|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **DEPARTAMENTO:** | **Ciencias de la Computación** | **CARRERA:** | **Ingenieria de Software** | | |
| **ASIGNATURA:** | Pruebas de software | **NIVEL:** | 6to | **ASIGNATURA:** | Pruebas de software |
| **DOCENTE:** | Ing. Luis Castillo | **PRÁCTICA N°:** | 1 | **DOCENTE:** | Ing. Luis Castillo |

**Manejo de Pruebas Unitarias y CI/CD en Node.js con Express y GitHub Actions**

**Bryan Roberto Quispe Romero**

**RESUMEN**

Durante el laboratorio se desarrolló un proyecto base en Node.js utilizando Express para crear un servidor web simple. Se configuraron las dependencias necesarias, incluyendo Jest para la ejecución de pruebas unitarias y ESLint para el análisis estático del código. Se implementaron funciones básicas como sum, factorial y fibonacci, junto con sus pruebas automatizadas usando Jest.  
Además, se configuró un flujo de Integración Continua (CI) con GitHub Actions, que ejecuta automáticamente las pruebas y validaciones de código en cada cambio subido al repositorio. Como parte de la práctica, se introdujo un error intencional en una función para observar la falla del flujo CI y posteriormente se corrigió para verificar la ejecución exitosa.  
**Palabras Claves:** Node.js, Express, Jest, ESLint, GitHub Actions, CI/CD, Pruebas Unitarias.

1. **INTRODUCCIÓN:**

El objetivo de esta práctica fue fortalecer el manejo de entornos Node.js y aplicar el enfoque de desarrollo guiado por pruebas (TDD) junto con técnicas de integración continua. Se buscó fomentar la disciplina en la construcción de aplicaciones modernas mediante la correcta configuración de un servidor Express, la implementación de pruebas unitarias con Jest y el uso de GitHub Actions para automatizar la validación del código.  
El laboratorio permitió al estudiante comprender la importancia de implementar pruebas desde las primeras fases del desarrollo y mantener un flujo de CI/CD que garantice la calidad del software antes de llegar a producción.

1. **OBJETIVO(S):**

**Objetivo General:**

* Implementar un proyecto Node.js con pruebas unitarias y un flujo de CI/CD usando GitHub Actions.

**Objetivos Específicos:**

* Crear la estructura base del proyecto con Node.js y Express.
* Configurar Jest para ejecutar pruebas unitarias.
* Configurar ESLint para aplicar buenas prácticas de código.
* Implementar funciones y sus pruebas (sum, factorial, fibonacci).
* Configurar un flujo de CI/CD con GitHub Actions para automatizar pruebas y validaciones.
* Simular fallos en el flujo CI y corregirlos para asegurar la estabilidad del código.

1. **MARCO TEÓRICO:**

* **Node.js:** Entorno de ejecución para JavaScript que permite desarrollar aplicaciones del lado del servidor.
* **Express:** Framework minimalista de Node.js para crear servidores y manejar rutas HTTP.
* **Jest:** Framework de pruebas de JavaScript que permite definir, ejecutar y verificar pruebas unitarias.
* **ESLint:** Herramienta de análisis estático que ayuda a encontrar y corregir problemas en el código.
* **GitHub Actions:** Plataforma de automatización que permite crear flujos de trabajo (workflows) para ejecutar tareas como pruebas, compilaciones y despliegues automáticamente.
* **CI/CD:** Práctica que integra cambios de código de manera continua y despliega aplicaciones con frecuencia, asegurando calidad y rapidez.

1. **DESCRIPCIÓN DEL PROCEDIMIENTO:**

**PARTE 1: Establecimiento de la estructura del proyecto base**

**Crear la carpeta del proyecto y abrirla en el terminal.**

**Inicializar package.json:**

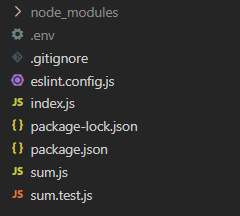
****

**Instalar Express:**

****

**Instalar Jest y ESLint como dependencias de desarrollo:**

****

****

**PARTE 2: Creación de archivos base**

**index.js**

* + **Se creó un servidor Express con un endpoint / que devuelve "Servidor funcionando".**
  + **Se levantó en el puerto 3000.**

**sum.js**

* + **Función que suma dos números y devuelve el resultado.**
  + **Exportada para su uso en pruebas.**

**sum.test.js**

* + **Prueba unitaria con Jest verificando la suma de dos números.**

**math.js**

* + **Funciones adicionales factorial y fibonacci.**

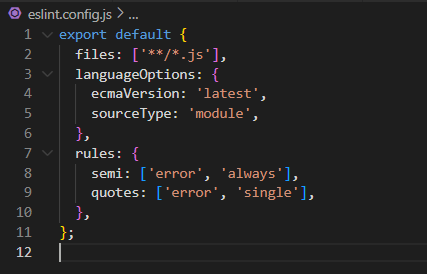
**math.test.js**

* + **Pruebas unitarias para factorial y fibonacci.**

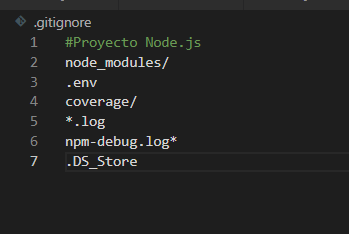
**package.json configurado con:**

****

**eslint.config.js con reglas básicas.**

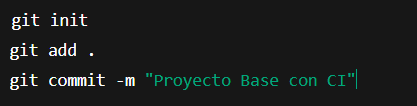
****

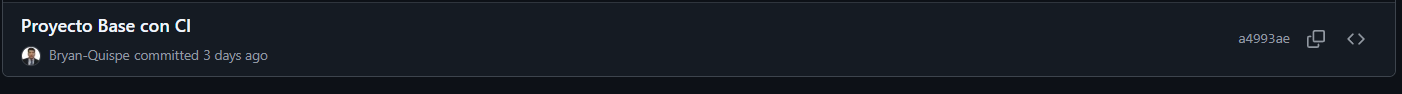
**.gitignore ignorando node\_modules/ y archivos temporales.**

****

**PARTE 3: Configuración de Git y subida al repositorio**

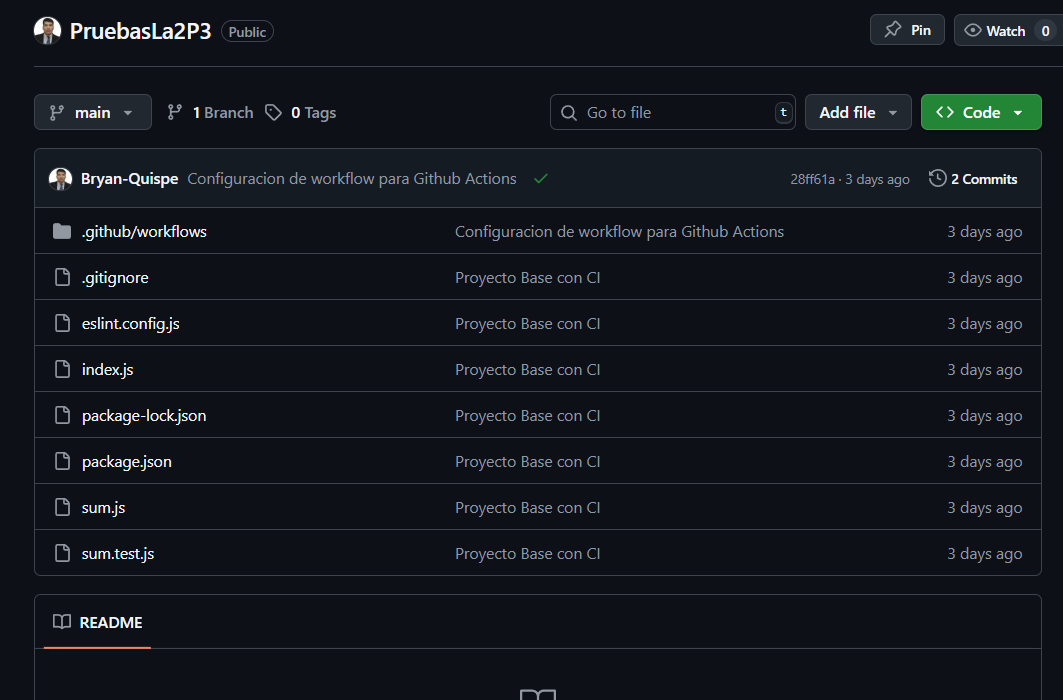
**Inicialización de Git en el proyecto local:**

****

****

**Conexión con el repositorio remoto creado en GitHub:**

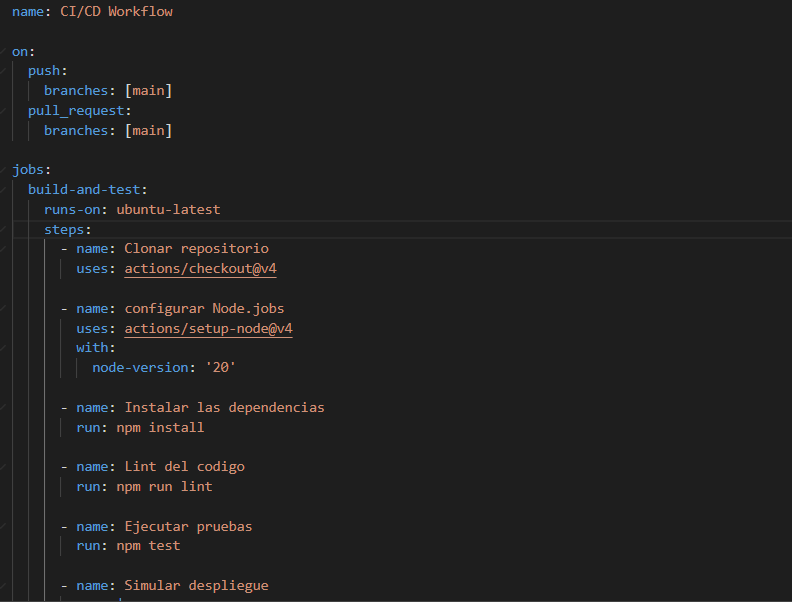
****

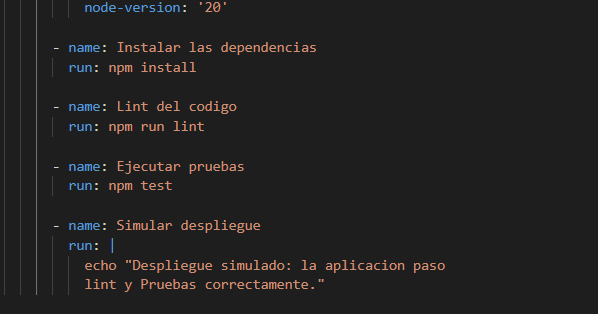
****

**PARTE 4: Configuración del workflow de GitHub Actions**

**Se creó la carpeta .github/workflows/ y dentro de ella el archivo ci.yml con la configuración del flujo CI/CD:**

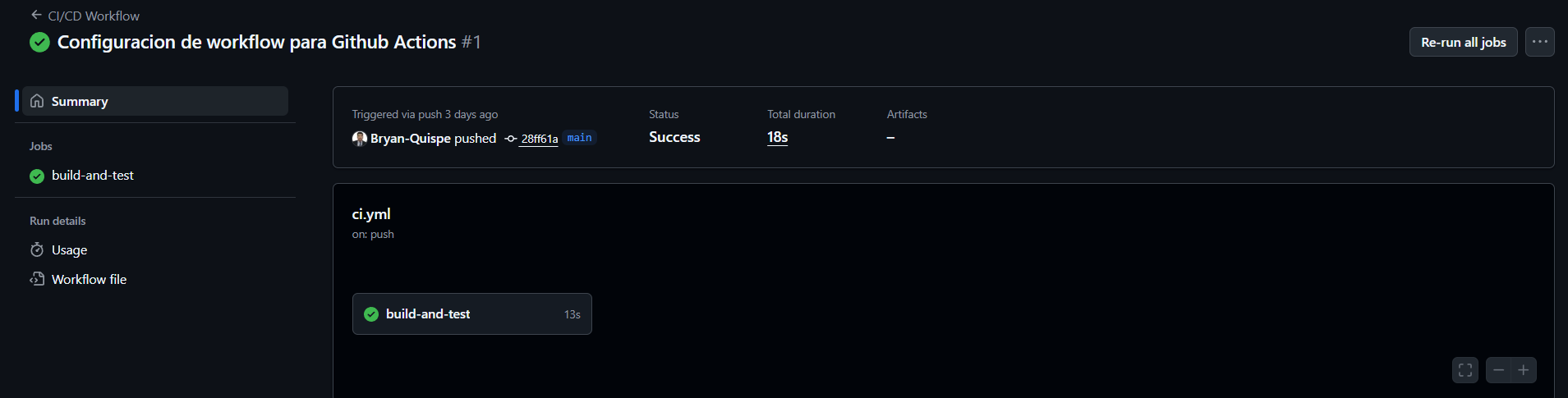
* **Instalación de dependencias.**
* **Ejecución de ESLint.**
* **Ejecución de Jest para las pruebas unitarias.**
* **Mensaje de despliegue simulado en caso de éxito**

****

****

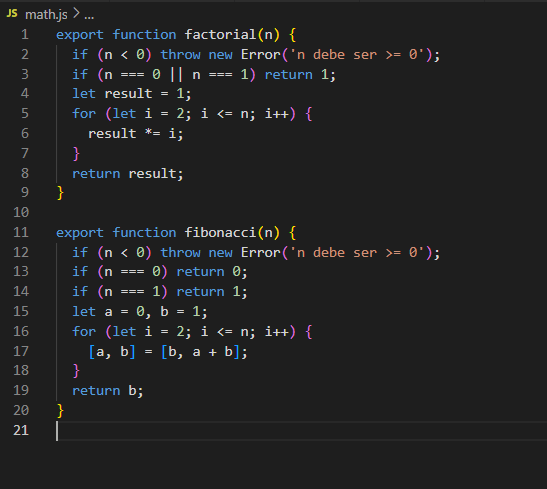
**Prueba del flujo CI/CD**

* **Se realizó un push al repositorio y se verificó en la pestaña Actions que el workflow se ejecutó correctamente.**
* **Se introdujo un error intencional en la función factorial() para que fallara el pipeline.**
* **Se corrigió el error y se volvió a ejecutar el flujo, confirmando su correcto funcionamiento.**

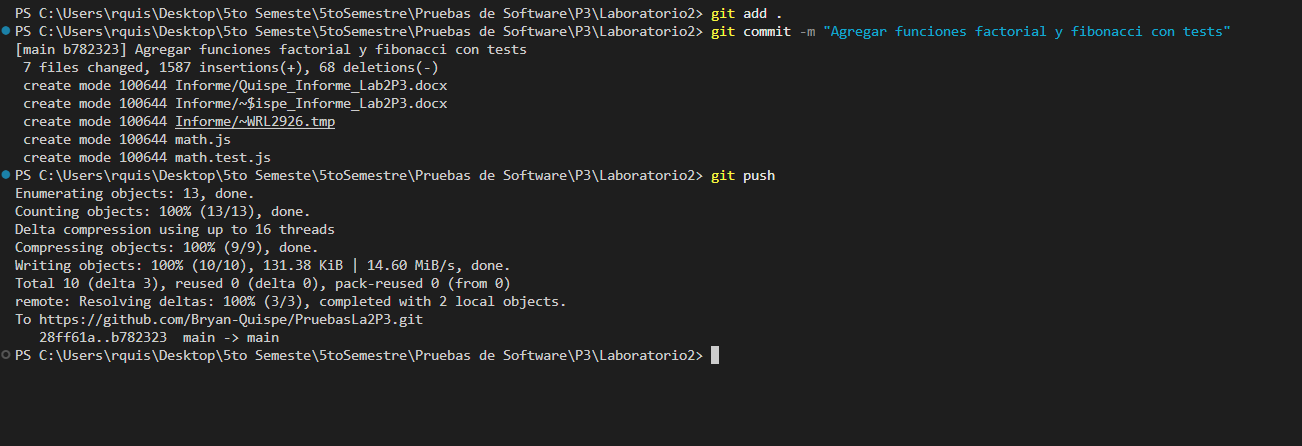
****

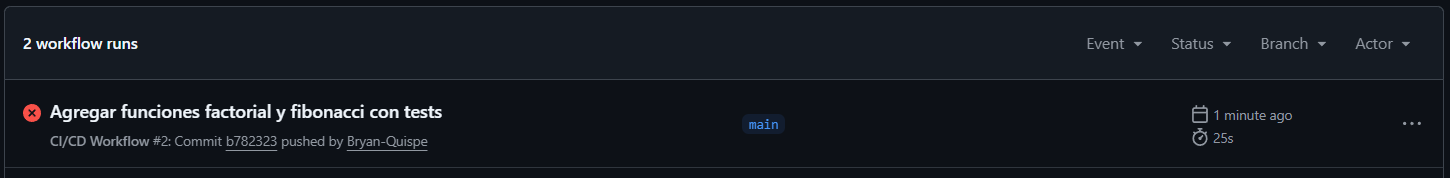
**SECCIÓN DE PREGUNTAS/ACTIVIDADES**

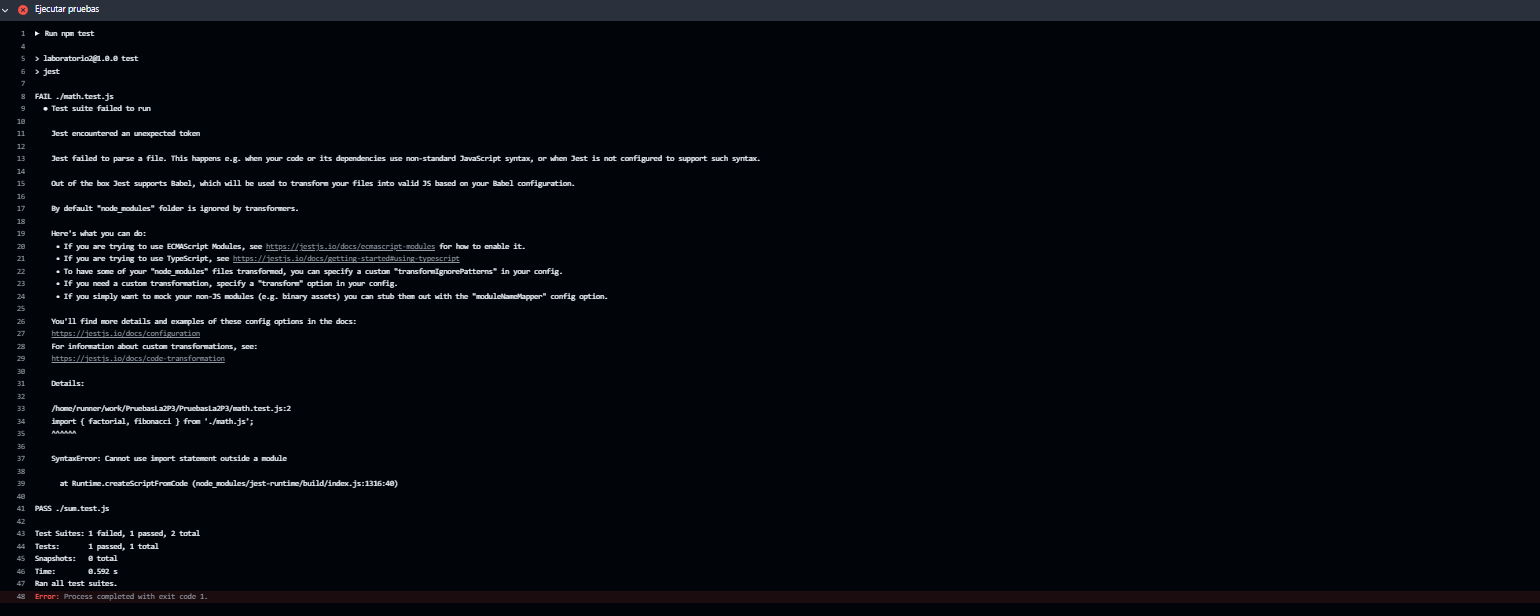
**Prueba falla**

****

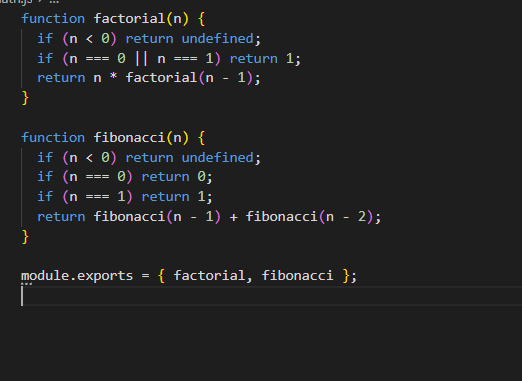
****

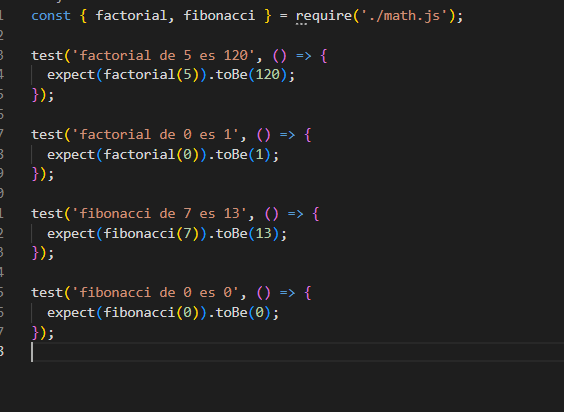
****

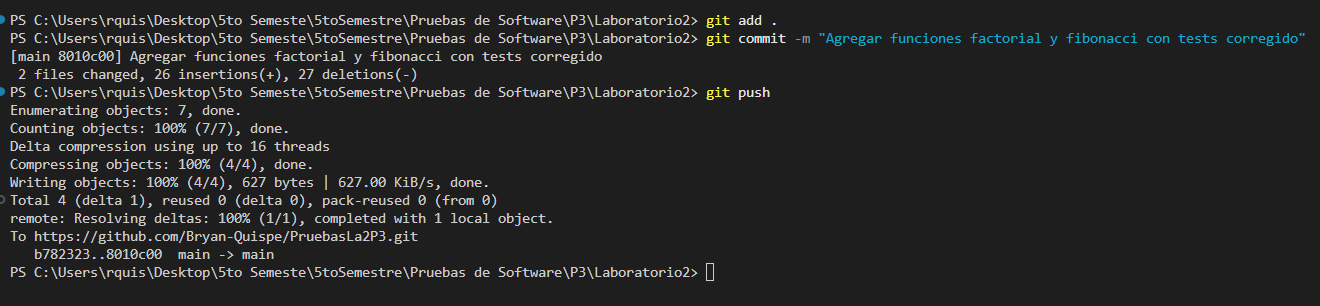
****

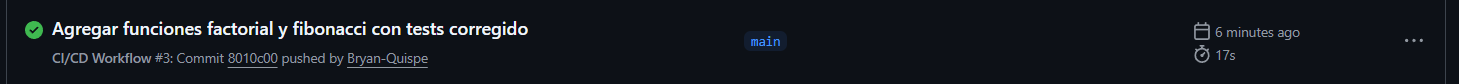
****

**Prueba Pasa**

****

****

****

****

1. **CONCLUSIONES:**

 GitHub Actions facilita la automatización del ciclo de integración y despliegue continuo, permitiendo detectar errores de forma temprana.

 Las pruebas unitarias son fundamentales para garantizar que las funciones implementadas (como factorial y fibonacci) cumplan con los resultados esperados.

 La correcta configuración del entorno de pruebas evita fallos innecesarios y asegura un flujo de desarrollo más estable y confiable.

1. **RECOMENDACIONES:**

 Usar **un estándar de módulos consistente**  
Evitar mezclar import/export con require/module.exports sin configurar correctamente el entorno. Esto previene errores de compatibilidad en Jest y en CI/CD.

 Ejecutar **pruebas localmente antes de subir cambios**  
Correr npm test antes de hacer git push reduce la probabilidad de que el flujo de GitHub Actions falle por errores evidentes.

 Configurar **el CI/CD con validaciones adicionales**  
Incluir análisis estático con herramientas como ESLint y validación de dependencias para detectar problemas antes de la fase de pruebas.

1. **BIBLIOGRAFÍA:**

 GitHub Actions Documentation: https://docs.github.com/actions

 Jest Testing Framework: <https://jestjs.io/docs>

 Node.js Modules Documentation: https://nodejs.org/api/modules.html