

LABORATORIO DE PRUEBAS AUTOMATIZADAS CON JEST

Juan Jose Santacruz Ferraro

1. INTRODUCCIÓN

Durante este laboratorio estuvimos trabajando con pruebas unitarias en React usando Jest. Lo que hicimos fue crear un contador interactivo que tiene varias funciones, y luego le hicimos pruebas automáticas para asegurarnos de que todo funcione como se espera.

2. OBJETIVO

- Usar Jest para crear pruebas automáticas en un componente de React.
- Revisar la cobertura de las pruebas con el comando `--coverage`.
- Comprobar que el contador funcione bien incluso en casos extraños o entradas incorrectas.

3. DESARROLLO

3.1 Creación del proyecto

Primero creamos el proyecto con el comando:

```
npm create vite@latest jest-contador -- --template react
```

Después instalamos todo lo necesario con:

```
npm install
```

Creamos un componente llamado `Counter.jsx` que hace lo siguiente:

- Suma y resta de uno en uno
- Suma y resta con un número que pongamos
- Reinicia el contador al valor inicial
- Valida que si escribimos letras o texto, no se dañe

3.2 Configuración de Jest y Babel

Instalamos estas dependencias para poder usar Jest y hacer las pruebas:

```
npm install --save-dev jest @testing-library/react @testing-library/jest-dom babel-jest
@babel/preset-env @babel/preset-react
```

También creamos los siguientes archivos:


- babel.config.js

```
B babel.config.js > ...
1  // babel.config.js
2  module.exports = {
3    presets: ['@babel/preset-env', '@babel/preset-react'],
4  };
5
```

- jest.config.js

```
jest.config.js > ...
1  module.exports = {
2    testEnvironment: 'jest-environment-jsdom',
3    transform: {
4      '^.+\\.jsx?$': 'babel-jest',
5    },
6    setupFilesAfterEnv: ['<rootDir>/src/setupTests.js'],
7    moduleFileExtensions: ['js', 'jsx'],
8  };
9
```

- src/setupTests.js (para que funcione bien Testing Library)

```
src >  setupTests.js
1   import '@testing-library/jest-dom';
2
```

Y en el package.json agregamos este script:

"coverage": "jest --coverage"

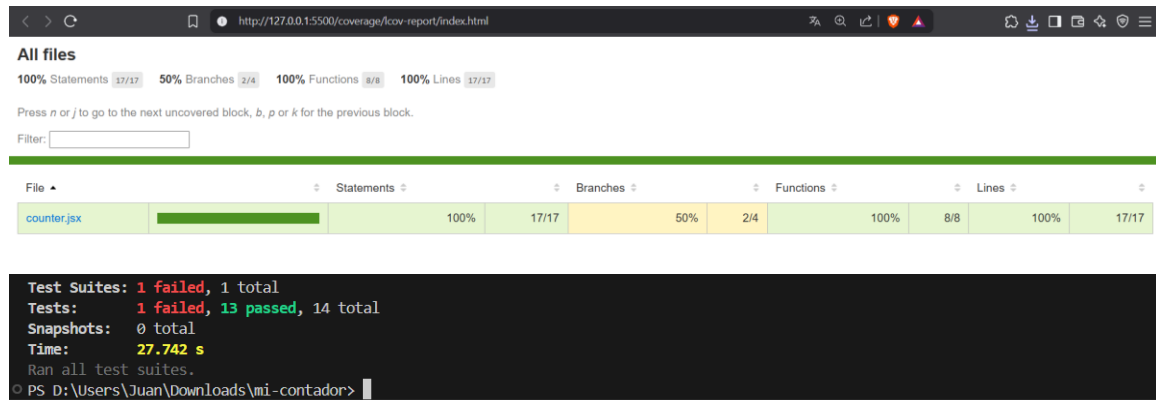
```
 package.json > ...
1   {
2     "name": "mi-contador",
3     "private": true,
4     "version": "0.0.0",
5     > Debug
6     "scripts": {
7       "dev": "vite",
8       "build": "vite build",
9       "lint": "eslint .",
10      "preview": "vite preview",
11      "test": "jest",
12      "coverage": "jest --coverage"
13    }
14 }
```

3.3 Pruebas unitarias

Creamos un archivo Counter.test.js con más de 10 pruebas. Algunas de las cosas que probamos fueron:

- Que se muestre bien el valor inicial
- Que sume y reste correctamente
- Que funcione bien con valores personalizados
- Que no se dañe si escribimos letras
- Que el botón de reset funcione bien
- Que incluso si ponemos espacios, lo entienda como número

4. RESULTADOS



Cobertura obtenida:

- Statements: 17/17 (100%)
- Functions: 8/8 (100%)
- Lines: 17/17 (100%)
- Branches: 2/4 (50%)

Esto muestra que la mayoría del componente fue probado correctamente. Sin embargo, una prueba falló. Fue la que verifica si el número con espacios es entendido correctamente. El fallo no es porque esté mal el componente, sino porque el texto en HTML está dividido y eso puede confundir a Testing Library.

5. DIFICULTADES

- Tuvimos problemas configurando Jest al inicio, porque hay diferencias entre cómo maneja los módulos.
- Al principio fue difícil pensar cómo hacer algunas pruebas, como cuando el usuario mete texto o deja vacío el input.
- Algunas condiciones internas no se probaron al 100%, pero no afectan el funcionamiento.
- La prueba que falló fue por cómo se busca el texto en la pantalla. A veces el número y el texto están en elementos distintos y eso hace que falle la búsqueda.

6. CONCLUSIONES

- Usar Jest fue una buena forma de comprobar que el componente funciona como debe.

- Las pruebas automáticas ayudan a evitar errores y a estar seguros de que todo está bien.
- Revisar la cobertura nos da una idea clara de qué tanto se está probando en realidad.