

TESTING EN REACT

1. Introducción al Testing

¿Por qué hacer testing?

El testing en aplicaciones React nos permite:

- Detectar errores antes de que lleguen a producción
- Documentar el comportamiento esperado de los componentes
- Refactorizar con confianza sabiendo que no rompemos funcionalidad existente
- Mejorar el diseño del código al pensar en cómo probarlo
- Reducir costos al encontrar bugs temprano en el desarrollo

2. Tipos de Testing

- Unit Tests (Pruebas Unitarias): Prueban componentes individuales de forma aislada
- Integration Tests (Pruebas de Integración): Prueban cómo varios componentes trabajan juntos
- End-to-End Tests (E2E): Prueban flujos completos de la aplicación como lo haría un usuario

En React, nos enfocaremos en pruebas unitarias y pruebas de integración.

3. Herramientas Principales

Jest: es el framework de testing más popular para JavaScript. Viene preconfigurado con Create React App y proporciona:

- Test runner (ejecutor de pruebas)
- Assertions (afirmaciones)
- Mocks y spies
- Coverage reports (reportes de cobertura)

React Testing Library (RTL)

Es la biblioteca recomendada para probar componentes React. Su filosofía es:

"Mientras más se parezcan tus tests a cómo los usuarios usan tu aplicación, más confianza te darán"

Principios de RTL:

- Probar el comportamiento, no la implementación
- Interactuar con el DOM como lo haría un usuario
- Evitar probar detalles internos (state, props internos)

4. Configuración Inicial

- a. Si se utilizó Create React App, Jest y RTL ya vienen configurados. Si se ha creado con Vite, debe instalar:

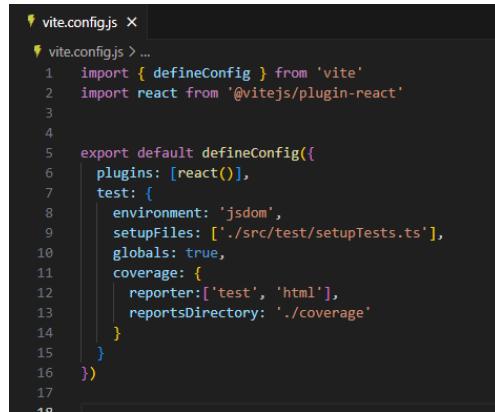
```
npm install --save-dev @testing-library/react @testing-library/jest-dom @testing-library/user-event
```

Este comando permite instalar:

| Paquete | Función |
|-----------------------------|--|
| vitest | Test runner (similar a Jest, rápido y compatible con Vite). |
| @testing-library/react | Permite renderizar y testear componentes de React. |
| @testing-library/jest-dom | Añade aserciones extra como toBeInTheDocument(). |
| @testing-library/user-event | Simula interacciones reales de usuario (click, type, tab, etc.). |
| jsdom | Entorno DOM simulado para ejecutar React en tests. |

b. Posteriormente configurar vitests en el proyecto:

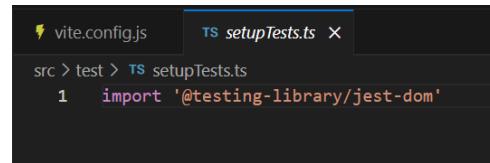
Agregar la sección de test en el archivo vite.config.ts o vite.config.js;



```
1 import { defineConfig } from 'vite'
2 import react from '@vitejs/plugin-react'
3
4
5 export default defineConfig({
6   plugins: [react()],
7   test: {
8     environment: 'jsdom',
9     setupFiles: ['./src/test/setupTests.ts'],
10    globals: true,
11    coverage: {
12      reporter: ['test', 'html'],
13      reportsDirectory: './coverage'
14    }
15  }
16})
17
```

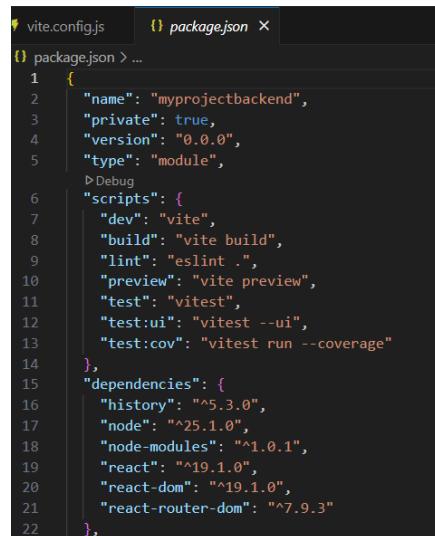
c. Crear el archivo de configuración global

En la carpeta src/ crear la carpeta test/ y luego el archivo: setupTests.ts (o .js)



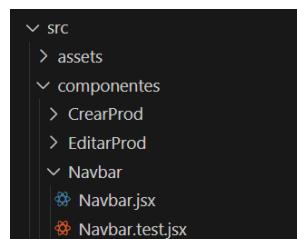
```
1 import '@testing-library/jest-dom'
```

d. Agregar scripts en package.json:



```
1 {
2   "name": "myprojectbackend",
3   "private": true,
4   "version": "0.0.0",
5   "type": "module",
6   "scripts": {
7     "dev": "vite",
8     "build": "vite build",
9     "lint": "eslint .",
10    "preview": "vite preview",
11    "test": "vitest",
12    "test:ui": "vitest --ui",
13    "test:cov": "vitest run --coverage"
14  },
15  "dependencies": {
16    "history": "^5.3.0",
17    "node": "^25.1.0",
18    "node-modules": "^1.0.1",
19    "react": "^19.1.0",
20    "react-dom": "^19.1.0",
21    "react-router-dom": "^7.9.3"
22  },
23}
```

e. Estructura de carpetas: se recomienda en cada carpeta de componente crear archivo de test:



5. Ejemplo:

- Aplicaremos testing en el componente navbar

```
vite.config.js  Navbar.test.jsx
src > components > Navbar > Navbar.test.jsx > describe('Navbar') callback > it('muestra la marca y los links con sus href correctos') callback
1 // src/components/Navbar/Navbar.test.jsx
2 import { render, screen } from '@testing-library/react'
3 import userEvent from '@testing-library/user-event'
4 import { MemoryRouter, Routes, Route } from 'react-router-dom'
5 import { Navbar } from './Navbar'
6
7 describe('Navbar', () => {
8
9   it('muestra la marca y los links con sus href correctos', () => {
10    render(
11      <MemoryRouter>
12        | <Navbar />
13      </MemoryRouter>
14    )
15
16    const brand = screen.getByRole('link', { name: '/mi página/i' })
17    expect(brand).toBeInTheDocument()
18    expect(brand).toHaveAttribute('href', '/')
19
20    expect(screen.getByRole('link', { name: '/home/i' }).toHaveAttribute('href', '/'))
21    expect(screen.getByRole('link', { name: '/inventario/i' }).toHaveAttribute('href', '/inventario'))
22    expect(screen.getByRole('link', { name: '/contacto/i' }).toHaveAttribute('href', '/contacto'))
23
24    const toggler = screen.getByRole('button')
25    expect(toggler).toHaveAttribute('data-bs-toggle', 'collapse')
26    expect(toggler).toHaveAttribute('data-bs-target', '#menuNav')
27  )
28
29
30  it('navega a /inventario al hacer click en el link', async () => {
31    const user = userEvent.setup()
32    render(
33      <MemoryRouter initialEntries={['/']}>
34        | <Navbar />
35        <Routes>
36          <Route path="/" element={<h1>Home</h1>} />
37          <Route path="/inventario" element={<h1>Inventario</h1>} />
38          <Route path="/contacto" element={<h1>Contacto</h1>} />
39        </Routes>
40      </MemoryRouter>
41    )
42
43    await user.click(screen.getByRole('link', { name: '/inventario/i' }))
44
45    expect(await screen.findByRole('heading', { name: '/inventario/i' })).toBeInTheDocument()
46  )
47})
48
```

- Ejecutar testing:

- npm run test
- npm run test:

En consola:

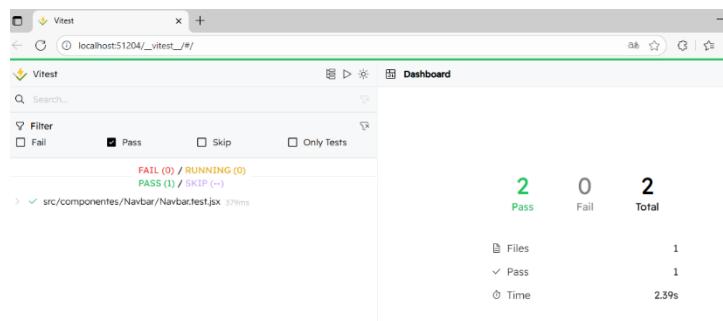
```
Test Files 0 passed (1)
  Tests 0 passed (0)
  Start at 17:38:34
  Duration 3.56s

C:\Users\viliv\Downloads\myprojectReactOctubre13>npm run test
> myprojectbackend@0.0.0 test
> vitest

vite v4.0.6 C:/Users/viliv/Downloads/myprojectReactOctubre13
  src/componentes/Navbar/Navbar.test.jsx (2 tests) 308ms
    └─ Navbar (2)
      └─ muestra la marca y los links con sus href correctos 207ms
      └─ navega a /inventario al hacer click en el link 100ms

  Test Files 1 passed (1)
  Tests 2 passed (2)
  Start at 17:38:43
  Duration 9.66s (transform 128ms, setup 1.30s, collect 1.92s, tests 308ms, environment 5.86s, prepare 17ms)

  PASS Waiting for file changes...
  press h to show help, press q to quit
```



Interfaz gráfica:

npm run test: ui

ui

6. Testeando componente producto:

```
vite.config.js Navbartest.jsx Productos.jsx Productos.test.jsx
src > components > Productos > Productos.test.jsx > describe('Productos.jsx') callback
1 import { render, screen, waitFor, within } from '@testing-library/react'
2 import userEvent from '@testing-library/user-event'
3 import { MemoryRouter } from 'react-router-dom'
4 import { vi } from 'vitest'
5 import { Productos } from './Productos'
6
7 const renderWithRouter = (ui) => render(<MemoryRouter>{ui}</MemoryRouter>)
8
9 describe('Productos.jsx', () => {
10   const originalFetch = global.fetch
11   const originalConfirm = window.confirm
12   const originalAlert = window.alert
13
14   beforeEach(() => {
15     global.fetch = vi.fn()
16     window.confirm = vi.fn().mockReturnValue(true)
17     window.alert = vi.fn()
18   })
19
20   afterEach(() => {
21     vi.clearAllMocks()
22   })
23
24   afterAll(() => {
25     global.fetch = originalFetch
26     window.confirm = originalConfirm
27     window.alert = originalAlert
28   })
29
30   test('carga y muestra productos', async () => {
31     const mockData = [
32       { id: 1, nombre: 'Teclado', descripcion: 'Mecánico', precio: 19990, activo: true },
33       { id: 2, nombre: 'Mouse', descripcion: 'Óptico', precio: 9990, activo: 'false' }
34     ]
35
36     global.fetch.mockResolvedValueOnce({
37       ok: true,
38       json: async () => mockData
39     })
40   })
41 })
```

Este test, cubre los siguientes casos:

- a. **Carga inicial:** renderiza productos, normaliza atributo 'activo', muestra estilos y botones correctos.
- b. **Botón desactivar:** invoca correctamente a Patch, actualiza lista y muestra alertas.
- c. **Cancelación confirm:** no llama Patch ni alerta si el usuario cancela.
- d. **Accesibilidad básica:** usa queries getByRole y findByText en vez de getByTestId.

(revisar componente cargado en AVA)