BootCamp Cypress en español- Un Framework desde cero (2024)

Preguntas y respuestas

cypress run - -headed

4. Como configurar el entorno de trabajo

Instalar VSCode
Instalar NodeJS
GIT

5. Selectores

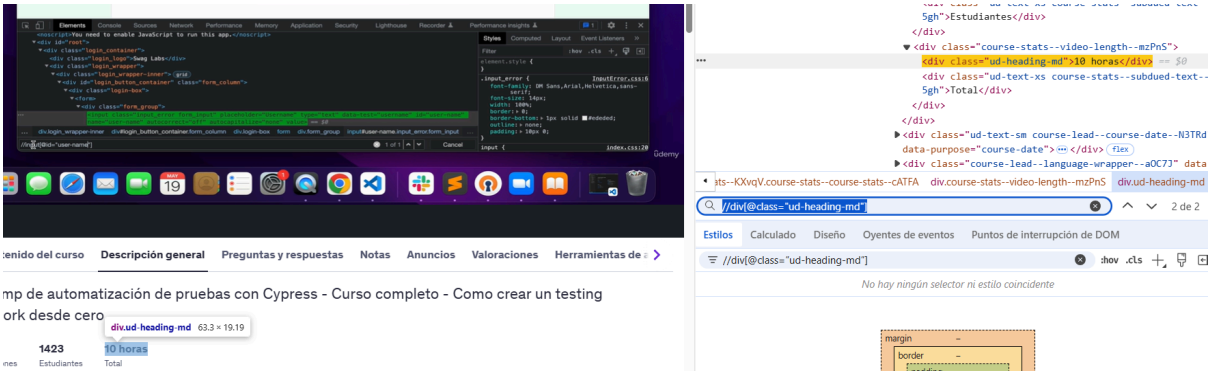
Lenguaje estandarizado, existen 2 formatos para escribir XPATH y CSS.
“saucedemo.com”

Formato XPATH

Como escribir o buscar selector en formato XPATH:

¿Cómo encontrar elementos?

//nombreEtiqueta[@nombreAtributo=”valor-atributo”]
//div[@class=”ud-heading-md”]



¿Cómo encontrar elementos por su inner text?

A través del método text()

//span[text()="Descripción general"]

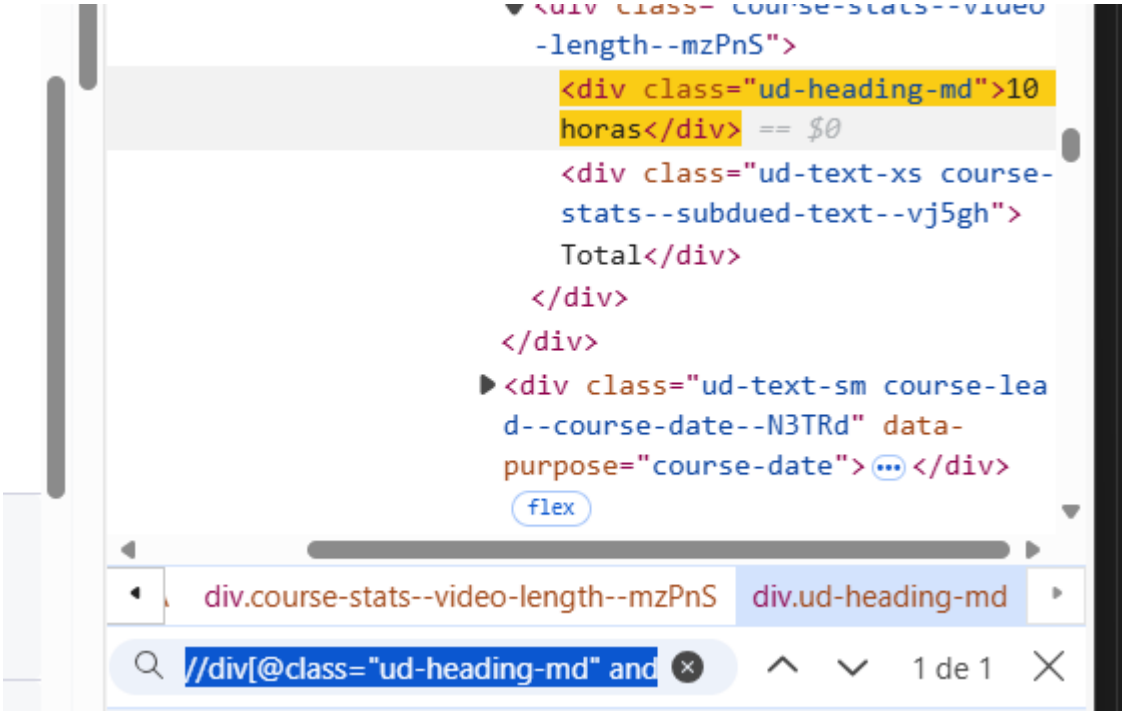
También podemos hacerlo con el prefijo ‘.’

//span[.="Descripción general"]

Selectores XPATH “or” y “and”

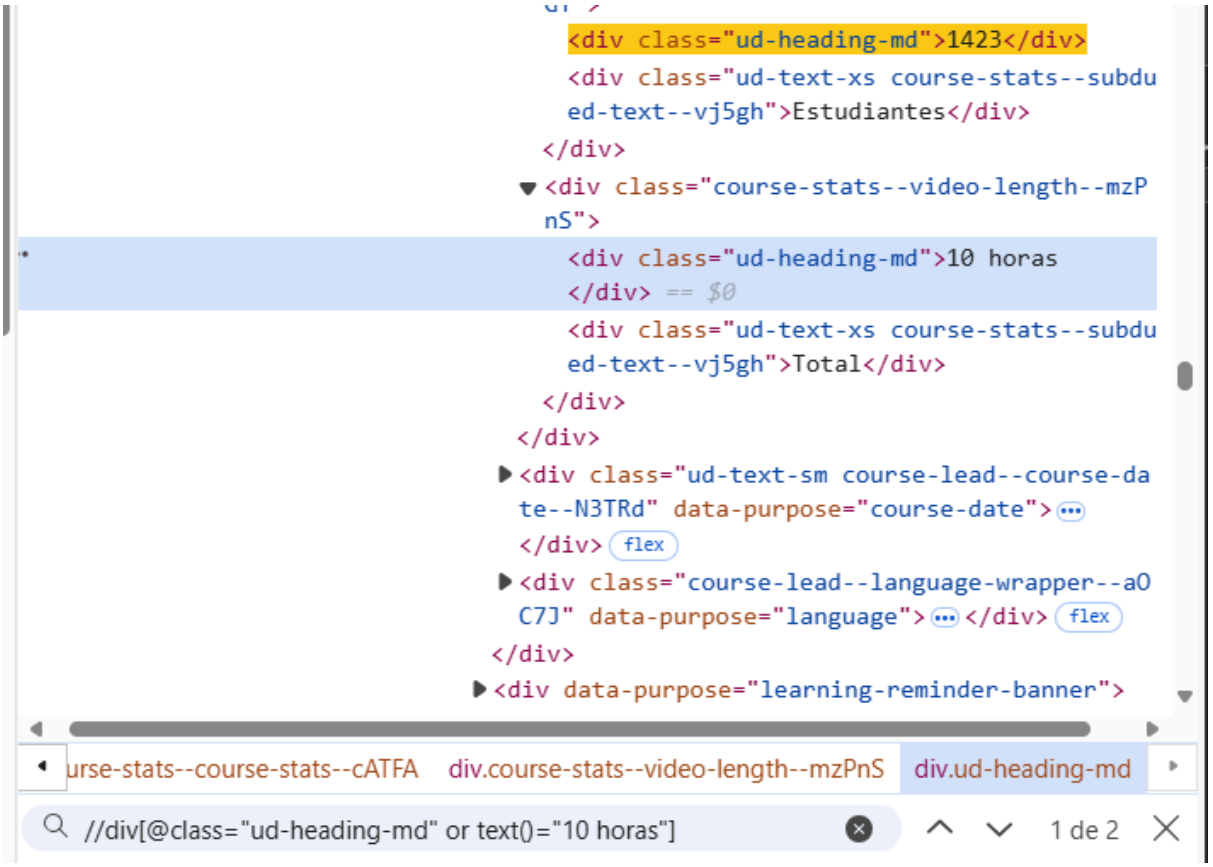
Usando el operador and:

//div[@class="ud-heading-md" and text()="10 horas"]



Usando el operador or:

//div[@class="ud-heading-md" or text()="10 horas"]



Cómo encontrar los elementos XPATH por su relación con otros elementos

//div[@class=”app--curriculum-item--vBwDu”]**/following-sibling::div**
//button[@class="ud-btn ud-btn-large ud-btn-ghost ud-heading-md ud-nav-button tabs-module--nav-button--DtB8V"]**/preceding-sibling::**

Formato CSS

Es similar, sin los dos slas ‘//’ y sin el ‘@’

div[class="ud-heading-md"]

Encontrar elementos por id

div[**id**="ud-heading-md"]
O también **poniendo #**

Encontrar elementos por class .

div[class="ud-heading-md"]
O también **poniendo .**

Relacionado con otro elemento

#tabs--3-content-2 div
Seleccionara todos los **div dentro del elemento id (#tabs--3-content-2)**

6. Fundamentos de Cypress

Creamos Repositorio

- Inicializamos el NPM sobre el proyecto:
  npm init -y
- Instalamos Cypress: (12.12.0)
  npm install cypress
- Abrir Cypress para su configuración
  npx cypress open
- Ejecutamos cypress
  npx cypress run

7. Selectors Cypress

Get
Contains
Find
first
Eq
Closest

8. Assertion cypress

```
describe('test assertions', ()=>{
    it('test case assertions cypress', ()=>{
        cy.visit('https://www.saucedemo.com/')
        cy.get('#user-name').should('exist') // comprobamos que el elemento con el nombre title debería existir
        cy.get('.login_logo').should('have.text', 'Swag Labs') // comprobamos debería tener el texto 'Swag Labs'
        cy.get('#user-name').type('hola') // insertamos valor para prueba should('have.value')
        cy.get('#user-name').should('have.value','hola') // comprobamos que tenga el valor 'hola'
        cy.get('.login_logo').should('contain.text','Labs') // comprobamos que contengala palabra labs
        cy.get('#user-name').type('navidad') // insertamos valor para prueba should('contain.value')
        cy.get('#user-name').should('contain.value','dad') // comprobamos que el valro contega la cadena de caracteres 'dad'
    })
})

describe('test assertions be.visible', ()=>{
    it('test case assertions be.visible cypress', ()=>{
        cy.visit('https://www.saucedemo.com/')
        // comprobamos que elemento existe en el DOM y no esta oculto. Primero hacemos login:
        cy.get('#user-name').type('standard_user') //buscamos el elemento con id: user-name y ponemos el valor indicado
        cy.get('#password').type('secret_sauce') //buscamos el elemento con id: password y ponemos el valor indicado
        cy.get('#login-button').click()
        cy.get('#react-burger-menu-btn').click()//desplegamos el subMenu para que este visible
        cy.get('.bm-menu-wrap').should('be.visible') // comprobamos que elemento existe en el DOM y no esta oculto
    })
})
```

```
describe('test assertions be.checked', ()=>{
    it('test case assertions be.checked cypress', ()=>{
        cy.visit('https://the-internet.herokuapp.com/checkboxes')
        cy.get('input[type="checkbox"]').eq(1).should('be.checked') // con eq
seleccionamos el checkbox del indice seleccionado

    })
})

describe('test assertions not...', ()=>{        //el not nos permite comprobar la
negación de cualquier assertion anterior
    it('test case assertions not... cypress', ()=>{
        cy.visit('https://the-internet.herokuapp.com/checkboxes')
        cy.get('input[type="checkbox"]').eq(0).should('not.be.checked') // con eq
seleccionamos el checkbox del indice seleccionado

    })
})
```

9. Click Type Check -Uncheck Wait-timeout

```
// método CLICK
describe('Login', ()=>{
    it('Login con credenciales de standard user', ()=>{
        cy.visit('https://www.saucedemo.com/')
         cy.get('#login-button').click({force: true}) //pasamos la propiedad force
    })
})


// método Type, permite editar información en un input y también permite agregar que se
pulse un tecla
describe('Login', ()=>{
    it('Login con credenciales de standard user', ()=>{
        cy.visit('https://www.saucedemo.com/')
         cy.get('#user-name').type('hola')
```

```
        cy.get('#user-name').type('{enter}')   //pulsamos la tecla enter
        cy.wait(10000)
    })
})


// método Check - Uncheck, para seleccionar radiobutton o checkboxes
describe('test assertions not...', ()=>{        //el not nos permite comprobar la
negación de cualquier assertion anterior
    it('test case assertions not... cypress', ()=>{
        cy.visit('https://the-internet.herokuapp.com/checkboxes')
        cy.get('input[type="checkbox"]').eq(0).check() // con eq seleccionamos el
checkbox del indice seleccionado
        cy.wait(2000)
        cy.get('input[type="checkbox"]').eq(0).uncheck() // con eq deselecinamos el
checkbox del indice seleccionado
        cy.wait(2000)
    })
})


// Wait, es recomendable no usarlo

const { timeout } = require("rxjs")

describe('test no wait', ()=>{
    it('test wait', ()=>{
        cy.visit('https://accounts.shopify.com/')
        cy.get('img[alt="Log in to Shopify"]', {timeout: 10000}).click() // intentara
encontrar el elemento imagén y esperará hast 10 segundos a ver si lo encuentra. si llega
a 10 y no lo encontró FALLARA
    })
})
```

11. Custom commands

Son métodos que creamos, pueden obtener o no parametros y devolver algo.

Caso prático el Login, crearemos un CustomCommands que contenga el Login par que pueda ser usado en diferentes casos y no duplicar el código.
Se crean dentro del directorio de nuestro proyecto **CYPRESS / SUPPORT / commands.js**

```
//(nombreMetodo, callback(peude recibir valores) )
Cypress.Commands.add('login', (username, password) =>{
    cy.visit('https://www.saucedemo.com/')
        cy.get('#user-name').type(username)
        cy.get('#password').type(password)
        cy.get('#login-button').click()
})
```

En el archivo de nuestro test, usaríamos nuestro custom commands de la siguiente manera:

```
//Accedemos a nuestro custom commadns de login
describe('test custom commands', ()=>{
    it('test custom commands', ()=>{
        cy.login('standard_user','secret_sauce')
        cy.wait(5000)
    })
})
```

12. Logs

Muestra ejemplo claro de como organizar los test de pruebas, usando métodos JS, para una mejor organización de los mismos.
Los métodos creados los encontramos en Cypress/Support/ logger.js

```
export class Logger{
    static pasoNumero(numero){
        const mensage = `Step # ${numero}`;
        cy.log(`**${mensage}**`);  // Se loguea en CYPREESS
        cy.allure().step(mensage);  // SE loguea en el REgistro de
Pruebas
    }

    static paso(description){
        const mensage = `Paso - ${description}`;
        cy.log(`**${mensage}**`);
        cy.allure().step(mensage);
    }

    static verificacion(description){
        const mensage = `Verificacion - ${description}`;
        cy.log(`**${mensage}**`);
        cy.allure().step(mensage);
    }
}
```

13. JS : clases, métodos estáticos y getters

Las **clases** son contenedores para almacenar código, por lo general incluye propiedades(valores irregulares) y métodos(acciones).
Clases
    Métodos
    Propiedades
class LoginPage
    title- propiedad
    login - metodo

- ¿Cómo guardar el título de la página?
    const tituloPagina = LoginPage.title
- ¿Cómo acceder a un método de una clase ?
    LoginPage.login(username, password)

Los **métodos static** permite que el resto del proyecto pueda acceder a dicho método sin tener que instanciar la clase, donde se encuentra este método. La llamada quedaría de la siguiente manera: `const titulo = LoginPage.obtenerTituloPagina() ;`

Los **Getters**, se podría decir que son métodos pero se utilizan como una propiedad, es decir sin ().

14.Proyecto Final _ Demoblaze

Demoblaze casos test, guia de proyector
    Casos de pruebas manuales.
    Casos de pruebas reflejen lo que el manual dice
    Testing en paralelo
    Implementar reporte de pruebas usando ALLURE Reports
    Usaremos CI/CD AWS

https://github.com/danigo89/Cypress_basicos.git

Configuramos username de git

Configuramos username de git y mail de git, para saber quien subio cambios:
    git config - -global user .name “Mi nombre”
    git config - -global user .email “nombre@mail.com”

Iniciamos proyecto:

1. Creamos archivo configuración del proyecto “package.json”
    a. npm init -y
2. Instalamos Cypress en nuestro proyecto:
    a. npm install cypress - -save-dev
3. Si no queremos compartir el node_modules, no pasa nada, cuando se descargue el proyecto haciendo “npm install” se descargan los archivos necesarios
    a. Si queremos **obviar la subida** de estos archivos en el control de versiones, creamos .gitignore e introducimos “**node_modules/***”
4. *Abrimos y Configuramos CYPRESS*
    a. Abrimos: npx cypress open
    b. Configuramos:
        i. E2E Testing
        ii. Todo seleccionado
        iii. Elegimos navegador
        iv. Creamos new spec, por defecto esta OK
5. *Agregar y configurar **ALLURE report** (reporte de pruebas)*
    a. Agregar: npm i @shelex/cypress-allure-plugin
    b. Configurar en cypress.config.js:
        i. const allureWriter = require('@shelex/cypress-allure-plugin/writer');:
        ii. allureWriter(on, config);
        iii. return config;
    c. Importar en e2e/support/ e2e.js:
        i. import '@shelex/cypress-allure-plugin';
    d. Creamos el directorio ejecutando cypress con variable de entorno(env) allure y su valor a true:
        i. **npx cypress run --headed --env allure=true**
        ii. Necesario **cambiar a la versión de Cypress** 13.1.0
            1. npm install cypress@13.1.0
        iii. o instalar otro paquete:
            1. npm i @mmisty/cypress-allure-adapter
6. *Para transformar los resultados en un reporte, usaremos ALLURE COMMAND LINE*
    a. npm i allure-commandline
    b. npm i allure-commandline@2.22.1 ??versión necesaria??
    c. Script que genera reporte de pruebas, agregamos a package.json:
        i.
```
"generar-reporte":"allure generate allure-results -o allure-report --clean"
```
            1. generar-reporte, nombre script
            2. allure generate, pedimos que genero reporte
            3. allure-results, carpeta donde estan los datos
            4. -o, salida/output
            5. allure-report, la carpeta de salida, donde guardar el reporte
            6. –clean limpia al final
        ii. Ejecutamos el nuevo script:
            1. **npx run generar-reporte ????????????**
                a. hará que cree el nuevo directorio allure-report

iii. Instalamos rimraf:

1. npm install rimraf --save-dev

7. *Problema con JAVa al ejecutar: Variable JAVA_HOME*
   a. allure open
   b. npm run test
   c. npm run generar-reporte
   d. allure generate allure-results -o allure-report --clean

```
dgomez@MERROW MINGW64 /c/Proyecto-Cypress/curso_cypress (main)
$ allure generate allure-results -o allure-report --clean

ERROR: JAVA_HOME is not set and no 'java' command could be found in your PATH

Please set the JAVA_HOME variable in your environment to match the
location of your Java installation.
```

24.Ejecutar prueba en especifico

Excluir pruebas

Varios test en un archivo **ONLY,** ejecutar un solo caso de pruebas de un archivo de pruebas. Pondremos el only desspués del it, pero ejecutara el resto de archivos

it.**only**(“...

Cuando tenemos varios archivos **SPEC**
Indicaremos con el spec el archivo que queremos ejecutar. En nuestro caso lo definimos en el package.json

cypress run - -headed - -env allure=true - **-spec \”cypress/e2e/tests/authentication.cy.js\”**

Eludir archivo o test usaremos la **X**
Agregamos una **X** delante → xdescribe(... o xit(...

25.Ejecución pruebas en paralelo

Instalamos paquete cypress-parallel

npm i cypress-parallel - -save-dev
Creamos Script correspondiente en pacjkage.json

-s → script que vamos a ejecutar
-t → threaders, número de cabezas o casos que queremos ejecutar a la vez
-d → directory o folder dónde estan nuestra pruebas
-a →parámetros Cypress, si quisiéramos pasar algún parámetro Cypress

```
"cy:parallel":"cypress-parallel -s test -t 2 -d ./cypress/e2e/tests"
```

26. Integración continua

Archivo buildspec.yaml

Archivo con diferentes scripts que se ejecutara desde el servidor.

1. Cramos cuenta AWS,
2. Tipo de proyecto **CodeBuild/ BuildProject**
3. Indicamos plataforma donde esta GITHUB