Proyecto Final

Curso de Sistemas Operativos y Laboratorio

Análisis de Rendimiento de Aplicaciones Web través de una API en Python

David Camilo García Echavarría
Cristian David Tamayo Espinosa
Silvio José Otero Guzmán
Gaia Ramirez Hincapié

Resumen

Este proyecto propone el desarrollo de una API en Python para monitorizar y analizar el rendimiento de aplicaciones web. La solución medirá métricas de CPU, memoria y dispositivos de E/S, y permitirá la creación de perfiles detallados de funciones específicas. Los datos se expondrán mediante endpoints REST y se garantizará compatibilidad multiplataforma.

Introducción

Necesidad o problema

Las aplicaciones web modernas requieren un monitoreo continuo de recursos para garantizar su estabilidad y escalabilidad. Sin herramientas adecuadas, los desarrolladores tienen dificultad para identificar cuellos de botella y optimizar funciones críticas.

Importancia tecnológica

En un entorno donde la eficiencia y la experiencia de usuario son primordiales, disponer de métricas precisas en tiempo real y perfiles de funciones facilita la toma de decisiones y mejora la calidad del software.

Antecedentes o marco teórico

Aspectos teóricos clave

- Conceptos de planificación y asignación de CPU en sistemas operativos.
- Gestión de memoria y detección de fugas.
- I/O subsystems: controladores, buffers y colas en discos y redes.
- Principios de perfilado de código: muestreo vs determinístico.

Relación con el curso de Sistemas Operativos

- En la parte teórica, se aplican conceptos de scheduling y administración de memoria vistos en clase.
- En laboratorio, se experimenta con herramientas de monitoreo y se profundiza en psutil, cProfile y otras utilidades de Python.

Objetivos (principal y específicos)

Objetivo principal

Desarrollar una API en Python que recolecta, exponga y permita analizar métricas de rendimiento de aplicaciones web en tiempo real.

Objetivos específicos

1. Implementar módulos de monitoreo de CPU, memoria y E/S usando bibliotecas multiplataforma.
2. Diseñar endpoints REST para exponer métricas sistematizadas.
3. Integrar perfiladores de funciones (cProfile, memory_profiler, py-spy) con mecanismos de activación remota.
4. Documentar y validar el funcionamiento en entornos Windows, Linux y macOS.

Metodología

Herramientas propuestas

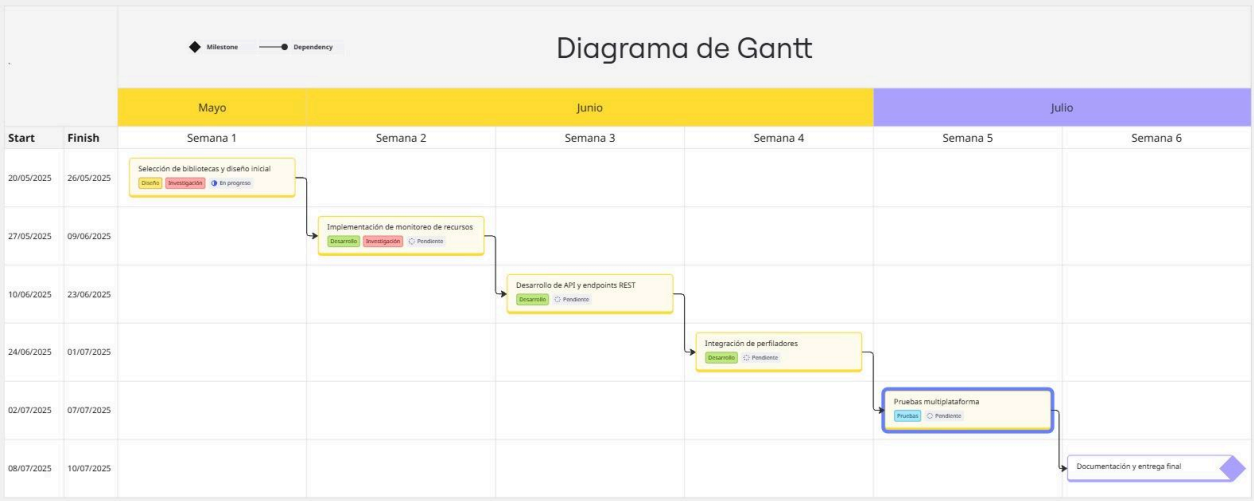
- **Python 3.8+** como lenguaje principal.
- **FastAPI** o **Flask** para la capa REST.
- **psutil** para métricas de sistema.
- **cProfile**, **memory_profiler** y **py-spy** para perfilado.
- **Prometheus Client** (opcional) para integración con herramientas de visualización.

Actividades principales

1. Revisión bibliográfica y selección de librerías.
2. Diseño de arquitectura modular (monitoreo, perfilado, API).
3. Implementación de módulos de monitoreo.
4. Desarrollo de endpoints REST.
5. Integración de perfiladores y pruebas unitarias.
6. Pruebas multiplataforma y ajustes.
7. Documentación y presentación final.

Cronograma

Actividad	Fecha de inicio	Fecha de fin
1. Selección de bibliotecas y diseño inicial	20/05/2025	26/05/2025
2. Implementación de monitoreo de recursos	27/05/2025	09/06/2025
3. Desarrollo de API y endpoints REST	10/06/2025	23/06/2025
4. Integración de perfiladores	24/06/2025	01/07/2025
5. Pruebas multiplataforma	02/07/2025	07/07/2025
6. Documentación y entrega final	08/07/2025	10/07/2025



Referencias

1. Repositorio oficial de psutil: https://github.com/giampaolo/psutil
2. Documentación de FastAPI: https://fastapi.tiangolo.com/
3. Python memory-profiler. (n.d.). *memory_profiler: Monitor Memory Usage of Python Code*. Recuperado el 22 de mayo de 2025, de https://pypi.org/project/memory-profiler/
4. Hettinger, R. (n.d.). *Profiling and Timing Code*. Python Software Foundation. Recuperado el 22 de mayo de 2025, de https://docs.python.org/3/library/profile.html
5. Denning, P. J. (1970). Virtual memory. *ACM Computing Surveys (CSUR)*, 2(3), 153–189. https://doi.org/10.1145/356586.356588