Enable Cross-Origin Requests (CORS) in ASP.NET Core

04/07/2019 • 13 minutes to read • +11

In this article

By Rick Anderson

This article shows how to enable CORS in an ASP.NET Core app.

Browser security prevents a web page from making requests to a different domain than the one that served the web page. This restriction is called the *same-origin policy*. The same-origin policy prevents a malicious site from reading sensitive data from another site. Sometimes, you might want to allow other sites make cross-origin requests to your app. For more information, see the Mozilla CORS article.

Cross Origin Resource Sharing (CORS):

- Is a W3C standard that allows a server to relax the same-origin policy.

- Is **not** a security feature, CORS relaxes security. An API is not safer by allowing CORS. For more information, see How CORS works.
- Allows a server to explicitly allow some cross-origin requests while rejecting others.
- Is safer and more flexible than earlier techniques, such as JSONP.

View or download sample code (how to download)

Same origin

Two URLs have the same origin if they have identical schemes, hosts, and ports (RFC 6454).

These two URLs have the same origin:

- `https://example.com/foo.html`
- `https://example.com/bar.html`

- `https://example.com/bar.html`

These URLs have different origins than the previous two URLs:

- `https://example.net` – Different domain
- `https://www.example.com/foo.html` – Different subdomain
- `http://example.com/foo.html` – Different scheme
- `https://example.com:9000/foo.html` – Different port

Internet Explorer doesn't consider the port when comparing origins.

CORS with named policy and middleware

CORS Middleware handles cross-origin requests. The following code enables CORS for the entire app with the specified origin:

C# Copy

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    readonly string MyAllowSpecificOrigins = "_myAllowSpecificOrigins";

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddCors(options =>
        {
            options.AddPolicy(MyAllowSpecificOrigins,
```

```
                builder =>
                {
                    builder.WithOrigins("http://example.com",
                                        "http://www.contoso.com");
                });
        });

        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseHsts();
        }


        app.UseCors(MyAllowSpecificOrigins);

        app.UseHttpsRedirection();
        app.UseMvc();
    }
}
```

The preceding code:

- Sets the policy name to "_myAllowSpecificOrigins". The policy name is arbitrary.
- Calls the UseCors extension method, which enables CORS.
- Calls AddCors with a lambda expression. The lambda takes a CorsPolicyBuilder object. Configuration options, such as `WithOrigins`, are described later in this article.

The AddCors method call adds CORS services to the app's service container:

C# Copy

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddCors(options =>
    {
        options.AddPolicy(MyAllowSpecificOrigins,
        builder =>
        {
            builder.WithOrigins("http://example.com",
                                "http://www.contoso.com");
        });
    });

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
}
```

For more information, see CORS policy options in this document .

The CorsPolicyBuilder method can chain methods, as shown in the following code:

C# Copy

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddCors(options =>
    {
        options.AddPolicy(MyAllowSpecificOrigins,
        builder =>
        {
            builder.WithOrigins("http://example.com",
                                "http://www.contoso.com")
                                .AllowAnyHeader()
                                .AllowAnyMethod();
        });
    });
```

```
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
}
```

The following highlighted code applies CORS policies to all the apps endpoints via CORS Middleware:

C# Copy

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseHsts();
    }


    app.UseCors();

    app.UseHttpsRedirection();
    app.UseMvc();
}
```

See Enable CORS in Razor Pages, controllers, and action methods to apply CORS policy at the page/controller/action level.

Note:

- `UseCors` must be called before `UseMvc`.
- The URL must **not** contain a trailing slash (`/`). If the URL terminates with `/`, the comparison returns `false` and no header is returned.

See Test CORS for instructions on testing the preceding code.

Enable CORS with attributes

The [EnableCors] attribute provides an alternative to applying CORS globally. The `[EnableCors]` attribute enables CORS for selected end points, rather than all end points.

Use `[EnableCors]` to specify the default policy and `[EnableCors("{Policy String}")]` to specify a policy.

The `[EnableCors]` attribute can be applied to:

- Razor Page `PageModel`
- Controller
- Controller action method

You can apply different policies to controller/page-model/action with the `[EnableCors]` attribute. When the `[EnableCors]` attribute is applied to a controllers/page-model/action method, and CORS is enabled in middleware, both policies are

applied. We recommend against combining policies. Use the `[EnableCors]` attribute or middleware, not both in the same app.

The following code applies a different policy to each method:

C# Copy

```
[Route("api/[controller]")]
[ApiController]
public class WidgetController : ControllerBase
{
    // GET api/values
    [EnableCors("AnotherPolicy")]
    [HttpGet]
    public ActionResult<IEnumerable<string>> Get()
    {
        return new string[] { "green widget", "red widget" };
```

```
    }

    // GET api/values/5
    [EnableCors]        // Default policy.
    [HttpGet("{id}")]
    public ActionResult<string> Get(int id)
    {
        switch (id)
        {
            case 1:
                return "green widget";
            case 2:
                return "red widget";
            default:
                return NotFound();
        }
    }
}
```

The following code creates a CORS default policy and a policy named `"AnotherPolicy"`:

C# Copy

```
public class StartupMultiPolicy
{
    public StartupMultiPolicy(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddCors(options =>
        {
            options.AddDefaultPolicy(
```

```
            options.AddDefaultPolicy(
                builder =>
                {

                    builder.WithOrigins("http://example.com",
                                        "http://www.contoso.com");
                });

            options.AddPolicy("AnotherPolicy",
                builder =>
                {
                    builder.WithOrigins("http://www.contoso.com")
                                        .AllowAnyHeader()
                                        .AllowAnyMethod();
                });

        });

        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseHsts();
        }

        app.UseHttpsRedirection();
        app.UseMvc();
    }
}
```

Disable CORS

The [DisableCors] attribute disables CORS for the controller/page-model/action.

CORS policy options

This section describes the various options that can be set in a CORS policy:

- Set the allowed origins
- Set the allowed HTTP methods
- Set the allowed request headers
- Set the exposed response headers
- Credentials in cross-origin requests
- Set the preflight expiration time

AddPolicy is called in `Startup.ConfigureServices`. For some options, it may be helpful to read the How CORS works section first.

Set the allowed origins

AllowAnyOrigin – Allows CORS requests from all origins with any scheme (`http` or `https`). `AllowAnyOrigin` is insecure because *any website* can make cross-origin requests to the app.

> **Note**
>
> Specifying `AllowAnyOrigin` and `AllowCredentials` is an insecure configuration and can result in cross-site request forgery. The CORS service returns an invalid CORS response when an app is configured with both methods.

`AllowAnyOrigin` affects preflight requests and the `Access-Control-Allow-Origin` header. For more information, see the Preflight requests section.

SetIsOriginAllowedToAllowWildcardSubdomains – Sets the IsOriginAllowed property of the policy to be a function that allows origins to match a configured wildcard domain when evaluating if the origin is allowed.

C# Copy

```
options.AddPolicy("AllowSubdomain",
    builder =>
    {
        builder.SetIsOriginAllowedToAllowWildcardSubdomains();
    });
```

Set the allowed HTTP methods

AllowAnyMethod:

- Allows any HTTP method:
- Affects preflight requests and the `Access-Control-Allow-Methods` header. For more information, see the Preflight requests section.

Set the allowed request headers

To allow specific headers to be sent in a CORS request, called *author request headers*, call WithHeaders and specify the allowed headers:

C# Copy

```
options.AddPolicy("AllowHeaders",
    builder =>
    {
        builder.WithOrigins("http://example.com")
               .WithHeaders(HeaderNames.ContentType, "x-custom-header");
```

```
    });
```

To allow all author request headers, call AllowAnyHeader:

C# Copy

```
options.AddPolicy("AllowAllHeaders",
    builder =>
    {
        builder.WithOrigins("http://example.com")
               .AllowAnyHeader();
    });
```

This setting affects preflight requests and the `Access-Control-Request-Headers` header. For more information, see the Preflight requests section.

A CORS Middleware policy match to specific headers specified by `WithHeaders` is only possible when the headers sent in `Access-Control-Request-Headers` exactly match the headers stated in `WithHeaders`.

For instance, consider an app configured as follows:

C# Copy

```
app.UseCors(policy => policy.WithHeaders(HeaderNames.CacheControl));
```

CORS Middleware declines a preflight request with the following request header because `Content-Language` (HeaderNames.ContentLanguage) isn't listed in `WithHeaders`:

Copy

```
Access-Control-Request-Headers: Cache-Control, Content-Language
```

The app returns a *200 OK* response but doesn't send the CORS headers back. Therefore, the browser doesn't attempt the cross-origin request.

Set the exposed response headers

By default, the browser doesn't expose all of the response headers to the app. For more information, see W3C Cross-Origin Resource Sharing (Terminology): Simple Response Header.

The response headers that are available by default are:

- `Cache-Control`
- `Content-Language`
- `Content-Type`
- `Expires`
- `Last-Modified`
- `Pragma`

The CORS specification calls these headers *simple response headers*. To make other headers available to the app, call WithExposedHeaders:

C# Copy

```
options.AddPolicy("ExposeResponseHeaders",
    builder =>
    {
        builder.WithOrigins("http://example.com")
               .WithExposedHeaders("x-custom-header");
    });
```

Credentials in cross-origin requests

Credentials require special handling in a CORS request. By default, the browser doesn't send credentials with a cross-origin request. Credentials include cookies and HTTP authentication schemes. To send credentials with a cross-origin request, the client must set `XMLHttpRequest.withCredentials` to `true`.

Using `XMLHttpRequest` directly:

JavaScript Copy

```
var xhr = new XMLHttpRequest();
xhr.open('get', 'https://www.example.com/api/test');
xhr.withCredentials = true;
```

Using jQuery:

JavaScript Copy

```
$.ajax({
  type: 'get',
  url: 'https://www.example.com/api/test',
  xhrFields: {
    withCredentials: true
  }
});
```

Using the Fetch API:

JavaScript Copy

```
fetch('https://www.example.com/api/test', {
    credentials: 'include'
});
```

The server must allow the credentials. To allow cross-origin credentials, call AllowCredentials:

C# Copy

```
options.AddPolicy("AllowCredentials",
    builder =>
    {
        builder.WithOrigins("http://example.com")
               .AllowCredentials();
    });
```

The HTTP response includes an `Access-Control-Allow-Credentials` header, which tells the browser that the server allows credentials for a cross-origin request.

If the browser sends credentials but the response doesn't include a valid `Access-Control-Allow-Credentials` header, the browser doesn't expose the response to the app, and the cross-origin request fails.

Allowing cross-origin credentials is a security risk. A website at another domain can send a signed-in user's credentials to the app on the user's behalf without the user's knowledge.

The CORS specification also states that setting origins to `"*"` (all origins) is invalid if the `Access-Control-Allow-Credentials` header is present.

Preflight requests

For some CORS requests, the browser sends an additional request before making the actual request. This request is called a *preflight request*. The browser can skip the preflight request if the following conditions are true:

- The request method is GET, HEAD, or POST.
- The app doesn't set request headers other than `Accept`, `Accept-Language`, `Content-Language`, `Content-Type`, or `Last-Event-ID`.
- The `Content-Type` header, if set, has one of the following values:

- The `Content-Type` header, if set, has one of the following values:
  - `application/x-www-form-urlencoded`
  - `multipart/form-data`
  - `text/plain`

The rule on request headers set for the client request applies to headers that the app sets by calling `setRequestHeader` on the `XMLHttpRequest` object. The CORS specification calls these headers *author request headers*. The rule doesn't apply to headers the browser can set, such as `User-Agent`, `Host`, or `Content-Length`.

The following is an example of a preflight request:

Copy

```
OPTIONS https://myservice.azurewebsites.net/api/test HTTP/1.1
Accept: */*
Origin: https://myclient.azurewebsites.net
Access-Control-Request-Method: PUT
Access-Control-Request-Headers: accept, x-my-custom-header
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; WOW64; Trident/6.0)
Host: myservice.azurewebsites.net
Content-Length: 0
```

The pre-flight request uses the HTTP OPTIONS method. It includes two special headers:

- `Access-Control-Request-Method`: The HTTP method that will be used for the actual request.
- `Access-Control-Request-Headers`: A list of request headers that the app sets on the actual request. As stated earlier, this doesn't include headers that the browser sets, such as `User-Agent`.

A CORS preflight request might include an `Access-Control-Request-Headers` header, which indicates to the server the headers that are sent with the actual request.

To allow specific headers, call WithHeaders:

C# Copy

```
options.AddPolicy("AllowHeaders",
    builder =>
    {
        builder.WithOrigins("http://example.com")
               .WithHeaders(HeaderNames.ContentType, "x-custom-header");
    });
```

To allow all author request headers, call AllowAnyHeader:

C# Copy

```
options.AddPolicy("AllowAllHeaders",
    builder =>
    {
        builder.WithOrigins("http://example.com")

               .AllowAnyHeader();
    });
```

Browsers aren't entirely consistent in how they set `Access-Control-Request-Headers`. If you set headers to anything other than `"*"` (or use AllowAnyHeader), you should include at least `Accept`, `Content-Type`, and `Origin`, plus any custom headers that you want to support.

The following is an example response to the preflight request (assuming that the server allows the request):

Copy

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 0
Access-Control-Allow-Origin: https://myclient.azurewebsites.net
```

```
Access-Control-Allow-Headers: x-my-custom-header
Access-Control-Allow-Methods: PUT
Date: Wed, 20 May 2015 06:33:22 GMT
```

The response includes an `Access-Control-Allow-Methods` header that lists the allowed methods and optionally an `Access-Control-Allow-Headers` header, which lists the allowed headers. If the preflight request succeeds, the browser sends the actual request.

If the preflight request is denied, the app returns a *200 OK* response but doesn't send the CORS headers back. Therefore, the browser doesn't attempt the cross-origin request.

Set the preflight expiration time

The `Access-Control-Max-Age` header specifies how long the response to the preflight request can be cached. To set this header, call SetPreflightMaxAge:

C# Copy

```
options.AddPolicy("SetPreflightExpiration",
    builder =>
    {
        builder.WithOrigins("http://example.com")
               .SetPreflightMaxAge(TimeSpan.FromSeconds(2520));
    });
```

How CORS works

This section describes what happens in a CORS request at the level of the HTTP messages.

- CORS is **not** a security feature. CORS is a W3C standard that allows a server to relax the same-origin policy.
  - For example, a malicious actor could use Prevent Cross-Site Scripting (XSS) against your site and execute a cross-site

request to their CORS enabled site to steal information.

- Your API is not safer by allowing CORS.
  - It's up to the client (browser) to enforce CORS. The server executes the request and returns the response, it's the client that returns an error and blocks the response. For example, any of the following tools will display the server response:
    - Fiddler
    - Postman
    - .NET HttpClient
    - A web browser by entering the URL in the address bar.
- It's a way for a server to allow browsers to execute a cross-origin XHR or Fetch API request that otherwise would be forbidden.
  - Browsers (without CORS) can't do cross-origin requests. Before CORS, JSONP was used to circumvent this restriction. JSONP doesn't use XHR, it uses the `<script>` tag to receive the response. Scripts are allowed to be loaded cross-origin.

The CORS specification introduced several new HTTP headers that enable cross-origin requests. If a browser supports CORS, it sets these headers automatically for cross-origin requests. Custom JavaScript code isn't required to enable CORS.

The following is an example of a cross-origin request. The `Origin` header provides the domain of the site that's making the request:

Copy

```
GET https://myservice.azurewebsites.net/api/test HTTP/1.1
Referer: https://myclient.azurewebsites.net/
Accept: */*
Accept-Language: en-US
Origin: https://myclient.azurewebsites.net
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; WOW64; Trident/6.0)
Host: myservice.azurewebsites.net
```

If the server allows the request, it sets the `Access-Control-Allow-Origin` header in the response. The value of this header either matches the `Origin` header from the request or is the wildcard value `"*"`, meaning that any origin is allowed:

Copy

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: text/plain; charset=utf-8
Access-Control-Allow-Origin: https://myclient.azurewebsites.net
Date: Wed, 20 May 2015 06:27:30 GMT
Content-Length: 12

Test message
```

If the response doesn't include the `Access-Control-Allow-Origin` header, the cross-origin request fails. Specifically, the browser disallows the request. Even if the server returns a successful response, the browser doesn't make the response available to the client app.

Test CORS

To test CORS:

1. Create an API project. Alternatively, you can download the sample.
2. Enable CORS using one of the approaches in this document. For example:

C# Copy

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
```