.NET Middleware: A Complete Guide with Examples

🔍What is Middleware in .NET Core?

Middleware in .NET Core is software that's assembled into an application pipeline to handle requests and responses. Each middleware component:

- Receives the incoming HTTP request.
- Optionally handles it.
- Passes it to the next middleware in the pipeline.
- Optionally handles the response.

Think of middleware as a **chain of delegates** (functions) that process HTTP requests.

⚖️Middleware vs Services

Feature	Middleware	Service (like DbContext)
Request-based	Yes	No
Executes per HTTP request	Yes	Depends on scope
Controls pipeline flow	Yes	No
Injected via DI	No (registered manually)	Yes
Example	UseAuthentication(), UseRouting()	DbContext, IEmailService

---

🕰️Built-in Middleware Examples

1. **UseRouting()**

2. Sets up request routing.

3. **UseAuthentication()**

4. Validates credentials and identity.

5. **UseAuthorization()**

6. Enforces access control based on policies/roles.

7. **UseEndpoints()**

8. Maps endpoints (e.g., controllers, Razor pages).
9. **UseCors()**
10. Enables Cross-Origin Resource Sharing.
11. **UseStaticFiles()**
12. Serves static content like HTML, CSS, JS.
13. **UseHttpsRedirection()**
14. Redirects HTTP requests to HTTPS.

💻Custom Middleware Example

Create Custom Logging Middleware:

```
public class RequestLoggingMiddleware
{
    private readonly RequestDelegate _next;

    public RequestLoggingMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    public async Task InvokeAsync(HttpContext context)
    {
        Console.WriteLine($"Request: {context.Request.Method}
{context.Request.Path}");
        await _next(context);
        Console.WriteLine($"Response: {context.Response.StatusCode}");
    }
}
```

Register it in Program.cs:

```
app.UseMiddleware<RequestLoggingMiddleware>();
```

🕰️Another Custom Middleware: Maintenance Mode

```
public class MaintenanceMiddleware
{
    private readonly RequestDelegate _next;
    private readonly bool _isInMaintenance = true;

    public MaintenanceMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    public async Task InvokeAsync(HttpContext context)
    {
        if (_isInMaintenance)
        {
            context.Response.StatusCode = 503;
            await
context.Response.WriteAsync("Site is under maintenance. Please try again
later.");
        }
        else
        {
            await _next(context);
        }
    }
}
```

Register:

```
app.UseMiddleware<MaintenanceMiddleware>();
```

🧮Middleware Execution Order

The order of middleware **matters**. They run in the sequence they're registered in `Program.cs`.

```
app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();
app.UseEndpoints(...);
```

For example, putting `UseAuthorization()` **before** `UseAuthentication()` will cause it to fail.

🔗Best Practices

- Always place middleware in the correct order.
- Keep middleware focused on one task (e.g., logging, validation).
- Use built-in middleware when possible.
- Avoid putting heavy logic in middleware; delegate to services if needed.

📄Common Use Cases

- Logging requests/responses
- Global exception handling
- Maintenance mode
- IP whitelisting or rate limiting
- Modifying request/response headers

📊Summary

Middleware in .NET Core is a powerful mechanism for handling requests globally. While it's not the same as services like `DbContext`, it plays a crucial role in request processing, security, and extensibility.

Use middleware when you want to:

- Intercept every request
- Short-circuit the pipeline
- Pre-process or post-process responses