|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **DEPARTAMENTO:** | **CIENCIAS DE LA COMPUTACIÓN** | **CARRERA:** | **SOFTWARE** | | |
| **ASIGNATURA:** | PRUEBAS DE SOFTWARE | **NIVEL:** | 6to | **FECHA:** | 10-08-2025 |
| **DOCENTE:** | Ing. Luis Castillo, Mgtr. | **PRÁCTICA N°:** | 2 – 3P | **CALIFICACIÓN:** |  |

**CI/CD usando GitHub Actions**

**Anthony Néstor Villarreal Macías**

**RESUMEN**

En esta práctica de laboratorio, se configuró un flujo de integración continua (CI) con GitHub Actions para una aplicación Node.js. Se desarrolló un proyecto base con Express, pruebas unitarias con Jest y análisis estático con ESLint. El objetivo fue automatizar la instalación de dependencias, pruebas y verificación de código. Se añadieron funciones factorial y Fibonacci con sus pruebas en Jest, y se provocó un error intencional para verificar el flujo CI, corrigiendo posteriormente el fallo. Las capturas de pantalla muestran las ejecuciones exitosa y fallida. GitHub Actions optimizó la detección temprana de errores, mejorando la calidad del software. La experiencia reforzó la importancia de automatizar procesos para mantener un desarrollo eficiente y confiable, destacando la facilidad de integración de herramientas modernas.

**Palabras Claves:** CI/CD, GitHub Actions, Pruebas Unitarias.

1. **INTRODUCCIÓN:**

La integración continua (CI) es una práctica clave en el desarrollo de software moderno que permite automatizar procesos para garantizar la calidad del código. Este laboratorio busca familiarizar al estudiante con la configuración de flujos CI/CD mediante GitHub Actions, utilizando una aplicación Node.js. Se realizaron actividades como la creación de pruebas unitarias con Jest, análisis estático con ESLint y simulación de despliegue automatizado, respetando las buenas prácticas de programación y la disciplina en el laboratorio.

1. **OBJETIVO(S):**
   1. Describir los alcances o metas de la práctica
      1. Configurar un flujo de integración continua en GitHub Actions que se active con cada push o pull request.
      2. Implementar pruebas unitarias con Jest para validar la lógica del sistema.
      3. Aplicar análisis estático con ESLint para garantizar buenas prácticas de codificación.
      4. Simular un proceso de despliegue automatizado, enfocándose en las etapas previas al despliegue continuo.
2. **MARCO TEÓRICO:**

GitHub Actions es una herramienta de automatización que permite crear flujos de trabajo (workflows) para CI/CD. Ejecuta tareas como instalación de dependencias, pruebas y análisis de código en respuesta a eventos en el repositorio, como push o pull requests. Jest es un framework de pruebas unitarias para JavaScript, conocido por su simplicidad y soporte para pruebas asíncronas. ESLint es una herramienta de linting que analiza el código estáticamente para detectar errores y asegurar consistencia en el estilo de programación. Express es un framework minimalista para Node.js que facilita la creación de APIs. Estas herramientas, combinadas, permiten desarrollar software robusto y automatizado.

1. **DESCRIPCIÓN DEL PROCEDIMIENTO:**

Se creó un proyecto Node.js con Express, configurando un servidor básico en el puerto 3000. Se implementó una función de suma en sum.js y pruebas unitarias en sum.test.js usando Jest. Se configuró ESLint con reglas básicas y se ignoró node\_modules en ‘.gitignore’. Se inicializó un repositorio Git, se subieron los cambios y se configuró un flujo en .github/workflows/ci.yml para ejecutar pruebas y linting automáticamente. Se añadieron funciones factorial y Fibonacci en math.js con pruebas en math.test.js. Finalmente, se provocó un error intencional en una prueba, se verificó el fallo en GitHub Actions, y se corrigió.

1. **PREGUNTAS/ACTIVIDADES:**

* Agregar más pruebas unitarias: Se implementaron funciones factorial y fibonacci en math.js. Se crearon pruebas en math.test.js para validar casos como factorial(5) = 120 y fibonacci(6) = 8. GitHub Actions ejecutó todas las pruebas con éxito, como se muestra en las capturas.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

* Provocar un error intencional y corregirlo: Se modificó la prueba de sum.test.js para esperar un resultado incorrecto (sum(2,2) = 5). El flujo CI falló, como se evidencia en la captura de GitHub Actions. Posteriormente, se corrigió la prueba (sum(2,2) = 4) y el flujo se ejecutó correctamente.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

1. **RESULTADOS OBTENIDOS:**

El proyecto se configuró exitosamente con un flujo CI/CD en GitHub Actions que automatizó dependencias, pruebas y linting. Las capturas muestran la ejecución exitosa y el fallo intencional. A continuación, se presentan los archivos de código actualizados:

* **.gitignore:**

**Texto

El contenido generado por IA puede ser incorrecto.**

* **Eslint.config.js:**

**Pantalla de computadora con letras

El contenido generado por IA puede ser incorrecto.**

* **Package.json:**

**Texto

El contenido generado por IA puede ser incorrecto.**

* **Index.js:**

**Texto

El contenido generado por IA puede ser incorrecto.**

* **Sum.js:**

**Pantalla de computadora con números

El contenido generado por IA puede ser incorrecto.**

* **Sum.test.js:**

**Texto

El contenido generado por IA puede ser incorrecto.**

* **Math.js:**

**Texto

El contenido generado por IA puede ser incorrecto.**

* **Math.test.js:**

**Texto

El contenido generado por IA puede ser incorrecto.**

* **.github/workflows/ci.yml:**

**Texto

El contenido generado por IA puede ser incorrecto.**

* **.env:**

**Icono

El contenido generado por IA puede ser incorrecto.**

* **GITHUB**

Captura de pantalla con la imagen de una pantalla

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

1. **CONCLUSIONES:**

* La configuración de GitHub Actions permitió automatizar pruebas y análisis de código, mejorando la detección temprana de errores.
* Las pruebas unitarias con Jest aseguraron la correcta funcionalidad de las funciones implementadas, fortaleciendo la calidad del software.

1. **RECOMENDACIONES:**

* Incluir más reglas en ESLint para proyectos más complejos, como compatibilidad con frameworks específicos.
* Explorar herramientas de cobertura de pruebas (como Jest coverage) para medir la calidad de las pruebas unitarias.

1. **BIBLIOGRAFÍA:**

* **GitHub.** (2025). GitHub Actions Documentation, GitHub Docs, https://docs.github.com/en/actions. Consultada: 10 de agosto de 2025.
* **Jest.** (2025). Jest Documentation, Jest, https://jestjs.io/docs/getting-started. Consultada: 10 de agosto de 2025.
* **ESLint.** (2025). ESLint Documentation, ESLint, https://eslint.org/docs/latest/. Consultada: 10 de agosto de 2025.