**ASP.NET Core CORS**

**CORS** stands for **C**ross-**O**rigin **R**esource **S**haring.

Per the ASP.NET Core docs, "CORS is a W3C standard that allows a server to relax the same-origin policy. Using CORS, a server can explicitly allow some cross-origin requests while rejecting others."

i.e. CORS gives you the power to allow cross-domain calls from the specified domains**.**

CORS is a server-based mechanism that essentially lets a server says “I allow cross domain calls from the domains I specify”

When browsers make cross domain calls using XHR, they request CORS headers to decide whether the target server allows access to the source domain.

In this case the source domain is Consumer\Client application(dev ex localhost:3000) server and the target server i.e. ASP.NET API service (dev exlocalhost:5000 (raw Kestrel) or (IIS/IIS Express)).

For Example

*http://amshekar.****me****/abc.html*

*http://amshekar****.me****/xyz.html*

The above URLs look almost identical, so let us call them, "same origin."

Now, after making a little twist, we have some more URLs, as shown below:

*http://amshekar****.in*** *//// different domain.*

*http://www.****amshekar.com****/xyz.html //// different sub domain*

*http://amshekar****.com:7000****/abc.html ////different ports!*

So, since the URLs in the above example do not look similar, we'll use the term "Different origin" to describe them.

If the CORS is not enabled then get an exception like the one below while trying to access another domain using an ajax call:

*“XMLHttpRequest cannot load http://www.mysite.com/. No 'Access-Control-Allow-Origin' header is present on the requested resource.”*

**CORS setup:**

* Register CORS functionality
* Configure CORS options
* Apply the functionality

There are a number of different ways to do this but the recommended\best approach is to create a CORS policy and then apply that policy either globally to all requests or specific controllers.

* It allows to add CORS and declare the policy in one place
* It allows the policy to be reused and be applied selectively

**Install CORS Nuget Package**

To install Microsoft ASP.NET Core Cross-Origin Support, run the following command in the Package Manager Console or search in Nuget

PM> Install-package Microsoft.AspNetCore.Cors

**Register and Define a Policy**

To do this start with registering CORS functionality in ConfigureServices() of Startup.cs:

public void ConfigureServices(IServiceCollection services)

{

// Add service and create Policy with options

services.AddCors(options =>

{

options.AddPolicy("CorsPolicy",

builder => builder.AllowAnyOrigin()

.AllowAnyMethod()

.AllowAnyHeader()

.AllowCredentials() );

});

services.AddMvc();

}

The AddCors() call above adds the CORS features to ASP.NET and creates a custom policy that can be reused in the application by name.

There are other policy options as follows:

* **AllowAnyOrigin()** - Allows any origin.
* **AllowAnyHeader()** - Allows all HTTP headers in the request.
* **WithHeaders()** - Allows only specific headers.
* **AllowAnyMethod()** -Allows all HTTP methods.
* **WithMethods()** - Allows only specific HTTP methods.
* **AllowCredentials()** - Credentials are passed with the cross-origin request, mostly used for cookie and HTTP authentication.

**Apply the Policy**

Once the policy has been defined it can be applied.Can apply the policy globally to every request in the application by call app.useCors() in the Configure() method of Startup:

public void Configure(IApplicationBuilder app)

{

// ...

// global policy - assign here or on each controller

app.UseCors("CorsPolicy");

// ...

// IMPORTANT: Make sure UseCors() is called BEFORE this

app.UseMvc(routes =>

{

routes.MapRoute(

name: "default",

template: "{controller=Home}/{action=Index}/{id?}");

});

}

UseCors() has to be called before UseMvc() so the middleware fires before the MVC pipeline gets control and terminates the request.

**Can apply the policy to individual controllers:**

[EnableCors("CorsPolicy")]

[ApiExceptionFilter]

public class MentoringApiController : Controller

**Check that it works**

When it's all said and done you now should get the appropriate CORS headers with your response:

HTTP/1.1 200 OK

Date: Tue, 27 Sep 2016 07:09:08 GMT

Content-Type: application/json; charset=utf-8

Server: Kestrel

Vary: Origin

Access-Control-Allow-Credentials: true

Access-Control-Allow-Origin: http://localhost:3000

Content-Length: 2851

**Note** that the actual headers sent may vary depending on what actual request needs. GET operations might have different CORS headers than a POST or OPTION request.

CORS headers are only sent on cross domain requests and the ASP.NET CORS module is smart enough to detect whether a same domain request is firing and if it is, doesn’t send the headers.