Obligatorio 2
Programación de Redes

Sebastián Escuder - 242739

Bruno Odella - 231665

Paula Cantera - 256680

Universidad ORT

Grupo M6A

Abril 2024

Acceso al repositorio:
https://github.com/ArqSis-PDR-2024-1/M6A_Ingenieria_242739_231665_256680

Migración a un Modelo Asincrónico

Cambios en ‘NetworkHelper’, ‘FileStreamHelper’, que generan cambios en el ‘Program.cs’ tanto del cliente como del servidor.

Se realizó la migración del sistema cliente-servidor de un modelo basado en hilos (Threads) a uno asincrónico utilizando Tasks y async/await. La misma se realizó para mejorar la eficiencia y escalabilidad. En el modelo basado en hilos, la creación y gestión de numerosos hilos consume muchos recursos del sistema. Además, las operaciones de entrada/salida (I/O) bloqueantes provocan que los hilos queden inactivos mientras esperan que finalicen, lo cual es un uso ineficiente de los recursos. Permite el paralelismo al igual que usar Tasks, pero no es el más eficiente cuando se trata de concurrencia elevada.

Al utilizar Tasks y async/await, cuando una operación de I/O está en espera, el hilo no se bloquea. En su lugar, la tarea asincrónica cede el control, permitiendo que el hilo realice otras tareas. Esto maximiza el uso eficiente de los recursos del sistema, mejorando la capacidad de respuesta y permitiendo que la aplicación maneje un mayor número de conexiones simultáneas sin una sobrecarga significativa.

En la implementación, se modificaron las firmas de los métodos relevantes para que sean asincrónicos y se utilizó await en las llamadas a métodos asincrónicos. En el servidor, la aceptación y manejo de nuevas conexiones de clientes se realizó de manera asincrónica, permitiendo gestionar múltiples conexiones sin bloqueos innecesarios.

Esta migración ha mejorado significativamente la eficiencia, capacidad de respuesta y escalabilidad de la aplicación, permitiendo un uso más eficiente de los recursos del sistema y una mejor experiencia para los usuarios, especialmente en escenarios de alta concurrencia.

Cambios en los repositorios del servidor

En el servidor, además de migrar el manejo de conexiones de clientes a un modelo asincrónico, también se migraron las operaciones de lectura y escritura de los repositorios de datos. Esta decisión se tomó para optimizar aún más el rendimiento y la escalabilidad del sistema. Originalmente, las operaciones de acceso a datos en los repositorios eran sincrónicas, lo que significaba que cada operación de lectura y escritura bloqueaba el hilo hasta que la operación se completaba. Este enfoque presenta varias desventajas, como el bloqueo de hilos y el uso ineficiente de recursos que se mencionaron en la sección anterior, especialmente en aplicaciones con alta concurrencia y grandes volúmenes de datos.

Además, se implementó un cambio en la gestión de semáforos dentro de los contextos de objetos, reemplazando `Semaphore` por `SemaphoreSlim`. Este cambio permite utilizar `WaitAsync` en lugar de `Wait`, facilitando que los hilos no se bloqueen y puedan continuar realizando otras tareas mientras esperan recursos, lo cual es crucial para las operaciones

asincrónicas. Migrar estas operaciones a un modelo asincrónico permite que las operaciones de lectura y escritura a la base de datos no bloqueen el hilo que las ejecuta. Cuando una operación de I/O está en progreso, el hilo cede el control y queda libre para realizar otras tareas, permitiéndonos manejar más solicitudes de manera eficiente.

Nueva funcionalidad(Opción 2)

Se agregó una funcionalidad al servidor que permite cerrarlo correctamente, permitiendo al administrador elegir entre esperar a que los clientes actuales se desconecten o desconectarlos forzosamente, utilizando las funcionalidades de cancelación de tareas proporcionadas por la clase ‘Task’. EL servidor ahora maneja el apagado de forma más robusta, asegurando que se cierren correctamente todas las conexiones de los clientes y que se liberen los recursos de manera adecuada. Para esto se agregó un ‘CancelationToken’ que se utiliza para cancelar las tareas de los clientes de manera ordenada. Este token fue propagado hacia todas las funciones del servidor, de forma tal que cualquier metodo asincrono sea cancelable. Por otro lado, se espera que si el servidor desea cerrar la conexión debe escribir la palabra ‘shutdown’ en la consola, y ahí se le consultará si desea esperar a que los clientes se desconecten o forzar la desconexión.