Reflexiones sobre el Desarrollo del Sistema de Gestión de Biblioteca

Lecciones del Código

Este proyecto en C++ me ha permitido pasar de la teoría a la práctica en la gestión de una aplicación real, con persistencia y lógica transaccional. La lección más importante no fue solo hacer que funcionara, sino entender por qué un diseño es mejor que otro.

Desafío 1: La Persistencia

Sinceramente, lo que de verdad me hizo **sudar** fue la gestión de archivos .txt para el guardado y la carga de datos (simulando CSV). Entendí que cada objeto (Libro, Autor, Préstamo) debe ser **serializado** (convertido en una línea de texto con comas) para guardarlo y luego **deserializado** para reconstruir el objeto al iniciar la aplicación.

Si fallaba un solo campo en una línea, el contador de IDs (nextId) se rompía, y la aplicación cargaba mal. Tuve que ser muy meticuloso con el uso de ifstream y getline para que todo encajara perfectamente.

Desafío 2: La Importancia de la Normalización

Antes de empezar, podría haber puesto todos los datos del autor dentro del libro, pero aprendí que esto es ineficiente y crea duplicados. La **normalización** me enseñó a:

1. Crear colecciones separadas para Autores, Estudiantes y Libros.
2. Usar solo el **ID del Autor** (id_autor) en la estructura Libro.
3. Implementar la lógica de búsqueda para simular un **JOIN**. Por ejemplo, para ver quién era el **Autor con más Libros**, tuve que contar cuántos libros apuntaban al mismo ID. Esto es mucho más limpio que manejar nombres repetidos. La creación de claves primarias (nextAutorId, nextLibroId) fue esencial para mantener esta integridad.

Desafío 3: Depuración y Sintaxis

La fase de depuración fue frustrante. En el main(), tuve un problema recurrente donde el cin >> opcion se comía el siguiente getline, lo que me obligó a añadir la línea db.publicLeerLinea() para **limpiar el buffer** y asegurar que la siguiente entrada de texto funcionara.

Además, corregir errores como la **redefinición de funciones** o el duplicate case value en el switch principal me recordó la necesidad de una revisión sintáctica exhaustiva. Incluso el pequeño detalle de olvidar **declarar** mostrarAutorConMasLibros() dentro de la clase me costó una hora de depuración.

Conclusión

El proyecto ha sido un excelente ejercicio para entender cómo interactúan las estructuras de datos, la POO y la persistencia. La necesidad de diseñar funciones transaccionales (Préstamo/Devolución) que afectaran a múltiples estructuras (cambiando la disponibilidad del Libro y añadiendo un Prestamo) me demostró cómo la lógica de la base de datos se traduce en código C++. Siento que mi comprensión de la arquitectura de software de bajo nivel ha mejorado significativamente.