

Examen-002

Objetivo

Evaluar que los estudiantes sepan aplicar buenas prácticas de desarrollo y configurar un flujo de integración continua usando GitHub Actions en el repositorio oficial del curso.

Integrantes:

Erick Medardo Alpusig Guamán
Saúl Gonzalo Tualombo Benavides
Claudio Omar Peñaherra Llulluna

Grupo: GR2SW

Fecha: 31 – 01 – 2026

Informe del Sistema de Gestión de Tareas Examen - Construcción y Evolución de Software

Resumen del Examen-002

Se presenta un sistema sencillo de gestión de tareas desarrollado en Python con integración continua mediante GitHub Actions. Este proyecto demuestra la implementación de buenas prácticas de desarrollo, incluyendo linting, formateo automático, pruebas unitarias con cobertura y un pipeline de CI/CD completo.

- **Autor:** Erick Alpusig – Saúl Tualombo – Claudio Peñaherrera
- **Fecha:** Enero 2026
- **Repositorio:** [alpusig-2025-b-emag-sw-gr2/Examen-002 at develop · 2025-b-sw-construccion-gr2/alpusig-2025-b-emag-sw-gr2](https://github.com/alpusig-2025-b-emag-sw-gr2/Examen-002)

1. Características Principales

El sistema permite realizar las siguientes acciones:

- Agregar tareas con título y descripción.
- Listar la totalidad de las tareas registradas.
- Marcar tareas como completadas.
- Eliminar registros de tareas.
- Filtrar tareas por estado (pendientes/completadas).
- Ejecución de pruebas unitarias completas.
- Integración de un pipeline de CI/CD con GitHub Actions.

2. Instalación y Ejecución Local

2.1. Requisitos Previos

- Python 3.8 o superior.
- pip (gestor de paquetes de Python).
- Git.

2.2. Pasos de Instalación

1. Clonar el repositorio:

```
git clone <URL_DEL_REPOSITORIO>
cd Examen-002
```

2. Crear un entorno virtual (recomendado):

```
python -m venv venv  
En Windows: venv\Scripts\activate  
En Linux/Mac: source venv/bin/activate
```

3. Instalar dependencias:

```
pip install -r requirements.txt
```

2.3. Ejecución del Proyecto

Ejemplo de uso de la lógica del sistema:

```
Python  
from src.logic import TaskManager  
  
manager = TaskManager()  
task1 = manager.add_task("Completar proyecto", "Proyecto de CI/CD")  
print(manager.get_all_tasks())  
manager.complete_task(1)
```

3. Calidad de Software y Pruebas

3.1. Pruebas Unitarias

Para garantizar la integridad del sistema, se utilizan los siguientes comandos:

- **Ejecución general:** `pytest tests/ -v`
- **Reporte de cobertura en terminal:** `pytest tests/ -v --cov=src --cov-report=term-missing`
- **Generación de reporte HTML:** `pytest tests/ --cov=src --cov-report=html`

3.2. Linter y Formato

Se aplican estándares de código mediante:

- **Flake8 (Estilo):** `flake8 src/ --count --show-source --statistics`
 - **Black (Formateo):** `black .`
-

4. Pipeline de CI/CD (GitHub Actions)

El proyecto utiliza GitHub Actions para automatizar el proceso de integración continua. El pipeline consta de 4 trabajos (jobs) que se ejecutan de forma secuencial:

Etap	Herramienta	Propósito
1. Lint	Flake8	Verificar calidad de código y cumplimiento de PEP 8.
2. Format	Black	Validar consistencia en el espaciado e indentación.
3. Test	Pytest	Ejecutar pruebas unitarias y medir la cobertura.
4. Build	Setuptools	Construir el paquete distribuible (.whl y .tar.gz).

Flujo de Ejecución:

Se activan disparadores (triggers) en cada push a las ramas main, develop y feature/*, así como en cada Pull Request.

5. Estructura del Proyecto

Plaintext

Examen-002/

```
├── .github/workflows/ # Configuración de CI/CD
├── docs/              # Documentación adicional
├── src/               # Lógica principal del sistema
├── tests/             # Suite de pruebas unitarias
├── example.py         # Script de demostración
├── requirements.txt   # Dependencias
└── setup.py          # Configuración de empaquetado
```

6. Evidencias de Implementación

6.1 Pipeline CI/CD Exitoso

✓ All checks have passed
4 successful checks

✓ CI Pipeline / Build del Proyecto (pull_request) Successful in 11s

✓ CI Pipeline / Lint con Flake8 (pull_request) Successful in 9s

✓ CI Pipeline / Pruebas Unitarias con Pytest (pull_request) Successful in 10s

✓ CI Pipeline / Verificación de Formato con Black (pull_request) Successful in 8s

✓ No conflicts with base branch
Merging can be performed automatically.

Merge pull request

You can also merge this with the command line. [View command line instructions.](#)

6.2 Revisión de Código y Pull Requests

ElErick-MG commented 2 minutes ago

View reviewed changes

github/workflows/ci.yml

ElErick-MG 2 minutes ago

Member Author ...

✓ Configurado correctamente para Examen-002/

✓ 4 jobs definidos (lint, format, test, build)

✓ Pasó todos los checks ✓

Reply...

Resolve conversation

github/workflows/ci.yml

83 + pytest tests/ -v --cov=src --cov-report=term-missing --cov-report=html

84 +

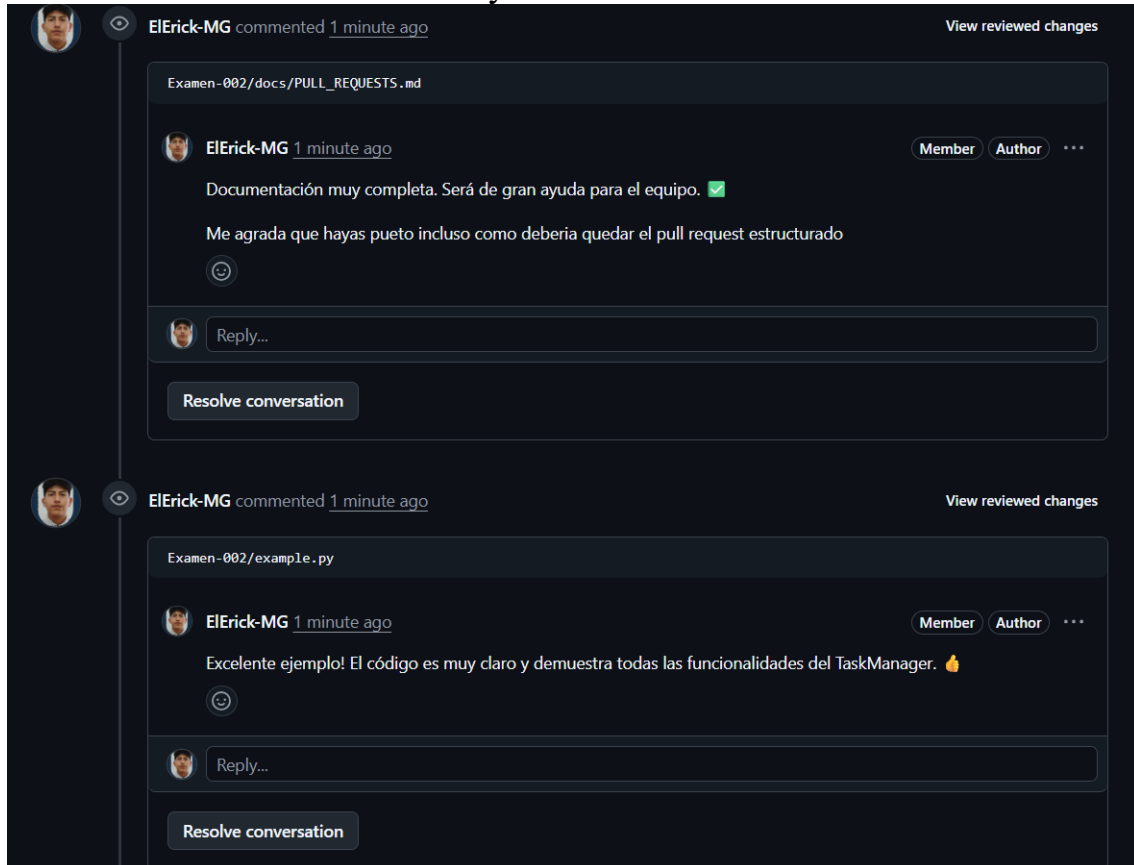
85 + - name: Subir reporte de cobertura

86 + uses: actions/upload-artifact@v4

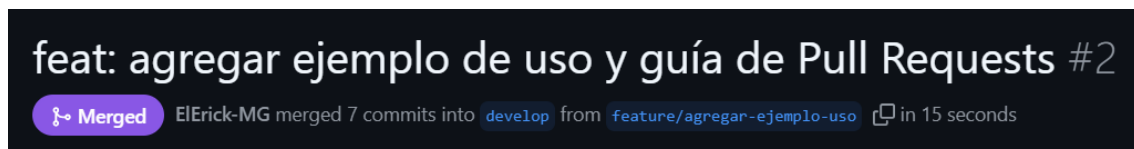
ElErick-MG 2 minutes ago

Member Author ...

Version actualizada de artifact@v4



6.3 Fusión de Ramas (Merge)



7. Conclusiones y Evaluación

Se ha determinado que el proyecto cumple satisfactoriamente con los criterios de evaluación establecidos para el curso de Construcción y Evolución de Software.

Logros alcanzados:

- Se alcanzó un **100% de cobertura** en las pruebas unitarias, incluyendo casos de borde y validación de entradas.
- Se automatizó el flujo de trabajo mediante un pipeline robusto.
- Se documentó correctamente el proceso de colaboración mediante Git Flow y revisiones por pares.

Tecnologías: Python 3.11+, Pytest, Flake8, Black y GitHub Actions.

¿Te gustaría que redactara una sección de "Recomendaciones" adicional para el informe o prefieres que ajuste algún apartado específico?