Manual Técnico

Proyecto: Gestor de Notas Académicas

Índice

1. Introducción

Este documento constituye el Manual Técnico para el proyecto "Gestor de Notas Académicas". Su objetivo principal es proveer una descripción de la arquitectura, el diseño, la implementación y el funcionamiento del sistema, sirviendo como guía de referencia para el equipo de desarrollo y mantenimiento, y facilitando la comprensión y administración del sistema por parte de personal técnico. El sistema es una aplicación de consola robusta, desarrollada íntegramente en Python, diseñada para centralizar y optimizar el seguimiento académico a nivel de cursos.

Permite la gestión de cursos y sus calificaciones, soportando las operaciones estándar de CRUD (Crear, Leer, Actualizar, Eliminar), junto con capacidades avanzadas de ordenamiento. Implementa módulos específicos, como un sistema de cola de solicitudes de revisión para gestionar apelaciones o correcciones, y un historial de cambios (log) que asegura la trazabilidad. El estado de la aplicación se guarda de forma persistente en un archivo JSON y los datos pueden exportarse a formato CSV. Es importante notar que el sistema no incluye una interfaz gráfica de usuario (GUI) ni está diseñado para operar como un servicio web de acceso multiusuario concurrente. Los objetivos clave del sistema son la centralización de datos, la trazabilidad de modificaciones, la agilidad en la consulta y la portabilidad gracias a su implementación en Python y el uso de archivos planos.

En cuanto a su arquitectura, la aplicación sigue un diseño modular básico que separa la lógica de negocio de la persistencia y la presentación. La Capa Lógica o Core contiene la lógica de negocio para cursos, colas e historial. Finalmente, la Capa de Persistencia abstrae la lectura y escritura de datos, utilizando JSON como base de datos principal para la persistencia del estado y CSV exclusivamente para la exportación. Las tecnologías clave son Python (v3.x) y sus bibliotecas estándar como json y csv, sin requerir dependencias externas significativas. Este manual está estructurado para guiar al lector a través de la instalación, la descripción detallada de los módulos, el flujo de datos y el manual de operaciones de la consola.

2. Alcance

El Gestor de Notas está pensado para usarse desde la consola por estudiantes o administrativos que deseen llevar un control sencillo de cursos y notas. Está diseñado para ser ligero, extensible y fácil de entender.

3. Requisitos

Software

- Python 3.8 o superior.

Librerías estándar requeridas

- json (incluido en Python)
- csv (incluido en Python)

Archivo de datos

- datos_academicos.json (creado automáticamente en la primera ejecución si no existe)

4. Estructura de archivos y datos

Archivos principales

- gestor_notas.py — código fuente principal que contiene toda la lógica.
- datos_academicos.json — archivo de persistencia donde se guardan cursos, historial y revisiones.
- mis_notas.csv (o nombre que el usuario elija) — archivo opcional generado por la función de exportación.

Estructura de datos (en memoria)

- cursos: lista de diccionarios. Cada diccionario tiene las claves:
    - nombre (string)
    - nota (int)
- historial_cambios: lista (pila) de strings con descripciones de las acciones realizadas.
- solicitudes_revision: lista (cola) de strings con los nombres de cursos que requieren revisión.

5. Manual Técnico

5.1 Arquitectura general

Aplicación de consola: un único módulo con funciones agrupadas por responsabilidades (persistencia, operaciones CRUD, utilidades de ordenamiento y búsqueda, gestión de estructuras LIFO/FIFO, exportación). El flujo principal está en main() que muestra un menú y despacha a funciones según la opción seleccionada.

5.2 Descripción de módulos / funciones

A continuación, se listan las funciones con su propósito, entradas y salidas.

Funciones de persistencia

- guardar_datos()
    - Propósito: serializar las estructuras cursos, historial_cambios y solicitudes_revision a ARCHIVO_DATOS (datos_academicos.json).
    - Entradas: variables globales.
    - Salidas: archivo en disco, mensajes en consola en caso de error.
- cargar_datos()
    - Propósito: cargar datos desde ARCHIVO_DATOS al iniciar la aplicación.
    - Errores manejados: archivo no encontrado, JSON corrupto, otros errores generales.

Funciones de la aplicación (operaciones del menú)

- mostrar_menu() — Imprime el menú principal.
- registrar_curso() — Solicita nombre y nota; valida duplicados; añade a cursos y registra en historial_cambios.
- mostrar_cursos() — Lista los cursos y notas.
- calcular_promedio() — Calcula promedio aritmético de las notas.
- contar_cursos_estado() — Cuenta aprobados (nota >= 60) y reprobados.
- buscar_curso_lineal() — Búsqueda lineal por nombre.
- actualizar_nota() — Actualiza la nota de un curso existente y registra la acción en historial_cambios.
- eliminar_curso() — Elimina un curso y registra la acción.
- ordenar_burbuja_nota() — Ordena una copia de cursos por nota usando Bubble Sort; pregunta si guardar el orden.
- ordenar_insercion_nombre() — Ordena una copia de cursos por nombre usando Insertion Sort; pregunta si guardar el orden.

- buscar_curso_binaria() — Realiza búsqueda binaria en una copia de cursos ordenada por nombre (case-insensitive).
- gestionar_cola_revision() — Submenú para encolar, procesar o listar solicitudes de revisión (FIFO).
- mostrar_historial_pila() — Muestra el historial_cambios en orden LIFO.
- exportar_a_csv() — Crea un archivo CSV con encabezados nombre_curso,nota.

Punto de entrada

- main() — Carga datos, ciclo principal de menú y guarda datos al salir.

5.3 Estructuras de datos

- cursos: List[Dict[str, Any]], por ejemplo: [{'nombre': 'Matemáticas', 'nota': 85}, ...].
- historial_cambios: List[str] donde cada entrada es una descripción de acción.
- solicitudes_revision: List[str] funcionando como cola FIFO.

5.4 Persistencia

- Formato: JSON con tres claves principales: cursos, historial, revisiones.
- Ventajas: legible, fácil de manipular.
- Consideraciones: no hay bloqueo de archivos; si dos instancias escriben simultáneamente pueden corromper el archivo. Para uso sencillo de un solo usuario en consola está bien.

5.5 Algoritmos implementados

- **Bubble Sort** (ordenar por nota): complejidad O(n^2). Implementado sobre una copia para no modificar la lista original salvo que el usuario lo confirme.
- **Insertion Sort** (ordenar por nombre): complejidad O(n^2). Implementado sobre una copia.
- **Búsqueda Lineal**: O(n), útil para listas no ordenadas.
- **Búsqueda Binaria**: O(log n), **requiere** lista ordenada alfabéticamente.

5.6 Manejo de errores y validaciones

- Validación de nombre no vacío al registrar.
- Prevención de duplicados (comparación case-insensitive).
- Validación de rango de nota (0-100) y manejo de ValueError en conversiones a entero.
- Manejo de FileNotFoundError y JSONDecodeError al cargar datos.
- Manejo de IOError al guardar o exportar CSV.

5.7 Pruebas y casos de prueba

Proponer pruebas unitarias y manuales.

Pruebas unitarias sugeridas

- Registrar curso válido → verificar que se agrega y que historial crece.
- Registrar curso duplicado → no debe agregarse.
- Actualizar nota válida → nota cambia y historial registra cambio.
- Eliminar curso existente → se elimina y historial registra.
- Ordenamiento por nota y por nombre → comprobar estabilidad (según implementación) y resultado.
- Búsqueda binaria → encontrar y no encontrar casos.
- Exportar CSV → archivo creado con cabecera correcta.
- Carga y guardado de JSON → persistencia consistente.

Casos de prueba manual

- Intentar ingresar notas no numéricas.
- Intentar ordenar cuando la lista está vacía.
- Procesar cola vacía.

5.8 Extensiones sugeridas y mejoras futuras

- Validación avanzada de nombres (caracteres, longitud máxima).
- Manejo de usuarios (login) para varios perfiles.
- Concurrencia y bloqueo de archivo o uso de base de datos ligera.
- Reportes estadísticos.

5.9 Notas de implementación y convenciones

- Convención de nombres: snake_case.
- Uso de variables globales para sencillez — en proyectos más grandes se recomienda encapsular en clases.
- Documentar con docstrings y comentar bloques críticos.