Automation step by step

Video 01:

Cypress is a test automation tool

Can test anything that runs on a web browser

Uses javascript

Cypress DOES NOT use Selenium, a lot of web browser automation tool uses selenium but not Cypress

Its open source

Cypress.io is the official site

How to use cypress

* Setup tests
* Write tests
* Run tests
* Debug

Above are the 4 major section while using cypress

We can also do cross browser testing with cypress

Video 02:

* Time travel takes snapshots as your tests run on every command. And if you go to command log you can see the corresponded snapshot.
* Debuggability readable errors and stack traces
* Automatic waiting automatically waits for commands and assertion before moving on
* Consistent results does not use Selenium or webDriver. Fast, consistent and reliable
* Screenshot and videos
* Cross browser testing locally or remotely (CI CD)

Cypress enables your to write all types of testing

* End to end
* Integration test
* Unit test

Video 03: IS nothing important just installation instruction.

Video 04:

* Lets a create a folder name CypressProject and drag it to vscode
* In terminal we type “npm init -y” this will start a node project and create package.json file.
* Then “npm install cypress” this will install the latest version of cypress but if you want to have any older version of cypress then just go to the official site and in the tab whats new, you can have your version and just copy and then type in the terminal “npm install cypress@yourversion”
* To watch the version: npx cypress –v
* To verify the cypress: npx cypress verify
* To start the cypress: npx cypress open

Cypress will be start and then 2 option E2E and component which we have to choose E2E and configure it. Then 4 browser option will be there and we can just start and we can see scaffold and create spec option. We will create our new file. Which will be in e2e folder. Where we’ll write all our test file and test script

Video 06:

Now at the very top of the testing file, just type this

/// <reference *types* = "cypress" />

Means all the code will be autocompletion will be done cypress package or cypress library.

Now lets run a test runner, which is mocha. Mocha is built in install in cypress.

/// <reference *types* = "cypress" />

it("google test", () => {

  cy.visit("https://github.com/jansir17");

});

Here, ‘it’ is mocha function and this is how we create a it block or test block.

Then tittle of the test is ‘google test’ , it’s the name of the test.

Then the function of the it block, what we want to do will be there. This function couild be declared as normal function or arrow. Both ways.

Cy is the gateway of all cypress command. Its an object to call everything.

And in this example we are calling my github id.

Lets npx cypress open now. ☺

Now we can see that 1 run , no error, no run, and time it has taken to run. This is called COMMAND LOG.

And right side is the application we visited.

Now if we change the visit site by any character or give any wrong site that does not exits. Then the moment I save the file after changing it, then test runner will automatically run and will give error.

This because, in our project file structured, cypress.json file has a command name   
{

“watchForFileChanges”: true

}

So if we make it false then test will not run automatically.

Now, lets have access the elements.

it("google test", () => {

  cy.visit("https://google.com");

  cy.get(".gLFyf").type("What is height of me");

});

You can see that now we are using get method of the object cy, and we are locating and class selector from the application selector button we can just click there and have the exact element class name and put it in our vscode and then .type is to type what we want to search in that element as the element is a search box. And then we can enter.

it("google test", () => {

  cy.visit("https://google.com");

  cy.get(".gLFyf").type("What is your name {enter}");

});

Video 07:

How to access elements.

it("google test", () => {

  cy.visit("https://google.com");

  cy.get(".gLFyf").type("Automation Step by Step {enter}");

  cy.get('[href="https://automationstepbystep.com/"] > .LC20lb').click();

});

So after selecting and search box and putting text and enter and then we are targeting any selector to click and go there. Simple

Now, lets step back and remove the 3rd line of getting the second element.and if we just change or give a wrong selector at the very first, then test runner will run and give a timeout after 4 seconds .

We can change that to anything we want .

it("google test", () => {

  cy.visit("https://google.com");

  cy.get(".gLFy", { timeout: 6000 }).type("Automation Step by Step {enter}");

  // cy.get('[href="https://automationstepbystep.com/"] > .LC20lb').click();

});

So we are changing the default timeout. In code dynamically.

But globally for all test we can change the default timeout by chaing in cypress.json file.

{

“defaultCommandTimeout: 5000”

}

There is another way to get into any element

it("google test", () => {

  cy.visit("https://google.com");

  cy.get(".gLFyf", { timeout: 6000 }).type("Automation Step by Step {enter}");

  // cy.get('[href="https://automationstepbystep.com/"] > .LC20lb').click();

  cy.contains("Images").click();

});

So where instead of selecting the selector we are just checking if it contains and then click.

We can also give some timeout in between the test run.

it("google test", () => {

  cy.visit("https://google.com");

  cy.get(".gLFyf", { timeout: 6000 }).type("Automation Step by Step {enter}");

  // cy.get('[href="https://automationstepbystep.com/"] > .LC20lb').click();

  cy.wait(4000);

  cy.contains("Images").click();

});

VIDEO 08: A login test

Now we are gonna have 2 it block

it("google test", () => {

  cy.visit("https://google.com");

  cy.get("div");

  cy.get(".gLFyf", { timeout: 6000 }).type("Automation Step by Step {enter}");

  // cy.get('[href="https://automationstepbystep.com/"] > .LC20lb').click();

  cy.wait(4000);

  cy.contains("Images").click();

});

it("login test", function () {

  cy.visit("https://github.com/jansir17");

});

If we add .only in one of the it block then only that block or test going to run.

Rememeber we can type describe and then put it block in it.

But if we have it block and describe block separated then, it block will run first even we put describe block first.

Remember, to get the descendent element of a selctor we can use find method.

Cy.get (‘.classSomething’).find(#savebutton).click()

VIDEO 10: Cypress Assertions

2 types of assertion

* Implicit
* Explicit

Implicit are in built assertions. We usually use command like should() check for the expected condition without having to use any assert or expect command separately

Explicit are NOT in built. NO reffered. Have to use specific assertion commands like assert() or expect()

If the explicit assertion passes you will see on the log and if not passed then wont see on the log.

So for implicit assertion we have

* Should()
* And()

For explicit assertion we have

* Expect()
* Assert()

/// <reference *types*= "cypress" />

it("Learning Assertions", () => {

  cy.visit("https://example.cypress.io/");

  cy.get(

    ":nth-child(4) > .row > .col-xs-12 > .home-list > :nth-child(1) > ul > :nth-child(1) > a"

  ).click();

  cy.get("#query-btn").should("contain", "Button");

});

Pretty self explainatory. That we are going to a site where are getting into and element and over there is a button which we are targeting and having implicit assertion.

Remember if we type in different case, it wont work. For both, contain and button. On the other hand if we write down properly but not completely, for ex: if we type Butt instead of Button. It will be alright. But remember again, its very case sensitive.

Lets have the idea about should-have

  cy.get("#query-btn")

    // .should("contain", "Butt")

    .should("have.class", "btn-primary");

});

What we are doing that querying that there should have a something like that. Remember over here there a class of 3 in a single element. we can check for any of those. But as previous style we can not have half of the name of the class. Either exact class or not. But it could any of the 3.

Now lets have idea about Should-be.

cy.get("#query-btn")

    .should("contain", "Butt")

    .should("have.class", "btn-primary")

    .should("be.visible")

    .should("be.enabled")

    .should("be.disabled");

});

Now lets have shoule-equal.

cy.get("#query-btn").invoke("attr", "id").should("equal", "query-btn");

now lets have and method or chaining method.

cy.get("#query-btn")

    .should("contain", "Button")

    .and("have.class", "btn-primary");

Nothing special about and. Just check this and that. That’s it.

EXPLICIT ASSERTION.

Expect method.

it("explicit CHeck", () => {

  cy.visit("https://example.cypress.io/");

  expect(true).to.be.true;

});

Now remember that if we give a value true than log wont show you anything. But if we give value false then log will show that

expect(true).to.be.false;

**assert**expected **true** to be false

Remember there are so many other method of

To.not.equall

To.be.a(‘string’)

To.be.true

To.be.false

To.be.null

Now Assert method

//assert

  assert.equal(4, 5, "numbers ain't equal");

});

That’s an error message if we have

VIDEO 11:

PAGE OBJECT MODEL

It’s a design principle,

Keep the object and method separate from test script.

Efficient reusability

Easier, efficient and faster changes and maintenance.

Its just a moduler way ot calling site to visit. Putting username , pass , loginbutton.

**How to upload file. (The Testing Academy)**

Just install firstly npm command of file upload. Google it and install it.

Then copy something which you want to upload, into the fixture folder which is in the file structure of your project folder. If the file is not there then cypress will throw an error. so its always going to look the file which we want to upload, in the fixture folder.

Now lets write the test script

/// <reference *types* = "cypress" />

describe("file uploading test suit", () => {

  it("uploading Moon", () => {

    const filePath = "Moon.jpg";

    cy.visit("https://the-internet.herokuapp.com/upload");

    cy.get("#file-upload").attachFile(filePath);

    cy.get("#file-submit").click();

    cy.get("#uploaded-files").contains("Moon.jpg");

  });

});

As we can see that we are saving file that we want to upload a filepath variable. And then attaching it.

Then simple stuff.

**How to download file**

/// <reference *types* = "cypress" />

describe("File Download", () => {

  it("Donwloading", () => {

    cy.downloadFile(

      "https://upload.wikimedia.org/wikipedia/en/a/a9/Example.jpg",

      "mydownloads",

      "example.jpg"

    );

  });

});

But had faced a lot of error regarding cy.task() has not been registered.

How to test REST API with CYPRESS.

/// <reference *types*= "cypress" />

describe("REST Api Testing", () => {

  it("GET", () => {

    cy.request({

      method: "GET",

      url: "https://httpbin.org/get",

    }).then((*response*) => {

      expect(*response*.body).have.property("url");

    });

  });

  it("POST", () => {

    cy.request({

      method: "POST",

      url: "https://httpbin.org/post",

      body: {

        name: "Jansir",

        age: 27,

      },

      headers: {

        "content-type": "application/json",

      },

    }).then((*response*) => {

      expect(*response*.body).have.property("json");

      expect(*response*.body.json).to.deep.equal({

        name: "Jansir",

        age: 27,

      });

    });

  });

});

Nothing to explain everything is selfexplaining.

Some Advance Command

Cy.visit(‘something’, {timeout: 3000})

Cy.visti(‘something’, {retryOnNetworkFailure: true});

**Accessing table**

/// <reference *types* ="cypress" />

describe("Advance Functionality", () => {

  it("Checking Table Data", () => {

    cy.visit("https://the-internet.herokuapp.com/tables");

    cy.get("#table1")

      .find("tbody>tr")

      .first()

      .find("td")

      .find("a")

      .first()

      .click();

    cy.url("match", "#edit");

  });

});

Printing a list of a table

it("printing Element", () => {

    cy.get(ul > li)

      .each((*$el*, *$index*, *$lis*) => {

        cy.log("Li element Text is" + *$el*.text());

      })

      .then(() => {

        expect($lis).to.have.length(3);

      });

    cy.window().its("innerWidth").should("eq", 1000);

  });

React testing project

/// <reference *types*= "cypress" />

describe("React Testing", () => {

  it("Sign UP", () => {

    cy.visit("https://react-redux.realworld.io/");

    // cy.viewport("mackbook-15");

    cy.get("a").contains("Sign up").click();

    cy.get('input[placeholder="Username"]').type(USERID());

    cy.get('input[placeholder="Email"]').type(USERID() + "@gmail.com");

    cy.get('input[placeholder="Password"]').type(USERID() + "776625");

    cy.get("button").contains("Sign in").click();

    cy.get("a").contains("Home").should("have.text", "Home");

    cy.wait(3000);

  });

  //fucntion to create random username/gmail/password

  function USERID() {

    var text = "";

    var possible =

      "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";

    for (var i = 0; i < 5; i++)

      text += possible.charAt(Math.floor(Math.random() \* possible.length));

    return text;

  }

  it("test2", () => {

    cy.get(".ion-compose").click();

    cy.get('input[placeholder="Article Title"]').type("TestDemo#1234");

    cy.get("#main> div fieldset:nth-child(2)>input").type(USERID());

    cy.get("#main> div fieldset:nth-child(3)>textarea").type("daasdadad");

    cy.get("button").contains("Publish Article").click();

    cy.get(".container h1").should("have.text", "TestDemo#1234");

  });

});

Its all self explanatory

HTML report generator

npm i mochawesome-report-generator

npm i mochawesome-merge

npm install mochawesome

npx cypress run --reporter mochawesome