

Or

Cy.contains(‘Top’).click(force:true)

Cy.url().should(‘include’, ‘top’)

**Frame handling:**

**Import ‘cypress-iframe’ 🡪 syntax to import iframe**

Cy.frameloaded(‘frameId’) // loading frame

Cy.iframe().find(“a[href\*=’mentorship’]”).eq(0).click() // switching frame

Cy.iframe().find(“h1[class\*=’pricing-title’]’”).should(‘have.length’, 2) - > //verify some elements on the identified iframe

**Working with Advanced Cypress commands**

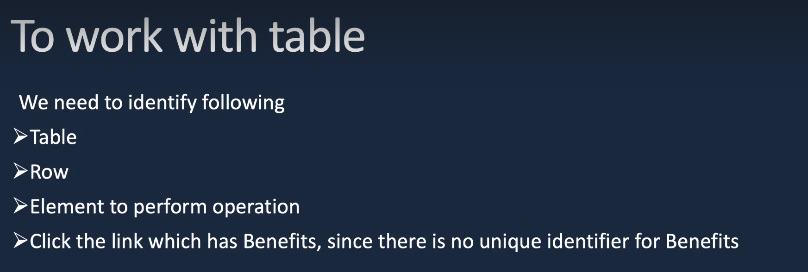
* Visit 🡪 visit’s any website,navigating to the url

Need to create object for cy..comes by default in cypress

cv.visit(‘https://www.cypress.io/’)

* Get 🡪 get an element from UI and also to get also the alias
* Wrap 🡪
* Parent 🡪 to obtain parent element from the present element
* Find 🡪 Get the descendant DOM elements of a specific selector , like query behavior of jquery
* Filter 🡪 Get the DOM elements that matches a specific selector
* Each 🡪 Iterate through each element



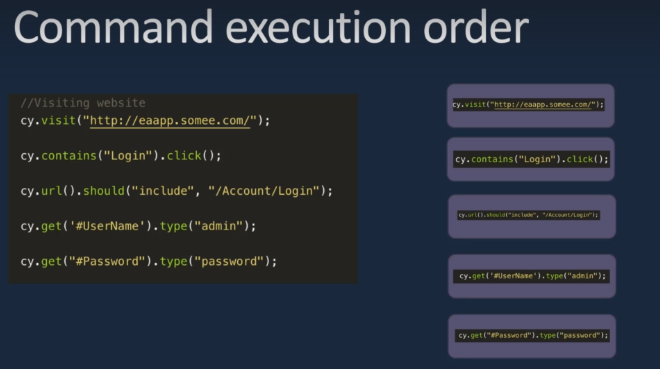
 

**Working with Cypress variables**:

Let, Var and Const: In cypress everything is asynchronous, in order to make synchronous need to use then () method, by resolving the promises.

Command execution order:

Cypress doesn’t execute everything line by line code instead, it accumulates into queue, then finally executes these queues in turn entire code.



<https://www.freecodecamp.org/news/var-let-and-const-whats-the-difference/#:~:text=var%20declarations%20are%20globally%20scoped%20or%20function%20scoped%20while%20let,the%20top%20of%20their%20scope>.

**const** declarations are block scoped

const **cannot be updated or re-declared**

let variables are block scoped.

**let** can be updated but not re-declared.

**var** variables can **be re-declared and updated**, is globally scoped because it exists outside a function

Because of the queue mechanism below line of code won’t work

Var a =cy.contains(“Login”); // linkedTex=contains

So you have to use then() method to resolve the promise by using the callback

cy.contains(“Login”).then(($link)=>{

const linkText=$link.text(); // the scope of variable is within this block only,

expect(linkText).is.eql(‘Login’); // chai assertion

}).click();

expect(linkText).is.eql(‘Login’); // this line will fail because variable is not accessible outside the block,🡪 **unexpected token**

var linkText

cy.get(“#Loginlink”).then(($link)=>{ // using element id

linkText=$link.text(); // the cope of variable is within this block only,

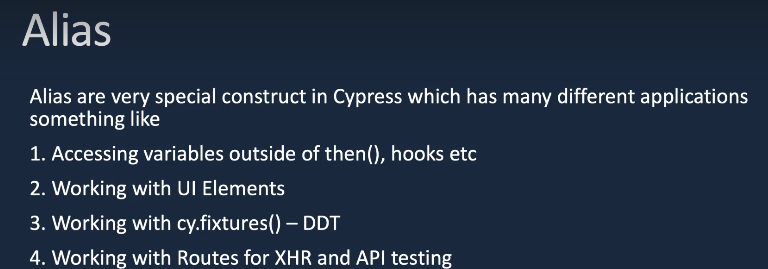
expect(linkText).is.eql(‘Login’); // chai assertion

}).click();

expect(linkText).is.eql(‘Login’); // this line will fail because variable is not accessible outside the block,🡪 **unexpected token**

In order to overcome from this problem where you want to use the value of variable outside the block of code, we can Alias , to use it in later part of code

**Understanding Alias:**



*cy.contains(“Login”).then(($link)=>{*

*return link.text();*

*}).as(linkText); // string being returned here, used an alias as linkText to access outside this scope*

*Cy.contains(“Login”).click();*

*Cy.get(“@linkText”).then(($x)=>{*

*Expect($x).is.eql(‘Login’) // use of this Alias linkText*

*});*

Other way to achieve the same thing:

*cy.contains(“Login”).invoke(‘text’).as(“LinkText”) // This is a change*

*Cy.contains(“Login”).click();*

*Cy.get(“@linkText”).then(($x)=>{*

*Expect($x).is.eql(‘Login’) // use of this Alias linkText*

*});*

**Working with Alias for UI operation**

Clicking a particular row’s Benefits --

cy.get(‘.table’).find(‘tr’).contains(‘Prashant’).parent().contains(‘Benifits’).click();

Clicking all rows using Alias

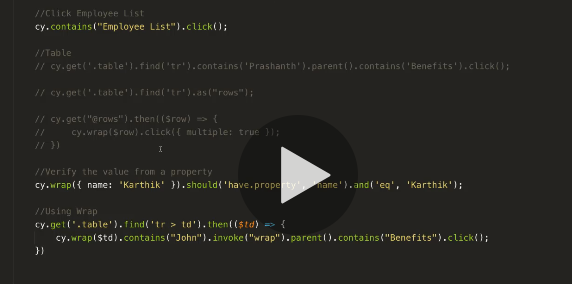
cy.get(‘.table’).find(‘tr’).as(‘rows’); // aliasing

cy.get(‘@rows’).then(($row) => { // wrap method usage

cy.wrap($row).click({multiple:true}) // wrap, brings all obj of particular row , click all rows

});

**Understanding Cypress Wrap method**



**Working with Assertions:**

Cypress bundles Chai assertion library and helpful extensions for sinons and jquery ,brining you dozens of powerful assertions for free

**Implicit and explicit wait**

-Implicit assertions 🡪expect()

-Explicit assertions 🡪should()

Cypress has a built in retry-ability in every command, the wait mechanism in assertions are really handy.

Usage for external time out in implicit wait

cy.get(“[aria-label=’Jump to a slide 2’]”,{timeout:6000}).should(‘have.class’,’ls-nav-avtive’);

Specify external time out in Explict await

cy.get(“[aria-label=’Jump to a slide 2’]”,{timeout:6000}).should(($x) =>

expect($x).to.have.class(’ls-nav-avtive’);

})

**Hooks of Cypress**

Mocha framework

Testing framework is required, for javascript we have Mocha and Jasmine.

But cypress recommending Mocha test framework in build

Describe – It’s like test suite

**SYNTAX:**

describe (‘My first test suite’, function() {

// it takes first argument as test suite name , second argument as function

// All it blocks

} )

it 🡪 All ‘it’ blocks are treated as Tests and enclosed within Describe

**SYNTAX:**

it(‘ Test case name’, function () {

// Test case steps/lines of code

} )

BeforeAll

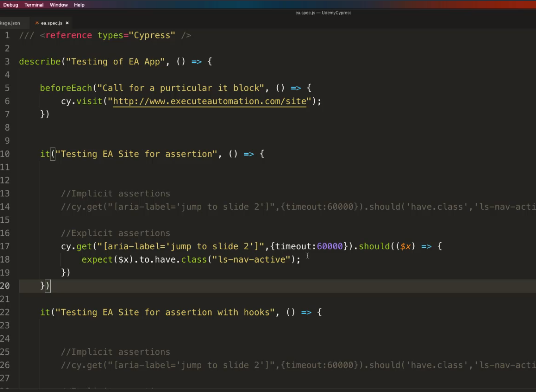
Before

After

BeforeEach

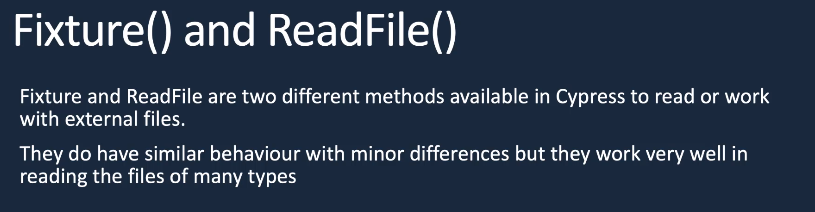
AfterEach

AfterAll

u

**Working with Data Driven:**

With Fixtures and read files



Keep json file under cypress/fixtures folder 🡪 name eauser.json

Create two properties

{

“UserName”:”admin”,

“Password”:”password”

}

Ea.spec.js

describe(“Test for json access”, () =>{

beforeEach(“Login to Application”, () =>{

cy.visit(<http://eaapp.somee.com/>”);

cy.fixture(“eauser”).as(“user”); // Usage of Alias & data driven ,accessing json ->eauser.json

})

It(“Usage of json values and alias outside the beforeEach block”, () =>{

Cy.get(“@user”).then((user) =>{ // Accessing alias using @

Cy.get(“#username”).type(user.UserName); // from json Key : UserName

Cy.get(“#password”).type(user.Password); // from json key : Password

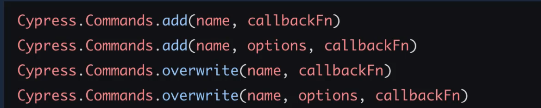
})

})

**15. Cypress custom commands.**

Cypress comes with its own API for creating own commands and overwriting an existing command. The built-in cypress commands use the very same API that’s defined below





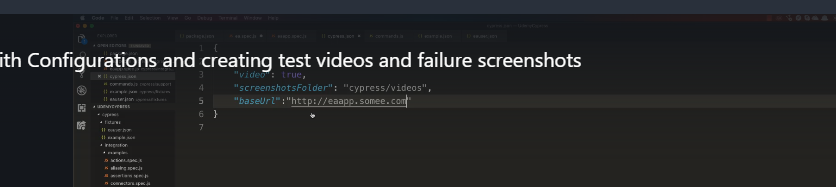
Cy.get(“#username”).type(user.UserName); // from json Key : UserName

Cy.get(“#password”).type(user.Password); // from json key : Password

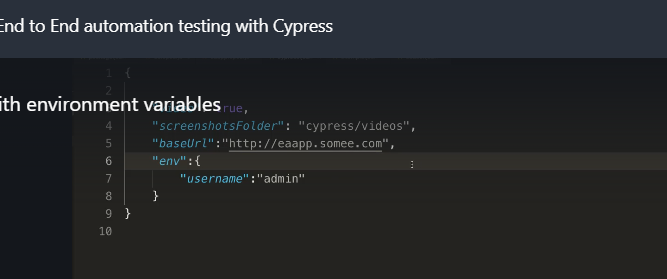
**Working with Cypress settings and environment variables**

**https://dev.to/hi\_iam\_chris/running-cypress-against-multiple-environments-54m9**

cypress.json to control cypress settings and env variable for ci/cd configurations



Working with env varibles



Cypress.env(“username”)

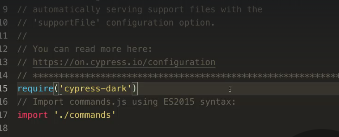


**Cypress plugIns**

Cypress-dark 🡪 Npm install –save-dev cypress-dark

And add below line in index.js

Require(‘cypress-dark’)

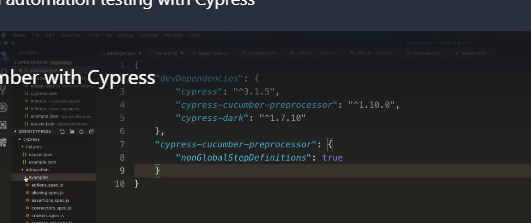


**Cucumber with cypress:**

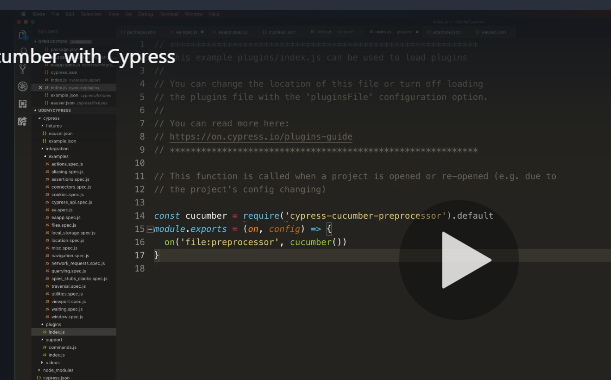
Plugin: Npm install –save-dev cypress-cucumber-preprocessor

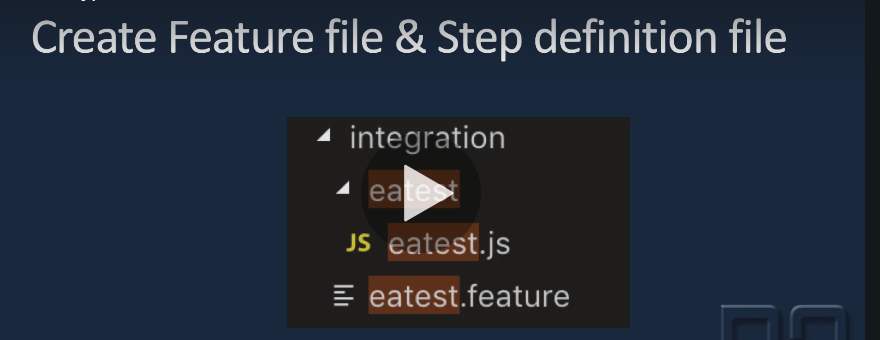
Two settings required:

1.

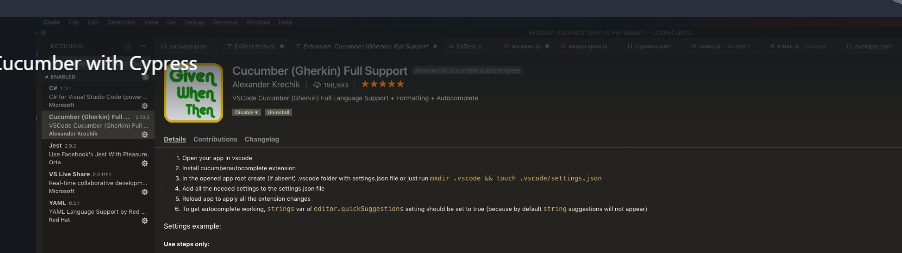


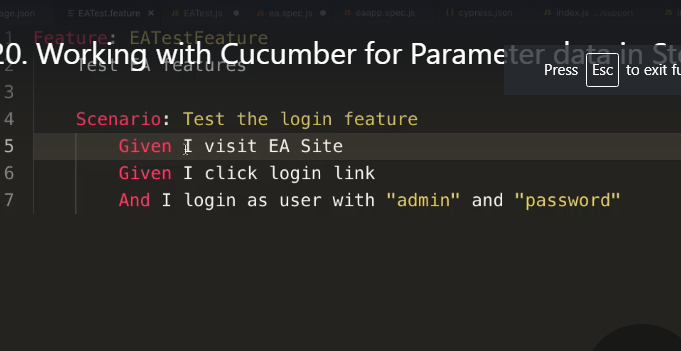
2. index.js

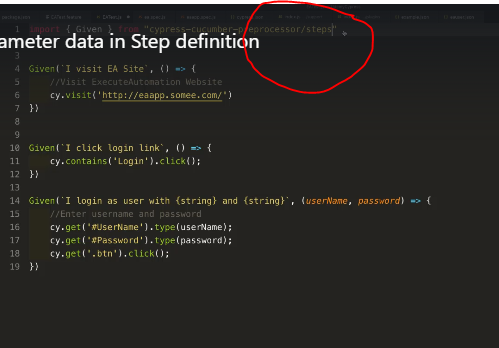
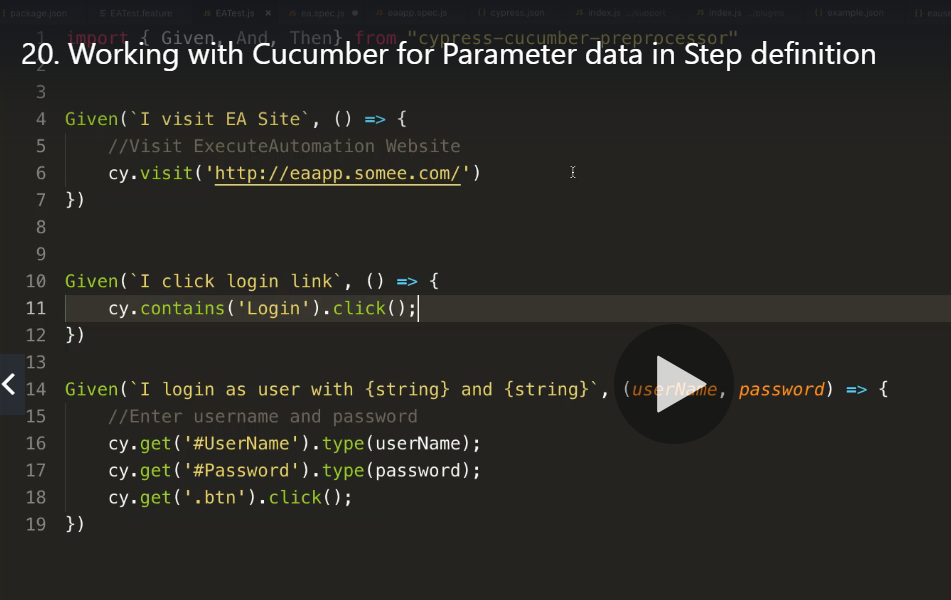


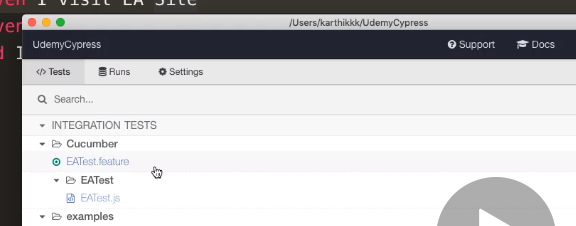


Install cucumber (Gherkin)Full support plug in , compatible with Protractor & Cypress.

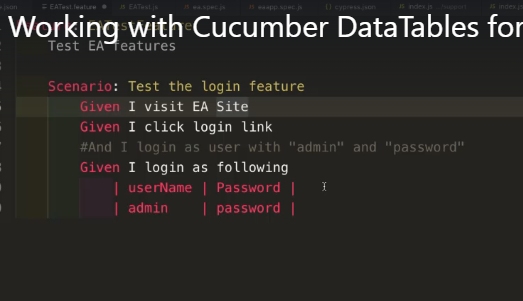






**Working with Data Tables for multiple data in cypress-cucumber**





You also have to add support for .features files to your Cypress configuration

cypress.json

**{**

**"testFiles": "\*\*/\*.{feature,features}"**

**}**

**{**

**“testFiles”:”\*\*/\*.{feature,features}”**

**}**

To run the bundled tests:

Cypress run - -spec \*\*/\*.feature

Feature: Smart Tagging

As a cucumber cypress plugin which handles Tags

I want to allow people to select tests to run if focused

So they can work more efficiently and have a shorter feedback loop

Scenario: This scenario should not run if @focus is on another scenario

Then this unfocused scenario should not run

@focus

Scenario: This scenario is focused and should run

Then this focused scenario should run

@this-tag-affects-nothing

Scenario: This scenario should also not run

Then this unfocused scenario should not run

@focus

Scenario: This scenario is also focused and also should run

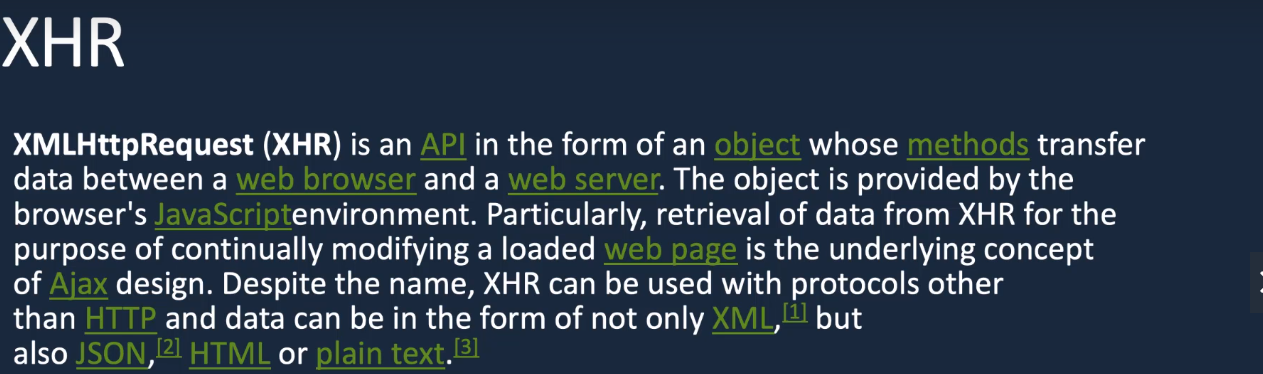
Then this focused scenario should run

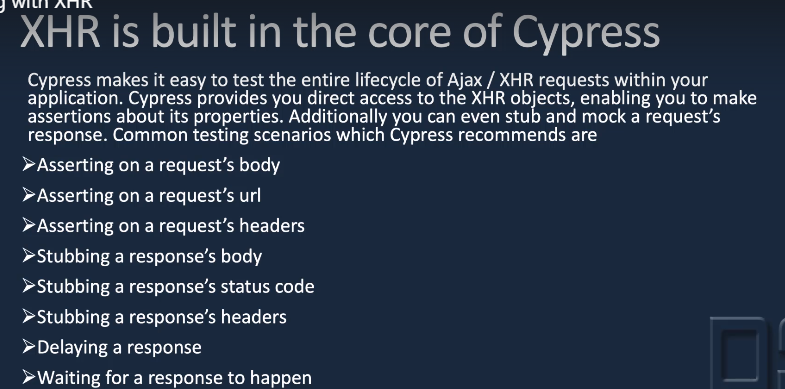
**To ignore any test files:**

"ignoreTestFiles": "\*.js"

"test": "cypress run --env TAGS='@e2e-test' --spec 'cypress/integration/\*\*/\*.feature'",

**Introduction to working with XHR**





Interacting with WebElements:

Time travel feature in Cypress.

1. To visit a url in browser

Cy.visit(‘http://ww.trainline.com’)

1. To find the url string

Cy.url().should(‘eq’, ‘http://ww.trainline.com’)

1. To find a webelement to operate

cy.get(‘input[name=username]’).should(‘be.enabled’).type(‘mercury’)

cy.get(‘input[name=password]’).should(‘be.enabled’).type(‘mercury’)

cy.get(‘input[name=login]’).should(‘be.eanbled’).click()

1. Validate Title of page

cy.title().should(‘eq’, ‘Page tile string’)

1. radio button

cy.get(‘input[name=rpundtrip]’).should(‘not.be.checked’).should(‘be.visible’).click()

**Page object model pattern:**

class LoginPage {

lanuchUrl(){

Cy.visit(‘http://www/w3schools.com’);

}

fillEmail(value){

const field=cy.get(‘input[name=username]’)

field.clear()

field.type(value)

return this;

} // similarly write for password Filed

submitLoginCred(){

Const subBtn=cy.get(‘input[name=submit]’)

subBtn.click()

}

}

export default LoginPage

Make use of the page class in other tests using

Import LoginPage from ‘../pageObjects/LoginPage’

Const lp=new LoginPage()

lp.launchUrl()

lp.fillEmail(‘sushantpatil@gamil.com’)

lp.fillPswd(‘password’)

lp.submitLoginCred()

cy.title().should(‘eq’,’Title of the Page’)

**JS Test Framework[Mocha] and Assertion library(Chai)**

<https://www.youtube.com/watch?v=zHzaUtHncQg>

npm install mocha

npm install chai

Mocha provided describe function – which takes first argument as Name of Function

Second parameter is a function, which holds the multiple tests [Test suite]

It function -🡪 A single test

It.skip(…………….) 🡪 This test won’t run

Describe.skip(…)🡪 This describe tests won’t run

It.only(….)-🡪 Only this test runs

Describe.only(…)🡪 Only this suite runs

.skip and .only can be used across multiple it and describe blocks

Chai is assertion library can be used with any JS framework

Chai provides 3 different assertion styles

Should: c, more explosive and readable

Expect:

Assert: TDD, more classical

Example:

Let name=’Sushant’

name.should.be.a(‘string’)

or

expect(name).to.be.a(‘String’)

or

assert.typeOf(name, ‘String’)

name.should.not.equal(‘Suprit’)

name.should.equal(‘Sushant’)

Mochawesome Reports:

<https://tsh.io/blog/cypress-and-mochawesome-reporting/>

Cypress and Xpath:

<https://www.youtube.com/watch?v=82c1u1qXkXI>

plugin xpath install

**Changing the browser behavior with Cypress Events:**

From execute automation

* Maximize the window
* Incognito Mode -- by default Cypress run test in new chrome profile (browser)
* Set chrome extension etc

Index.js

Const cucumber=require(‘cypress\_cucumber\_preprocessor’).default

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

On(‘file-preprocessor’, cucumber());

On(‘before.browser.launch’, (browser ={}, args)=>{

If(browser.name===’Chrome’){

args.push(‘—start-fullscreen’)

}else if(browser.name===’Electron’){

args.push(‘—start-fullscreen’)

}else{

}

**Debugging with Cypress:**

**Jenkins integration:**

**java -jar jenkins.war**

1. Password after running Jenkins.war file using: java –jar Jenkins.war

**dad1a8db9d164dc9a92cec510dec1995**

1. **UserName : SushAdmin**
2. **Password : SushPassword**
3. [**Sushant.patil@softvision.com**](mailto:Sushant.patil@softvision.com)
4. [**http://localhost:8080/**](http://localhost:8080/)

**https://dev.to/walmyrlimaesilv/testing-copy-to-clipboard-with-cypress-1414**

cy.contains('button', 'Copy to clipboard').click()

cy.assertValueCopiedToClipboard('Hello World')

cy.get('#copy-to-clipboard-input-field')

.clear()

.type('Foo bar baz')

cy.contains('button', 'Copy to clipboard').click()

cy.assertValueCopiedToClipboard('Foo bar baz')

})

Cypress.Commands.add('assertValueCopiedToClipboard', value => {

cy.window().then(win => {

win.navigator.clipboard.readText().then(text => {

expect(text).to.eq(value)

})

})

})

**Difference between cy.get and cy.contains()**

[**https://dev.to/walmyrlimaesilv/a-simple-cypress-exercise-2lfe**](https://dev.to/walmyrlimaesilv/a-simple-cypress-exercise-2lfe)

**Testing Shadow dom elements using selectorshub chrome plugin for cssSelector/xpath:**

Shadow DOM **allows hidden DOM trees to be attached to elements in the regular DOM tree** — this shadow DOM tree starts with a shadow root, underneath which you can attach any element, in the same way as the normal DOM

A dom tree within another , shadow dom starts with Shadow root

[**https://sanjayselectorshub.medium.com/automate-shadow-dom-in-selenium-cypress-and-playwright-507acdd96e09**](https://sanjayselectorshub.medium.com/automate-shadow-dom-in-selenium-cypress-and-playwright-507acdd96e09)

Automating shadow DOMs have been always a quite challenging and time taking task because of couple of below reasons -

1- Identifying shadow dom (shadow root) /element

2- Writing and verifying selectors for shadow root element as we can’t verify selector for shadow dom elements using devtools ctrl+f. solution is use selectorhub plugin for chrome

3- XPath doesn’t support shadow dom elements so we have to use on css selector.

Element under shadow dom , text field , type Sushant

* Cy.get(‘#username’).shadow().find(‘#kills’).type(‘Sushant’)

Element under nested shadow dom , text feild surname -Patil

* Cy.get(‘#username’).shadown().find(#app2).shadow().find(#surname).type(‘Patil’)

**Selectorshub chrome plugin provides a feature to identify shadow elements for all tools cypress and selenium, playwright**

**Selenium with Java code:**

Element under shadow dom , text field , type Sushant

**SearchContext shadow =driver.findElement(By.cssselactor(“#username”)).getShadowroot();**

**Shodow.findElement(By.cssSelctor(“#kills”).sendKeys(“Sushant”);**

Element under nested shadow dom , text feild surname –Patil

SearchContext shadow0 =driver.findElement(By.cssselactor(“#username”)).getShadowroot();

SearchContext shadow1 = shadow0.findElement(ByCssSelevtor(“#app2”).getShadowroot();

shadow1.findElement(By.cssSelector(“#surname”).sendKeys(‘Patil’)

**Testing shadow dom elements inside iframe**

**Handling invisible or hidden elements in Cypress**

https://www.tutorialspoint.com/cypress/cypress\_hidden\_elements.htm#:~:text=Cypress%20has%20another%20technique%20for,and%20we%20can%20click%20it. ss: