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

10/04/2021 • 40 minutes to read •      +12

In this article

By Rick Anderson and Kirk Larkin

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 to 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`

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

Enable CORS

There are three ways to enable CORS:

- In middleware using a named policy or default policy.
- Using endpoint routing.
- With the [EnableCors] attribute.

Using the [EnableCors] attribute with a named policy provides the finest control in limiting endpoints that support CORS.

> ⚠ **Warning**
>
> **UseCors** must be called in the correct order. For more information, see **Middleware order**. For example, `UseCors` must be called before **UseResponseCaching** when using `UseResponseCaching`.

Each approach is detailed in the following sections.

CORS with named policy and middleware

CORS Middleware handles cross-origin requests. The following code applies a CORS policy to all the app's endpoints with the specified origins:

C# Copy

```
var  MyAllowSpecificOrigins = "_myAllowSpecificOrigins";

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddCors(options =>
{
    options.AddPolicy(name: MyAllowSpecificOrigins,
                      builder =>
                      {
                          builder.WithOrigins("http://example.com",
                                              "http://www.contoso.com");
                      });
});

// services.AddResponseCaching();

builder.Services.AddControllers();

var app = builder.Build();
app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();

app.UseCors(MyAllowSpecificOrigins);

app.UseAuthorization();

app.MapControllers();

app.Run();
```

The preceding code:

- Sets the policy name to `_myAllowSpecificOrigins`. The policy name is arbitrary.
- Calls the UseCors extension method and specifies the `_myAllowSpecificOrigins` CORS policy. `UseCors` adds the CORS middleware. The call to `UseCors` must be placed after `UseRouting`, but before `UseAuthorization`. For more information, see Middleware order.
- Calls AddCors with a lambda expression. The lambda takes a CorsPolicyBuilder object. Configuration options, such as `WithOrigins`, are described later in this article.
- Enables the `_myAllowSpecificOrigins` CORS policy for all controller endpoints. See endpoint routing to apply a CORS policy to specific endpoints.

- When using Response Caching Middleware, call UseCors before UseResponseCaching.

With endpoint routing, the CORS middleware **must** be configured to execute between the calls to `UseRouting` and `UseEndpoints`.

See Test CORS for instructions on testing code similar to the preceding code.

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

C# Copy

```
var  MyAllowSpecificOrigins = "_myAllowSpecificOrigins";

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddCors(options =>
{
    options.AddPolicy(name: MyAllowSpecificOrigins,
                      builder =>
                      {
                          builder.WithOrigins("http://example.com",
                                              "http://www.contoso.com");
                      });
});

// services.AddResponseCaching();

builder.Services.AddControllers();

var app = builder.Build();
app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();

app.UseCors(MyAllowSpecificOrigins);

app.UseAuthorization();

app.MapControllers();

app.Run();
```

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

The CorsPolicyBuilder methods can be chained, as shown in the following code:

C# Copy

```
var MyAllowSpecificOrigins = "_myAllowSpecificOrigins";

var builder = WebApplication.CreateBuilder(args);

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

builder.Services.AddControllers();

var app = builder.Build();
app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();

app.UseCors(MyAllowSpecificOrigins);

app.UseAuthorization();

app.MapControllers();

app.Run();
```

Note: The specified URL must **not** contain a trailing slash (`/`). If the URL terminates with `/`, the comparison returns `false` and no header is returned.

CORS with default policy and middleware

The following highlighted code enables the default CORS policy:

C# Copy

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddCors(options =>
{
    options.AddDefaultPolicy(
        builder =>
        {
            builder.WithOrigins("http://example.com",
                                "http://www.contoso.com");
```

```
        });
});

builder.Services.AddControllers();

var app = builder.Build();

app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();

app.UseCors();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

The preceding code applies the default CORS policy to all controller endpoints.

Enable Cors with endpoint routing

Enabling CORS on a per-endpoint basis using `RequireCors` ***does not support automatic preflight requests***. For more information, see this GitHub issue and Test CORS with endpoint routing and [HttpOptions].

With endpoint routing, CORS can be enabled on a per-endpoint basis using the RequireCors set of extension methods:

C# Copy

```
var MyAllowSpecificOrigins = "_myAllowSpecificOrigins";

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddCors(options =>
{
    options.AddPolicy(name: MyAllowSpecificOrigins,
                      builder =>
                      {
                          builder.WithOrigins("http://example.com",
                                              "http://www.contoso.com");
                      });
});

builder.Services.AddControllers();
builder.Services.AddRazorPages();
```

```
var app = builder.Build();

app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();

app.UseCors();

app.UseAuthorization();

app.UseEndpoints(endpoints =>
{
    endpoints.MapGet("/echo",
        context => context.Response.WriteAsync("echo"))
        .RequireCors(MyAllowSpecificOrigins);

    endpoints.MapControllers()
              .RequireCors(MyAllowSpecificOrigins);

    endpoints.MapGet("/echo2",
        context => context.Response.WriteAsync("echo2"));

    endpoints.MapRazorPages();
});

app.Run();
```

In the preceding code:

- `app.UseCors` enables the CORS middleware. Because a default policy hasn't been configured, `app.UseCors()` alone doesn't enable CORS.
- The `/echo` and controller endpoints allow cross-origin requests using the specified policy.
- The `/echo2` and Razor Pages endpoints do **not** allow cross-origin requests because no default policy was specified.

The [DisableCors] attribute does **not** disable CORS that has been enabled by endpoint routing with `RequireCors`.

See Test CORS with endpoint routing and [HttpOptions] for instructions on testing code similar to the preceding.

Enable CORS with attributes

Enabling CORS with the [EnableCors] attribute and applying a named policy to only those endpoints that require CORS provides the finest control.

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

- `[EnableCors]` specifies the default policy.
- `[EnableCors("{Policy String}")]` specifies a named policy.

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

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

Different policies can be applied to controllers, page models, or action methods with the `[EnableCors]` attribute. When the `[EnableCors]` attribute is applied to a controller, page model, or 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("Policy1")]
    [HttpGet("{id}")]
    public ActionResult<string> Get(int id)
    {
        return id switch
        {
            1 => "green widget",
            2 => "red widget",
            _ => NotFound(),
```

```
        };
    }
}
```

The following code creates two CORS policies:

C# Copy

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddCors(options =>
{
    options.AddPolicy("Policy1",
        builder =>
        {
            builder.WithOrigins("http://example.com",
                                "http://www.contoso.com");
        });

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

builder.Services.AddControllers();

var app = builder.Build();

app.UseHttpsRedirection();

app.UseRouting();

app.UseCors();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

For the finest control of limiting CORS requests:

- Use `[EnableCors("MyPolicy")]` with a named policy.
- Don't define a default policy.
- Don't use endpoint routing.

The code in the next section meets the preceding list.

See Test CORS for instructions on testing code similar to the preceding code.

Disable CORS

The [DisableCors] attribute does **not** disable CORS that has been enabled by endpoint routing.

The following code defines the CORS policy `"MyPolicy"`:

C# Copy

```
var MyAllowSpecificOrigins = "_myAllowSpecificOrigins";

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddCors(options =>
{
    options.AddPolicy(name: "MyPolicy",
        builder =>
        {
            builder.WithOrigins("http://example.com",
                                "http://www.contoso.com")
                   .WithMethods("PUT", "DELETE", "GET");
        });
});

builder.Services.AddControllers();
builder.Services.AddRazorPages();

var app = builder.Build();

app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();

app.UseCors();

app.UseAuthorization();

app.MapControllers();
app.MapRazorPages();

app.Run();
```

The following code disables CORS for the `GetValues2` action:

C# Copy

```
[EnableCors("MyPolicy")]
[Route("api/[controller]")]
[ApiController]
public class ValuesController : ControllerBase
{
    // GET api/values
    [HttpGet]
    public IActionResult Get() =>
        ControllerContext.MyDisplayRouteInfo();

    // GET api/values/5
    [HttpGet("{id}")]
    public IActionResult Get(int id) =>
        ControllerContext.MyDisplayRouteInfo(id);

    // PUT api/values/5
    [HttpPut("{id}")]
    public IActionResult Put(int id) =>
        ControllerContext.MyDisplayRouteInfo(id);


    // GET: api/values/GetValues2
    [DisableCors]
    [HttpGet("{action}")]
    public IActionResult GetValues2() =>
        ControllerContext.MyDisplayRouteInfo();

}
```

The preceding code:

- Doesn't enable CORS with endpoint routing.
- Doesn't define a default CORS policy.
- Uses [EnableCors("MyPolicy")] to enable the `"MyPolicy"` CORS policy for the controller.
- Disables CORS for the `GetValues2` method.

See Test CORS for instructions on testing the preceding code.

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 *Program.cs*. 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

```
var MyAllowSpecificOrigins = "_MyAllowSubdomainPolicy";

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddCors(options =>
{
    options.AddPolicy(name: MyAllowSpecificOrigins,
        builder =>
        {
            builder.WithOrigins("https://*.example.com")
                .SetIsOriginAllowedToAllowWildcardSubdomains();
        });
});

builder.Services.AddControllers();
```

```
var app = builder.Build();
```

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

```
using Microsoft.Net.Http.Headers;

var MyAllowSpecificOrigins = "_MyAllowSubdomainPolicy";

var builder = WebApplication.CreateBuilder(args);

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

builder.Services.AddControllers();

var app = builder.Build();
```

To allow all author request headers , call AllowAnyHeader:

C# Copy

```
var MyAllowSpecificOrigins = "_MyAllowSubdomainPolicy";

var builder = WebApplication.CreateBuilder(args);
```

```
builder.Services.AddCors(options =>
{
    options.AddPolicy(name: MyAllowSpecificOrigins,
        builder =>
        {
            builder.WithOrigins("https://*.example.com")
                   .AllowAnyHeader();
        });
});

builder.Services.AddControllers();

var app = builder.Build();
```

`AllowAnyHeader` 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

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddCors(options =>
{
    options.AddPolicy("MyExposeResponseHeadersPolicy",
        builder =>
        {
            builder.WithOrigins("https://*.example.com")
                   .WithExposedHeaders("x-custom-header");
        });
});

builder.Services.AddControllers();

var app = builder.Build();
```

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

Using the Fetch API :

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

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

```csharp
var builder = WebApplication.CreateBuilder(args);

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

builder.Services.AddControllers();

var app = builder.Build();
```

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 OPTIONS request before making the actual request. This request is called a preflight request. The browser can skip the preflight request if **all** 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:
  - `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 response similar to the preflight request made from the **[Put test]** button in the Test CORS section of this document.

Copy

```
General:
Request URL: https://cors3.azurewebsites.net/api/values/5
Request Method: OPTIONS
Status Code: 204 No Content

Response Headers:
Access-Control-Allow-Methods: PUT,DELETE,GET
Access-Control-Allow-Origin: https://cors1.azurewebsites.net
Server: Microsoft-IIS/10.0
Set-Cookie: ARRAffinity=8f8...8;Path=/;HttpOnly;Domain=cors1.azurewebsites.net
Vary: Origin
```

```
Request Headers:
Accept: */*
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Access-Control-Request-Method: PUT
Connection: keep-alive
Host: cors3.azurewebsites.net
Origin: https://cors1.azurewebsites.net
Referer: https://cors1.azurewebsites.net/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: cross-site
User-Agent: Mozilla/5.0
```

The preflight request uses the HTTP OPTIONS method. It may include the following 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`.
- Access-Control-Allow-Methods

If the preflight request is denied, the app returns a `200 OK` response but doesn't set the CORS headers. Therefore, the browser doesn't attempt the cross-origin request. For an example of a denied preflight request, see the Test CORS section of this document.

Using the F12 tools, the console app shows an error similar to one of the following, depending on the browser:

- Firefox: Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at `https://cors1.azurewebsites.net/api/TodoItems1/MyDelete2/5`. (Reason: CORS request did not succeed). Learn More
- Chromium based: Access to fetch at 'https://cors1.azurewebsites.net/api/TodoItems1/MyDelete2/5' from origin 'https://cors3.azurewebsites.net' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource. If an opaque response serves your needs, set the request's mode to 'no-cors' to fetch the resource with CORS disabled.

To allow specific headers, call WithHeaders:

C# Copy

```
using Microsoft.Net.Http.Headers;

var builder = WebApplication.CreateBuilder(args);

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

builder.Services.AddControllers();

var app = builder.Build();
```

To allow all author request headers , call AllowAnyHeader:

C# Copy

```
using Microsoft.Net.Http.Headers;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddCors(options =>
{
    options.AddPolicy("MyAllowAllHeadersPolicy",
        builder =>
        {
            builder.WithOrigins("https://*.example.com")
                   .AllowAnyHeader();
        });
});

builder.Services.AddControllers();

var app = builder.Build();
```

Browsers aren't consistent in how they set `Access-Control-Request-Headers`. If either:

- Headers are set to anything other than `"*"`
- AllowAnyHeader is called: Include at least `Accept`, `Content-Type`, and `Origin`, plus any custom headers that you want to support.

Automatic preflight request code

When the CORS policy is applied either:

- Globally by calling `app.UseCors` in *Program.cs*.
- Using the `[EnableCors]` attribute.

ASP.NET Core responds to the preflight OPTIONS request.

Enabling CORS on a per-endpoint basis using `RequireCors` currently does **not** support automatic preflight requests.

The Test CORS section of this document demonstrates this behavior.

[HttpOptions] attribute for preflight requests

When CORS is enabled with the appropriate policy, ASP.NET Core generally responds to CORS preflight requests automatically. In some scenarios, this may not be the case. For example, using CORS with endpoint routing.

The following code uses the [HttpOptions] attribute to create endpoints for OPTIONS requests:

C# Copy

```
[Route("api/[controller]")]
[ApiController]
public class TodoItems2Controller : ControllerBase
{
    // OPTIONS: api/TodoItems2/5
    [HttpOptions("{id}")]
    public IActionResult PreflightRoute(int id)
    {
        return NoContent();
    }

    // OPTIONS: api/TodoItems2
    [HttpOptions]
    public IActionResult PreflightRoute()
    {
        return NoContent();
    }

    [HttpPut("{id}")]
    public IActionResult PutTodoItem(int id)
    {
        if (id < 1)
```

```
        {
            return BadRequest();
        }

        return ControllerContext.MyDisplayRouteInfo(id);
    }
```

See Test CORS with endpoint routing and [HttpOptions] for instructions on testing the preceding code.

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

```
var builder = WebApplication.CreateBuilder(args);

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

builder.Services.AddControllers();

var app = builder.Build();
```

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 Cross-Site Scripting (XSS) against your site and execute a cross-site request to their CORS enabled site to steal information.
- An API isn't 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 PUT test button on the deployed sample

The following is an example of a cross-origin request from the Values test button to `https://cors1.azurewebsites.net/api/values`. The `Origin` header:

- Provides the domain of the site that's making the request.
- Is required and must be different from the host.

General headers

Copy

```
Request URL: https://cors1.azurewebsites.net/api/values
Request Method: GET
Status Code: 200 OK
```

Response headers

Copy

```
Content-Encoding: gzip
Content-Type: text/plain; charset=utf-8
Server: Microsoft-IIS/10.0
Set-Cookie: ARRAffinity=8f...;Path=/;HttpOnly;Domain=cors1.azurewebsites.net
Transfer-Encoding: chunked
Vary: Accept-Encoding
X-Powered-By: ASP.NET
```

Request headers

```
Accept: */*
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Connection: keep-alive
Host: cors1.azurewebsites.net
Origin: https://cors3.azurewebsites.net
Referer: https://cors3.azurewebsites.net/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: cross-site
User-Agent: Mozilla/5.0 ...
```

In `OPTIONS` requests, the server sets the **Response headers** `Access-Control-Allow-Origin: {allowed origin}` header in the response. For example, the deployed sample , Delete [EnableCors] button `OPTIONS` request contains the following headers:

General headers

```
Request URL: https://cors3.azurewebsites.net/api/TodoItems2/MyDelete2/5
Request Method: OPTIONS
Status Code: 204 No Content
```

Response headers

```
Access-Control-Allow-Headers: Content-Type,x-custom-header
Access-Control-Allow-Methods: PUT,DELETE,GET,OPTIONS
Access-Control-Allow-Origin: https://cors1.azurewebsites.net
Server: Microsoft-IIS/10.0
Set-Cookie: ARRAffinity=8f...;Path=/;HttpOnly;Domain=cors3.azurewebsites.net
Vary: Origin
X-Powered-By: ASP.NET
```

Request headers

```
Accept: */*
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Access-Control-Request-Headers: content-type
Access-Control-Request-Method: DELETE
Connection: keep-alive
Host: cors3.azurewebsites.net
Origin: https://cors1.azurewebsites.net
Referer: https://cors1.azurewebsites.net/test?number=2
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: cross-site
User-Agent: Mozilla/5.0
```

In the preceding **Response headers**, the server sets the Access-Control-Allow-Origin header in the response. The `https://cors1.azurewebsites.net` value of this header matches the `Origin` header from the request.

If AllowAnyOrigin is called, the `Access-Control-Allow-Origin: *`, the wildcard value, is returned. `AllowAnyOrigin` allows any origin.

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.

Display OPTIONS requests

By default, the Chrome and Edge browsers don't show OPTIONS requests on the network tab of the F12 tools. To display OPTIONS requests in these browsers:

- `chrome://flags/#out-of-blink-cors` or `edge://flags/#out-of-blink-cors`
- disable the flag.
- restart.

Firefox shows OPTIONS requests by default.

CORS in IIS

When deploying to IIS, CORS has to run before Windows Authentication if the server isn't configured to allow anonymous access. To support this scenario, the IIS CORS module needs to be installed and configured for the app.

Test CORS

The sample download has code to test CORS. See how to download. The sample is an API project with Razor Pages added:

C# Copy

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddCors(options =>
{
    options.AddPolicy(name: "MyPolicy",
                      builder =>
                      {
                          builder.WithOrigins("http://example.com",
                              "http://www.contoso.com",
                              "https://cors1.azurewebsites.net",
                              "https://cors3.azurewebsites.net",
                              "https://localhost:44398",
                              "https://localhost:5001")
                                  .WithMethods("PUT", "DELETE", "GET");
                      });
});

builder.Services.AddControllers();
builder.Services.AddRazorPages();

var app = builder.Build();

app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();

app.UseCors();

app.UseAuthorization();

app.MapControllers();
app.MapRazorPages();

app.Run();
```

> ⚠ **Warning**
>
> `WithOrigins("https://localhost:<port>");` should only be used for testing a sample app similar to the **download sample code** .

The following `ValuesController` provides the endpoints for testing:

C# Copy

```
[EnableCors("MyPolicy")]
[Route("api/[controller]")]
[ApiController]
public class ValuesController : ControllerBase
{
    // GET api/values
    [HttpGet]
    public IActionResult Get() =>
        ControllerContext.MyDisplayRouteInfo();

    // GET api/values/5
    [HttpGet("{id}")]
    public IActionResult Get(int id) =>
        ControllerContext.MyDisplayRouteInfo(id);

    // PUT api/values/5
    [HttpPut("{id}")]
    public IActionResult Put(int id) =>
        ControllerContext.MyDisplayRouteInfo(id);


    // GET: api/values/GetValues2
    [DisableCors]
    [HttpGet("{action}")]
    public IActionResult GetValues2() =>
        ControllerContext.MyDisplayRouteInfo();

}
```

MyDisplayRouteInfo is provided by the Rick.Docs.Samples.RouteInfo NuGet package and displays route information.

Test the preceding sample code by using one of the following approaches:

- Use the deployed sample app at https://cors3.azurewebsites.net/ . There is no need to download the sample.
- Run the sample with `dotnet run` using the default URL of `https://localhost:5001`.
- Run the sample from Visual Studio with the port set to 44398 for a URL of `https://localhost:44398`.

Using a browser with the F12 tools:

- Select the **Values** button and review the headers in the **Network** tab.

- Select the **PUT test** button. See Display OPTIONS requests for instructions on displaying the OPTIONS request. The **PUT test** creates two requests, an OPTIONS preflight request and the PUT request.

- Select the **`GetValues2 [DisableCors]`** button to trigger a failed CORS request. As mentioned in the document, the response returns 200 success, but the CORS request is not made. Select the **Console** tab to see the CORS error. Depending on the browser, an error similar to the following is displayed:

  Access to fetch at `'https://cors1.azurewebsites.net/api/values/GetValues2'` from origin `'https://cors3.azurewebsites.net'` has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. If an opaque response serves your needs, set the request's mode to 'no-cors' to fetch the resource with CORS disabled.

CORS-enabled endpoints can be tested with a tool, such as curl , Fiddler , or Postman . When using a tool, the origin of the request specified by the `Origin` header must differ from the host receiving the request. If the request isn't *cross-origin* based on the value of the `Origin` header:

- There's no need for CORS Middleware to process the request.
- CORS headers aren't returned in the response.

The following command uses `curl` to issue an OPTIONS request with information:

Bash Copy

```
curl -X OPTIONS https://cors3.azurewebsites.net/api/TodoItems2/5 -i
```

Test CORS with endpoint routing and [HttpOptions]

Enabling CORS on a per-endpoint basis using `RequireCors` currently does **not** support automatic preflight requests. Consider the following code which uses endpoint routing to enable CORS:

C# Copy

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddCors(options =>
{
    options.AddPolicy(name: "MyPolicy",
```

```
                builder =>
                {
                    builder.WithOrigins("http://example.com",
                        "http://www.contoso.com",
                        "https://cors1.azurewebsites.net",
                        "https://cors3.azurewebsites.net",
                        "https://localhost:44398",
                        "https://localhost:5001")
                            .WithMethods("PUT", "DELETE", "GET");
                });
});

builder.Services.AddControllers();
builder.Services.AddRazorPages();

var app = builder.Build();

app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();

app.UseCors();

app.UseAuthorization();

app.MapControllers();
app.MapRazorPages();

app.Run();
```

The following `TodoItems1Controller` provides endpoints for testing:

C# Copy

```
[Route("api/[controller]")]
[ApiController]
public class TodoItems1Controller : ControllerBase
{
    // PUT: api/TodoItems1/5
    [HttpPut("{id}")]
    public IActionResult PutTodoItem(int id)
    {
        if (id < 1)
        {
            return Content($"ID = {id}");
        }

        return ControllerContext.MyDisplayRouteInfo(id);
    }

    // Delete: api/TodoItems1/5
    [HttpDelete("{id}")]
```

```
    public IActionResult MyDelete(int id) =>
        ControllerContext.MyDisplayRouteInfo(id);

    // GET: api/TodoItems1
    [HttpGet]
    public IActionResult GetTodoItems() =>
        ControllerContext.MyDisplayRouteInfo();

    [EnableCors]
    [HttpGet("{action}")]
    public IActionResult GetTodoItems2() =>
        ControllerContext.MyDisplayRouteInfo();

    // Delete: api/TodoItems1/MyDelete2/5
    [EnableCors]
    [HttpDelete("{action}/{id}")]
    public IActionResult MyDelete2(int id) =>
        ControllerContext.MyDisplayRouteInfo(id);
}
```

Test the preceding code from the test page of the deployed sample .

The **Delete [EnableCors]** and **GET [EnableCors]** buttons succeed, because the endpoints have `[EnableCors]` and respond to preflight requests. The other endpoints fails. The **GET** button fails, because the JavaScript sends:

JavaScript Copy

```
 headers: {
      "Content-Type": "x-custom-header"
 },
```

The following `TodoItems2Controller` provides similar endpoints, but includes explicit code to respond to OPTIONS requests:

C# Copy

```
[Route("api/[controller]")]
[ApiController]
public class TodoItems2Controller : ControllerBase
{
    // OPTIONS: api/TodoItems2/5
    [HttpOptions("{id}")]
    public IActionResult PreflightRoute(int id)
    {
        return NoContent();
    }

    // OPTIONS: api/TodoItems2
```

```
    [HttpOptions]
    public IActionResult PreflightRoute()
    {
        return NoContent();
    }

    [HttpPut("{id}")]
    public IActionResult PutTodoItem(int id)
    {
        if (id < 1)
        {
            return BadRequest();
        }

        return ControllerContext.MyDisplayRouteInfo(id);
    }

    // [EnableCors] // Not needed as OPTIONS path provided
    [HttpDelete("{id}")]
    public IActionResult MyDelete(int id) =>
        ControllerContext.MyDisplayRouteInfo(id);

    [EnableCors]  // Rquired for this path
    [HttpGet]
    public IActionResult GetTodoItems() =>
        ControllerContext.MyDisplayRouteInfo();

    [HttpGet("{action}")]
    public IActionResult GetTodoItems2() =>
        ControllerContext.MyDisplayRouteInfo();

    [EnableCors]  // Rquired for this path
    [HttpDelete("{action}/{id}")]
    public IActionResult MyDelete2(int id) =>
        ControllerContext.MyDisplayRouteInfo(id);
}
```

Test the preceding code from the test page of the deployed sample. In the **Controller** drop down list, select **Preflight** and then **Set Controller**. All the CORS calls to the `TodoItems2Controller` endpoints succeed.

Additional resources

- Cross-Origin Resource Sharing (CORS)
- Getting started with the IIS CORS module

Is this page helpful?

Yes No

Recommended content

Migrate from ASP.NET Core 2.2 to 3.0

Learn how to migrate an ASP.NET Core 2.2 project to ASP.NET Core 3.0.

Bundle and minify static assets in ASP.NET Core

Learn how to optimize static resources in an ASP.NET Core web application by applying bundling and minification techniques.

Enforce HTTPS in ASP.NET Core

Learn how to require HTTPS/TLS in an ASP.NET Core web app.

URL Rewriting Middleware in ASP.NET Core

Learn about URL rewriting and redirecting with URL Rewriting Middleware in ASP.NET Core applications.

Show more