|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **DEPARTAMENTO:** | **Ciencias de la Computación** | **CARRERA:** | **Ingeniería en Software** | | |
| **ASIGNATURA:** | Pruebas de Software | **NIVEL:** | 6to | **FECHA:** | 10/08/2025 |
| **DOCENTE:** | Ing. Luis Castillo | **PRÁCTICA N°:** | 6 | **CALIFICACIÓN:** |  |

**CI-CD usando GitHub Actions**

**Elkin Andres Pabón González**

**RESUMEN**

La presente práctica tuvo como objetivo implementar un flujo de Integración Continua (CI) en una aplicación Node.js utilizando GitHub Actions. Se configuró un repositorio en GitHub y se desarrolló un servidor básico con Express, junto con funciones y pruebas unitarias usando Jest. Se aplicó análisis estático de código mediante ESLint para garantizar la calidad y buenas prácticas de programación. El flujo de CI se configuró para ejecutarse automáticamente ante cada push o pull request a la rama principal, validando la instalación de dependencias, la ejecución de pruebas y la verificación de estilo. Además, se agregaron funciones adicionales para ampliar la cobertura de pruebas y se provocaron errores intencionales para comprobar el correcto funcionamiento del flujo de integración continua. Finalmente, se evidenció la importancia de la detección temprana de errores y de la automatización en el ciclo de desarrollo, mejorando la eficiencia y la confiabilidad del software.

**Palabras Claves:** Integración Continua, GitHub Actions, Node.js

1. **INTRODUCCIÓN:**

La presente práctica se enfoca en la implementación de un flujo de Integración Continua (CI) utilizando GitHub Actions en un proyecto Node.js. Esta actividad permite automatizar tareas como la instalación de dependencias, la ejecución de pruebas unitarias con Jest y el análisis estático de código con ESLint, garantizando así la calidad y la estabilidad del software. Durante el desarrollo, se fomentó el manejo responsable de los recursos del laboratorio y el cumplimiento de las normas establecidas para el uso del equipo y del entorno de trabajo. Además, se incentivó la disciplina en el versionamiento del código mediante el uso de Git y GitHub, asegurando un control preciso de cambios y facilitando la colaboración. La práctica busca reforzar competencias técnicas y organizativas esenciales para entornos de desarrollo profesional.

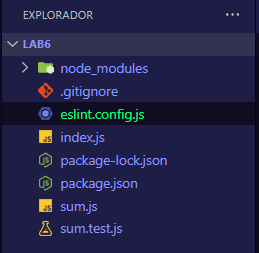
1. **OBJETIVO(S):**
   1. Implementar un flujo de Integración Continua en un proyecto Node.js utilizando GitHub Actions para automatizar tareas esenciales del desarrollo.
   2. Configurar workflows que se activen automáticamente en cada actualización del repositorio, verificando dependencias, ejecución de pruebas y análisis de código.
   3. Desarrollar y ejecutar pruebas unitarias con Jest que validen el correcto funcionamiento de las funciones y detecten errores tempranamente.
   4. Aplicar análisis estático de código con ESLint para mantener estándares de calidad y reforzar buenas prácticas de programación.
2. **MARCO TEÓRICO:**

* **Integración Continua (CI):** Es una práctica de desarrollo de software que consiste en integrar el código de forma frecuente en un repositorio compartido, ejecutando automáticamente pruebas y validaciones para detectar errores de manera temprana. Su objetivo es mantener la calidad y estabilidad del producto durante todo el ciclo de desarrollo.
* **Entrega Continua (CD):** Es la extensión de la integración continua que permite que el software esté siempre listo para ser desplegado en producción. Automatiza las fases posteriores a la integración, como el empaquetado y la liberación del software, reduciendo tiempos y riesgos en el despliegue.
* **GitHub Actions:** Es una herramienta de automatización integrada en GitHub que permite ejecutar flujos de trabajo (workflows) definidos en archivos YAML. Estos flujos pueden ejecutarse en respuesta a eventos como commits, pull requests o programación programada, y permiten automatizar pruebas, compilaciones, despliegues y otras tareas.
* **Node.js:** Es un entorno de ejecución de JavaScript del lado del servidor basado en el motor V8 de Google Chrome. Permite construir aplicaciones escalables y de alto rendimiento, siendo ampliamente utilizado en proyectos que requieren rapidez y concurrencia.
* **Jest:** Es un framework de pruebas para JavaScript que facilita la creación y ejecución de pruebas unitarias. Ofrece funcionalidades como aserciones, mocks y medición de cobertura de código, siendo muy utilizado en proyectos Node.js y React.
* **ESLint:** Es una herramienta de análisis estático para JavaScript que permite identificar y corregir patrones problemáticos o no conformes a las reglas de estilo definidas. Ayuda a mantener un código limpio, consistente y libre de errores comunes.

1. **DESCRIPCIÓN DEL PROCEDIMIENTO:**

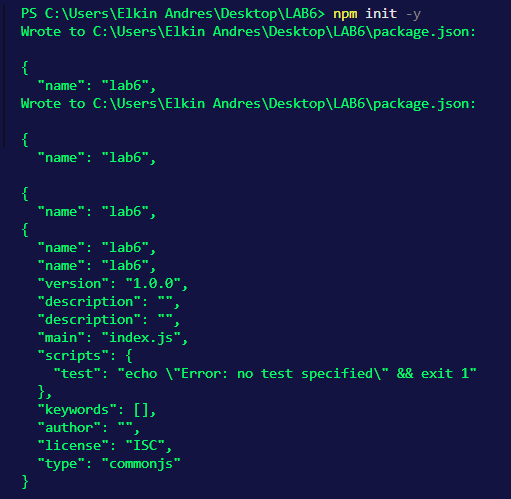
**PARTE 1: Establecimiento de la estructura del proyecto base**

**Paso 1: Creación de la estructura básica.**

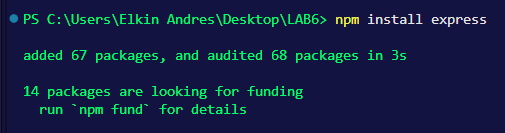


**Paso 2: Instalación de dependencias necesarias.**

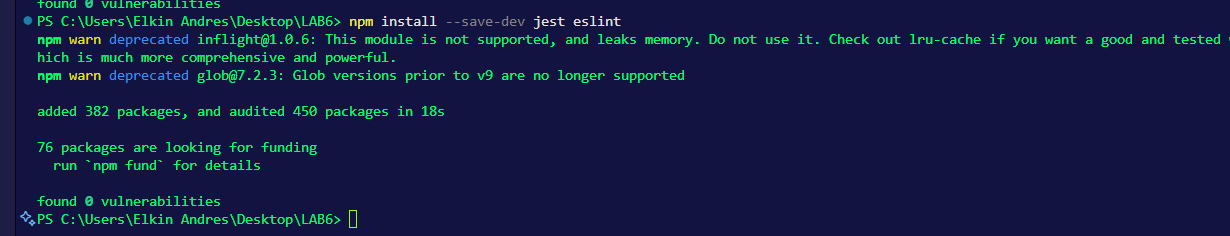
1. Creamos el archivo package.json para cargar las dependencias npm init -y



1. Instalamos la dependencia de Express npm install express



1. Instalamos las dependencias de Jest y ESLint npm install --save-dev jest eslint para que se puedan ejecutar en modo desarrollador



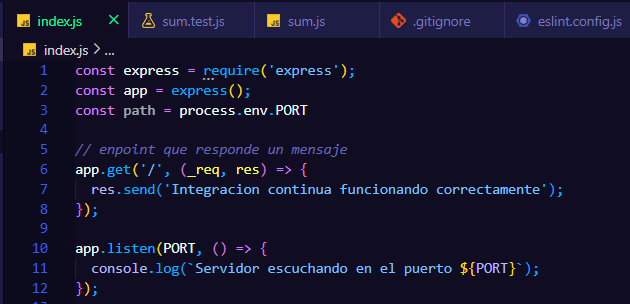
**PARTE 2: Creación de archivos base**

**Paso 1: Crear archivo index.js.**

a. Usar el servidor express

b. Implementar un endpoint sencillo que responda con un mensaje

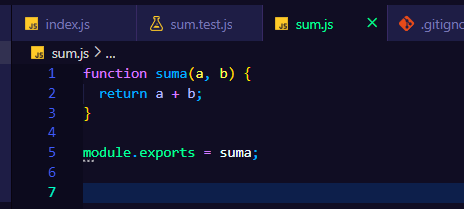
c. Levantar el servidor en el puerto 3000



**Paso 2: Crear archivo sum.js.**

a. Crear una función que sume dos números pasados como parámetros

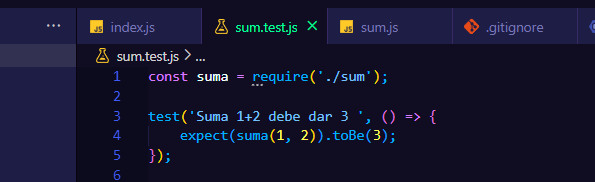
b. Exportar la función.



**Paso 3: Crear archivo sum.test.js.**

a. Usar el archivo con la función de suma

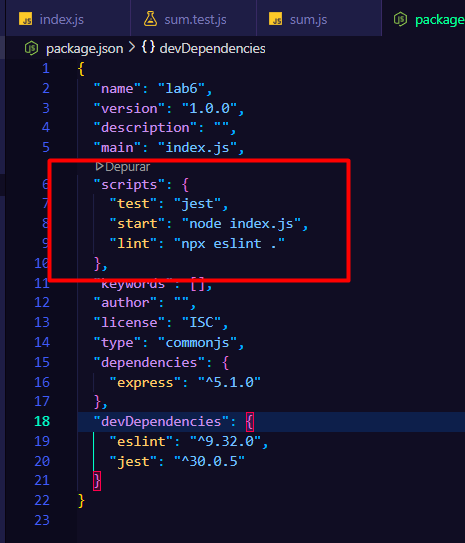
b. Crear una prueba para la función de suma.



**Paso 4: Configurar package.json.**

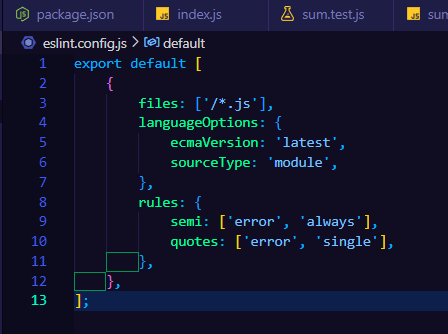
a. Agregar o editar los scripts para start, test y lint

b. Agregar la característica type para que ESLint funcione como módulo.



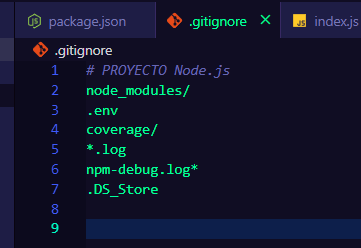
**Paso 5: Crear el archivo ESLint.**

1. Trabajar con reglas sencillas



**Paso 6: Ignorar node\_modules.**

1. En el archivo .gitignore ignorar todos los archivos que puedan causar conflictos para un proyecto NodeJS

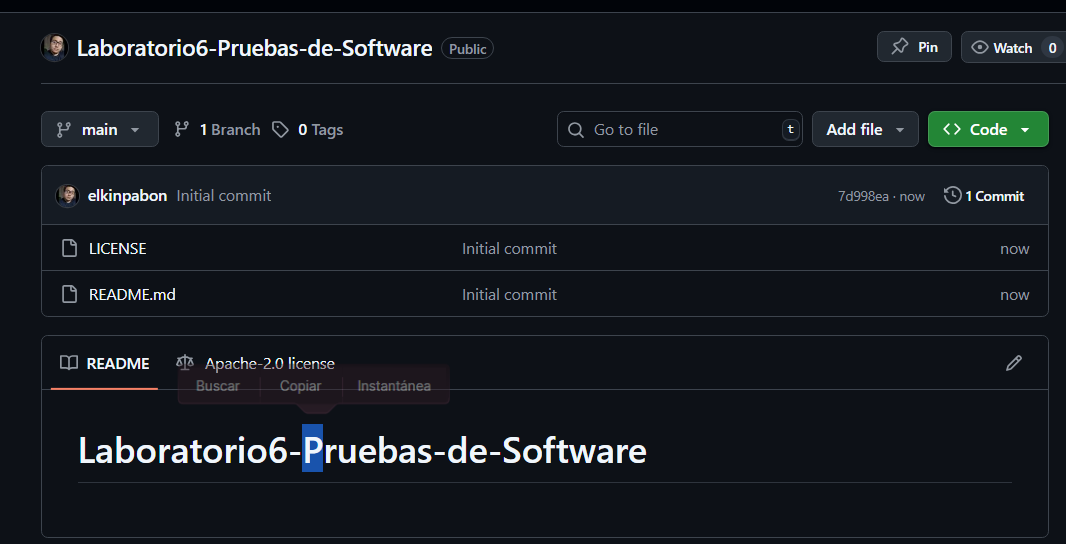


**PARTE 3: Configuración de Git**

**Paso 1: Crear repositorio en la cuenta de Git.**

a. Abrir la cuenta de Git en el navegador

b. Crear un nuevo repositorio vacío

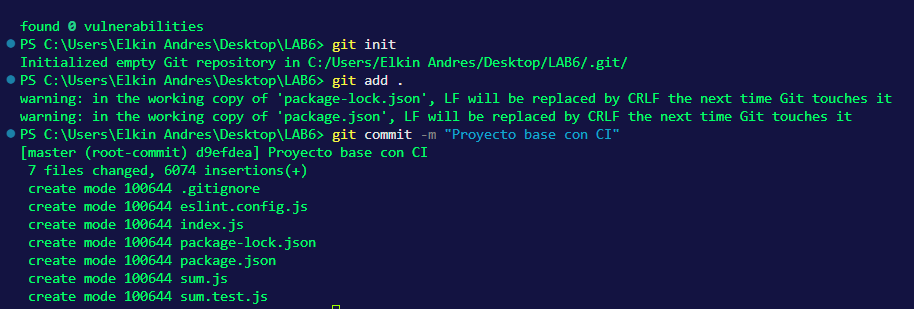


**Paso 2: Ejecución de comandos para clonar al repositorio.**

a. git init

b. git add .

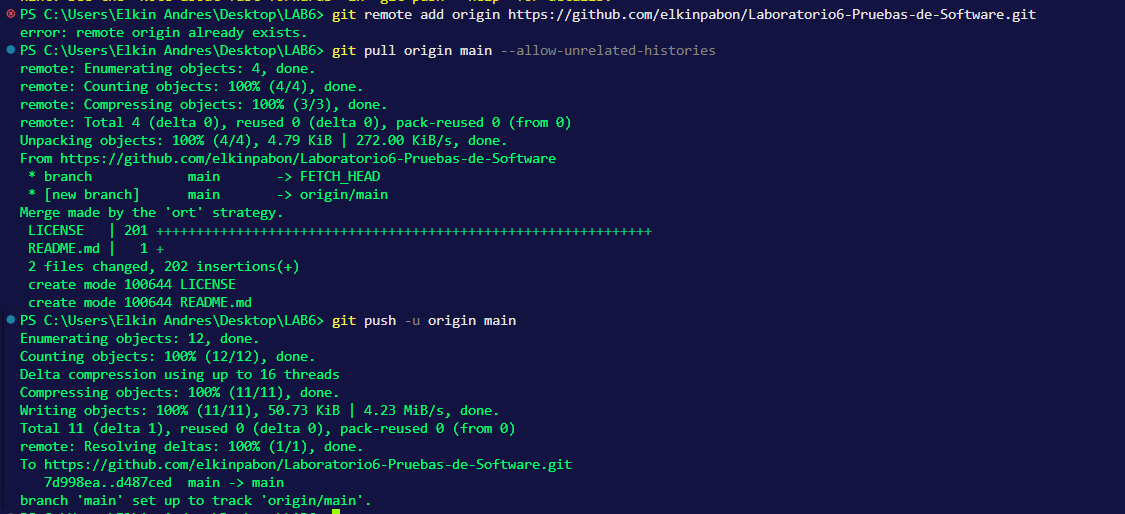
c. git commit -m "Proyecto base con CI"

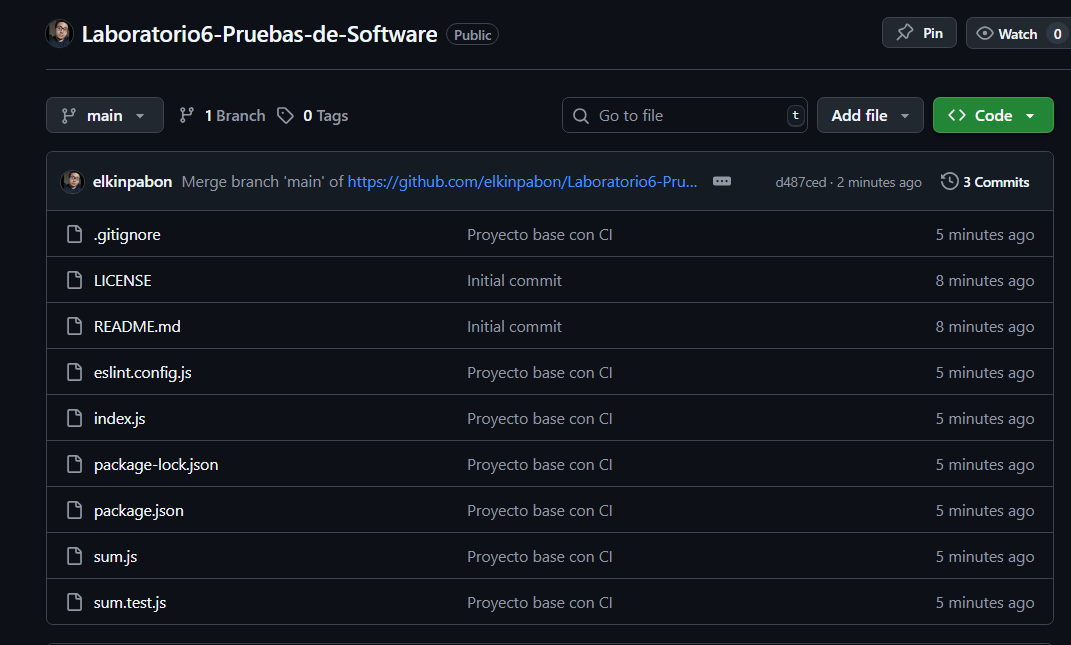


d. git branch -M main

e. git remote add origin https://github.com/TU\_USUARIO/nombreRepositorio.git

f. git push -u origin main





**Paso 3: Crear el workflow de GitHub Actions.**

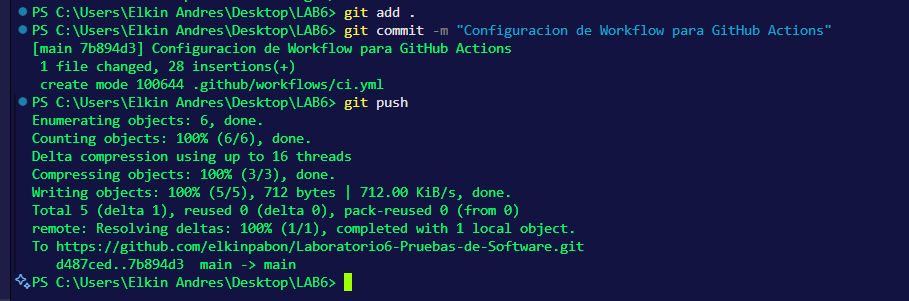
a. Crear un archivo nuevo para el workflow .github/workflows/ci.yml.

b. Configurar los triggers.

c. Configurar los trabajos a realizar

d. Configurar dentro de los trabajos los pasos a ejecutarse.



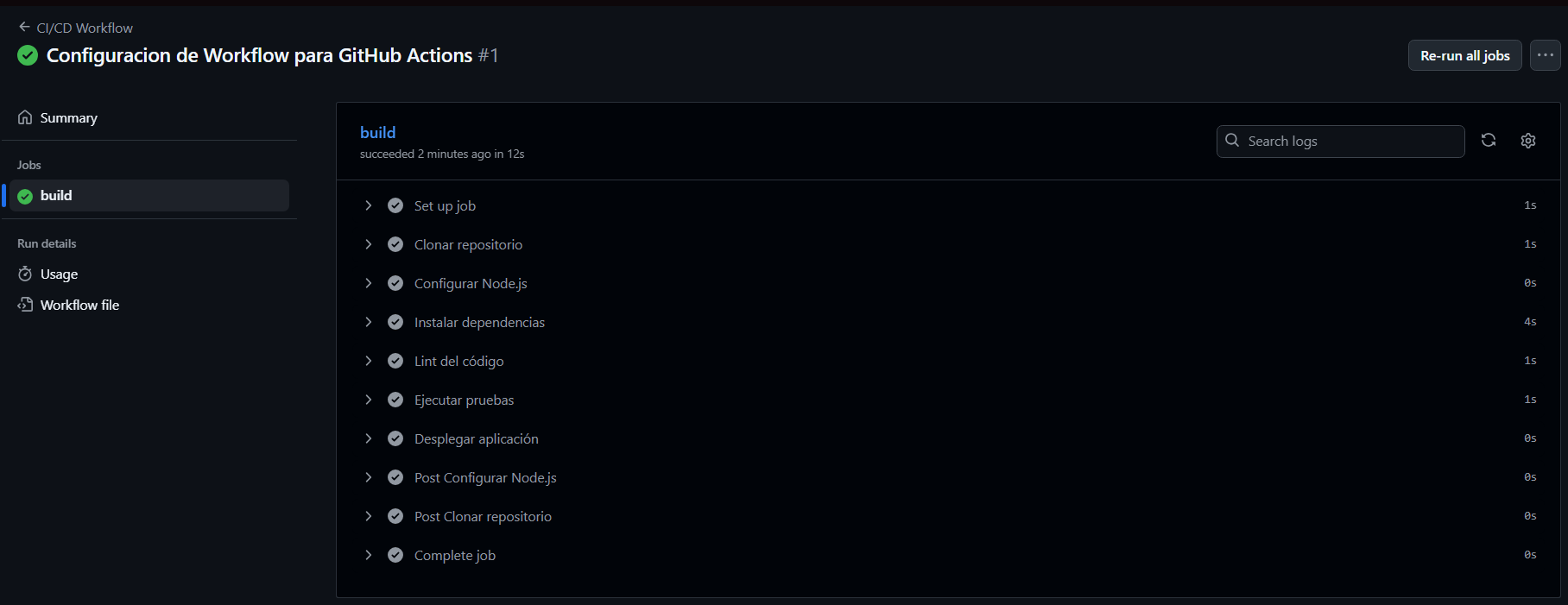


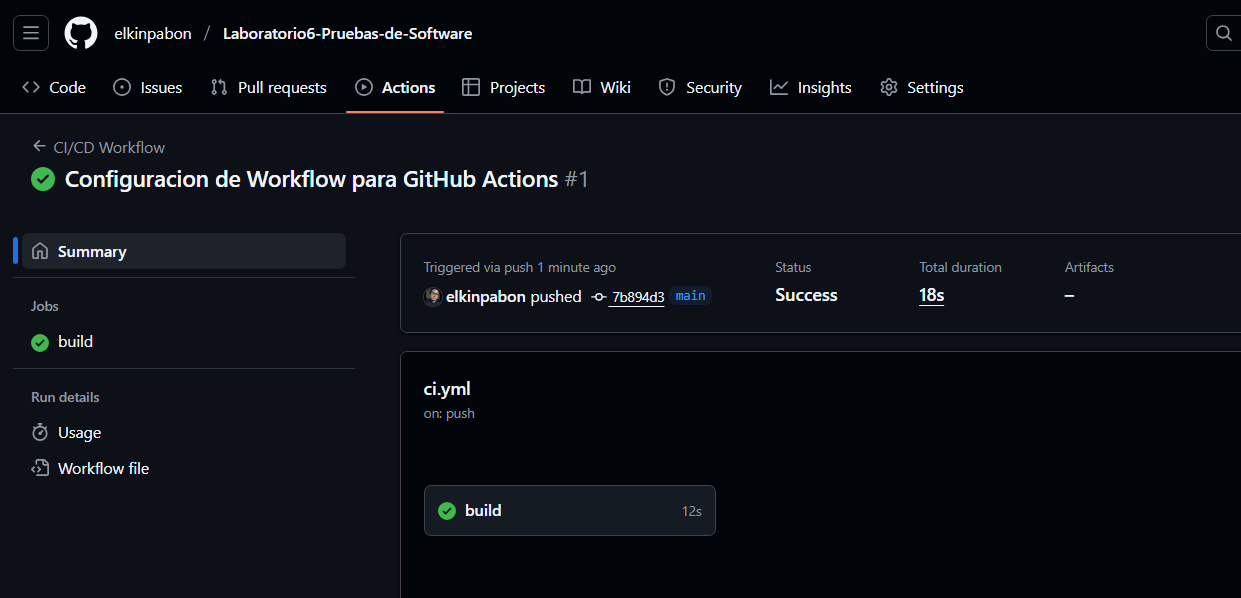
**Paso 4: Probar la CI.**

a. Realizar un cambio al código.

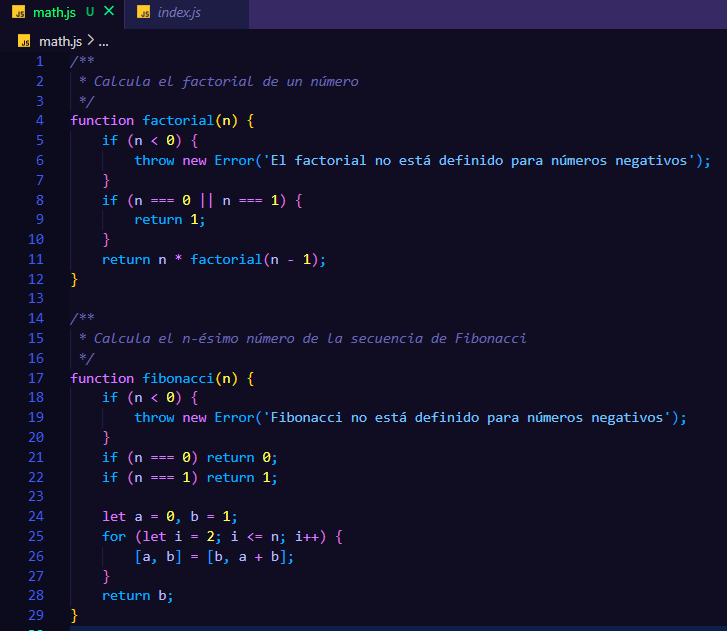
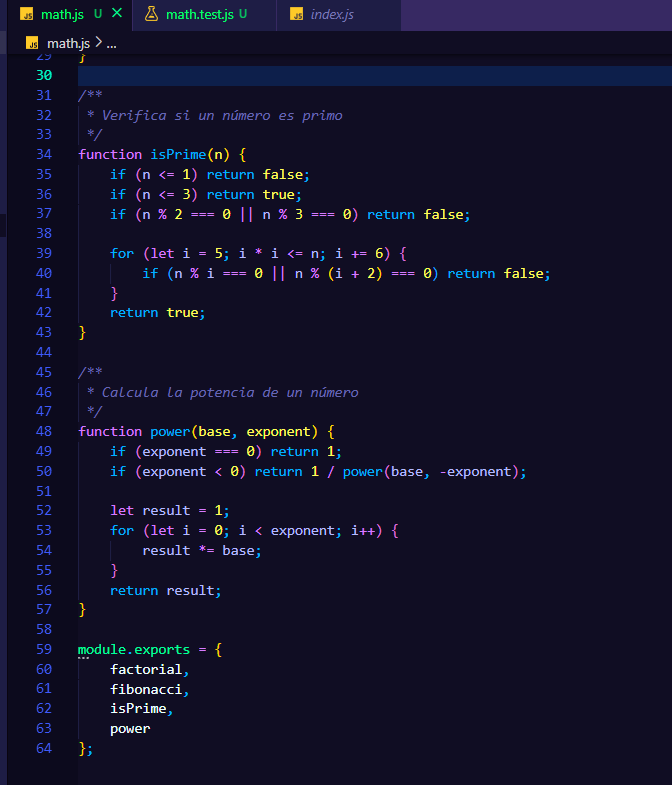
b. Ejecutar de nuevo los comandos para realizar un nuevo push.

c. Revisar en GitHub dentro del repositorio, en la pestaña Actions, como se ejecutan los Worflows

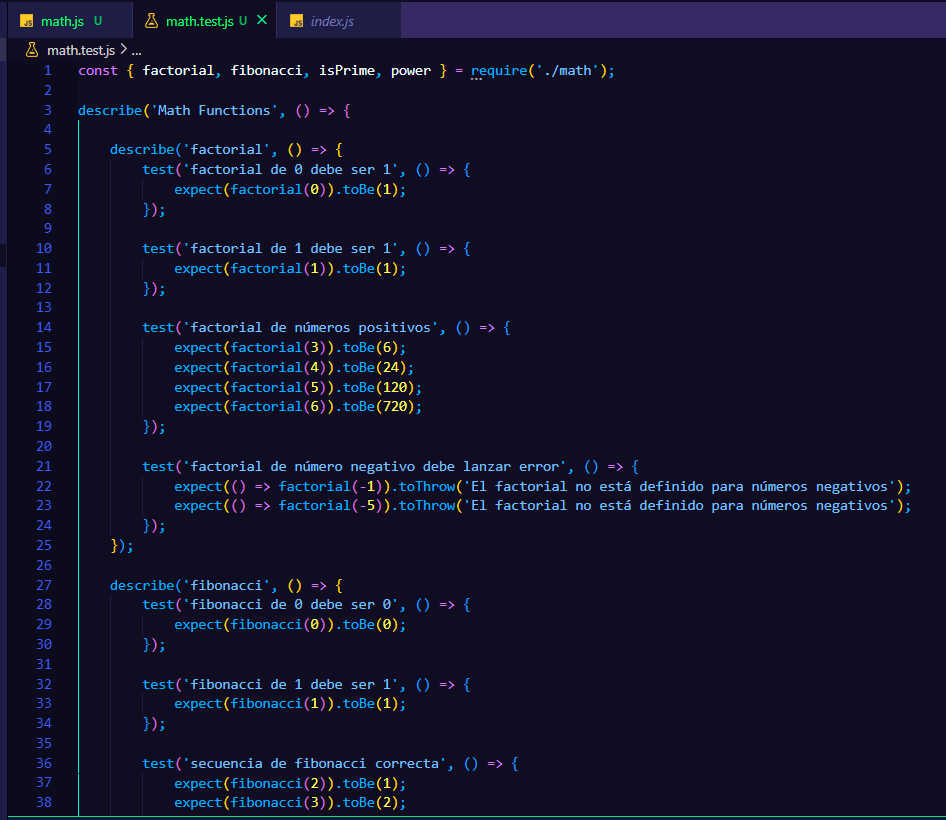




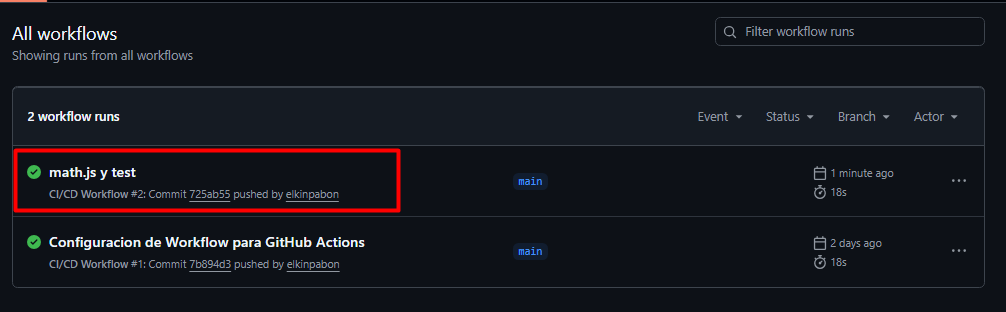
1. **PREGUNTAS/ACTIVIDADES:**
2. **Agregar más pruebas unitarias**
   1. Agregar al menos 2 funciones nuevas (por ejemplo, factorial, fibonacci) en un archivo math.js.

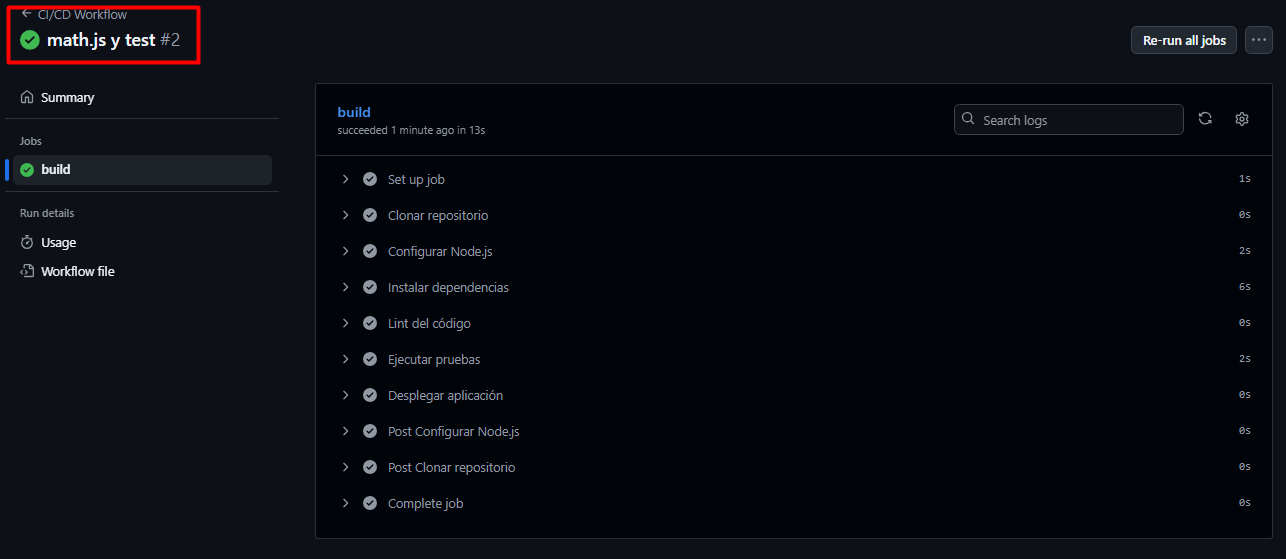
 

* 1. Crear su correspondiente archivo math.test.js con pruebas Jest.

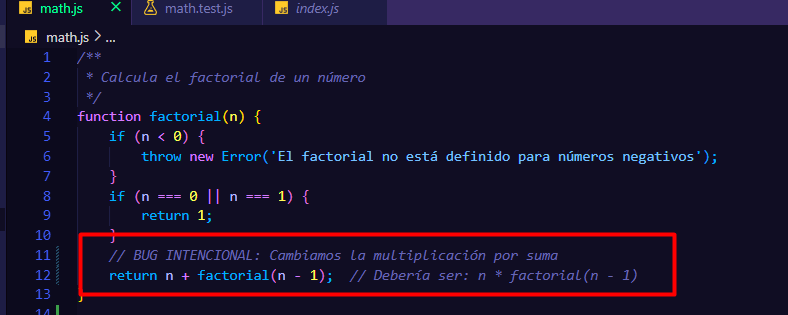


* 1. Asegurarse de que GitHub Actions ejecute todas las pruebas con éxito.

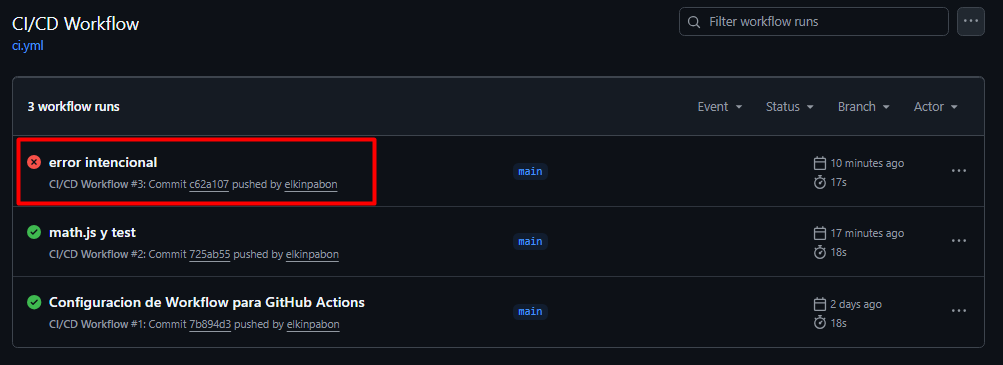


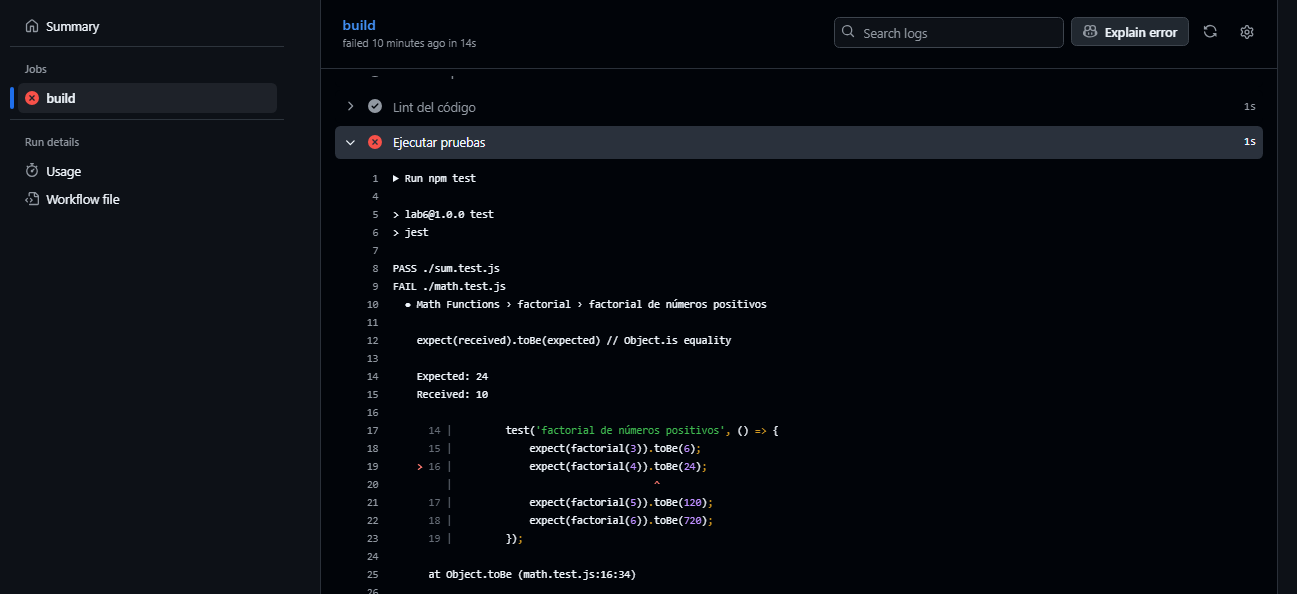


1. **Provocar un error intencional y corregirlo**
   1. Modificar cualquier función o el test de alguna de ellas para que falle intencionalmente.

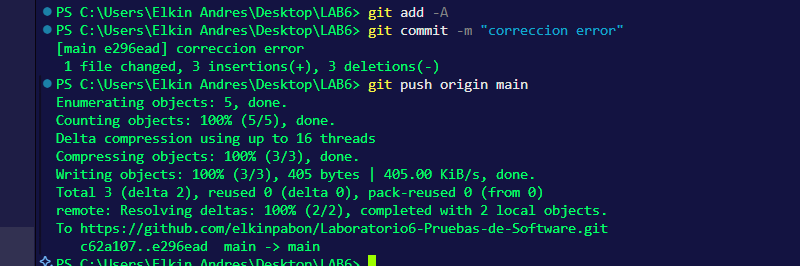


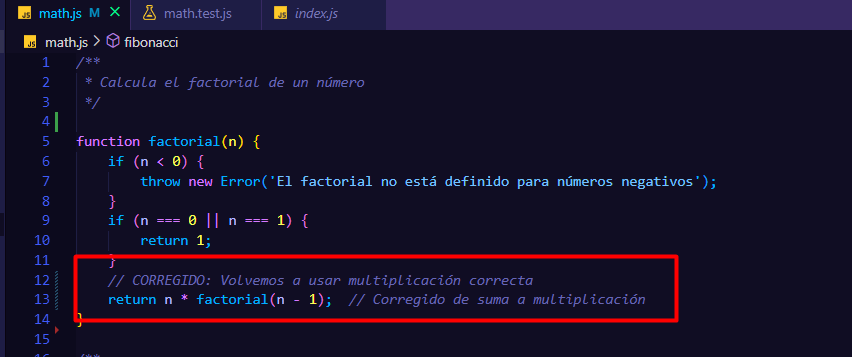
* 1. Subir los cambios y verificar que el flujo CI falla.



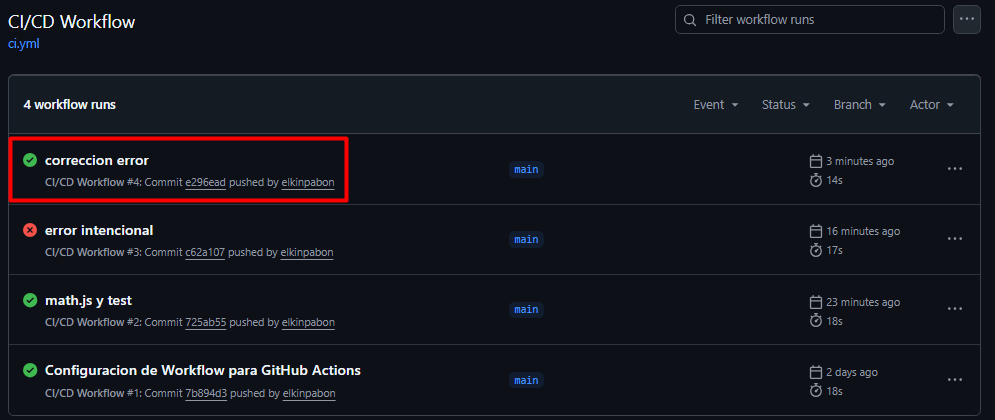


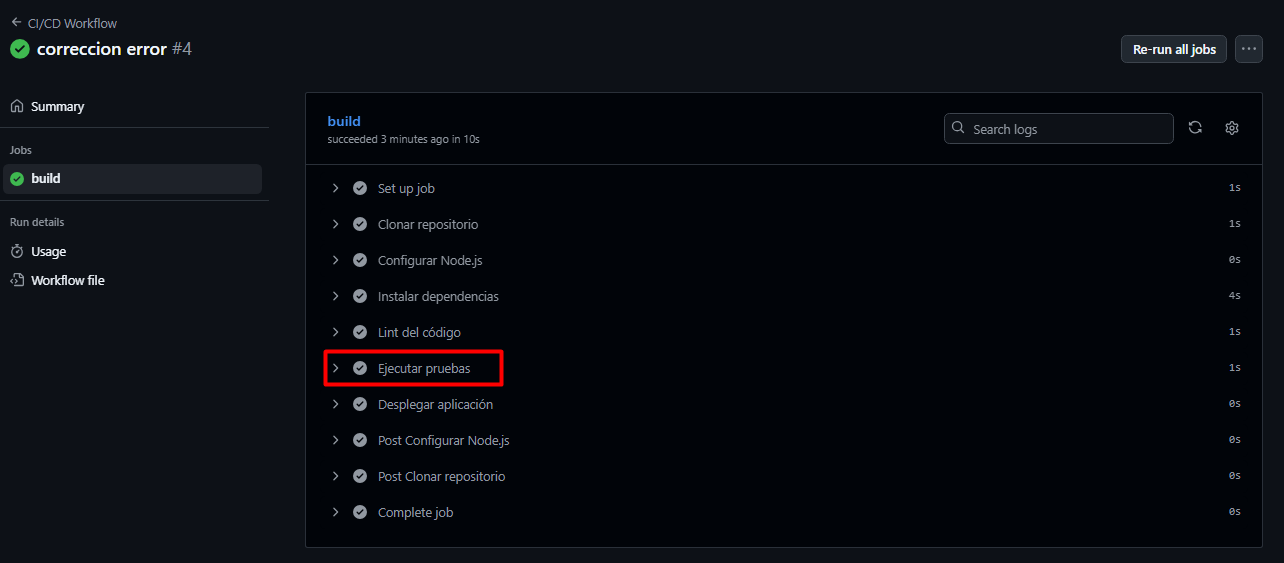
* 1. Corregir el error y volver a subir.





* 1. Adjuntar captura de ambas ejecuciones en GitHub Actions





1. **CONCLUSIONES:**

* La implementación de un flujo de Integración Continua mediante GitHub Actions demostró ser una herramienta eficaz para automatizar tareas críticas del ciclo de desarrollo, como la instalación de dependencias, la ejecución de pruebas unitarias y el análisis estático de código, lo que permitió reducir significativamente la intervención manual y minimizar el riesgo de errores humanos.
* La integración de Jest como framework de pruebas permitió validar el correcto funcionamiento de las funciones implementadas en cada actualización del repositorio. Esto aseguró que las nuevas funcionalidades no afectaran negativamente las ya existentes, reforzando la estabilidad y confiabilidad del sistema.
* El uso de ESLint como herramienta de análisis estático contribuyó a mantener un código consistente y libre de errores comunes, promoviendo el cumplimiento de estándares de calidad y buenas prácticas de programación. Esto repercute directamente en la mantenibilidad del software a largo plazo.
* La práctica permitió comprender la importancia de incorporar CI en los proyectos de desarrollo moderno, sentando las bases para una posterior implementación de Entrega Continua (CD) que optimizaría aún más los tiempos de despliegue y la calidad final del producto.

1. **RECOMENDACIONES:**

* Mantener actualizados los archivos de configuración del flujo de CI y las dependencias del proyecto para garantizar compatibilidad con nuevas versiones de librerías y herramientas, evitando fallos inesperados durante la ejecución de los workflows.
* Ampliar la cobertura de pruebas unitarias incluyendo casos de prueba que contemplen escenarios límite, datos atípicos y condiciones de error, con el fin de robustecer la validación del sistema y anticipar fallos potenciales en producción.
* Integrar notificaciones automáticas en GitHub Actions, como mensajes a correo electrónico o plataformas de mensajería (Slack, Teams), para recibir alertas inmediatas ante fallos, permitiendo una respuesta más rápida y eficiente del equipo de desarrollo.
* Considerar la incorporación progresiva de prácticas de Entrega Continua (CD) y despliegue automatizado, vinculando el flujo actual de CI con entornos de staging o producción controlados, para reducir el tiempo entre el desarrollo y la entrega final del software.

1. **BIBLIOGRAFÍA:**

* Fowler, M. 2020. Continuous Integration. ThoughtWorks, Addison-Wesley Professional. Disponible en: https://martinfowler.com/articles/continuousIntegration.html. Fecha de consulta: 09 de agosto de 2025.
* Kim, G., Humble, J., Debois, P., Willis, J. 2021. The DevOps Handbook. IT Revolution Press. Páginas 45–87. Fecha de consulta: 09 de agosto de 2025.
* Turnbull, J. 2014. The Node Beginner Book. Leanpub. Disponible en: https://leanpub.com/nodebeginner. Fecha de consulta: 09 de agosto de 2025.
* Raj, P., Seybold, C. 2018. Practical Node.js: Building Real-World Scalable Web Apps. Apress. Páginas 112–134. Fecha de consulta: 09 de agosto de 2025.