Persistencia de DoltSharp

1. Almacenamiento de Datos

* Meta: Asegurar la persistencia de datos de usuarios, tareas, eventos y configuraciones en archivos legibles y manejables.
* Implementación:
  + Utilizar archivos en formato JSON o XML para almacenar la información. Estos formatos son fáciles de leer y escribir en C# y son adecuados para un equipo de desarrolladores junior.
  + Los datos persistidos deben incluir: información de usuario (nombre, email, contraseña hasheada), detalles de las tareas (nombre, descripción, prioridad, fecha límite), y configuración de eventos.

2. Organización de Archivos

* Meta: Mantener los archivos organizados para evitar la corrupción y facilitar su acceso.
* Implementación:
  + Almacenar los datos en archivos separados por cada entidad (usuarios, tareas, eventos). Por ejemplo:
    - Un archivo usuarios.json para la información de los usuarios.
    - Un archivo tareas.json para las tareas.
    - Un archivo eventos.json para los eventos.
  + Si los datos son demasiado grandes, se pueden dividir los archivos por categorías, usuarios o fechas para evitar archivos de gran tamaño que puedan ser difíciles de manejar.

3. Escritura y Lectura Asíncrona

* Meta: Asegurar que las operaciones de persistencia no bloqueen el rendimiento de la aplicación.
* Implementación:
  + Usar operaciones asíncronas de lectura y escritura de archivos utilizando async/await en C#. Esto permite que la aplicación mantenga su rendimiento sin bloquear el hilo principal de la interfaz gráfica.
  + Utilizar buffering al escribir datos grandes para evitar que el sistema se ralentice.

4. Copia de Seguridad y Recuperación

* Meta: Proteger los datos contra la corrupción o pérdida accidental.
* Implementación:
  + Implementar una funcionalidad de copias de seguridad automáticas que almacene una copia de los archivos importantes (usuarios, tareas, eventos) en un directorio separado o una unidad externa. Estas copias pueden hacerse:
    - En intervalos regulares (diario, semanal).
    - Antes de cualquier operación crítica (como eliminar o modificar un evento/tarea).
  + Proporcionar una funcionalidad de restauración de copia de seguridad en caso de que un archivo se corrompa o se pierda, permitiendo al usuario restaurar los datos desde la copia más reciente.

5. Sincronización y Escritura en Archivos

* Meta: Asegurar que múltiples operaciones de escritura no corrompan los archivos.
* Implementación:
  + Usar bloqueo de archivos (file locks) para garantizar que solo una operación pueda escribir en un archivo a la vez, evitando la corrupción de datos cuando múltiples usuarios o procesos intenten acceder al archivo simultáneamente.
  + Realizar verificaciones antes de modificar archivos para asegurar que no estén en uso por otro proceso.

6. Integridad de Datos

* Meta: Asegurar que los datos almacenados no sean corrompidos y mantengan su integridad.
* Implementación:
  + Implementar verificación de integridad después de cada operación de escritura en archivos. Se pueden generar sumas de verificación (checksums) para asegurarse de que los datos se han escrito correctamente.
  + Crear mecanismos de rollback que permitan volver a un estado anterior en caso de que ocurra un fallo durante una operación de escritura.

7. Persistencia de Configuraciones

* Meta: Asegurar que las configuraciones de la aplicación (preferencias del usuario, ajustes de la interfaz, etc.) se guarden de manera persistente.
* Implementación:
  + Las configuraciones de usuario y las preferencias del sistema deben guardarse en archivos separados, como config.json. Esto puede incluir:
    - Preferencias de la interfaz (tema, idioma, etc.).
    - Recordatorios o notificaciones configuradas por el usuario.
  + Estas configuraciones deben cargarse al inicio de la aplicación y permitir su modificación en cualquier momento, guardándose automáticamente al cerrar la aplicación o cambiar una configuración.

8. Tolerancia a Fallos

* Meta: Asegurar que la aplicación pueda recuperarse de fallos y mantener la persistencia de datos.
* Implementación:
  + Implementar una función de guardado en intervalos regulares que almacene los cambios no guardados en un archivo temporal. En caso de un fallo inesperado, los datos pueden recuperarse del archivo temporal al reiniciar la aplicación.
  + Si una operación de escritura falla, mostrar un mensaje de error amigable y ofrecer la opción de reintentar la operación.

9. Manejo de Versiones de Archivos

* Meta: Gestionar los cambios en los archivos de datos a lo largo del tiempo.
* Implementación:
  + Usar un sistema de versiones para los archivos de datos, donde cada cambio significativo guarde una nueva versión del archivo (por ejemplo, tareas\_v1.json, tareas\_v2.json). Esto permite mantener un historial de versiones en caso de que sea necesario revertir a una versión anterior.

10. Compatibilidad de Archivos

* Meta: Asegurar que los archivos sean compatibles con futuras versiones de la aplicación.
* Implementación:
  + Estructurar los archivos de datos de manera flexible para que, si se añaden nuevos campos en versiones futuras, la aplicación pueda seguir manejando los archivos antiguos sin problemas.
  + Incluir un campo de versión en cada archivo para gestionar posibles migraciones de datos o actualizaciones del formato.