Requisitos de Rendimiento de DoltSharp

### **Transacciones por Segundo (TPS)**

* **Meta**: El sistema debe procesar al menos **100 transacciones por segundo (TPS)** en condiciones normales.
* **Implementación**:
  + Utilizar archivos **JSON** o **XML** para el almacenamiento de datos, ya que son formatos fáciles de manipular en C# y comprensibles para programadores junior.
  + Para optimizar, cargar los datos en memoria y solo escribir en los archivos cuando se produzcan cambios significativos. Usar **MemoryCache** para almacenar datos temporales y evitar leer/escribir archivos repetidamente.

### 2. **Latencia de Red**

* **Meta**: La latencia de red no debe superar los **200 ms**.
* **Implementación**:
  + Dado que la aplicación usa archivos locales, la latencia de red solo será relevante si se implementa algún tipo de sincronización con la nube o servidores externos. Mantener todo local inicialmente.
  + Si se requiere sincronización en el futuro, usar **HTTP/2** o tecnologías ligeras como **gRPC** para mantener la latencia baja.

### 3. **Tiempo de Carga de Formularios**

* **Meta**: Los formularios deben cargarse en menos de **2 segundos**.
* **Implementación**:
  + Usar **carga diferida (lazy loading)** para solo cargar los datos relevantes cuando el usuario los necesite.
  + Mantener datos en memoria y usar **binding** dinámico para evitar sobrecargar los formularios con información innecesaria al momento de cargarse.
  + Las operaciones de lectura de archivos se deben hacer de forma asíncrona para que no bloqueen la interfaz gráfica.

### 4. **Tiempo de Respuesta en Consultas**

* **Meta**: El tiempo de respuesta en consultas debe ser menor de **300 ms** para consultas complejas y **100 ms** para simples.
* **Implementación**:
  + Usar **lectura asíncrona de archivos** para obtener los datos necesarios, optimizando las operaciones de E/S.
  + Mantener los datos más utilizados en memoria y usar estructuras de datos eficientes (listas o diccionarios) para que las búsquedas en memoria sean rápidas.
  + Evitar cargar todos los datos al mismo tiempo; cargar solo lo necesario según el contexto.

### 5. **Capacidad de Manejar Carga (Escalabilidad)**

* **Meta**: La aplicación debe ser capaz de escalar para soportar hasta **500 usuarios** concurrentes inicialmente, con la posibilidad de expandirse a **1000**.
* **Implementación**:
  + Dado que se usará almacenamiento basado en archivos, mantener los archivos bien organizados (uno por usuario, evento, o tarea) para evitar la sobrecarga de acceso a un solo archivo.
  + Utilizar **bloqueo de archivos** (file locks) para evitar problemas de concurrencia si varios usuarios intentan acceder/modificar un archivo al mismo tiempo.
  + Escalar la aplicación dividiendo la lógica de almacenamiento de archivos en diferentes módulos o servicios si fuera necesario en un futuro.

### 6. **Optimización de Recursos del Sistema**

* **Meta**: Mantener el uso de CPU por debajo del **70%** y el uso de memoria por debajo del **80%**.
* **Implementación**:
  + Usar operaciones asíncronas para las lecturas y escrituras en archivos, utilizando async/await en C# para optimizar el rendimiento y liberar recursos del sistema.
  + Cargar en memoria solo los datos necesarios y evitar mantener información innecesaria en memoria, minimizando el consumo de recursos.
  + Evitar el uso excesivo de recursos de la interfaz gráfica; los controles en **Windows Forms** deben ser ligeros y no sobrecargar la CPU con redibujados innecesarios.

### 7. **Concurrencia**

* **Meta**: El sistema debe ser capaz de manejar al menos **100 solicitudes simultáneas**, escalando a **200** en fases posteriores.
* **Implementación**:
  + Utilizar **hilos en segundo plano** o **Task.Run()** para manejar múltiples operaciones a la vez sin bloquear la interfaz de usuario.
  + Implementar **asincronía** en todas las operaciones que accedan a archivos para evitar que múltiples solicitudes bloqueen el sistema.
  + Usar técnicas de sincronización para evitar conflictos de concurrencia en el acceso a los archivos.

### 8. **Monitoreo de Rendimiento**

* **Meta**: Implementar un sistema de monitoreo continuo para evaluar el rendimiento del sistema.
* **Implementación**:
  + Utilizar herramientas de logging ligeras como **Serilog** para registrar eventos clave del sistema (lecturas/escrituras en archivos, errores, etc.).
  + Registrar los tiempos de respuesta de las operaciones con archivos, el uso de recursos, y cualquier error que ocurra para analizar el comportamiento del sistema.
  + En caso de ser necesario en el futuro, integrar soluciones como **Application Insights** para un monitoreo más detallado en caso de expandir la aplicación a un entorno en la nube.