HERRAMIENTAS AUTOMATIZADAS

Para testing

TOP 3 HERRAMIENTAS DE TESTING UNITARIO PARA FRONTEND CON REACT

Jest

Ventajas

- Todo en uno: assertions, mocks, coverage.
- Integración con React.
- Muy buena documentación y comunidad grande.

Desventajas

- Puede ser más lento en proyectos grandes sin Vite.
- La configuración puede crecer si lo combinas con muchas herramientas.

Vitest

Ventajas

- Muy rápido (basado en Vite, usa ESM por defecto).
- API similar a Jest, fácil migración.
- Ideal para proyectos modernos con Vite + React.

Desventajas

- Comunidad más pequeña que Jest (aunque está creciendo rápido).
- No tan probado en grandes proyectos legacy.

React Testing Library (RTL)

Ventajas

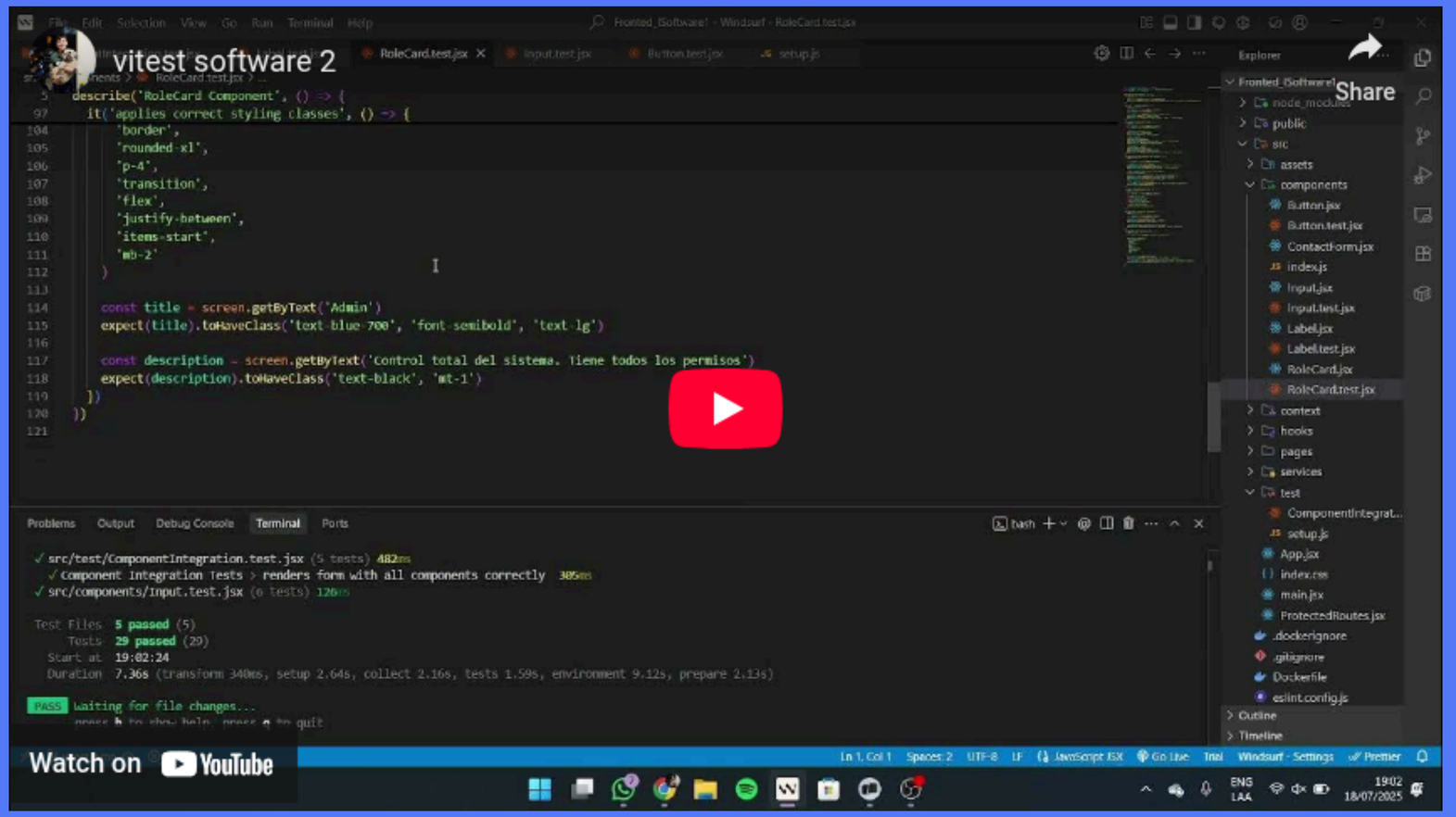
- Fomenta pruebas desde la perspectiva del usuario.
- Funciona muy bien con Jest.
- Minimiza el acoplamiento a la implementación interna.

Desventajas

- No es un framework completo, necesitas Jest u otro runner.
- Puede ser más difícil testear lógica compleja sin apoyo externo.

HERRAMIENTA SELECCIONADA - VITEST

- Integración nativa con proyectos Vite (ideal para React, Vue, etc.).
- API similar a Jest, facilita la migración.
- Muy rápido (basado en ESM y Vite).
- Soporte para testing de componentes con @testing-library.
- Coverage reporting integrado (--coverage).


vitest software 2
Share
describe('RoleCard Component', () => {
it('applies correct styling classes', () => {
'border',
'rounded-xl',
'p-4',
'transition',
'flex',
'justify-between',
'items-start',
'mb-2'
)
const title = screen.getByText('Admin')
expect(title).toHaveClass('text-blue-700', 'font-semibold', 'text-lg')
const description = screen.getByText('Control total del sistema. Tiene todos los permisos')
expect(description).toHaveClass('text-black', 'mt-1')
})
})
Problems
Output
Debug Console
Terminal
Ports
✓ src/test/ComponentIntegration.test.jsx (5 tests) 482ms
✓ Component Integration Tests > renders form with all components correctly 305ms
✓ src/components/Input.test.jsx (6 tests) 126ms
Test Files 5 passed (5)
Tests 29 passed (29)
Start at 19:02:24
Duration 7.36s (transform 340ms, setup 2.64s, collect 2.16s, tests 1.59s, environment 9.12s, prepare 2.13s)
PASS Waiting for file changes...
Watch on YouTube

TOP 3 HERRAMIENTAS DE TESTING UNITARIO PARA BACKEND CON NODE JS

Mocha + Chai

Framework flexible para pruebas en Node.js, generalmente usado con Chai para aserciones.

Muy popular
Extensible con plugins
Buen soporte para pruebas asíncronas

Requiere más configuración que Jest
Necesita librerías adicionales (Chai, Sinon)

Jest

Framework de testing desarrollado por Facebook, soporta pruebas unitarias, de integración y mocking.

Fácil configuración
Soporte para async/await
Gran documentación
Snapshots testing

Puede ser lento en proyectos grandes

Supertest

Librería para probar APIs HTTP (complementa Mocha/Jest).

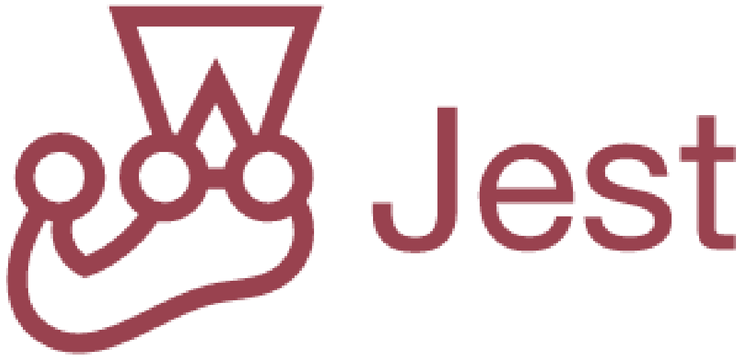
Ideal para endpoints REST
Fácil integración con Express

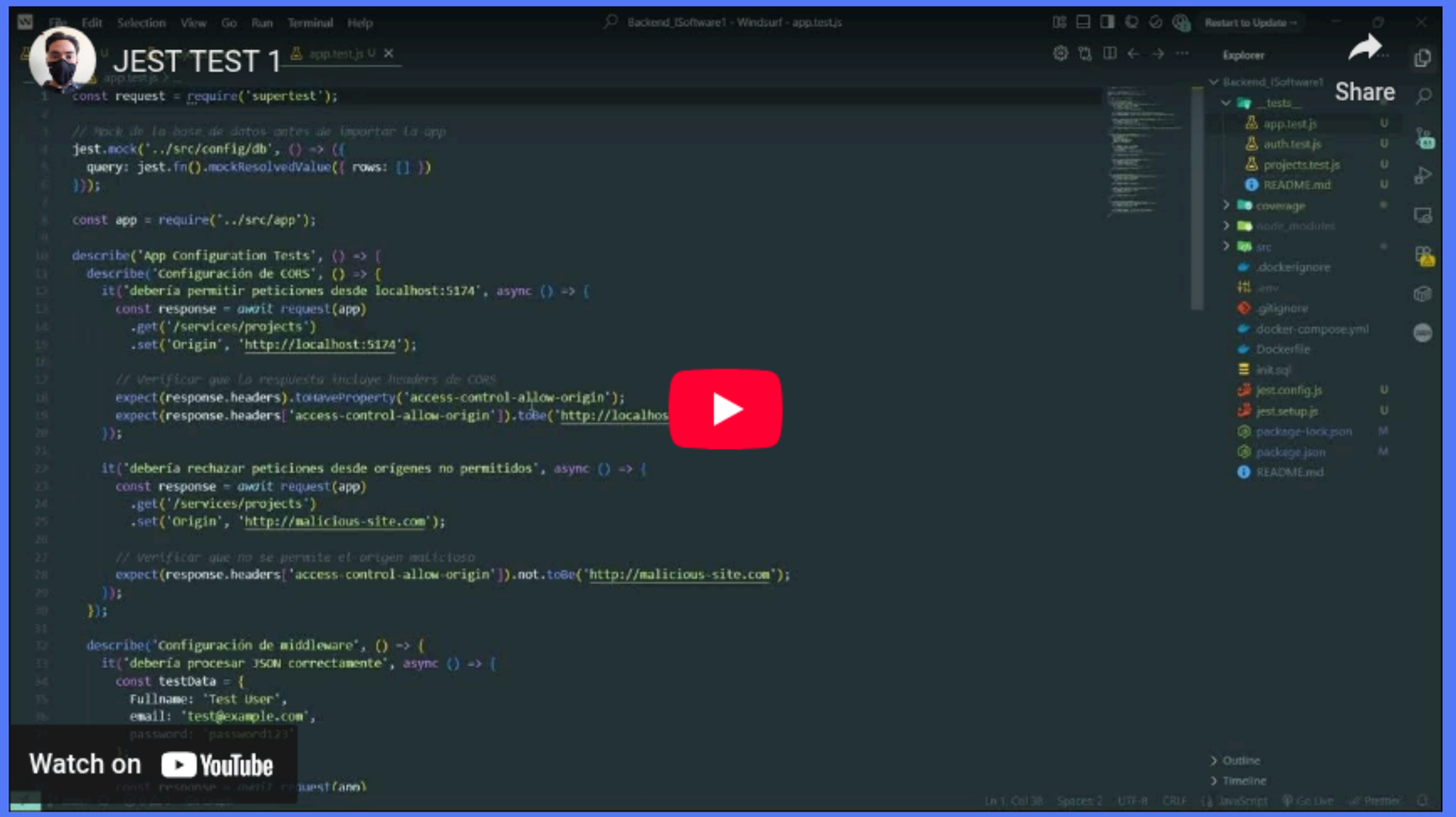
Solo para pruebas HTTP

HERRAMIENTA SELECCIONADA
JEST

Razones de la elección:

- Integración sencilla con proyectos Node.js.
- Soporte para mocking de bases de datos (PostgreSQL).
- Documentación amplia y comunidad activa.
- Velocidad y paralelización de pruebas.
- Coverage reporting integrado (--coverage).




JEST TEST 1
Share
Watch on YouTube

CONCLUSIONES

1. Jest fue fácil de configurar y usar, con buena documentación y mensajes claros.
2. Escribir los tests fue sencillo gracias a su sintaxis clara y soporte para async/await.
3. Supertest se integró sin complicaciones y facilitó las pruebas de endpoints REST.
4. El tiempo invertido fue bajo: los tests corren rápido y dan feedback inmediato.
5. La cobertura alcanzó 63.5%, mostrando que la herramienta ayuda a detectar partes no testeadas.
6. En general, la experiencia fue positiva: aprendimos rápido, automatizamos bien, y vimos resultados útiles.

GRACIAS!