PROYECTO: Asistente Inteligente de Archivos y Auditoría del Sistema

OBJETIVO GENERAL DEL PROYECTO:

Desarrollar una herramienta robusta y fácil de usar, operada desde la línea de comandos (CLI), que nos ayude a gestionar eficientemente los archivos y directorios de nuestros sistemas. Además, esta herramienta debe tener capacidades de auditoría, registrando y permitiendo el análisis de eventos clave del sistema. La idea es automatizar tareas repetitivas y obtener una visión clara de la actividad en el sistema de archivos.

REQUERIMIENTOS CLAVE:

1. Gestión Centralizada de Archivos y Directorios:

\* Navegación: La herramienta debe permitirnos listar el contenido de cualquier directorio, mostrando información relevante como nombre, tipo (archivo/directorio), tamaño y fechas de modificación.

\* Manipulación: Necesitamos poder crear nuevos directorios, así como renombrar, mover, copiar y eliminar archivos y directorios existentes.

\* Búsqueda Avanzada: Debe incluir una funcionalidad para buscar archivos por nombre (completo o parcial) o por extensión dentro de un directorio específico y sus subdirectorios.

2. Sistema de Auditoría y Registro (Logging):

\* Registro Automático: La aplicación debe registrar automáticamente todas las operaciones críticas que realice (ej. crear, eliminar, mover un archivo/directorio) y cualquier error o excepción que ocurra durante su ejecución.

\* Detalles del Log: Cada entrada de log debe incluir una marca de tiempo precisa, el tipo de evento (informativo, advertencia, error) y una descripción clara de lo sucedido.

\* Análisis de Logs: La herramienta debe tener una función para leer y analizar estos archivos de log. Esto incluye poder filtrar eventos por tipo, rango de fechas o palabras clave, y generar resúmenes (ej. contar errores por día).

3. Arquitectura Robusta y Escalable:

\* La solución debe ser diseñada utilizando Programación Orientada a Objetos (POO). Esto significa que conceptos como clases, objetos, encapsulamiento, herencia y polimorfismo deben ser aplicados para organizar el código de manera lógica y mantenible.

\* El manejo de excepciones es fundamental para garantizar que la aplicación sea tolerante a fallos y proporcione mensajes de error útiles al usuario.

\* Se requiere el uso eficiente de generadores e iteradores, especialmente para operaciones que involucren un gran número de archivos o la lectura de logs extensos, para optimizar el uso de memoria y rendimiento.

\* El código debe estar bien modularizado, organizándose en diferentes archivos (.py) y, si el proyecto crece, en una estructura de paquete Python.

CONCEPTOS ADICIONALES PARA EXPLORAR DURANTE EL PROYECTO:

\* Estructuras de Datos: Aprender a usar de forma óptima listas, diccionarios, sets, y comprender conceptos como árboles (para el recorrido de directorios) a medida que surjan las necesidades.

\* Principios SOLID: Aplicar de forma iterativa para mejorar la calidad del diseño del código (responsabilidad única, abierto/cerrado, etc.).

\* Patrones de Diseño: Identificar oportunidades para aplicar patrones comunes (ej. Singleton para el Logger, Factory Method para eventos) para soluciones elegantes y probadas.

\* Decoradores: Útiles para añadir funcionalidad transversal como el logging automático a métodos.

\* Expresiones Regulares: Cruciales para el análisis avanzado y la extracción de información de los archivos de log.