Dot Net/C# (ServiceStack),...xử lý/giải quyết các cơ chế đa nhiệm, đa luồng, memory leak, high cpu, threads

Cần tìm hiểu

Đa Nhiệm và Đa Luồng:

- **Async/Await:** Sử dụng từ khóa `async` và `await` có thể chờ đợi các tác vụ bất đồng bộ mà không làm đóng băng luồng chính
- **Parallel Programming:** Sử dụng `Task` sẽ tạo ra các luồng riêng biệt

→ việc trong phương thức xuất hiện **Task** và **async / await** nó có thể chờ tác vụ bất đồng bộ mà không đóng băng luồng chính

Memory Leak:

> 💡 Tránh tạo đối tượng không cần thiết
>
> - **Sử dụng StringBuilder**: Khi thực hiện nhiều thao tác nối chuỗi, sử dụng `StringBuilder` thay vì `string` để giảm thiểu việc tạo nhiều đối tượng chuỗi.
> - **Object Pooling**: Tái sử dụng các đối tượng thay vì tạo mới chúng mỗi lần. Ví dụ, sử dụng các pattern như Object Pooling cho các đối tượng tốn kém.

Giải phóng tài nguyên không cần thiết

- **Dispose Pattern**: Sử dụng pattern `IDisposable` để giải phóng tài nguyên (như kết nối cơ sở dữ liệu, file stream) khi không cần nữa.
- **Using Statements**: Sử dụng từ khóa `using` để tự động giải phóng tài nguyên.

Tối ưu hóa bộ nhớ heap và stack

- **Struct vs. Class**: Sử dụng `struct` thay vì `class` khi cần lưu trữ các đối tượng nhỏ và không cần các tính năng của lớp (như thừa kế).
- **Avoid Boxing/Unboxing**: Tránh việc boxing/unboxing không cần thiết bằng cách sử dụng các kiểu dữ liệu generic.

Garbage Collection (GC)

Hiểu và tối ưu hóa quá trình GC:

- **GC.Collect()**: Tránh gọi `GC.Collect()` thủ công, để .NET tự động quản lý quá trình này.
- **Large Object Heap (LOH)**: Giảm thiểu việc phân bổ các đối tượng lớn để tránh phân mảnh bộ nhớ trên LOH.

Tối ưu hóa bộ nhớ trong ASP.NET Core

- **Response Caching**: Sử dụng caching để giảm tải cho server và tiết kiệm bộ nhớ.
- **Pooling Connections**: Sử dụng connection pooling cho cơ sở dữ liệu để giảm thiểu việc tạo và hủy kết nối liên tục.
- **Session State**: Tránh lưu trữ quá nhiều dữ liệu trong session state.

💡 **Minimize Middleware**

- Giảm thiểu số lượng middleware trong pipeline để giảm thiểu tiêu thụ bộ nhớ.

Khi xử lý file lớn và nhỏ :

💡 **Tệp lớn**: Sử dụng `FileStream` hoặc `MemoryMappedFile` để tối ưu bộ nhớ và hiệu suất.

💡 **Tệp nhỏ**: Sử dụng `File` class hoặc `StreamReader` / `StreamWriter` để đơn giản hóa việc xử lý.

💡 **Tối ưu CPU**: Sử dụng buffer lớn hơn và các phương thức không đồng bộ để cải thiện hiệu suất và giảm tải CPU.

Khi xử lý số lượng file :

💡 **Xử lý song song (Parallel Processing)**: Sử dụng thư viện Parallel hoặc PLINQ (Parallel LINQ) của .NET

💡 **Xử lý tuần tự (Sequential Processing)**: Đây là cách đơn giản nhất, xử lý các tệp một cách tuần tự

💡 **Lập trình hàng đợi (Queue-based Programming)**: Sử dụng một hàng đợi để lưu trữ các tệp cần xử lý. Một hoặc nhiều luồng consumer sẽ lấy tệp ra khỏi hàng đợi và xử lý. Cách này giúp kiểm soát tốt hơn số lượng luồng đồng thời và phân phối tải trọng.

💡 **Sử dụng đám mây (Cloud Computing)**: Nếu số lượng tệp rất lớn, bạn có thể sử dụng các dịch vụ đám mây như AWS Lambda, Azure Functions hoặc Google Cloud

💡 **Tối ưu CPU**: Sử dụng buffer lớn hơn và các phương thức không đồng bộ để cải thiện hiệu suất và giảm tải CPU.

🥏 S-O-L-I-D

🎈 yield return

dotnet ef dbcontext scaffold "Persist Security Info=True;User ID=sa;Password=1234567890;Initial Catalog=TicketTravel;Data Source=DESKTOP-GRU7MA8\SQLEXPRESS;Connection Timeout=100000" Microsoft.EntityFrameworkCore.SqlServer -o Database