

CYPRESS:

What is Cypress?

Cypress is a **purely JavaScript-based front-end testing tool** built for the modern web.

Why cypress?

It aims to address the pain points developers or QA engineers face while testing an application. Cypress is a more developer-friendly tool that uses a unique DOM manipulation technique and operates directly in the browser. Cypress also provides a unique interactive test runner in which it executes all commands.

Cypress Ecosystem:

Cypress consists of a **free, open source, locally installed application** and **Cypress Cloud for recording your tests**.

- **First:** Cypress helps you set up and start writing tests every day while you build your application locally. *TDD at its best!*

- **Later:** After building up a suite of tests and integrating tests with your CI Provider, Cypress Cloud can record your test runs. You'll never have to wonder: *Why did this fail?*

Installing cypress:

We all know that cypress is a JavaScript based project so we have to create the environment that should have JavaScript runtime environment

- First of all we have to install **nodeJs** (Open source, backend JavaScript runtime), (For window users it is recommended to install .msi based on system configuration.)

- Install **VS code** and then create the file **Package.json** (It will store all your metadata helps to manage your project dependencies)
- From this path we will hit **npm install** (node package manager) whoever will developed software which is JavaScript based will host all their package in npm
- Likewise when u hit npm install command through package.Json it will scan your package where u have given u need cypress 10.11 or any other specific version and it automatically connects to its repository.

Installing package.Json:

- First open VS code click on left end and then open terminal.
- Give command **mkdir CypressAutomation**. //mkdir is command to open new file while cypressautomation is folder name.
- After that give command **cd CypressAutomation** to navigate that folder.
- Then hit **npm -I init** to download package.Json.
- After installing type yes.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\LAPTOP LINKS> mkdir cypressparactice

Directory: C:\Users\LAPTOP LINKS

Mode                LastWriteTime         Length Name
----                -
d-----          12/21/2022  11:12 PM             cypressparactice

PS C:\Users\LAPTOP LINKS> cd cypressparactice
PS C:\Users\LAPTOP LINKS\cypressparactice> npm -i init
npm WARN config global '--global', '--local' are deprecated. Use '--location=global' instead.
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (cypressparactice)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\Users\LAPTOP LINKS\cypressparactice\package.json:

{
  "name": "cypressparactice",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Installing Cypress:

Again go to terminal of package.Json then install cypress using this command.

npm install cypress --save-dev

```
{ } package.json > ...
10   "license": "ISC",
11   "devDependencies": {
12     "cypress": "^12.2.0"
13   }
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\LAPTOP LINKS\cypressparactice> npm install cypress --save-dev
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.

> cypress@12.2.0 postinstall C:\Users\LAPTOP LINKS\cypressparactice\node_modules\cypress
> node index.js --exec install

Installing Cypress (version: 12.2.0)

✓ Downloaded Cypress
✓ Unzipped Cypress
✓ Finished Installation C:\Users\LAPTOP LINKS\AppData\Local\Cypress\Cache\12.2.0

You can now open Cypress by running: node_modules\.bin\cypress open

https://on.cypress.io/installing-cypress

npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN cypressparactice@1.0.0 No description
npm WARN cypressparactice@1.0.0 No repository field.

+ cypress@12.2.0
added 166 packages from 176 contributors and audited 166 packages in 825.99s

28 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

New major version of npm available! 6.14.17 -> 9.2.0
Changelog: <https://github.com/npm/cli/releases/tag/v9.2.0>
Run `npm install -g npm` to update!

Latest version of cypress is installed.

A folder called **node-module** is automatically created where all and contain related dependencies

Opening cypress:

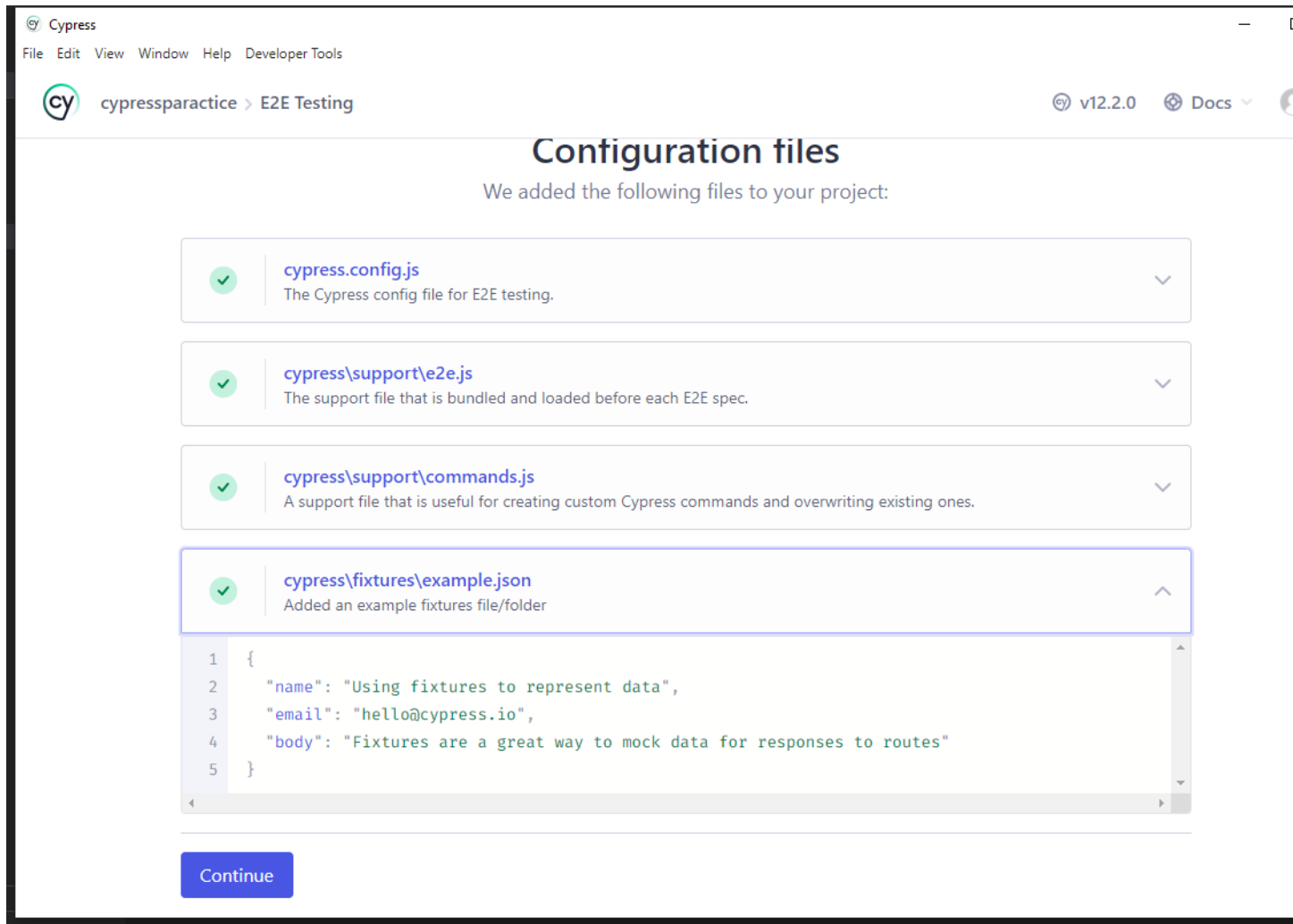
We use the code to open cypress:

➤ **`npx cypress open`**

Choose testing type...

As a QA tester we will be focusing on **end-to-end testing**.

Here when we click on continue a folder name **cypress.config.js**



Choose a browser:

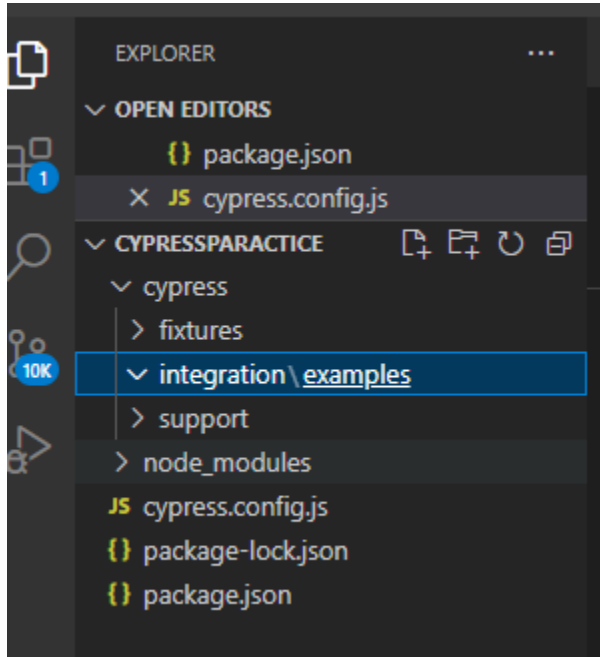
We can do testing in different browser

(Chrome, Edge, Firefox, Electron)

➤ **For electron (`npx cypress run --headed`)**

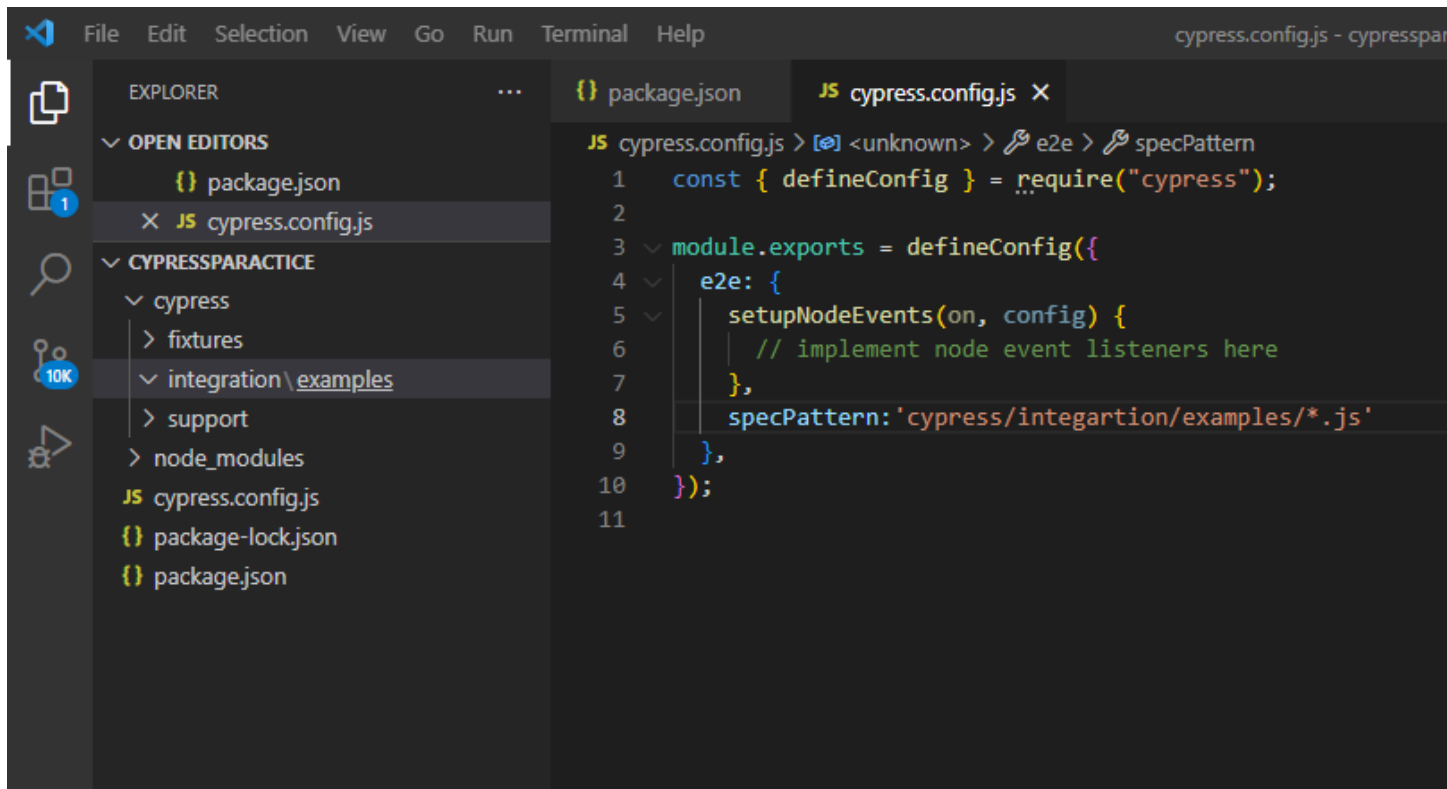
➤ **For Chrome(npx cypress run –browser chrome)**

First of all we will create a new folder named integration where we have to write all our test cases.



Lets suppose we have given **test1.js** name to our test case.

Now we have to create spec files. How our test runner know where we have written our test cases that's why we have to write in the folder cypress.config.js the path through which it can easily approach our test cases.



First of all write the structure(syntax) for writing the text

```
describe('my firsttest suite',function()

{
    it('my firstTest case',function(){
//all the test case is written inside it
})
})
```

LOCATORS:

Cypress only supports the Css selectors while other automation tools like selenium use XPath, Css selectors and many others.

- ❖ If we put # before id as #id it become Css selector.
- ❖ If we put a dot before classname as .classname it is also a Css selector. // remember there is no space in class name if there is any put a dot there.
- ❖ If we want a unique Css selector we will put tagname.classname or tagname#idname.

- ❖ You can also take any attribute and write Css selector for it as
tagname[attribute=value] // Tagname is always optional.

ASSERTIONS:

Cypress bundles the popular [Chai](#) assertion library, as well as helpful extensions for [Sinon](#) and [jQuery](#), bringing you dozens of powerful assertions for free.

1. Should():

.should() is an **assertion**, and **it** is safe to chain further commands that **use** the subject. ... Use Cypress commands before or after **.should()** instead.

Syntax:

```
.should(chainers)
```

Example:

We have to assert checkbox is disabled.

```
Cy.get('.checkbox').should('be.disabled')
```

2. And():

And is an alias of should().

Syntax:

```
.and(chainers)
```

Example:

```
Cy.get('.button').should('have.active', 'class').and('not.be.disabled')
```

Difference between then() and should()/and()?

Using **.then()** allows you to use the yielded subject in a callback function and should be used when you need to manipulate some values or do some actions.

When using a callback function with `.should()` or `.and()`, on the other hand, there is special logic to rerun the callback function until no assertions throw within it. You should be careful of side affects in a `.should()` or `.and()` callback function that you would not want performed multiple time.

Common Assertions:

Here are some common assertions.

3. Length:

```
// retries until we find 3 matching<li.selected>
Cy.get('li.selected').should('have.length',3)
```

4. Text context:

```
//assert the element text content is exactly the given text
Cy.get('[data-testid= "user-name"]').should('have.text', 'joe Smith')
```

5. Visibility:

```
//Try until some elements are visible
Cy.get('.li').should('be.visible')
//Try until every element is invisible
Cy.get('li.hidden').should('not.be.visible')
```

6. State:

```
//Try until radio is checked
Cy.get(':radio').should('be.checked')
```

Chai assertions:

Chainers	Examples
true	expect(true).to.be.true
string	expect('testing').to.have.string('test')

Commands:

1) Cy.visit:

To visit a remote URL.

```
Cy.visit(url)
```

2) Cy.wait:

Wait for a number of milliseconds or wait for an aliased resource to resolve before moving on to the next command.

```
Cy.wait(time)
```

3) Cy.wrap:

Yield the object passed into `.wrap()`. If the object is a promise, yield its resolved value.

```
Cy.wrap(subject)
```

4) Cy.then:

Enables you to work with the subject yielded from the previous

5) Cy.pause:

Stop `cy` commands from running and allow interaction with the application under test. You can then "resume" running all commands or choose to step through the "next" commands from the Command Log.

It is [unsafe](#) to chain further commands that rely on a DOM element as the subject after `.pause()`.

```
Cy.pause()
```

6) Cy.go:

Navigate back or forward to the previous or next URL in the browser's history.

```
Cy.go(direction)
```

Actions:

1. Type:

Type into a DOM element.

```
Cy.get('input').type(text)
```

2. Select:

Select an `<option>` within a `<select>`.

```
Cy.go('input').select(value)
```

3. Check:

Check checkbox(es) or radio(s).

```
Cy.get('input').check()
```

4. Click:

Click a DOM element.

```
Cy.get('.btn').click
```

Queries:

1. As:

Assign an alias for later use. Reference the alias later within a `cy.get()` query or `cy.wait()` command with an `@` prefix.

```
//.as(aliasname)  
cy.get('.main-nav').find('li').first().as('firstNav')
```

2. Eq:

Get A DOM element at a specific index in an array of elements.

```
//.eq(index)  
cy.get('tbody>tr').eq(0)
```

3. Filter:

Get the DOM elements that match a specific selector.

```
//.filter(selector)
cy.get('td').filter('.users')
```

4. Get:

Get one or more DOM elements by selector or alias.

```
//cy.get(selector)
cy.get('.list > li') // Yield the <li>'s in .list
```

5. Invoke:

Invoke a function on the previously yielded subject.

```
//.invoke(functionName)
cy.get('.input').invoke('val').should('eq', 'foo') // Invoke the 'val' function
```

6. Title:

Get the `document.title` property of the page that is currently active.

```
cy.title() // Yields the documents title as a string
```

7. Url:

Get the current URL of the page that is currently active.

```
cy.url()
```

This is an alias of `cy.location(href)`.

Events:

Cypress emits a series of events as it runs in your browser. These events are useful not only to control your application's behavior, but also for debugging purposes.

Name:	window:confirm
Yields:	the confirmation text (String)
Description:	Fires when your app calls the global <code>window.confirm()</code> method. Cypress will auto accept confirmations. Return false from this event and the confirmation will be canceled.
Event	Details
Name:	window:alert
Yields:	the alert text (String)
Description:	Fires when your app calls the global <code>window.alert()</code> method. Cypress will auto accept alerts. You cannot change this behavior.
Event	Details
Name:	window:before:load End-to-End Only
Yields:	the remote window (Object)
Description:	Fires as the page begins to load, but before any of your applications JavaScript has executed. This fires at the exact same time as <code>cy.visit()</code> <code>onBeforeLoad</code> callback. Useful to modify the window on a page transition.
Event	Details
Name:	window:load
Yields:	the remote window (Object)
Description:	Fires after all your resources have finished loading after a page transition. This fires at the exact same time as a <code>cy.visit()</code> <code>onLoad</code> callback.
Event	
Name:	window:before:unload
Yields:	the actual beforeunload event (Object)

Event	
Description:	Fires when your application is about to navigate away. The real event c
	on.

Cypress is asynchronous in nature:

As cypress is a java-based project and javascript is asynchronous itself, Cypress is also an asynchronous in nature.

- Cypress is asynchronous in nature and there is no guarantee in sequence of execution but cypress takes care of this.
- While other testing tools such as Selenium are synchronous.

Dynamic and static dropdown:

How to handle static dropdown using cypress?

- If it is static the tagname will always be “**selected**”. It is also html rule
- Here is a small difference of handling the static dropdown and dynamic dropdown.

```
//static dropDown
cy.get('select').select('option2').should('have.value','option2')

//Dynamic Dropdown
cy.get('#autocomplete').type("ind")
```

Cypress Framework:

Here is a sample framework for automating a website.

Code Structure

Page Object Model

Importing Pages in Test Class

For Example in Below Example

- Importing Home Page and Products Page inside Test Class
- Importing Test data from Fixtures

- Similarly as in Selenium we are following Page Object Model here **Importing Pages Inside Test Classes**

```
/// <reference types="Cypress" />
import HomePage from'../../support/PageObjects/HomePage'
import products from'../../support/ProductsPage/products'
describe('My Second Test Suite', function()
{
  before(function(){
    cy.fixture("example").then(function(data)
    {
      this.data=data
    })
    // runs before all tests in the block
  })
  //we have to resolve the promise for the value that data returns
  // this.data and data are different variables this.data is global
  it('My FirstTest case',function() {
    const homePage=new HomePage()
    const Products=new products()
    cy.visit(Cypress.env('url')+"/angularpractice/")
    //This is example of subdomain means main domain is url itself while
    subdomain which is angular paractice is concatenated with domain
    // cy.visit('https://rahulshettyacademy.com/angularpractice/')
    Cypress.config('pageLoadTimeout', 100000)
    //for only one test case is we want to manage runtime

    homePage.getEditBox().type(this.data.name)
    homePage.getGender().select(this.data.gender)
    homePage. getTwowaydatabinding().should('have.value',this.data.name)
    homePage.getEditBox().should('have.attr','minlength','2')
    homePage.getEnterprenuer().should('be.disabled')
    //Cypress.config('pageLoadTimeout', 100000)
    // for one specific click
    homePage. getShopTab().click()

    //cy.pause() this is only used for debugging the test case
    this.data.ProductName.forEach(function(element){

      cy.SelectProduct(element)

    });
  });
});
```

```

Products.getcheckout().click()
var sum=0
cy.get('tr td:nth-child(4) strong').each(($el, index, $list) => {

  const amount=$el.text()
  var res=amount.split(" ")
  res=res[1].trim()
  sum=Number(sum)+Number(res)
}).then(function()
{
  cy.log(sum)
})
cy.get('h3 strong').then(function(element){
  const amount=element.text()
  var res=amount.split(" ")
  var total=res[1].trim()
  expect(Number(total)).to.equal(sum)
})
cy.contains('Checkout').click()
cy.get('#country').type('India')
cy.get('.ng-untouched > .btn').click()

cy.get('#country').click({force:true})
cy.get('input[type="submit"]').click
cy.get('.alert').then(function(element)

{
  const Actualtext=element.text()

  expect(Actualtext.includes("Success")).to.be.true
})

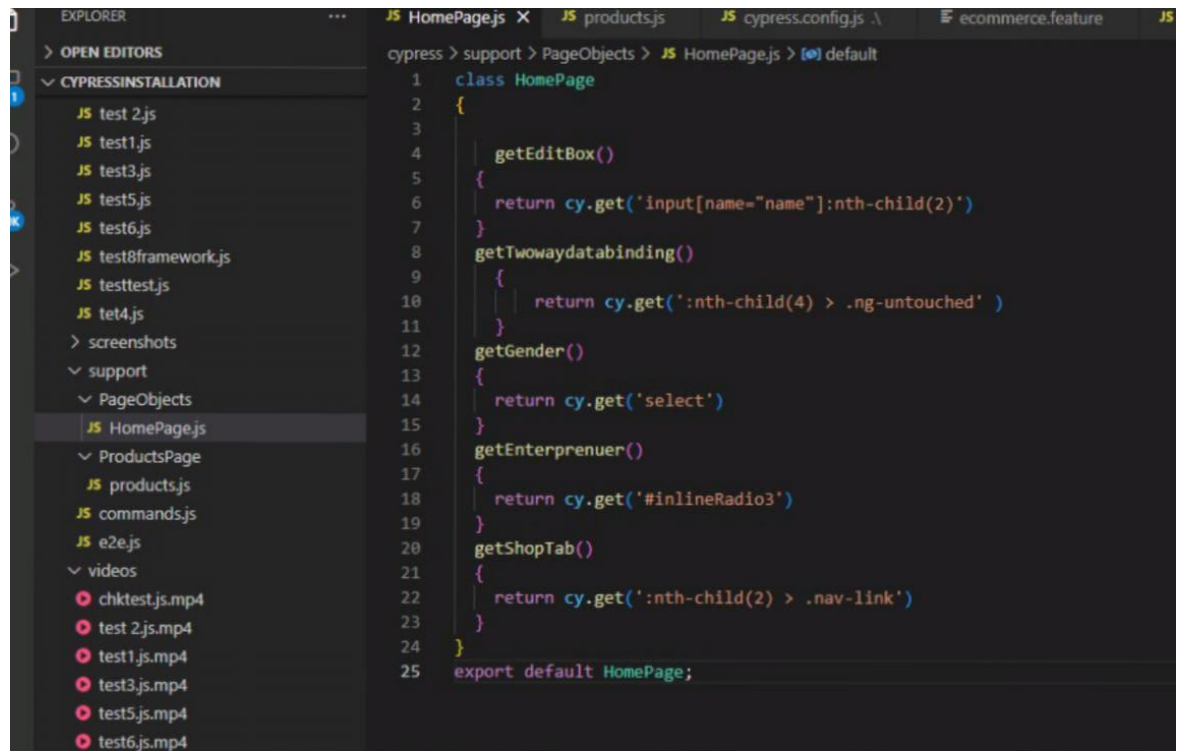
})
})

```

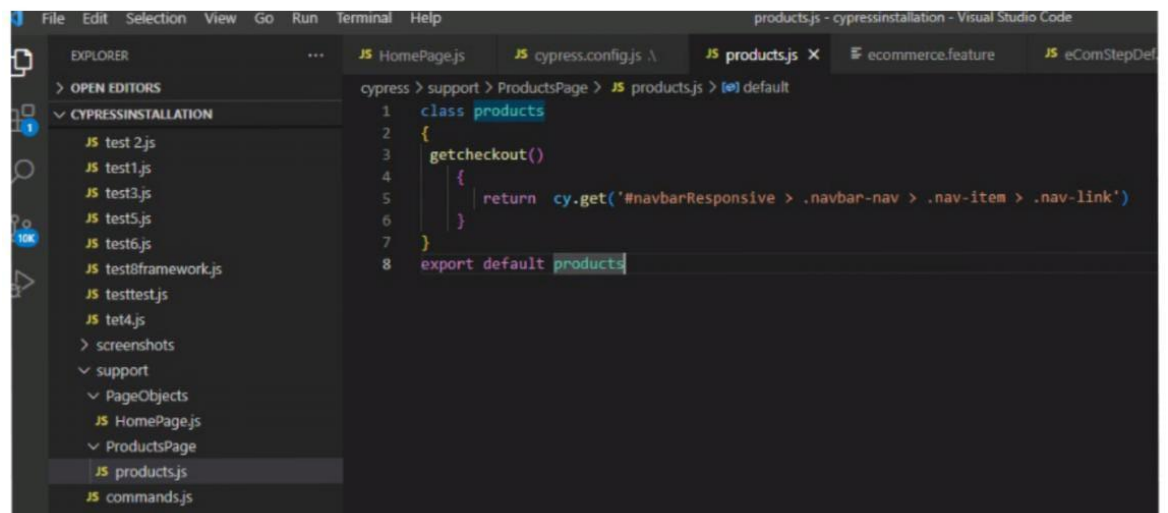
- First of all we have to navigate the url. In this framework we import the url from **cypress.config.js**.


```
JS cypress.config.js > ...
13
14 module.exports = defineConfig({
15   chromeSecurityWebsite: false,
16   defaultCommandTimeout: 8000,
17   pageLoadTimeout: 30000,
18
19   // reporter: 'mochawesome',
20   env: {
21     url: "https://rahulshettyacademy.com",
22   },
23
24   retries: {
25     runMode: 1,
26   },
27
28   e2e: {
29     // setupNodeEvents,
```

- We also import from homepage and products class in our test case.



```
1 class HomePage
2 {
3
4   getEditBox()
5   {
6     return cy.get('input[name="name"]:nth-child(2)')
7   }
8   getTwowaydatabinding()
9   {
10    | return cy.get(':nth-child(4) > .ng-untouched' )
11  }
12   getGender()
13   {
14     return cy.get('select')
15   }
16   getEnterprenuer()
17   {
18     return cy.get('#inlineRadio3')
19   }
20   getShopTab()
21   {
22     return cy.get(':nth-child(2) > .nav-link')
23   }
24 }
25 export default HomePage;
```



```
1 class products
2 {
3   getcheckout()
4   {
5     return cy.get('#navbarResponsive > .navbar-nav > .nav-item > .nav-link')
6   }
7 }
8 export default products;
```

HomePage and products are classes while objects are written inside the classes.

Java Script Fundamentals for Automation:

- **JavaScript Hello word program:**

Simply the code for printing is

```
console.log("Hello word")
```

For single line comment we use // in the start of line.

For multiple line code we use /* at start of code while */ in the end of the code.

- **Declaring the variables in Java Script:**

In Java Script we need not to write data types to declare the variables.

Commonly **Let** keyword is used to declare the variable however in ES6 Engine we also use **Var and Const** Keywords.

We cannot redeclare variable with Let but it is possible with var.

Using Let we can reassign the value to variable.

An example of declaring the variables is:

```
let a=6  
console.log(a)
```

We declare and then print the variable.

- **Understanding the datatypes in JavaScript:**

If we want to find the data type of variable we use the method **typeof()**

```
console.log(typeof(a))
```

In Java script there Is no variation of integer float

As if we write b=20.987

The integer type remains integer.

String is declared as:

```
let c="CYPRESS"
```

```
console.log(c)
```

Boolean which is also a data type has only two values (true and false) can be declared as:

```
let d=true  
console.log(typeof(d))
```

Null and undefined are also data types.

We can also use the assignment operators.

```
var c=a+b  
console.log(c)
```

There is also a Not operator which applies only on Boolean datatype represent by!

```
console.log(!c)
```

• **Loops and condition in JavaScript:**

➤ **IF-ELSE condition:**

If condition is usually used in JavaScript. If there is more than one conditions elseif condition is used.

```
let flag=true  
if(!flag)  
{  
  console.log("condition is satisfied")  
}  
else{  
  console.log("condition is not satisfied")  
}
```

Here is a tricky thing that the value of flag remain true but condition is not satisfied because we use **Not** operator.

➤ **While loop:**

- When we use While condition the conditions executes infinite times. So, whenever we use while condition its important to tell how many times we want to execute the code.

```
let i=0
```

```
while(i<10)
{
    i++
    console.log(i)
}
```

➤ Do-while loop:

If we use do-while condition the code is executed once before checking the condition.

```
do
{
    i++
}
while(i<10);
console.log(i)
```

➤ For loop:

In for loop we initialize the variable, then satisfy the condition and then increment or decrement of the variable is done.

```
for(let k=0;k<10;k++)
{
    console.log(k)
}
```

• Array and its methods:

If we have a set of values, we have store all of them in a container called array.

The array is declared as following ways:

We may initialize it as:

```
var marks=Array(6)
var marks=new Array(20,34,54,23,43,23,45)
//Here if we use Let keyword it will throw an error because we cannot redeclare marks y using let.
```

Other methods to initialize the array is:

```
var marks= [20,34,54,23,43,23,45]
```

The square brackets shows that this is an array.

The array can be printed as:

```
console.log(marks[2])
```

Print using the For loop:

```
for(let i=0;i<marks.length;i++)  
{  
    console.log(marks[i])  
}
```

Now if we reassign value at any index .It will changed....

```
marks[3]=56  
console.log(marks[3]) //56
```

Now we have to find the length of the array.

```
console.log(marks.length)
```

Now we have to add new element at the end of array.

```
marks.push(65)  
console.log(marks) // [20,34,54,23,43,23,45,65]
```

Now we let assume we have to need the previous original array.

```
marks.pop(65) // [20,34,54,23,43,23,45]
```

If we want to add the array at the start of the array.

```
marks.unshift(12) //[12,20,34,54,23,43,23,45]
```

Index of the array can be found as:

```
console.log(marks.indexOf(54)) //2
```

To check whether the variable is present in the array or not?

```
console.log(marks.includes(98)) //false
```

Array can be cut into half or some digits are removed as:

```
console.log(marks.slice(2,5)) //[54,56,43]
```

Let suppose we want to sum all the elements of an array.

```
Var sum=0
```

```
for(let i=0;i<marks.length;i++)
{
    sum=sum+marks[i]
}
console.log(sum)
```

We can also sum all the values by using the **Reduce Filter** method:

If you want to update or iterate the values reduce method is used.

```
let total=marks.reduce((sum,marks)=>sum+marks,0)
console.log(total)
```

If you want to filter your array according to the given condition.

```
let even=marks.filter(marks=>marks%2==0)
console.log(even)
```

Mapping is all about from mapping from one value to another value. Filter only works when condition is satisfied.

```
let mappedarray=marks.filter(marks=>marks*3)
console.log(mappedarray)
```

Array can be sorted as:

```
marks.sort()
console.log(marks)
```

Sometimes it may throw errors while sorting as if we write 003 it cannot be sorted correctly using this.

Then we will use the anonymous function to sort as we use reduce, filter and map.

```
console.log(marks.sort((a,b)=>a-b))
```

For reversing

```
marks.reverse()
console.log(marks)
//For strings this method works
```

For number data type:

```
console.log(marks.sort((a,b)=>b-a))
```

• **Functions in array:**

Functions are the blocks of code.

The simple code for function in JavaScript is:

```
function add(a,b)
{
    return a+b
}
let sum=add(2,3)
console.log(sum)
```

For anonymous functions (that do not have function name // function expression)

```
let sumofintegers =function(c,d)
{
    return c+d
}
```

To make our code more clear and simple we also remove function keyword and code as:

```
let sumofintegers =(c,d)=> c+d
console.log(sumofintegers(2,3))
```

• **Strings in JavaScript:**

Simple way to declare a string in JavaScript is:

```
let day='tuesday '
console.log(day.length) //8
```

here day.length shows the length of array.

If we want to remove some of letters we use **slice** method

```
let subday =day.slice(0,4)
console.log(subday)
```

To print on the specific index:

```
console.log(day[1])//1
```

To split the string:

```
let splitday =day.split("s")
console.log(splitday)
```

To convert string into number:

```
let date='23'
let nextdate ='27'
let diff=parseInt(nextdate)-parseInt(date)
```



```
console.log(diff)
```

To convert number into string:

```
let newquote=day+"is funday"  
console.log(newquote)
```