

PRUEBAS DE SOFTWARE

DIEGO ALEJANDRO MESA VASQUEZ



CENTRO DE TECNOLOGIAS
AGROINDUSTRIALES

MEDELLÍN

2024

Introducción

En el desarrollo de software, las pruebas son una etapa crucial que ayuda a asegurar la calidad y funcionalidad de una aplicación. Existen diferentes tipos de pruebas, cada una con su enfoque y propósito, que permiten identificar errores, validar funcionalidades y garantizar que el producto cumpla con los requisitos establecidos. En este informe, exploraremos las pruebas de software más comunes, discutiremos cuáles son las más adecuadas para nuestro proyecto y describiremos el uso de una herramienta de pruebas específica, Jest, que instalamos para llevar a cabo pruebas básicas en nuestra solución de software. A través de este proceso, buscamos no solo mejorar la calidad de nuestro producto, sino también adquirir una comprensión más profunda de las mejores prácticas en la industria del desarrollo de software.

Preguntas de Reflexión sobre Pruebas de Software

¿Qué tipos de pruebas de software existen? Explique sus características y beneficios.

Existen varios tipos de pruebas de software, entre los más comunes están:

- **Pruebas Unitarias:** Verifican que cada componente individual del software funcione correctamente. Son rápidas de ejecutar y ayudan a detectar errores temprano en el desarrollo.
- **Pruebas de Integración:** Aseguran que los diferentes módulos o servicios de una aplicación funcionen bien juntos. Son útiles para detectar problemas en la interacción entre componentes.
- **Pruebas Funcionales:** Evalúan si el software cumple con los requisitos funcionales. Se centran en lo que el sistema debe hacer, más que en cómo lo hace.
- **Pruebas de Rendimiento:** Miden cómo se comporta el software bajo ciertas cargas de trabajo. Ayudan a identificar cuellos de botella y problemas de escalabilidad.
- **Pruebas de Usabilidad:** Evalúan la facilidad con la que los usuarios pueden interactuar con el sistema. Son esenciales para asegurar una buena experiencia de usuario.

Según la consulta que realizó, ¿qué tipos de pruebas se adaptan mejor al proyecto de software que está desarrollando?

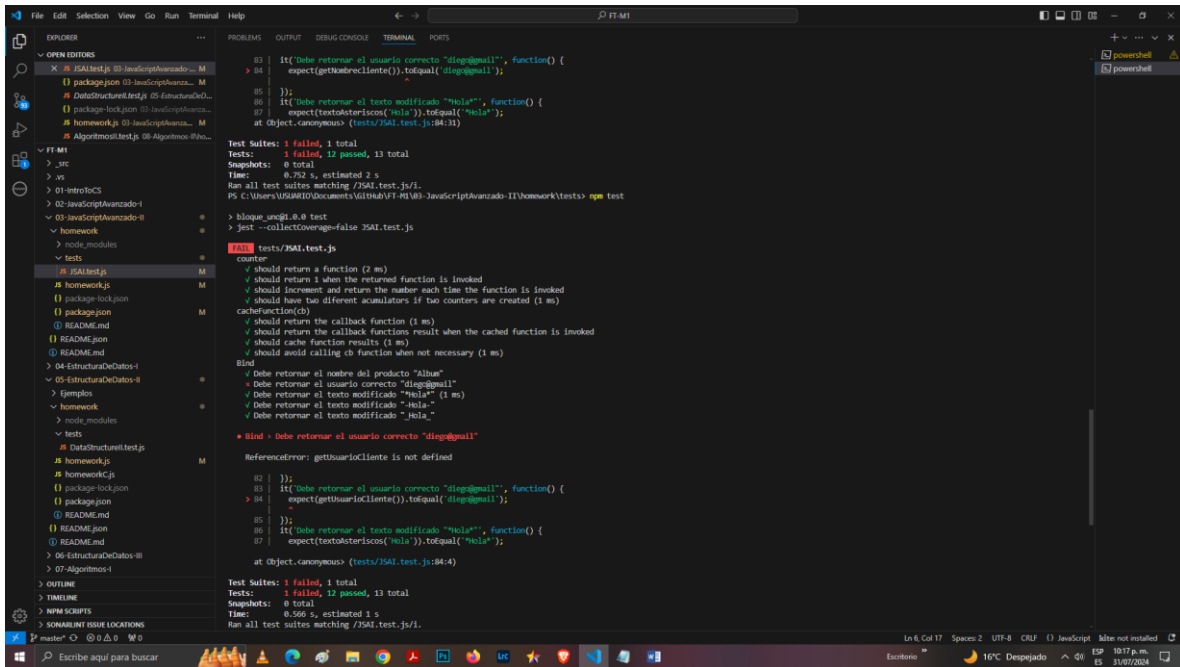
Para mi proyecto, que está enfocado en una aplicación web, las pruebas unitarias y pruebas de integración son esenciales. Las pruebas unitarias nos aseguran que cada función individual funcione bien, mientras que las pruebas de integración garantizan que todos los componentes de la aplicación trabajen juntos sin problemas. Además, considero importantes las pruebas funcionales para verificar que la aplicación cumpla con los requisitos esperados por los usuarios.

Investigue e instale unas herramientas de pruebas de software en su computador, “una de su gusto”.

He decidido usar Jest como herramienta de pruebas. Jest es un framework de pruebas muy popular para aplicaciones JavaScript, especialmente adecuado para proyectos que utilizan React, pero también es muy versátil para otros tipos de aplicaciones.

He realizado pruebas básicas utilizando Jest para verificar que las funciones de mi aplicación se comporten como se espera. Las capturas de pantalla de estas pruebas se incluirán en el documento final.

Error de configuración de las pruebas:



The screenshot shows the VS Code interface with a file explorer on the left, a code editor in the center, and a terminal on the right. The code editor displays a Jest test file with several test cases. The terminal shows the output of running the tests, indicating a configuration error. The error message is: `ReferenceError: getUsuarioCliente is not defined`. This error occurs in the test case `it('Debe retornar el usuario correcto "diego@gmail"', function() { expect(getUsuarioCliente()).toEqual('diego@gmail'); });`. The test suite summary shows 1 failed, 12 passed, and 13 total tests.

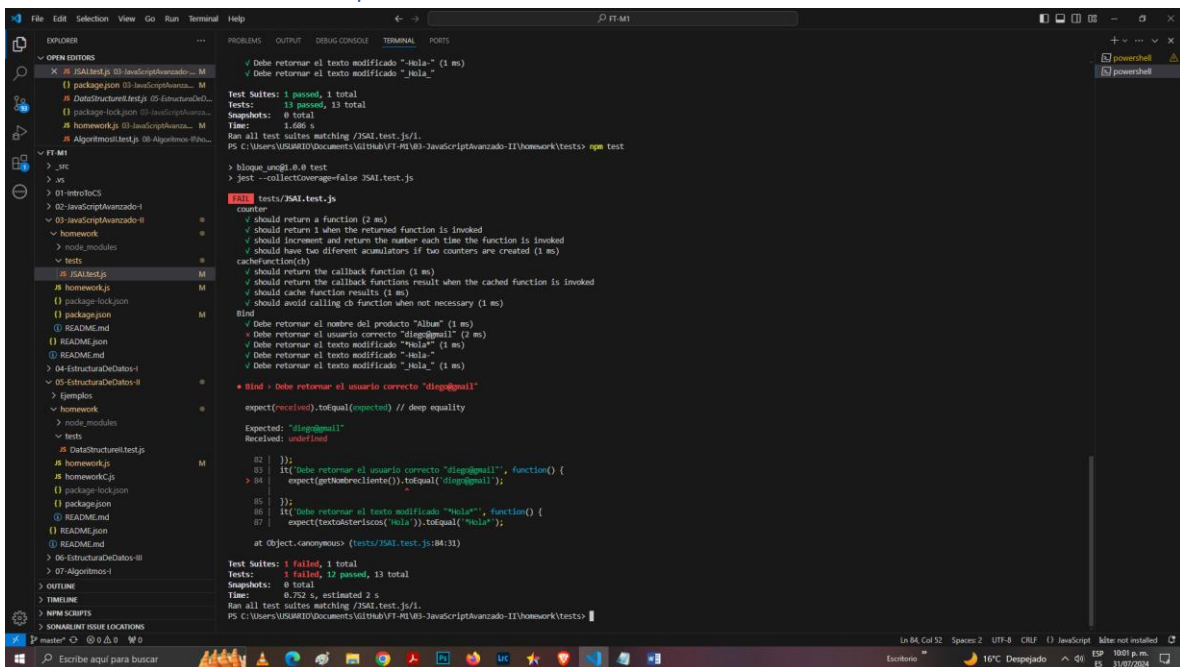
```
Test Suites: 1 failed, 1 total
Tests: 1 failed, 12 passed, 13 total
Snapshots: 0 total
Time: 0.752 s, estimated 2 s
Run all test suites matching /35AI.test.js/i.
PS C:\Users\USUR10\Documents\GitHub\VF-PI-V8-JavascriptAvanzado-II\homework\tests> npm test

> bloque_unq01.0.0 test
> jest --collectCoverage=false 35AI.test.js

FAIL tests/35AI.test.js
  counter
    ✓ should return a function (2 ms)
    ✓ should return 1 when the returned function is invoked
    ✓ should increment and return the number each time the function is invoked
    ✓ should have two different accumulators if two counters are created (1 ms)
  cacheFunction(cb)
    ✓ should return the callback function (1 ms)
    ✓ should return the callback functions result when the cached function is invoked
    ✓ should cache function results (1 ms)
    ✓ should avoid calling cb function when not necessary (1 ms)
  Bind
    ✓ Debe retornar el nombre del producto "Albar" (1 ms)
    ✓ Debe retornar el usuario correcto "diego@gmail" (2 ms)
    ✓ Debe retornar el texto modificado "Hola" (1 ms)
    ✓ Debe retornar el texto modificado "Hola_" (1 ms)
    ✓ Debe retornar el texto modificado "Hola_" (1 ms)
  • Bind • Debe retornar el usuario correcto "diego@gmail"
    ReferenceError: getUsuarioCliente is not defined
      82 |   });
      83 |   it('Debe retornar el usuario correcto "diego@gmail"', function() {
      84 |     expect(getUsuarioCliente()).toEqual('diego@gmail');
        |     ^
      85 |   });
      86 |   it('Debe retornar el texto modificado "Hola"', function() {
      87 |     expect(textoAsteriscos("Hola")).toEqual("Hola");
        |     ^
      88 |   });
      89 |   at Object.canonycous (tests/35AI.test.js:84:4)

Test Suites: 1 failed, 1 total
Tests: 1 failed, 12 passed, 13 total
Snapshots: 0 total
Time: 0.566 s, estimated 1 s
Run all test suites matching /35AI.test.js/i.
```

Error de resultado de las pruebas



The screenshot shows the VS Code interface with a file explorer on the left, a code editor in the center, and a terminal on the right. The code editor displays a Jest test file with several test cases. The terminal shows the output of running the tests, indicating a result error. The error message is: `Expected: "diego@gmail" Received: undefined`. This error occurs in the test case `it('Debe retornar el usuario correcto "diego@gmail"', function() { expect(getUsuarioCliente()).toEqual('diego@gmail'); });`. The test suite summary shows 1 failed, 12 passed, and 13 total tests.

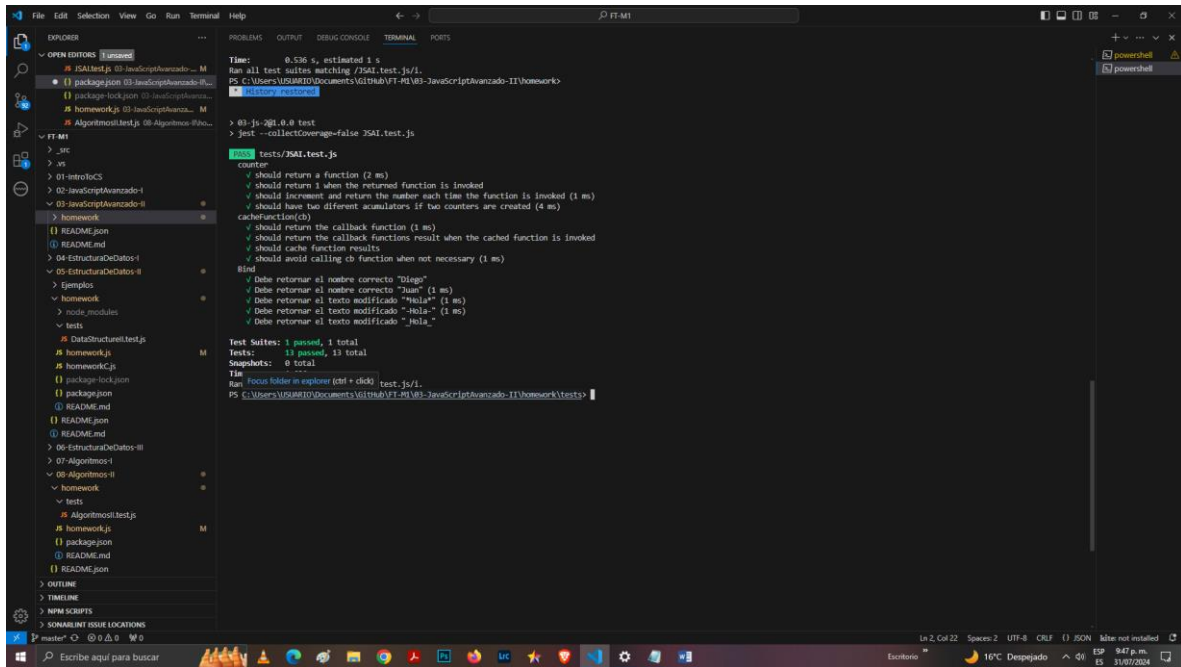
```
Test Suites: 1 failed, 1 total
Tests: 1 failed, 12 passed, 13 total
Snapshots: 0 total
Time: 0.752 s, estimated 2 s
Run all test suites matching /35AI.test.js/i.
PS C:\Users\USUR10\Documents\GitHub\VF-PI-V8-JavascriptAvanzado-II\homework\tests> npm test

> bloque_unq01.0.0 test
> jest --collectCoverage=false 35AI.test.js

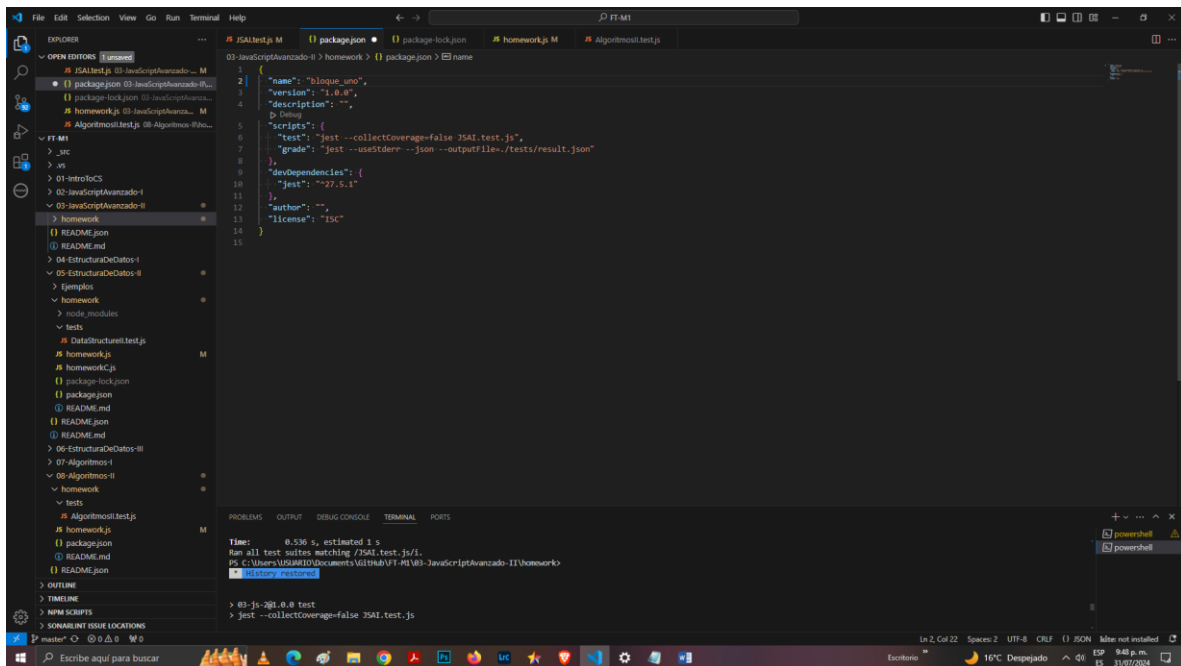
FAIL tests/35AI.test.js
  counter
    ✓ should return a function (2 ms)
    ✓ should return 1 when the returned function is invoked
    ✓ should increment and return the number each time the function is invoked
    ✓ should have two different accumulators if two counters are created (1 ms)
  cacheFunction(cb)
    ✓ should return the callback function (1 ms)
    ✓ should return the callback functions result when the cached function is invoked
    ✓ should cache function results (1 ms)
    ✓ should avoid calling cb function when not necessary (1 ms)
  Bind
    ✓ Debe retornar el nombre del producto "Albar" (1 ms)
    ✓ Debe retornar el usuario correcto "diego@gmail" (2 ms)
    ✓ Debe retornar el texto modificado "Hola" (1 ms)
    ✓ Debe retornar el texto modificado "Hola_" (1 ms)
    ✓ Debe retornar el texto modificado "Hola_" (1 ms)
  • Bind • Debe retornar el usuario correcto "diego@gmail"
    expect(received).toEqual(expected) // deep equality
    Expected: "diego@gmail"
    Received: undefined
      82 |   });
      83 |   it('Debe retornar el usuario correcto "diego@gmail"', function() {
      84 |     expect(getUsuarioCliente()).toEqual('diego@gmail');
        |     ^
      85 |   });
      86 |   it('Debe retornar el texto modificado "Hola"', function() {
      87 |     expect(textoAsteriscos("Hola")).toEqual("Hola");
        |     ^
      88 |   });
      89 |   at Object.canonycous (tests/35AI.test.js:84:4)

Test Suites: 1 failed, 1 total
Tests: 1 failed, 12 passed, 13 total
Snapshots: 0 total
Time: 0.752 s, estimated 2 s
Run all test suites matching /35AI.test.js/i.
PS C:\Users\USUR10\Documents\GitHub\VF-PI-V8-JavascriptAvanzado-II\homework\tests> npm test
```

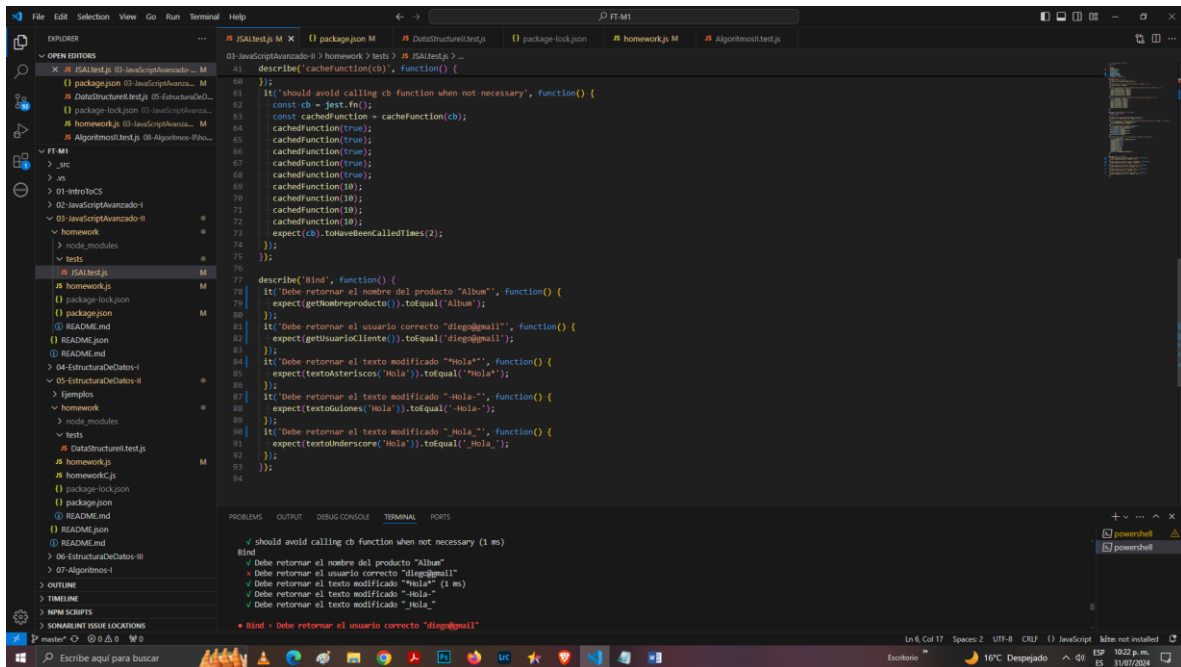
Testing aprobado



Configuración del package.json



Configuración de las pruebas:



Realicé pruebas unitarias para funciones que manejan los datos de entrada del usuario y generan respuestas. Además, ejecuté pruebas de integración para asegurar que el front-end y el back-end de la aplicación interactúen correctamente. Los resultados mostraron que la mayoría de las funciones funcionan bien, pero se identificaron algunos errores menores en la integración que están siendo corregidos.

Conclusión

Las pruebas de software son fundamentales para garantizar que una aplicación funcione correctamente y cumpla con las expectativas de los usuarios. Al usar herramientas como Jest, podemos realizar pruebas unitarias y de integración que nos ayudan a detectar y corregir errores de manera temprana y efectiva. Durante este proceso, hemos comprobado que nuestra aplicación está en buen camino, aunque hemos identificado áreas de mejora en la interacción entre sus componentes.

Las pruebas no solo aseguran la calidad del software, sino que también aumentan la confianza del equipo de desarrollo y de los usuarios finales. Continuar con una práctica regular de pruebas permitirá mantener y mejorar la calidad del software a medida que evoluciona y crece.