NET Core CORS Setup

Here’s a quick rundown on how to setup CORS in a NET Core API.

It takes three pieces:

- CORS Policy Name
- AddCors to Services
- Add UseCors to Middleware

All three of the above elements are done in your Startup.cs.

1. Define a string, somewhere in your Startup.cs to name your CORS policy, like this:

```
public class Startup : OGA.WebAPI_Base.WebAPI_Startup_Base
{
    private string MyAllowSpecificOrigins = "_myAllowSpecificOrigins";

    public Startup(IConfiguration configuration, IWebHostEnvironment env) : base(configuration, env)
    {
        dashboard.service.Program.Config = configuration;
    }
}
```

2. Register Cors with the Service Provider, like this:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddCors(options =>
    {
        options.AddPolicy(name: MyAllowSpecificOrigins,
                          policy =>
                          {
                              policy.WithOrigins("http://192.168.1.109:4200",
                                                 "http://192.168.1.109:5000");
                          });
     });
}
```

3. Add CORS to your middleware stack, like this:

```
public void Configure(IApplicationBuilder app, IHostApplicationLifetime lifetime, IServiceProvider sp)
{
   //app.UseStaticFiles();
   app.UseRouting();

   app.UseCors(MyAllowSpecificOrigins);
   app.UseAuthentication();
   app.UseAuthorization();

   app.UseEndpoints(endpoints =>
   {
      endpoints.MapControllers();
   });
}
```

> NOTE: We've added 'app.UserCors' after UseRouting, and before Authentication.

This is the proper order of calls, based on this article: NET Core Middleware Registration Order

Advanced Handling

If your application requires more complex Origin allowance determination than can be statically set in a WithOrigins clause of a CORS policy, you can defer origin checks to your own method call, by pointing the policy to a custom method, like this:

```
services.AddCors(options =>
{
   options.AddPolicy(name: MyAllowSpecificOrigins,
                     policy =>
                     {
                        policy.SetIsOriginAllowed(this.CORSValidateOrigin);
                     });
});
```

> NOTE: The above configured the CORS policy to call `this.CORSValidateOrigin` for every received web request.

> This allows you full control over what origins are allowed.

The callback you would use, accepts a string and returns a bool. Here’s what a CORS callback would look like that allows all origins (during development):

```
/// <summary>
/// Custom handling method that evaluates incoming web request origins for allowance by CORS.
/// Specifically, we allow all in this application.
/// </summary>
/// <param name="arg"></param>
/// <returns></returns>
private bool CORSValidateOrigin(string arg)
{
    // Allow ALL origins...
    return true;
}
```

Your custom origin validation method can allow web requests, however you deem fit. Could be from a json file, an in-memory lookup, database query, or a hardcoded list.

You just have to be mindful of any latency penalty your check method adds to request performance.

Revision #1
Created 19 March 2025 15:32:27 by glwhite
Updated 19 March 2025 15:37:23 by glwhite