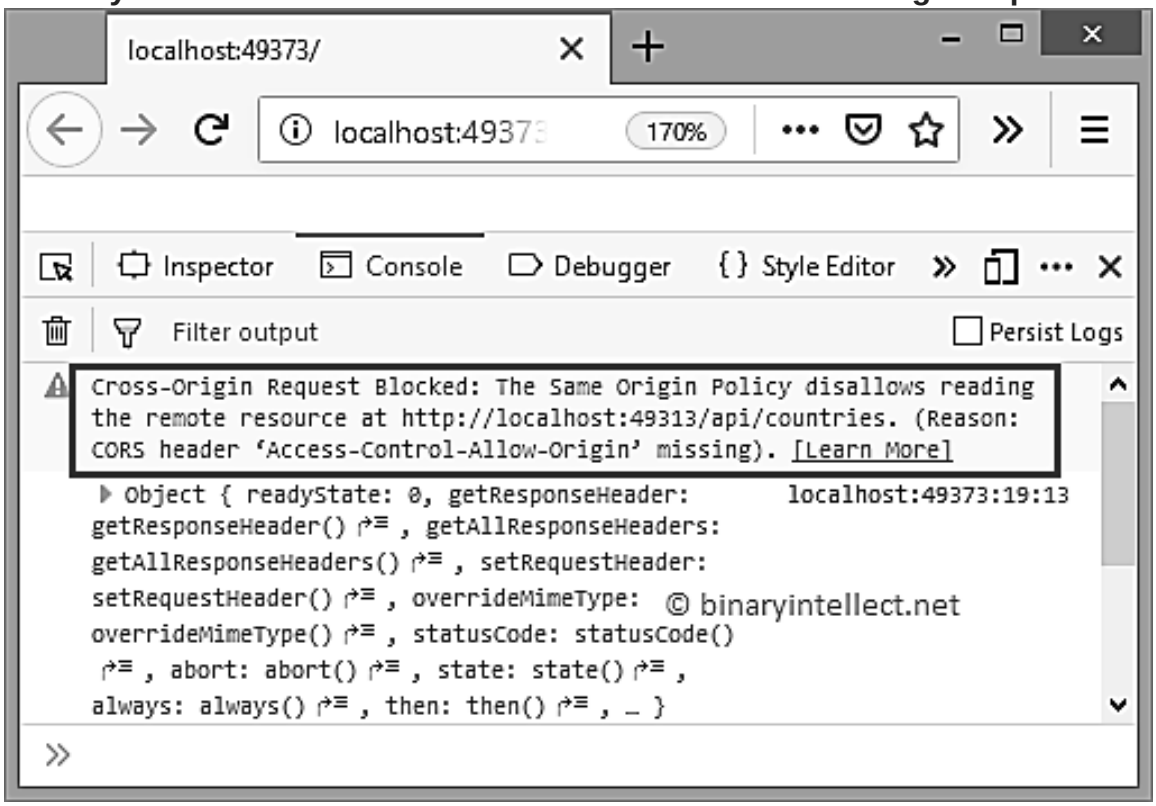
HOW TO ENABLE CORS IN ASP.NET CORE

Take advantage of the CORS middleware in ASP.Net Core to bypass the security restrictions of the web browser and allow cross-origin requests



The same-origin policy is a standard security mechanism in web browsers that allows communications between two URLs only if they share the same origin, meaning the same protocol, port, and host. For example, a client or script at **http://localhost:6000** will not be able to access a server application **at http://localhost:5080** because these two URLs have different port addresses. Security restrictions in your web browser will not allow requests to a server application in another domain.

Here is where CORS (Cross-Origin Resource Sharing) comes to the rescue. CORS is a W3C standard that allows you to get around the default same-origin policy adopted by the browsers. In short, you can use CORS to allow some cross-origin requests while preventing others. In this article we'll examine how CORS can be enabled and configured in ASP.Net Core.

Next, add the cross-origin resource sharing services to the pipeline. To do this, invoke the AddCors method on the IServiceCollection instance in the ConfigureServices method of the Startup class as shown in the code snippet below.

```
public void ConfigureServices(IServiceCollection services)
                                                                {
            services.AddCors();

        }
```

Configure CORS policy in ASP.Net Core

You can configure CORS policy in various ways in ASP.Net Core. As an example, the following code snippet allows only a specific origin to be accessed.

```
services.AddCors(options                                        =>
                                                                 {
                        options.AddPolicy("AllowSpecificOrigin",
                                          builder                =>
builder.WithOrigins("http://localhost:60571"));
            });
```

Apart from the WithOrigins method, ASP.Net gives us a number of other methods related to other policy options. These include the following:

- ✓ **AllowAnyOrigin — used to allow access to the resource from any origin**
- ✓ **AllowAnyHeader— used to allow all HTTP headers in the request**
- ✓ **AllowAnyMethod— used to allow any HTTP methods to be accessed**
- ✓ **AllowCredentials — used to pass credentials with the cross-origin request**
- ✓ **WithMethods — used to allow access to specific HTTP methods only**
- ✓ **WithHeaders — used to allow access to specific headers only**

If you want to allow more than one origin to access a resource, you can specify the following in the ConfigureServices method.

```
services.AddCors(options =>
            {
                options.AddPolicy("AllowSpecificOrigin",
                    builder =>
builder.WithOrigins("http://localhost:60571",
"http://localhost:60890"));
            });
```

If you want to allow any origin to access a resource, you should use the AllowAnyOrigin method instead of the WithOrigins method. The code snippet given below illustrates how you can allow CORS requests from all origins with any scheme.

```
services.AddCors(options =>
            {
                options.AddPolicy("AllowAllOrigins",
                    builder => builder.AllowAnyOrigin());
            });
```

CORS is a useful mechanism that allows us to flexibly bypass the restrictions of the same-origin policy of web browsers. When we want to allow cross-origin access to our server applications, we can use CORS middleware in ASP.Net Core to do so while taking advantage of a variety of cross-origin access policies.

In STARTUP FILE:

```
services.AddControllersWithViews();
            services.AddCors(o =>
o.AddPolicy("AllowMyOrigin", builder =>
            {
                builder.AllowAnyOrigin()
                       .AllowAnyMethod()
                       .AllowAnyHeader();
      }));
```

In CONTROLLER FILE

```
namespace JwtServer.Controllers
{
   [EnableCors("AllowMyOrigin")]
    [Authorize]
    [Route("api/[controller]")]
    [ApiController]
    public class MyDataController : ControllerBase
    {
        // GET: api/MyData
        [HttpGet]
        public IEnumerable<string> Get()
        {
            return new string[] { "value1", "value2" };
               }
```