Taller: Implementación de un esquema de pruebas con Cypress

Julian F. Latorre

11 de septiembre de 2024

Índice

1. Introducción

Este taller está diseñado para guiar a los estudiantes en la implementación de un esquema de pruebas utilizando Cypress para aplicaciones fullstack. Durante las próximas dos horas, exploraremos los conceptos fundamentales de Cypress y cómo aplicarlos eficazmente en un entorno de desarrollo fullstack.

2. Objetivos

Al finalizar este taller, los estudiantes serán capaces de:

- Comprender la importancia de las pruebas en el desarrollo fullstack
- Configurar Cypress en un proyecto existente
- Escribir pruebas básicas y avanzadas con Cypress
- Implementar buenas prácticas en la escritura de pruebas
- Integrar pruebas de Cypress en un flujo de trabajo de CI/CD

3. Requisitos previos

- Conocimientos básicos de JavaScript
- Familiaridad con el desarrollo fullstack (frontend y backend)
- Node.js y npm instalados en el sistema
- Un editor de código (como VS Code)
- Un proyecto fullstack existente (preferiblemente con React en el frontend y Node.js en el backend)

4. Parte 1: Introducción a Cypress (30 minutos)

4.1. ¿Qué es Cypress?

Cypress es una herramienta de pruebas de próxima generación construida para la web moderna. Aborda los puntos débiles que los desarrolladores y los ingenieros de control de calidad enfrentan al probar aplicaciones modernas.

4.2. Ventajas de Cypress

- Configuración sencilla
- Debugging en tiempo real
- Esperas automáticas
- Captura de instantáneas y videos
- Soporte para pruebas de UI y API

4.3. Actividad de reflexión

En grupos, discutan:

- ¿Qué desafíos han enfrentado al probar sus aplicaciones fullstack?
- ¿Cómo creen que Cypress podría abordar estos desafíos?
- ¿Qué tipos de pruebas consideran más importantes para sus proyectos actuales?

5. Parte 2: Configuración de Cypress (20 minutos)

5.1. Instalación

En tu proyecto, ejecuta:

```
npm install cypress --save-dev
```

5.2. Configuración inicial

Ejecuta Cypress por primera vez para generar la estructura de directorios:

```
npx cypress open
```

5.3. Estructura del proyecto

Explora la estructura de directorios generada por Cypress:

- `cypress/e2e`: Aquí irán tus pruebas
- `cypress/fixtures`: Para datos de prueba
- `cypress/support`: Para comandos personalizados

5.4. Actividad práctica

Configura Cypress en tu proyecto y familiarízate con la estructura de directorios.

6. Parte 3: Escribiendo tu primera prueba (30 minutos)

6.1. Anatomía de una prueba Cypress

```
describe('Mi primera prueba', () => {
  it('Visita la página principal', () => {
    cy.visit('http://localhost:3000')
    cy.contains('Bienvenido').should('be.visible')
  })
})
```

6.2. Selectores en Cypress

Cypress utiliza selectores similares a jQuery. Algunos ejemplos:

- `cy.get('.mi-clase')`
- `cy.get('#mi-id')`
- `cy.get('[data-testid=mi-elemento]')`

6.3. Actividad práctica: Escribiendo tu primera prueba

Sigue estos pasos para crear tu primera prueba en Cypress:

1. Abre tu editor de código y navega hasta la carpeta de tu proyecto.
2. Crea un nuevo archivo en la carpeta `cypress/e2e` y nómbralo `home_page_spec.cy.js`.
3. Abre el archivo y comienza con la estructura básica de una prueba:

```
describe('Página de inicio', () => {
  beforeEach(() => {
    cy.visit('http://localhost:3000') // Ajusta la URL seg
ún tu aplicación
  })

  it('Debería mostrar el título de la página', () => {
    // Aquí irá tu primera aserción
  })
})
```

4. Ahora, añade una aserción para verificar que el título de la página esté presente:

```
cy.get('h1').should('contain', 'Bienvenido a Mi Aplicación
')
```

5. Agrega otra prueba para interactuar con un elemento de la página. Por ejemplo, si tienes un botón de "Leer más":

```
it('Debería poder hacer clic en el botón "Leer más"', ()
=> {
  cy.get('button').contains('Leer más').click()
  cy.url().should('include', '/about') // Asume que el bot
ón lleva a la página "About"
})
```

6. Guarda el archivo.
7. Abre Cypress con el comando `npx cypress open` en tu terminal.
8. En la interfaz de Cypress, selecciona ”E2E Testing” y luego elige tu navegador preferido.
9. Deberías ver tu nuevo archivo de prueba listado. Haz clic en él para ejecutar las pruebas.
10. Observa cómo Cypress ejecuta tus pruebas y muestra los resultados.

Desafío adicional: Si terminas pronto, intenta añadir una prueba más compleja:

- Encuentra un formulario en tu aplicación (por ejemplo, un formulario de contacto o de búsqueda).
- Escribe una prueba que complete el formulario y lo envíe.
- Verifica que la aplicación responda correctamente (por ejemplo, mostrando un mensaje de éxito o navegando a una nueva página).

7. Parte 4: Pruebas avanzadas (30 minutos)

7.1. Pruebas de API

Cypress puede realizar solicitudes de red directamente:

```
cy.request('POST', '/api/usuarios', { nombre: 'Juan' })
  .then((response) => {
    expect(response.status).to.eq(200)
    expect(response.body).to.have.property('id')
  })
```

7.2. Fixtures: Datos de prueba estáticos

Los fixtures en Cypress son archivos de datos estáticos que se utilizan para proporcionar datos consistentes a tus pruebas. Estos son especialmente útiles para simular respuestas de API, llenar formularios con datos predefinidos, o cualquier situación donde necesites datos de prueba consistentes.

7.2.1. ¿Qué son los fixtures?

- Archivos JSON, CSV, o de texto que contienen datos de prueba
- Se almacenan en el directorio `cypress/fixtures`
- Permiten separar los datos de prueba de la lógica de la prueba
- Facilitan la reutilización de datos en múltiples pruebas

7.2.2. Creando y usando un fixture

Primero, crea un archivo JSON en `cypress/fixtures/usuario.json`:

```
{
  "nombre": "Juan Pérez",
  "email": "juan@ejemplo.com",
  "password": "contraseña123"
}
```

Luego, usa el fixture en tu prueba:

```
describe('Formulario de registro', () => {
  it('debería llenar el formulario con datos del fixture', () => {
    cy.visit('/registro')

    cy.fixture('usuario').then((usuario) => {
      cy.get('#nombre').type(usuario.nombre)
      cy.get('#email').type(usuario.email)
      cy.get('#password').type(usuario.password)
    })

    cy.get('button[type="submit"]').click()
    cy.contains('Registro exitoso').should('be.visible')
  })
})
```

7.2.3. Ventajas de usar fixtures

- Mantenibilidad: Los datos se pueden actualizar fácilmente sin cambiar el código de la prueba
- Consistencia: Asegura que todas las pruebas usen los mismos datos
- Legibilidad: El código de la prueba se enfoca en el comportamiento, no en los datos
- Flexibilidad: Puedes tener múltiples conjuntos de datos para diferentes escenarios

7.3. Comandos personalizados

Los comandos personalizados en Cypress te permiten crear funciones reutilizables específicas para tu aplicación, extendiendo así la funcionalidad de Cypress.

7.3.1. ¿Qué son los comandos personalizados?

- Funciones JavaScript que extienden la API de Cypress
- Se definen en el archivo `cypress/support/commands.js`
- Permiten encapsular secuencias de acciones comunes
- Mejoran la legibilidad y mantenibilidad de las pruebas

7.3.2. Creando un comando personalizado

Abre el archivo `cypress/support/commands.js` y añade tu comando personalizado:

```
Cypress.Commands.add('login', (email, password) => {
  cy.visit('/login')
  cy.get('#email').type(email)
  cy.get('#password').type(password)
  cy.get('button[type="submit"]').click()
  cy.url().should('include', '/dashboard') // Asume redirección al
    dashboard tras login exitoso
})
```

Ahora puedes usar este comando en tus pruebas:

```
describe('Dashboard', () => {
  it('debería mostrar el nombre del usuario después de iniciar sesi
    ón', () => {
    cy.login('usuario@ejemplo.com', 'contraseña123')
    cy.get('#user-welcome').should('contain', 'Bienvenido, Usuario'
    )
  })
})
```

7.3.3. Ventajas de los comandos personalizados

- Reutilización: Reduce la duplicación de código en tus pruebas
- Abstracción: Oculta los detalles de implementación, haciendo las pruebas más legibles
- Mantenibilidad: Si la lógica cambia, solo necesitas actualizar el comando en un lugar
- Organización: Permite agrupar lógica relacionada en un solo lugar

7.3.4. Combinando fixtures y comandos personalizados

Puedes combinar fixtures y comandos personalizados para crear pruebas aún más potentes y flexibles:

```
Cypress.Commands.add('loginWithFixture', (fixtureFile) => {
  cy.fixture(fixtureFile).then((user) => {
    cy.login(user.email, user.password)
  })
})

describe('Pruebas de usuario', () => {
  it('debería permitir que un administrador acceda al panel de
    control', () => {
    cy.loginWithFixture('admin-user')
    cy.get('#admin-panel').should('be.visible')
  })

  it('debería mostrar error para un usuario normal intentando
    acceder al panel de admin', () => {
    cy.loginWithFixture('normal-user')
    cy.get('#admin-panel').should('not.exist')
    cy.get('#error-message').should('contain', 'Acceso denegado')
  })
})
```

7.4. Actividad práctica

1. Crea un fixture para un usuario de prueba en `cypress/fixtures/test-user.json`.
2. Crea un comando personalizado llamado `loginAndNavigate` que use este fixture para iniciar sesión y navegar a una página específica.
3. Escribe una prueba que use este comando personalizado para verificar que cierto contenido está presente en la página después de iniciar sesión.

8. Parte 5: Integración con CI/CD y mejores prácticas (10 minutos)

8.1. Ejecución en CI/CD

Cypress puede ejecutarse en modo headless para CI/CD:

```
npx cypress run
```

8.2. Buenas prácticas

- Usa selectores de datos específicos para pruebas (`data-testid`)
- Mantén las pruebas independientes entre sí
- Simula el estado del servidor cuando sea posible
- Utiliza `beforeEach()` para configurar el estado inicial

9. Conclusión y reflexión final

9.1. Recapitulación

Repasa los conceptos clave cubiertos en el taller.

9.2. Discusión grupal

En grupos, discutan:

- ¿Cómo planean implementar Cypress en sus proyectos actuales?
- ¿Qué desafíos anticipan y cómo planean abordarlos?
- ¿Qué otros aspectos de las pruebas les gustaría explorar?

9.3. Recursos adicionales

- Documentación oficial de Cypress: `https://docs.cypress.io`
- Ejemplos de Cypress: `https://github.com/cypress-io/cypress-example-recipes`
- Curso en línea: "Testing React with Cypress" en Egghead.io