Testing y pruebas - Jest

Julian F. Latorre

6 de agosto de 2024

Índice

1. Introducción

Este documento proporciona una guía exhaustiva y detallada para implementar pruebas utilizando Cypress, un poderoso framework de pruebas de extremo a extremo para aplicaciones web. Cada paso se explica en profundidad, incluyendo el propósito de cada acción y una descripción detallada de los fragmentos de código utilizados.

2. Pasos de Implementación Detallados

1. **Instalación de Node.js**

   - **Propósito:** Node.js es esencial para ejecutar Cypress y gestionar las dependencias del proyecto.
   - **Pasos detallados:**
     a) Visita `https://nodejs.org`
     b) Descarga la versión LTS (Long Term Support) de Node.js. Esta versión es estable y tiene soporte a largo plazo.

c) Ejecuta el instalador y sigue las instrucciones en pantalla.

d) Verifica la instalación abriendo una terminal y ejecutando:

```
node --version
npm --version
```

Estos comandos deberían mostrar las versiones instaladas de Node.js y npm respectivamente.

2. **Creación del proyecto**

- **Propósito:** Establecer una estructura de directorios para nuestro proyecto de pruebas Cypress.
- **Pasos detallados:**
  a) Abre una terminal o línea de comandos.
  b) Ejecuta: `mkdir proyecto-cypress`
     - Este comando crea un nuevo directorio llamado "proyecto-cypress".
  c) Navega al directorio: `cd proyecto-cypress`
     - Cambia el directorio de trabajo actual al nuevo directorio creado.
  d) Inicializa el proyecto: `npm init -y`
     - Este comando crea un archivo `package.json` con la configuración predeterminada.
     - La opción `-y` acepta automáticamente todas las opciones predeterminadas.

3. **Instalación de Cypress**

- **Propósito:** Instalar Cypress como una dependencia de desarrollo en nuestro proyecto.
- **Paso detallado:**
  a) Ejecuta: `npm install cypress -save-dev`
     - Este comando instala Cypress y lo agrega a las dependencias de desarrollo en `package.json`.
     - `-save-dev` indica que Cypress es una dependencia de desarrollo, no de producción.

4. **Apertura de Cypress**

- **Propósito:** Iniciar la interfaz gráfica de Cypress para configurar y ejecutar pruebas.
- **Paso detallado:**
  a) Ejecuta: `npx cypress open`
     - `npx` es una herramienta de ejecución de paquetes de Node.js.

- Este comando abre la interfaz gráfica de Cypress, permitiéndote configurar y ejecutar pruebas.

5. **Configuración inicial**
   - **Propósito:** Configurar Cypress para pruebas de extremo a extremo (E2E) y seleccionar un navegador.
   - **Pasos detallados:**
     a) En la interfaz de Cypress, selecciona ."E2E Testing".
        - Esto configura Cypress para pruebas de extremo a extremo, que simulan la interacción real del usuario con la aplicación.
     b) Elige tu navegador preferido de la lista disponible.
        - Cypress soporta varios navegadores como Chrome, Firefox, y Edge.
        - La elección del navegador depende de las necesidades específicas de tu proyecto.

6. **Creación de un archivo de prueba**
   - **Propósito:** Crear un archivo donde escribiremos nuestras pruebas de Cypress.
   - **Paso detallado:**
     a) En la carpeta `cypress/e2e`, crea un nuevo archivo llamado `mi_prueba.cy.js`
        - La extensión `.cy.js` es una convención de Cypress para identificar archivos de prueba.
        - Cypress automáticamente reconocerá y cargará este archivo en su interfaz.

7. **Escritura de una prueba básica**
   - **Propósito:** Escribir una prueba simple para demostrar la funcionalidad básica de Cypress.
   - **Paso detallado:**
     a) Abre `mi_prueba.cy.js` y añade el siguiente código:

```
describe('Mi primera prueba', () => {
  it('Visita Google', () => {
    cy.visit('https://www.google.com')
    cy.get('input[name="q"]').type('Cypress testing')
    cy.get('input[name="btnK"]').click()
    cy.title().should('include', 'Cypress testing')
  })
})
```

Explicación línea por línea:

- `describe('Mi primera prueba', () =>{ ... })`: Define un grupo de pruebas relacionadas.

- `it('Visita Google', () =>{ ... })`: Define una prueba específica dentro del grupo.
- `cy.visit('https://www.google.com')`: Navega a la página de Google.
- `cy.get('input[name="q"]').type('Cypress testing')`: Encuentra el campo de búsqueda y escribe Çypress testing".
- `cy.get('input[name="btnK"]').click()`: Encuentra el botón de búsqueda y hace clic en él.
- `cy.title().should('include', 'Cypress testing')`: Verifica que el título de la página incluya Çypress testing".

8. **Ejecución de la prueba**
   - **Propósito:** Ejecutar la prueba que acabamos de escribir y observar su funcionamiento.
   - **Pasos detallados:**
     a) En la interfaz de Cypress, haz clic en `mi_prueba.cy.js`
     b) Observa cómo Cypress ejecuta la prueba automáticamente
        - Cypress abrirá el navegador seleccionado y ejecutará cada paso de la prueba.
        - Podrás ver en tiempo real cómo se realiza cada acción especificada en la prueba.

9. **Análisis de resultados**
   - **Propósito:** Interpretar los resultados de la prueba ejecutada.
   - **Pasos detallados:**
     a) Revisa los resultados en la interfaz de Cypress
        - La interfaz mostrará cada paso de la prueba y su estado (éxito o fallo).
        - Si hay errores, Cypress proporcionará información detallada sobre el fallo.
     b) Verifica que la prueba haya pasado correctamente
        - Una prueba exitosa se mostrará en verde.
        - Si la prueba falla, se mostrará en rojo con detalles sobre el error.

10. **Mejora y expansión de las pruebas**
    - **Propósito:** Desarrollar un conjunto más completo de pruebas para tu aplicación.
    - **Pasos detallados:**
      a) Añade más pruebas al archivo `mi_prueba.cy.js`

- Puedes agregar más bloques `it()` dentro del `describe()` existente.
- Ejemplo:

```
it('Verifica resultados de búsqueda', () => {
  cy.get('#search').should('contain', 'Cypress')
})
```

b) Crea nuevos archivos de prueba para diferentes funcionalidades
- Por ejemplo, `login.cy.js` para pruebas de inicio de sesión.
- Organiza tus pruebas en archivos separados para mantener una estructura clara.

3. Conclusión

Has completado la implementación detallada de pruebas con Cypress. Este proceso te ha proporcionado una base sólida para comenzar a escribir y ejecutar pruebas automatizadas para tus aplicaciones web. Recuerda que Cypress ofrece muchas más funcionalidades avanzadas, como el manejo de estados, la simulación de peticiones de red, y la creación de comandos personalizados. Te recomendamos continuar explorando la documentación oficial de Cypress para aprovechar al máximo esta potente herramienta de pruebas.

4. Ejemplo de Pruebas Cypress para una Aplicación Fullstack

A continuación, se presenta un ejemplo de cómo escribir pruebas end-to-end con Cypress para una aplicación fullstack de lista de tareas.

4.1. Configuración inicial

Primero, asegúrate de que Cypress está instalado en tu proyecto:

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

4.2. Archivo de pruebas

Crea un nuevo archivo llamado `todo_list.spec.js` en la carpeta `cypress/integration/` con el siguiente contenido:

```
describe('Todo List App', () => {
  beforeEach(() => {
    // Visitar la página principal antes de cada prueba
    cy.visit('http://localhost:3000')
  })

  it('Carga la aplicación correctamente', () => {
```

```
  // Verificar que el título de la página es correcto
  cy.title().should('eq', 'Todo List App')

  // Verificar que el encabezado principal está presente
  cy.get('h1').should('contain', 'Lista de Tareas')
})

it('Puede agregar una nueva tarea', () => {
  const newTask = 'Comprar leche'

  // Escribir una nueva tarea en el campo de entrada
  cy.get('input[placeholder="Nueva tarea"]').type(newTask)

  // Hacer clic en el botón para agregar la tarea
  cy.get('button').contains('Agregar').click()

  // Verificar que la nueva tarea aparece en la lista
  cy.get('ul.task-list').should('contain', newTask)
})

it('Puede marcar una tarea como completada', () => {
  const taskToComplete = 'Comprar leche'

  // Encontrar la tarea en la lista y marcarla como completada
  cy.get('li').contains(taskToComplete)
    .find('input[type="checkbox"]')
    .check()

  // Verificar que la tarea está marcada como completada
  cy.get('li').contains(taskToComplete)
    .should('have.class', 'completed')
})

it('Puede eliminar una tarea', () => {
  const taskToDelete = 'Comprar leche'

  // Encontrar la tarea en la lista y hacer clic en el botón de
  eliminar
  cy.get('li').contains(taskToDelete)
    .find('button.delete')
    .click()

  // Verificar que la tarea ya no está en la lista
  cy.get('ul.task-list').should('not.contain', taskToDelete)
})

it('Muestra el número correcto de tareas pendientes', () => {
  // Agregar algunas tareas
  cy.get('input[placeholder="Nueva tarea"]').type('Tarea 1{enter}
  ')
  cy.get('input[placeholder="Nueva tarea"]').type('Tarea 2{enter}
  ')
  cy.get('input[placeholder="Nueva tarea"]').type('Tarea 3{enter}
  ')

  // Marcar una tarea como completada
  cy.get('li').contains('Tarea 2')
```

```
      .find('input[type="checkbox"]')
      .check()

    // Verificar que el contador de tareas pendientes es correcto
    cy.get('.pending-count').should('contain', '2 tareas pendientes
    ')
  })
})
```

4.3. Explicación del código

Este archivo de pruebas realiza las siguientes acciones:

1. **Carga de la aplicación:** Verifica que la aplicación se carga correctamente y muestra el título y encabezado esperados.
2. **Agregar una tarea:** Prueba la funcionalidad de agregar una nueva tarea a la lista.
3. **Marcar una tarea como completada:** Verifica que se puede marcar una tarea como completada y que se refleja visualmente.
4. **Eliminar una tarea:** Comprueba que se puede eliminar una tarea de la lista.
5. **Contador de tareas pendientes:** Asegura que el contador de tareas pendientes se actualiza correctamente al agregar y completar tareas.

4.4. Ejecución de las pruebas

Para ejecutar estas pruebas, asegúrate de que tu aplicación está corriendo en `http://localhost:3000` y luego ejecuta Cypress con el siguiente comando:

```
npx cypress open
```

Selecciona el archivo `todo_list.spec.js` en la interfaz de Cypress para ejecutar las pruebas.

4.5. Consideraciones adicionales

- Este ejemplo asume que la aplicación tiene una estructura y nombres de clases específicos. Ajusta los selectores según sea necesario para que coincidan con tu implementación real.
- En una aplicación real, considera añadir pruebas para manejar errores, como intentar agregar una tarea vacía o manejar problemas de conexión con el backend.
- Para aplicaciones más complejas, podrías necesitar configurar un estado inicial de la base de datos antes de las pruebas, o mockear ciertas respuestas de la API para pruebas más controladas.