
# Delegating Handler in ASP.NET Core

Aram Tchekrekjian

@AramT87

# Delegating Handler

A DelegatingHandler is a class in .NET that lets you intercept, inspect, or modify an HttpRequestMessage or HttpResponseMessage

This can happen before and after it's handled by the next component in the pipeline.

It's like an HttpClient middleware.

@AramT87

# Chained Delegating Handler

When you add multiple DelegatingHandlers to an HttpClient, they form a pipeline, where each handler:

1. Can process the request before passing it on to the next handler.
2. Can process the response after receiving it from the next handler.

@AramT87

# Delegating Handler Use Cases

Logging requests/responses

Retrying failed calls

Injecting headers like Authorization

Adding resilience with Polly (when chained)

@AramT87

# Authorization

When calling external APIs in ASP.NET Core using HttpClient, it’s common to repeat the same logic for certain actions, like adding authentication headers.

What if you could intercept every outgoing HTTP request and inject a JWT token automatically?

That’s exactly what DelegatingHandler helps you do.

Just like middleware intercepts incoming HTTP requests in ASP.NET Core.

@AramT87

# JWT Auth Delegating Handler

```
using System.Net.Http.Headers;

public class JwtAuthHandler(ITokenProvider tokenProvider)
                : DelegatingHandler
{
    protected override async Task<HttpResponseMessage> SendAsync(
        HttpRequestMessage request,
        CancellationToken ct)
    {
        var token = await tokenProvider.GetTokenAsync(ct);

        request.Headers.Authorization =
                            new AuthenticationHeaderValue(
                                    "Bearer", token);

        return await base.SendAsync(request, ct);
    }
}
```

@AramT87

# Token Provider

```
using Microsoft.Extensions.Caching.Memory;

public interface ITokenProvider
{
    Task<string> GetTokenAsync(CancellationToken ct);
}

public class TokenProvider(IAuthService authService, IMemoryCache cache)
                : ITokenProvider
{
    private const string CacheKey = "auth_token";

    public async Task<string> GetTokenAsync(CancellationToken ct)
    {
        if (cache.TryGetValue(CacheKey, out string? token))
        {
            return token ?? "";
        }

        var (accessToken, expiresIn) =
            await authService.GetAccessTokenAsync(ct);

        if (string.IsNullOrWhiteSpace(accessToken))
        {
            throw new InvalidOperationException(
                "Received empty access token from AuthService."
                );
        }

        cache.Set(CacheKey, accessToken, new MemoryCacheEntryOptions
        {
            AbsoluteExpirationRelativeToNow =
                TimeSpan.FromSeconds(expiresIn - 60)
        });

        return accessToken;
    }
}
```

# Auth Service (Placeholder)

```
public interface IAuthService
{
    Task<(string? accessToken, int expiresIn)>
        GetAccessTokenAsync(CancellationToken ct);
}
public class AuthService : IAuthService
{
    public Task<(string? accessToken, int expiresIn)>
        GetAccessTokenAsync(CancellationToken ct)
    {
        // Code Removed for Brevity
        // Ideally this should call some Auth API
        // to authenticate and generate a JWT token
    }
}
```

# External Api Service + DI Reg

```
public class ExternalApiService(
                IHttpClientFactory httpClientFactory)
{
    private readonly HttpClient _httpClient =
                httpClientFactory.CreateClient("ExternalApi");

    public async Task<string> GetBearerDataAsync(CancellationToken ct)
    {
        // /bearer is a test endpoint under httpbin project
        // that requires a valid JWT token to be passed
        var response = await _httpClient.GetAsync("/bearer", ct);
        response.EnsureSuccessStatusCode();
        return await response.Content.ReadAsStringAsync(ct);
    }
}
```

```
// Inside Program.cs

builder.Services.AddMemoryCache();

builder.Services.AddSingleton<IAuthService, AuthService>();
builder.Services.AddSingleton<ITokenProvider, TokenProvider>();
builder.Services.AddTransient<JwtAuthHandler>();

// Configure named HttpClient with JWT handler (JwtAuthHandler)
builder.Services.AddHttpClient("ExternalApi", client =>
{
    client.BaseAddress = new Uri("https://httpbin.org");
})
.AddHttpMessageHandler<JwtAuthHandler>();

builder.Services.AddScoped<ExternalApiService>();
```

# Found this useful?

# Consider Reposting

Aram Tchekrekjian

<Coding𝄞onata />

@AramT87

# Thank You

## Follow me for more content

Aram Tchekrekjian

Microsoft® MVP Most Valuable Professional

AramT87

Get Free Tips and Tutorials in .NET and C#

Join 800+ Readers

CodingSonata.com/newsletters

<Codingsonata />