Requisitos de Rendimiento de DoltSharp

1. Transacciones por Segundo (TPS):

* Meta: El sistema debe ser capaz de procesar al menos 100 transacciones por segundo en condiciones normales.
* Implementación: Para mejorar el rendimiento, se puede usar caching con MemoryCache en ASP.NET Core o Redis para reducir las consultas innecesarias a la base de datos en tareas repetitivas.

1. Latencia de Red:

* Meta: La latencia de red entre el cliente y el servidor no debe superar los 200 ms en condiciones normales.
* Implementación: Utilizar tecnologías como gRPC o SignalR para optimizar la comunicación en tiempo real y minimizar la latencia en interacciones cliente-servidor.

1. Tiempo de Carga de Formularios:

* Meta: Los formularios de la aplicación deben cargarse completamente en menos de 2 segundos.
* Implementación: Usar carga diferida (lazy loading) para los elementos que no son necesarios inmediatamente. Además, se puede mejorar la respuesta del frontend mediante la optimización de las consultas a la base de datos con Entity Framework Core o Dapper.

1. Tiempo de Respuesta en Consultas:

* Meta: El tiempo de respuesta de las consultas a la base de datos no debe exceder los 300 ms para consultas complejas y 100 ms para consultas simples.
* Implementación: Optimizar las consultas utilizando índices en las tablas de la base de datos y aplicar consultas parametrizadas para garantizar eficiencia y seguridad.

1. Capacidad de Manejar Carga (Escalabilidad):

* Meta: El sistema debe escalar automáticamente para manejar hasta 1,000 usuarios concurrentes sin degradar el rendimiento.
* Implementación: Usar Azure App Services o AWS Elastic Beanstalk para escalar automáticamente la aplicación. También se puede utilizar caching distribuido con Redis o Memcached para mejorar la gestión de recursos en situaciones de alta carga.

1. Optimización de Recursos del Sistema:

* Meta: El uso de CPU no debe exceder el 70% en picos de carga, y el uso de memoria debe mantenerse por debajo del 80%.
* Implementación: Utilizar hilos asíncronos con async/await en C# para mejorar la eficiencia del uso de recursos y evitar bloqueos innecesarios.

1. Concurrencia:

* Meta: El sistema debe permitir la concurrencia de al menos 200 solicitudes simultáneas sin deteriorar el tiempo de respuesta.
* Implementación: Usar mecanismos de asincronía y paralelismo en C# para manejar múltiples solicitudes simultáneamente sin bloquear recursos.

1. Monitoreo de Rendimiento:

* Meta: Implementar un sistema de monitoreo continuo para evaluar el rendimiento del sistema en tiempo real.
* Implementación: Utilizar herramientas como Application Insights en Azure o New Relic para monitorear el rendimiento de la aplicación, incluyendo tiempos de respuesta, uso de CPU, memoria y capacidad de manejo de transacciones.