Day 58/60 - Why You Should Use CancellationToken in Async Operations

When working with asynchronous tasks in .NET, cancelling unnecessary operations is crucial for performance and resource management. CancellationToken allows you to gracefully stop an ongoing task instead of letting it run indefinitely.

1. How Does CancellationToken Work?
• It signals a task to stop without forcibly terminating it.
• Helps in improving responsiveness by freeing up resources.
• Prevents wasted computation on operations that are no longer needed.

2. Using CancellationToken in an Async Task

```
public async Task DownloadFileAsync(CancellationToken token)
{
 for (int i = 0; i < 10; i++)
 {
 if (token.IsCancellationRequested)
 {
 Console.WriteLine("Download canceled.");
 return; // Gracefully exit
 }

 await Task.Delay(1000); // Simulating work
 Console.WriteLine($"Downloading... {i * 10}%");
 }
}
```

Calling the method with cancellation:

```
var cts = new CancellationTokenSource();
var task = DownloadFileAsync(cts.Token);

await Task.Delay(3000); // Let it run for 3 seconds
cts.Cancel(); // Request cancellation
await task;
```

3. Why Use CancellationToken?

✔ Prevents unnecessary resource usage.
✔ Ensures graceful shutdown of async operations.
✔ Avoids hanging tasks that are no longer needed.

4. When to Use It?
• Long-running operations like downloads or database queries.
• Background services where resources should be freed when stopping.
• UI applications where users may cancel an action before completion.

Cancellation tokens improve efficiency and prevent wasted processing in async workflows. Start using them to write responsive and efficient .NET applications.