Rahul shetty 29 Nov start

cypress

1) What is cypress

→ cypress is a next generation front end Automation testing tool build for modern web application (like selenium QTP earlier tool

Why cypress

① if there is any element which is in disable mode and it will be enable after few seccten cypress will automatically wait for this. (timeout)
unlike selenium we need to add w8 to acheive same.

explaination

② force click. (if there is any dropdown which open another one and because of that below element get hide so if we found that element with the help of force click we can do it.
↑ explain as above.

explaination

actual

1) Cypress automatically waits for commands and assertion before moving on. | No more asyn hell.

2) [ability] to test edge Testcase by mocking the server response.
(means with handling frontend UI we can doing API call make API responce also)
out of box testing possible with the help of cypress

3) Cypress takes snapshot as our tests run. we can hover each commands in command log to see exactly what happend at each step.

4) Because of Architecture design. cypress delivered fast, consistent & reliable test execution compared to other Automation tools.

Be AatmaNirbhar. Use Kaagaz

[illegible] in real time by manipulating Dom & altering Network request & response on the fly

cypress open doors to New kind of testing & having ultimate control over your application. (Front & back)

(unit testing / integration testing / end to end t
it feet end & every layer of SDLE

cypress Browser support.
Major Browser are
chrome
firefox & IE.
Electron → default Browser.
→ one of the version of chrome. which renderd engine which ~~used~~ chrome also but very optim way & does not having all pluging which chrome has.
→ lightweighted & test run faster than other

cypress Component.
Test Runner
Dash Board Service

Which enviroment require for cypress.
node. js (node install)

Be AatmaNirbhar. Use Kaagaz

(npmjs.com)

NPM → Node package manager

1) package.json

* A package.json is a json file that exists at the root of a Javascript / Node project.
* it holds metadata relevant to the project & it is used for managing the project dependencies.

To Download cypress

1) npm init → create package.json file for us.

1) create folder → go to that folder
2) 1) npm init → create package.json file.
   2) npm install cypress
      npm install cypress --save-dev (create entry in package.json)
   3) npm install npx
   4) npx cypress open

- For Javascrip there are framework like → jasmin मोका mocha.

cypress recommonded to used mocha framework.

(suite)

ex1)
```
describe ('Test Scenario', () => {
    it ('Test Case', () => {
        cy.visit (" url ")

    }
})
```

(object)

* cy → globle command which help to invoke any cypress command & it comes default.

Be AatmaNirbhar. Use Kaagaz

For debbuging cypress is best

1) cy.visit ("") ⇒ To visit any URL

[There are 4 browser support cypress]

1) chrome
2) Firefox
3) ~~IE~~ (Michrosoft edge) Ms edge
4) Electron (default Browser) (highly suggested by cypress)
   - (electron is also kind of chrome, it used to rendering html instead of pulling all chrome option)
   - more stable & High performance & Fast.
   - cypress is invoke electron by default when we installing.

* We are call each & every file in javascript as spec file

⇒ [Headless]

1) Through command line cypress run in headless mode
   NPX ~~run~~ cypress run.

npx cypress ~~npx~~ run filepath ⇒ open specific file.
(execution time save)

[Headed]

npx cypress run --headed

npx cypress run

npx cypress run ~~filename~~ --spec "filepath"

npx cypress run --headed

npx cypress run --headed --browser chrome

Be AatmaNirbhar. Use Kaagaz

* Project structure
→ when we give command npx cypress open by default cypress folder made in which there are 4 folder in cypress.
1) fixture
2) plug integration
3) plugins
4) support
5) video

1) Fixture ⇒

Cypress
- Fixture
- integration
- plugins
- support
- video

node modules
1) cypress.json
2) Package-lock.json
3) Package.json.

1) fixture → [To Store Test Data]
- we r not hardcode the data in our Test Case
- it driven from external file like excel json
- cypress suggest to put all data in fixture so we can directly used it in test case without adding code to drive

2) Integration → [To write Test Case]

Be AatmaNirbhar. Use Kaagaz

< reference type = "cypress" />
Autosuggestion

3) Plugin → It is used as listner (लीस्नर)
To handle cypress event (listener) mean if this event happen the what to do.

4) Support → To write customized command & reusable method
i.e we create one utility and place all method in that folder.
~~resyn~~ reused it.

5) Video → For every Test run we have video

6) Node-Module → it create when we run command npm install cypress.
if we dont have that module we cannot run any cypress command.

7) cypress.json → we have some default configuration it testrunner (cypress)
Setting > Configuration
but if we want to override that default values
we ~~used~~ write code for cypress.json folder.

- cypress uses the cypress.json file, if we want to specify any custom configuration or override any custom configuration we can used cypress.json folder.

Be AatmaNirbhar. Use Kaagaz

chropath plying element highlight!

* Identify element on web page.

→ cypress only support 'CSS selector'

id #idname tagname# idname → make more unit.

class .classname

tagname.classname → to make more unique

attribute tagname [attribute='value']

* Traverse parent to child by using tag

→ parent tagname space child tagname.

* cy.get() → get is the method which actually get the locator present on your web page. i.e. element which present on web page.

* Basic Assertion (chai (library) Assertion)

1) should → it is assertion which will help you to assert on property return by the stap element

should ('have.length')

* cy.wait(ms) → for waiting cypress to load new element

Be AatmaNirbhar. Use Kaagaz

Handling invisible element with cypress

PAGE NO.
DATE :

* in order to select only visible element in JQuery there is one copy to remove only visible element

(Q3) cy.get ('.product : visible') should ('have.length' 4)
JQuery

20 → 1
80 → 4

* Understanding get & find command
parent child chaining
⇒ cy.get ('.products'). find ('.product'). should ('have.length' 4)
⇒ cy.get ('.products'). find ('.product'). eq (2)

cy.get
cy.contains

1) Find ()
2) eq ()
3) each ()
4) text ()
5) cy.wrap ()

Be AatmaNirbhar. Use Kaagaz

rahulshetty academy.com Promise.

* iterating over the array of web element by using each

* Asynchronous (As it der in nodejs)

* Javascript & cypress are Asynchronous in nature
* cy. And there is No gurentee to sequence of execution by cypress takes the care of this.
  1) script work Asynchronously
  2) execution work synchronously

Be AatmaNirbhar. Use Kaagaz

* if we write console.log( ) in between cypress code. we get console.log() command 1st than other. that means for execution of each and every command it take some time but cypress can't wait it execute another command that way we get console.log as first.

* Selenium & java is synchronus
cypress is Asynchronous.

* But how cypress handle Asynchronous behaviour cause we see each & every step execution done in step by step mean synchronous manner.

* Basic nature is Asynchroous in cypress.

- **Cypress Commands are Asynchronous**

So After all command execution it queued all the command
& cypress begin it running them in order.

- **So in Cypress execution is in Synchronous**

* In Asynchronous every step is written in **Promise**

* What is promise ?
→ Promise is nothing but the state and behaviour of your step.
What state your step is in.

Be AatmaNirbhar. Use Kaagaz

executed or pending

There are 3 ~~state~~ type of Promise.

1) Resolve ⇒ means step is executed
2) Rejected ⇒ means error in step
3) Pending ⇒ means step is yet to executed
trying to load but you're not willing to wait there.

* Ideally you have to wait until promise get resolve.

* So how we know promise is resolve & step is executed so there is one method called
[.then()]

ex →

```
cy.visit("www.google.com")
.then (()=> {
  cy.get ('#id')
})
```

In order to make cypress synchronously we have to concatinate then() block to each line of code but cypress ppl handle this internally so we dont need to give then() method.

Be AatmaNirbhar. Use Kaagaz

let logo = cy.get ('.brand')
logo.text() // this line of code give error.

* As cypress handle promise internally but when we manypullate anything it cannot handle it cause cypress cant recognized that ~~this~~ thing.

So we need to handle promises manually as like

cy.get ('.brand').then (c logo) => {
logo.text()
})

* When we concatinate two cypress command then internally it handle the promises.
→ cy.get ('#id').text()

* cypress also supports Jquery method & text() is Jquery method

* Jquery is dev. tool available to manypulate dom

* Non cypress command cannot resolve promise by themselve we need to manually resolve it by [then()]

Be AatmaNirbhar. Use Kaagaz

23 → running

* Aliasing → as

cy.get ('#1').as ('productlocator')
cy.get ('@productlocator') (reused locator) → I used

⇒ What is need of aliasing
→ if in future web element get changed then instead of changing everywhere you can just change at one placed. so we store element in variable by using as() & call it ('@product')

2nd ans

console.log ()	cy.log()
1) Not related to cypress (JS)	1) related to cypress.
2) output print in browser console.	2) output print on test runner information

* If we want to used Javascript code in cypress we have to resolve promise manually by putting then (cy.)
(Nacypress command)

* Assertion → in order to compare actual value with expected value we ~~you~~ used assertion.
→ assertion come from chait library
1) should ('have.length',4)
2) should ('have.text', 'Green Cart')

Be AatmaNirbhar. Use Kaagaz

Rahulshettyacademy.com/automation practice

PAGE NO.
DATE:

Handle checkBoxes in cypress

*

CheckBox

1) [check() method] should ('be.checked')
~~chec~~ click() / option1]

2) [uncheck() method] should ('not.be.checked')

* if we validating the [Property] of checkbox we have assertion as ⇒ should ('have) this Property
~~should ('hav~~
~~should ('be.checked')~~

* For validating [behaviour] should ('be.checked')

* For combination of assertion we used [.and]
.should ('be.checked').and ('have.value', 'option1')

* Multiple checkbox

1) find common property
cy.get ('input[type="checkbox"]').check(['option1', 'option2'])

in check method we have to pass, value property of checkbox in array for multiple checkbox.

Be AatmaNirbhar. Use Kaagaz

[parent stem child]

[Handling Static Dropdown in cypress]

2 type

* 1) static Dropdown
2) dynamic Dropdown → based on input (suggession get)

1) Static Dropdown

→ if the Dropdown is static then by default its tagname is 'select' (its a html role)

cy.get('select').select('PassArgument')

we are validating Property

so → should('have.value', 'option')
should('have.contain', 'option1')

2) Dynamic Dropdown

→ in dynamic dropdown we have to type any word and it shown all matching it know as dynamic dropdown.

so logic is. find common element after typing & used each method & used condition if

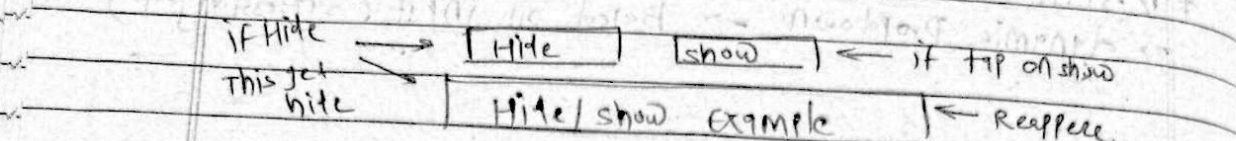
each(el) ⇒ el.text() ===' India'
el.click()

Assertion

cy.get('#id').should('have.value', 'India')

Be AatmaNirbhar. Use Kaagaz

Handling visible & invisible element by using Assertions in Cypress



1) should ('be.visible')
should ('not.be.visible')

* Radio Button

1) used click()
2) used check()

Be AatmaNirbhar. Use Kaagaz

behaviour should be
property should have

(*) How cypress auto handle Alert in web page

1)* Popup handling by using cypress

⇒ cypress auto accept the alert. cypress itself click on [OK]
⇒ cypress not concentrate of child window it only concentrate on parent window only

* Two Types of Popup.
1) Alert → in which we have to click on [OK] button only
2) Confirm → in which we have 2 option [OK] [Cancel]
cypress automatically confirm the Popup.

• But how we know cypress handle the popup.
⇒ on Test runner log we display that thing

* [cypress auto accepts alerts & popups.]

* [cypress also have capability to listen browser events.]

IMP. Whenever any alert come on page automatically [window: alert] event get fired on browser.

* Cypress have ability to control to manupulate our browser DOM (html element)
* cypress take control of our DOM & provide direction to our browser by manupulating our code.
(Selenium cannot control & manupulate DOM)

Be AatmaNirbhar. Use Kaagaz

How To handle popup

1) By using cypress we can trigger [window: alert] event
2) When this event trigger cypress open the alert popup
3) then we capture the text over which written on Alert popup.

— In this way only we handle the popup.

Note → manually or automatically we cannot read the Alert popup
you can read on browser view only.

* cy.on() [For alert]

This method help to trigger any event

ex) example.

```
event → response
cy.on ('window: alert', (string) => {
↓ expect (string). to equal ('actual string')
```

We cannot see popup on UI cypress internally handle it but we can perform action with the help of code.

* our entire testing run on mocha framework.

* if want to compare two value, string assertion is
[expect (value). to. equal ('actual value')]

* cy.on (take two argument)
cy.on ('Firing', 'output of')
event event

imp Que
1)
2)
3)

Be AatmaNirbhar. Use Kaagaz

imp

Que How to validate alert in cypress.

→ 1) cypress have capability to firing browser event.
2) window: alert is one of the browser event which get fired on alert open
3) so you are firing that event through cypress to get access to that alert.

For confirm alert

```
cy.on ('window: confirm', (str) =>
{
expect (str). to equal ('actual string')
})
```

— * —

Be AatmaNirbhar. Use Kaagaz

Flacky → Non Consistance

* How to Handle child tab.

* If we tap on any tab it navigte you to another tab then how to handle it ?

⇒ 1) cypress do not have knowledge to handle or switching to child browser.

* If our html element contain ['target'] attribute then only while tapping on any tab it navigte or open new browser window
  - If we remove target atribute then we remain in same window.

* cypress has ability to manypulated the Dom.

* Invoke() method ⇒

1) invoke() is Command in cypress
2) invoke a function whatever we pass in argument.
3) why we required invoke()

→

1) There is function called ['removeattr'] in JQuery which help to remove atribute of give locator
2) If we have to remove the target atribute
3) So invoke is required to invoke the JQuery function.

ex → cy.get('#opentab').invoke('removeAttr', 'target')
  - click()

1st arg JQuery function — 2nd arg which attr remove

cy.go('back')

Be AatmaNirbhar. Use Kaagaz

* Browser Navigation *.

Req * How to come back to original page after navigating to child browser.

→ cy.go()

↳ used to navigate back or forward to the privious or next URL in the browser's history

cy.go ('back') ⇒ go back to original page

cy.go ('forward') ⇒ go ~~to~~ forward next page.

↳ validation

cy.url() → to get current url of the page that is currently active

assertion → substring

cy.url().should ('include', 'qaclickacadamy')

To check whether this string present in this url

Be AatmaNirbhar. Use Kaagaz

PAGE NO.
DATE:

* Handling web tables with cypress using each ~~compar~~ Command.

Problem → in table find element whoes text is Python & Compare this price to ~~actu~~ expect

* Code ⇒

cy.get ('tr td > td : nth-child(2))

cy.get ['tr td : nth-child (1)']

Be AatmaNirbhar. Use Kaagaz

* Handling mouse hover popup using cypress

- cypress not support mouse hover but we can handle it by using JQuery function or force click
- In JQuery we have function to trigger that mouse hover by using [show() method]

1) [show()] method in JQuery
⇒ it used to display the hidden & selected element.
[NOTE] show method should be applied on immediate parent of hidden element then only it gives output.

ex→ cy.get('.dive.mouse-hover-content').invoke('show')
cy.contains('Top').click()
cy.url().should('include', 'top')

- cypress have ability to click on hidden element as well by using {force: true} arg. in click()

ex Problem ||⇒ verify the popup is displaying when you mouseover & then click on Top button. the ans is ab

ex Problem || → if the requirement is only tap on 'Top' button then we need to your following approach

ans { cy.contains('Top').click({force: true})
cy.url().should('include', 'top')

* How to handle element ~~dy~~ which is invisible mode make it visible.
1) by using JQueries show() method
2) by using {force: click} in click method.
pass arg

Be AatmaNirbhar. Use Kaagaz

both r diff
child tab → diff
child window → diff

amitchikte@gmail.com
9657363602

PAGE NO.
DATE:

* **Handling child window using cypress**

1) Cypress cannot handle the child window ~~also~~ too.
But we have two way to handle

1) way

⇒ to remove target attribute by using removeAttr() function by using of JQuery.

2) way

- If we have child tab & child window that mean your locator or element having 'href' attribute
- From href attribute if we fetch the value we get url
→ so by using that url if we type
cy.visit(url)
→ we can handle child window.

* In order to find any element attribute value we have one method in JQuery called **Prop()**

* **Prop()** → Prop() is the method which help us to get Properties value that mean attribute value.

* cy.get('#opentab').prop('href') ⇒ Not work

cypress method ✗ JQery method

Be AatmaNirbhar. Use Kaagaz

2nd method to work So we Prefer target

cy.get('#openTab').then(el => {
const url = el.prop('href')
cy.visit(url)
})

but 2nd method not work on Rahul shetty Guided

1st url = https://rahulshettyacademy.com/AutomationPractice/
main base domain subdomain

2nd url = https://www.qaclickacademy.com
main domain

Here we change the main domain but we can change subdomain as per security issue Cypress not handle it.

Be AatmaNirbhar. Use Kaagaz

Ifreme

* Handling Frames with cypress using real time examples

- What are Frames ?

→ The Frame is nothing but the HTML document which ~~are~~ is embedded with another HTML document.

*) 1st way by using JQuery & JS ⇒ already done.
*) 2nd way (Not teach by sir).

finding element in ifrae

1) Need to install plugin (make sure you r in Project folder)
[npm install -D cypress-iframe]

2) Need to import by using write command at top of code
[import 'cypress-iframe']

3) For Autosuggestion →
<reference type = 'cypress' />
<reference type = 'cypress-iframe' />

* 1st of all find (ifreme id) so it move focus on frame

Command

- cy.FremLoaded ('cssselector Pass') → cypress get knowledge about Frame

- cy.iframe () → to switch in ifreme mode

Be AatmaNirbhar. Use Kaagaz

< reference types = "cypress" />
<reference types = "cypress-iframe" />

[Code] import 'cypress-iframe'
describe ('Iframe verification', () => {
it ('Iframe verify', () =>
// Real code
cy.visit ("Rahul shetty Practice website")
cy.frameLoaded ('#courses-iframe')
finding element in iframe → cy.iframe().find ("a[href-'mentorship"]).eq(0).click()
// validation

cy.iframe().find ("elem").should ('have.length', 2)
})
})

[Chrom web scraping]

Be AatmaNirbhar. Use Kaagaz

* Hooks

⇒ Test hooks ⇒ which takes from Mocha Fremwork

Why hooks

⇒ • Hooks are helpful to set condition that you want to run before a set of tests or before each test.
• They are also helpful to clean up condition after a set of tests or after each test.

```
describe ('Hook', () => {
        (run once only)
before all it block. → before (() => {
        // runs once before all tests in the block.
used in → initialise test data / to invoke browser or
        heat the url
    })

        (run once only)
    after (() => {
        // runs once after all tests in the block
used in → delete cache or cookies or delete Testdata
    })

        (run multiple time)
before each it block → beforeEach (() => {
        // runs before each test in the block
    })

        (run mutiple time)
    afterEach (() => {
    // runs ~~befor~~ after each test in the block
    })
```


Be AatmaNirbhar. Use Kaagaz

rahul sheth AngularPractice website | data driven by using fixture method

understand how fixture work in driven data

(45)*

- cypress is providing us a feature to drive the data from external sources
- so, if you place your data in [' fixture'] folder cypress have automatically knowledge on that & it take to all json file which are presenting in fixture folder & interact with them & load this data in your test.

ex → cypress
↳ fixtures
↳ {} example.json

- cypress suggested to have 1 json file in fixture for test data for 1 test case.

- if there is 10 Test Case then we have 10 fixture.json file in fixture folder.

- All test data should be written in before () hook block.

→ cy.fixture ()
↳ This method help to talk with files present in [Fixture] Folder

return promise resolve promise

* cy.fixture ('filename').then ((data) ⇒
example
{
the scope of data is present in only this block
but we need that data in everywhere.
solution scope is in external class. → this.data = data
local variable
global variable
}

Be AatmaNirbhar. Use Kaagaz

{angular practice website}

* validating attribute properties & there behaviour with cypress assertions

- Assertion

1)) cy.get(' ').should('have.value', 'expected')

if want to validate attribute.

2)) cy.get(' ').should('have.attr', 'minilength', '2')

3)) disable button

cy.get(' ').should('be.disabled')

Be AatmaNirbhar. Use Kaagaz

PAGE NO.
DATE:

* Building customized cypress Commands for reusing the code.

→ If we have list and we need to iterate it then we used each() method.

- we added customized cypress command in [cypress > support > cy commands.js] folder

- Command How to declared. (syntax)

cypress.Commands.add ('login', (email, Password) => {

command name

parameter need to pass.

Common Function

})

customized

⇒ our command is become cy.

cy.login ('mayur@gmail', '1111')

cy.log

- Why we used customized command

⇒ if you think any logic may need to execute again & again or think logic is used for multiple test case then you used customized command and used it.

Be AatmaNirbhar. Use Kaagaz

* Parameterized the test Data From Json file by using each command

* in Fixture folder we can store the data in array format from that we apply the For each method to take each component or element.

↳ Fixture
↳ example.Json

```
{
  "user" : "admin"
  "Password": "admin123"
  "Product": [ "banana", "chiky", "santra"]
}
```

* Test Script ⇒

```
this.data.Product.ForEach ((el) ⇒
    cy.selectdata (el)
}
```

customize command

Be AatmaNirbhar. Use Kaagaz

PAGE NO.
DATE:

* Test Debugging & Pause with cypress

1) cy.pause() → if from perticular test case upnext test case get fails then we can pause that test case & after that we can resume from that perticular test case

2) cy.debug() → you can apply that command on event also
like
.click().debug()
it perform same function as like pause(). method

Be AatmaNirbhar. Use Kaagaz

Pom

(POM)

* Implementing Page object Design Patten into the cypress.

* Need of page object Design Patten.

① If in our application there are multiple pages like 1> Home page
2> Shop page
3> checkout page

② So, in POM we collect all element of one page (element of Home page). we we write them in one specific class

ele called as object

③ means if in our application there are 4 pages then we have to create 4 JS file and all object belongs to that perticular page goes into that respective file.

④ because in future if any object element of any page in application get changed & we write all code in single file only then it is very hard to find

* So in integration folder itself we made the pageobject folder & in that we create js file for each page in an application

↳ integration
↳ ~~PageObject~~
↳ ~~Home.js~~
Test Case1.JS.

support
↳ PageObject
↳ Home.JS.

Be AatmaNirbhar. Use Kaagaz

each ele place in one method in class

Home.js

```
class HomePage {

  {
    getEditBox()
    {
return cy.get ('input [name ="name')
    }

    getTwoWayDataBinding()
    {
      return cy.get ('css selector')
    }

  }
export default HomePage
```

used

Js makesure that this class need to available in all other file in your fremwork.

Be AatmaNirbhar. Use Kaagaz

[Normal js file.]

```
import HomePage from "../pageObject/HomePage;

describe (' ', function () {

    it (' ', function () {
let HomeP = new HomePage () ← creating the object

    HomeP.getEditBox().type (this.data.name)
    HomeP.getTwoWayDataBinding().should ('have.value', this.data.na
```

```
    })
})
```


Be AatmaNirbhar. Use Kaagaz

* Modifying existing tests into page object pattern as per cypress standard.

* If for specific element we want more timeout we ~~mean wait~~ used the command as
Cypress.Config ('defaultCommandTimeout', 8000)

Not suggested to used this.

Be AatmaNirbhar. Use Kaagaz

If we want to compare text contain then we can used this assertion

* assertion → expect (undefined.includes("success")).to.be.true

* implementing the sum of product functionality with cypress on ecommerce websites.

⇒ tr td:nth-child(3) strong

if there is price in format ₹. 5000
space.

```
cy.get('.alert').then(function(ele)
{ var sum = 0
  const actualText = element.text()
  var res = actualText.split(" ")

  res = res[1].trim()
  sum = sum + Number(res)
```

NOTE

cypress.
JSON

Be AatmaNirbhar. Use Kaagaz

Javascript

Importance of environmental variables in cypress framework

*→ What are environmental variables in general

→ ① This are the variable which are globaly declared for your frameworK, so that they will have access to each & every test case.

ex. like → URL → you can declared URL globaly & it can be access to each & every test case.

② Per before going Project in Production need to check in different environment like

1) QA → https://RedditQA.com/login

2) UAT → https://RedditmyUAT.com/login

so URL get change in both env.

③ so we maintain globle URL & we can change it according to our environment.

NOTE → in cypress.json file we cant declared any user defined variable which are not in cypress Testrunner > setting > configuration variable

we have to declared it in env mean sub json

cypress.json

```
{
  "defaultCommandTimeout" : 40000
  X "url" : "www.google.com" X Not allow
  "env" : {
      "url" : "www.google.com"
  }
}
```


Be AatmaNirbhar. Use Kaagaz

POM syntax

* Setting env. variable through command line argument

if we want to fetch this set value in our test file.

- cy.visit (cypress.env('url'))

* We can override variable which set in cypress.json by using command promp.

⟹ npx cypress run --spec cypress/integration/examples/Test8.js

Normal command — one specific test file.

npx cypress run --spec cypress/integration/examples/Test8.js --env url = https://google.com --headed

setting env variable runtime.

* if Test case is fail then it takes the screenshot. and for each & every test case there is video get generated.

Be AatmaNirbhar. Use Kaagaz

* ~~Monitoring Test Execution videos & screenshots through cypress dashboards~~

* Exploring cypress Dashboard & its feature for framework development
* 1st way to get all test case report
* cypress Dashbard → cypress dashbard is a service that gives you access to recorded tests

* How to login through Dashboard
1> Test runner
2> setting
3> runs
↳ we can login through 1> github
2> login through google

- Project Id → uniquely identify our Project
- Record key →
with the help of this Dashbard & our Project is connected.

2nd way PTO

Be AatmaNirbhar. Use Kaagaz

Generating mocha Awesome reports for cypress execution

2) Mochawesome

1) npm install --save--dev mochawesome
2) npm install --sav --dev mocha
3) cypress.json →

```
{
    "reporter": "mochawesome"
}
```

4) npx cypress run --reporter mochawesom

Normally if written in regular give / single slash
in written in Packge.json give // double slash

" npx cypress run --spect " file folder path/* "

Be AatmaNirbhar. Use Kaagaz

* Scripting command & how to create

1) in package.json
↳ There is script.

* you can run ~~any~~ any ~~command~~ script by calling command
[npm run testname] (which given in package.json script section)
↑
script

* Why we used script command
→ • because instead of calling big command we can wrap it into one variable & call it.
• Also in real time it is very hard to remember all this command so simply we write command as
[npm run name]
↑ script variable name.

1) npx cypress run → headless
2) npx cypress run --headed → head mode (electron)
3) npx cypress run --browser chrome --headed → chrome browser.

* Running multiple specs file on fly from cypress scripting commands.

Pro 1) way →
we can write the same command & give, for each file.

→ "npx cypress run --spec Test1.js, Test2.js"

2way → Create folder for which Tc we want to run.
→ "npx cypress run -- spe "folder path/*" "

Be AatmaNirbhar. Use Kaagaz

Note

* cypress Provides the feature of retrive means if the test case is fail then cypress automatically run it one more time to confirm is this fail because of cypress issue (inconsistency) or bug.

So that we need to add this command in cypress.json

```
{
  "retries": {
    "runmode": 1
  }
}
```


Be AatmaNirbhar. Use Kaagaz

Jenkins

* Jenkins → download
  ↳ generic java package (download)
* go command prompt → go where Jenkin download
  To start Jenking command is
  java - jar jenkins.war - http Port = 9090
* Type → localhost : 9090 in our browser
  enter → username
  Password → provide Password
* integrating cypress framework into Jenkins CI tool
  → New item
  → Name → cypressAutomation
* With the help of Jenkin we can build fremcourk
  Mean we get report of test case of like
  dashboard Mochawesome or allure report

Be AatmaNirbhar. Use Kaagaz

* What is BDD & cucumber & Benifits of it

1) What is cucumber & what is it need to used in our ~~fre~~ cypress fremeowrk

→ ① in this you can describe your test cse in the Form of behaviour

② in this
our Test can be drive in behaviour driven style
The syntax in BDD which include given, when then called ~~er~~ ~~Gerkein~~ syntax
gherkin [गेरकीन

③ given → what is given
when → when you perfor that action
And → when you perform multiple action you concatinate this with when syntax
Then → what is outcome

④ & then we link this line with our actual code.

* Feature : → Test scenario which include multiple TC

Feature description (optional)

Scenario : → each scenario told one Test cse

Given :
When :
And
Then

here we hide backend & write functionality of
or Fronend mean we encapsulated backend &
show frontend which we make more userfriendly

Be AatmaNirbhar. Use Kaagaz

Scenario: login with orange HRM

Given I Navigate to url → cy.visit ('https://www.orangehrm)

When I Enter username as "Admin" & Password as "admin123"

→ cy.get ("#").type (username)

cy.get ("#").type (Password)

And I tap of login button

→ cy.get ("#").click ()

Then I navigate of dashboard

cy.get ("#").should ('be.visible')

- with the help of this above any one can easily understand what our testcase is exactly doing
- & ultimatly we can tie up our code with given, when, then line

* BDD Framework → So we called this as 'Behaviour driven development'

* Cucumber → cucumber is one of tool which support BDD framework

* Base of ~~our~~ this behaviour we are driven our development. development is nothing but our actual code

1) Describe our behaviour

2) Based on behaviour we drive our development (actual code)

Be AatmaNirbhar. Use Kaagaz

used of → | --save-dev → entry of app get store in package.json file |

* Where we can write this code

→ In cucumber ther is concept called
• StepDefination,
↳ & we write the code there.

Given, When, Then, And ⇒ Keyword

Be AatmaNirbhar. Use Kaagaz

* Install cucumber cypress Preposesses plug
set it in index.js

* Converting cypress Mocha Test to feature file with gherkin syntax.

* Feature (TS) → which explain what exactly feature file doing

↳ it may have multiple test scenario. (i.e. TC)

* some rules or convection need to follow

↳ If we create feature file with one name called (X).feature. the create another folder which name is also (X)

* Cause this file look the feature file with the same name

* [Where we can write the hooks in cucumber BDD.]

→ i.e. before, after, beforeEach AfterEach
it is recommed to write the all hook in seperate ~~folder~~ File.

*. BDD cucumber support the mocha ~~framework~~ related Hooks

* when we are using mocha hook not used arrow function

* We are run feature file in BDD

Be AatmaNirbhar. Use Kaagaz

Nov-29

29 —
30 —
1 —
2 —
3 —
4 —
5 —
6 —

* Data Driven Testing using cucumber Data table feature for Cypress

1) First in BDD we take data from fixture folder
2) Second in BDD we declared the data & take this directly
3) by using scenario outline.

* in cucumber BDD we can reused the code

@ smoke
Scenario: filling the form to shop
Given I open ECommerce page
When I fill the form details
| name | gender |
| bobz | male |
Then validate the form behaviour
And select the shop page.

⇒ JS file [[name, gender], [bobz, male]]

When ('I fill the form details', function (dataTable) {

homepage.getEditBox().type(dataTable.rowTable[1][0])
homepage.getGender().select(dataTable.rowTable[1][1])
})

Be AatmaNirbhar. Use Kaagaz

* Tagging implementation to control Test execution for cucumber scenario

* by using tag in scenario BDD like @ focus.
then we used the tags.

Command
→ npx cypress-tags run -e TAGS = "@smoke" --headed --browser chrome

* Why Tagging used ? in feture file.
→ control our execution within feature file by using tagname over scenario

if want to execute all file in feature then give command as.
* npx cypress run --spec Folderpath* Feature

PTO.

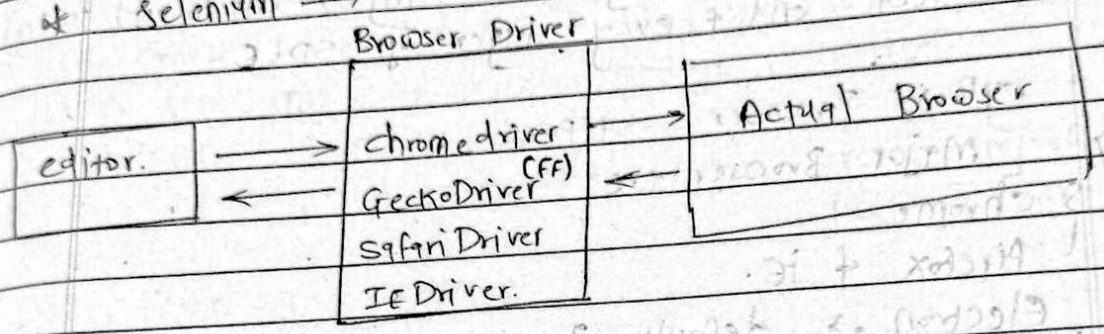
Be AatmaNirbhar. Use Kaagaz

5) view videos of your entire tests execution when run from cypress.

* cypress build on Nodejs & comes package as an npm module
* As it is build Node JS it uses Javascrip for writing tests.

Architecture

* Selenium →



in selenium

1) We write code in editor, wrap the code in Json file. & send to Browser driver (not original Browser) & that intrupt our all code & give command to our original browser & then give responce back to Browser driver & from that on editor.

1) But in cypress Browser itself executing the code.

* Cypress Architecture -

1) Most testing tools (like selenium) operate by running outside of the browser & executing remote command across the network. But cypress engine directly operates inside the browser. in other word. it is the browser that is executing our test code.

Be AatmaNirbhar. Use Kaagaz