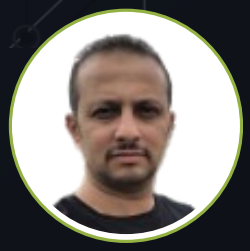
5 HIDDEN GEMS IN ASP.NET CORE YOU PROBABLY AREN'T USING



Ayman Anaam

More →

Repost

1️⃣ IHostedService for Background Tasks

Ever needed to run recurring background jobs without adding external libraries? Say hello to IHostedService. It’s a straightforward way to handle tasks like sending emails, cleanup operations, or periodic notifications in the background.

How It Works:

Implement the IHostedService interface and add it to the service collection

```
ublic class BackgroundTask : IHostedService, IDisposable
{
    private Timer? _timer;

    public Task StartAsync(CancellationToken cancellationToken)
    {
        _timer = new Timer(DoWork, null, TimeSpan.Zero, TimeSpan.FromMinutes(1));
        return Task.CompletedTask;
    }

    private void DoWork(object? state)
    {
        Console.WriteLine("Running background task...");
    }

    public Task StopAsync(CancellationToken cancellationToken)
    {
        _timer?.Change(Timeout.Infinite, 0);
        return Task.CompletedTask;
    }

    public void Dispose() => _timer?.Dispose();
}

builder.Services.AddHostedService<BackgroundTask>();
```

Ayman Anaam

Repost

2️⃣ Built-in Health Checks

Monitoring your app's health is critical, and ASP.NET Core makes it simple with built-in Health Checks. Whether it's a database connection, disk space, or custom dependency, you can easily add health checks to your app.

How To Use:

Add the Health Checks service and configure it:

```
builder.Services.AddHealthChecks()
    .AddCheck("Sample Health Check", () =>
        HealthCheckResult.Healthy("The app is healthy!"));

app.UseHealthChecks("/health");
```

Why It Matters:

It's a lightweight solution to ensure your app's critical dependencies are always in good shape.

Ayman Anaam

Repost

3️⃣ Endpoint Filters in ASP.NET Core 8

With Endpoint Filters, introduced in ASP.NET Core 8, you can apply reusable logic to specific endpoints. Think of them as middleware but with more granular control.

Use Case:

Add validation, logging, or even modify responses across multiple routes

```
app.MapPost("/api/data", (DataModel data) => Results.Ok(data))
    .AddEndpointFilter(async (context, next) =>
    {
        var data = context.GetArgument<DataModel>(0);
        if (string.IsNullOrWhiteSpace(data.Name))
            return Results.BadRequest("Name is required!");
        return await next(context);
    });
```

Why It's Useful:

Avoid repetitive code and create cleaner APIs with centralized logic.

Ayman Anaam

Repost

4 HTTP/3 Support

Performance is king in modern web apps, and HTTP/3 is the latest standard offering reduced latency and faster data transfer. ASP.NET Core supports HTTP/3 out of the box, letting you take advantage of these performance gains with minimal effort.

Enable HTTP/3:

In your appsettings.json, configure Kestrel:

```
"Kestrel": {
  "Endpoints": {
    "HttpsDefault": {
      "Url": "https://localhost:5001",
      "Protocols": "Http1AndHttp2AndHttp3"
    }
  }
}
```

Tip:

Use HTTP/3 for scenarios like streaming video or gaming applications where speed matters most.

Ayman Anaam

Repost

5 Rate Limiting Middleware

If you’re building public APIs, managing traffic is critical to prevent abuse. ASP.NET Core’s new Rate Limiting Middleware makes it easy to control request rates, ensuring your services remain stable under heavy loads.

```
builder.Services.AddRateLimiter(options =>
{
    options.GlobalLimiter = PartitionedRateLimiter.Create<HttpContext, string>(
        context => RateLimitPartition.GetFixedWindowLimiter(
            "default", _ => new FixedWindowRateLimiterOptions
            {
                PermitLimit = 5,
                Window = TimeSpan.FromSeconds(10)
            }));
});

app.UseRateLimiter();
```

Why Use It?

Protect your APIs from overuse and maintain consistent performance.

Ayman Anaam

Repost

Thank you!

Share this to help others gain insight—what you know could be exactly what they need!

Ayman Anaam

Repost