# Cypress

FWK de automatización E2E

Únicamente se puede trabajar con JavaScript y TyperScript

## **Ventajas y Desventajas**

Cypress inyecta directamente el código JavaScript en los navegadores y eso lo hace muy útil. El set up es muy fácil.

### **Cypress vs Selenium**

Ventajas sobre selenium:

* Setting inicial mucho más sencillo. Con selenium hay que configurar jdk, o Python, una building tool, hay que bajar un web driver… Cuando el browser se actualiza genera problemas…
* No uso de web driver
* SI cambias algo del código, el browser se actualiza automáticamente con los cambios modificados
* EN cada paso ves que hay en la página, como un snapchot por cada paso

Desventajas:

* Si abre tabs distintos u otro dominio mediante el testeo no puede
* Iterar datos es complejo
* Con esperas asíncronas
* Hay versiones nuevas cada muy poco tiempo (Tu versión puede quedarse obsoleta)

## **Instalación y Setup**

Package Manager -> Chocolatey.org

Instalar node(Viene con npm) > npm install cypress –save-dev

Abrir cypress: npx cypress open

## **Runner de Cypress y el CLI**

**describe:** Nos da el nombre del conjunto de test

**it:** Cada caso de prueba

**Elegir navegador mediante código**: npm run cypress run -b “navegador”

### **Estructura proyecto cypress**

* **Downloads**: Se guardan descargas de vínculos (cypress no puede abrirlos, se descargan y se evaluan en la carpeta)
* **Fixtures**: Donde se guardan información a usar en los test (usernames, passwords…)
* **E2e**: Donde se definen los tests
* **Plugins**: Definir que plugins vamos a usar,
* **Screenshots**: Evidencias de cypress cuando algo falla
* **Support:** 
  + **Commands.js:** Añadir comandos personalizados
  + **Components.**js
  + **E2e.**js
* **Videos**: Videos de las ejecuciones

## **Automatizando con Cypress**

### **Selectores**

Selectores por Css por defecto únicamente en cypress sin plugins: Selector que todo elemento web tiene que puede ser un id, un atributo, la clase.

Seleccionar selector: Pulsar open selector “Mirilla” y al pasar por elementos se muestra el selector. Este se puede copiar.

Get: Devuelve elemento según el css selector. Normalmente si el elemento que se quiere seleccionar tiene un id, lo coge por defecto.

Within: Para encadenar get. Dentro de una selección mayor, búscame esto.

Get vs Find: Get te busca desde el root, y find desde el contexto en el que tya se está.

### **Plugin XPath**

https://www.npmjs.com/package/cypress-xpath

Instalación:

* npm install -D cypress-xpath
* Añadir carpeta supports del proyecto: require('cypress-xpath');

Ya el objeto cy, debe tener el método cy.xpath()

* Se deshabilita el open selector, se busca elemento, botón derecho inspeccionar y copiar xpath.

### **Encontrar elemento por su texto (No xpath ni css)**

1. Pinchar el open selector “mirilla”
2. Seleccionar opción **cy.contains**
3. Escribir el texto a buscar

Usar cuando hay un solo elemento con dicho nombre.

### **Filtros**

Se añaden al selector.

Implementación -> Selector : filtro

Tipos: enable, disable, checked, empty, visible,

### **Assertions**

Tipos de test más comunes:

* Validar el lenght -> .should(‘have,length’, integer)
* Verificar clase de un web element -> should(‘have.class’, nombre\_clase)
* Comprobación texto -> (have.text’, nombre texto)
* Comprobación valor -> (have.value, valor)
* Verificar que algo está en la pantalla -> should(‘be.visible)
* Verificar que el elemento existe aunque no se muestre -> should(‘exist’)
* Comprobar el estado de elemento -> should(‘be.checked’)

### **Debugging**

cy.log(‘string’) -> Deja comentarios en el panel de la ejecución de cypress

.pause() -> Pausar la ejecución del runner y nos permite correr hasta la próxima pausa

.debug() -> No pausa la ejecución. Nos muestra en la consola el momento determinado donde se ponga el debug.

“Funcion Viajar en el tiempo”: Si si se hace un hover sobre alguna opción del panel de ejecución, se muestra que se estuvo haciendo y el estado.

## **Comandos de Cypress**

* Get -> Devuelve un elemento get asociado al selector. Hay que anexarle una función. Siempre pide un locator o alias.
* Find -> Sirve para encontrar un elemento dependiente de otro. Encontrar por ejemplo un elemento dentro de un formulario.Necesita que alguien le diga donde buscar, un padre. Find no funciona con alias, solo locators.
* Contains() -> Encuentra elemento por su texto
* Wait -> Tiempo de espera (No se aconseja)
* Invoke() -> Invocar funciones para que se ejecuten sobre los elementos
* First() -> Contenido dinamico y se quiere elegir un elemento en concreto. Valida el primer elemento. Last() el último. Eq(index) index se corresponde al elemento que se quiere seleccionar.
* Parent() -> Devuelve el elemento padre del elemento que se le pasa previamente
* Type() -> Escribe lo que se pasa por parámetro en campos de texto
* Check()/uncheck() -> Para marcar o desmarcar un check
* Select() -> Seleccionar de un desplegable, se le puede pasar nombre de la opción o posición del elemento.
* Clear() -> Limpiar campo de texto
* Rightclick() -> Botón derecho
* cy.on("window:alert",(alert)=>{ expect(alert).to.equal("You selected a context menu")

## Sesiones y Cookies

### Jquery (No muy importante)

Jquery: Liibreria de JavaScript, que nos permita viajar a través del dom, manipularlo…

Jquery es síncrono y el get de cypress es asíncrono.

Cy.$(‘Css\_selector’)

Si no encuentra elementos en el dom hay que especificar que pasa si no se encuentra.

$ Sirve para más cosas que para el dom

$$ solo para el dom

### Sesiones

Cy.ssesion(‘nombre\_sesion’, funcion) ->. Guarda la sesión par autilizarlo en otros test y no tener que repetir los comandos de login.

### Cookies

Cy.getCookies() -> Devuelve todas las cookies

Cy.getCookie(‘nombrecookie’) -> Obtiene la cookie

Cy.clearCookies() -> Borra las cookies

Cy.setCookie(‘nombre\_cookie’,’valor\_cookie’) -> Crea cookie

## API requesty responses

### Usando cy.request

## Fixtures y Custom Commands

Fixtures: Generar datos para llamarlos en los testCases.

Cy.fixtures(‘nombreJson’)

Los comandos se pueden crear en commands.js y después llamarlos en los testCases.

## Dashboards

Mochaweome para generas reportes

Lanzar pruebas: npm run cy:run:chrome -- --reporter mochawesome --spec "cypress/e2e/Pruebas/APIS.cy.js"

Se genera informe en mochawesome.html

Botón derecho > run to cursor -> Se abre reporte en el navegador

## TEST DE ACCESIBILIDAD

Dos dependencias:

Axe-core -> npm install axe-core –save-dev

Cypress-axe -> npm install cypress-axe –save-dev