Response Caching Middleware in ASP.NET Core

02/07/2020 • 14 minutes to read • 👥👥🦖👤👤 +6

In this article

By John Luo

This article explains how to configure Response Caching Middleware in an ASP.NET Core app. The middleware determines when responses are cacheable, stores responses, and serves responses from cache. For an introduction to HTTP caching and the [ResponseCache] attribute, see Response Caching.

View or download sample code   (how to download)

Configuration

Use the Microsoft.AspNetCore.App metapackage or add a package reference to the Microsoft.AspNetCore.ResponseCaching   package.

In `Startup.ConfigureServices`, add the Response Caching Middleware to the service collection:

C#   📋 Copy

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddResponseCaching();
    services.AddMvc()
        .SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
}
```

Configure the app to use the middleware with the UseResponseCaching extension method, which adds the middleware to the request processing pipeline in `Startup.Configure`:

C#   📋 Copy

```csharp
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
    }

    app.UseStaticFiles();

    app.UseResponseCaching();

    app.Use(async (context, next) =>
    {
        context.Response.GetTypedHeaders().CacheControl =
            new Microsoft.Net.Http.Headers.CacheControlHeaderValue()
            {
                Public = true,
                MaxAge = TimeSpan.FromSeconds(10)
            };
        context.Response.Headers[Microsoft.Net.Http.Headers.HeaderNames.Vary] =
            new string[] { "Accept-Encoding" };

        await next();
    });

    app.UseMvc();
}
```

The sample app adds headers to control caching on subsequent requests:

- Cache-Control   : Caches cacheable responses for up to 10 seconds.
- Vary   : Configures the middleware to serve a cached response only if the Accept-Encoding   header of subsequent requests matches that of the original request.

C#   📋 Copy

```csharp
// using Microsoft.AspNetCore.Http;

app.Use(async (context, next) =>
```

```
{
    context.Response.GetTypedHeaders().CacheControl =
        new Microsoft.Net.Http.Headers.CacheControlHeaderValue()
        {
            Public = true,
            MaxAge = TimeSpan.FromSeconds(10)
        };
    context.Response.Headers[Microsoft.Net.Http.Headers.HeaderNames.Vary] =
        new string[] { "Accept-Encoding" };

    await next();
});
```

The preceding headers are not written to the response and are overridden when a controller, action, or Razor Page:

- Has a [ResponseCache] attribute. This applies even if a property isn't set. For example, omitting the VaryByHeader property will cause the corresponding header to be removed from the response.

Response Caching Middleware only caches server responses that result in a 200 (OK) status code. Any other responses, including error pages, are ignored by the middleware.

> ⚠ **Warning**
>
> Responses containing content for authenticated clients must be marked as not cacheable to prevent the middleware from storing and serving those responses. See **Conditions for caching** for details on how the middleware determines if a response is cacheable.

Options

Response caching options are shown in the following table.

Option	Description
MaximumBodySize	The largest cacheable size for the response body in bytes. The default value is `64 * 1024 * 1024` (64 MB).
SizeLimit	The size limit for the response cache middleware in bytes. The default value is `100 * 1024 * 1024` (100 MB).

Option	Description
UseCaseSensitivePaths	Determines if responses are cached on case-sensitive paths. The default value is `false`.

The following example configures the middleware to:

- Cache responses with a body size smaller than or equal to 1,024 bytes.
- Store the responses by case-sensitive paths. For example, `/page1` and `/Page1` are stored separately.

```C#
services.AddResponseCaching(options =>
{
    options.MaximumBodySize = 1024;
    options.UseCaseSensitivePaths = true;
});
```

VaryByQueryKeys

When using MVC / web API controllers or Razor Pages page models, the [ResponseCache] attribute specifies the parameters necessary for setting the appropriate headers for response caching. The only parameter of the `[ResponseCache]` attribute that strictly requires the middleware is VaryByQueryKeys, which doesn't correspond to an actual HTTP header. For more information, see Response caching in ASP.NET Core.

When not using the `[ResponseCache]` attribute, response caching can be varied with `VaryByQueryKeys`. Use the ResponseCachingFeature directly from the HttpContext.Features:

```C#
var responseCachingFeature =
context.HttpContext.Features.Get<IResponseCachingFeature>();

if (responseCachingFeature != null)
{
    responseCachingFeature.VaryByQueryKeys = new[] { "MyKey" };
}
```

Using a single value equal to `*` in `VaryByQueryKeys` varies the cache by all request query parameters.

HTTP headers used by Response Caching Middleware

The following table provides information on HTTP headers that affect response caching.

Header	Details
Authorization	The response isn't cached if the header exists.
Cache-Control	The middleware only considers caching responses marked with the `public` cache directive. Control caching with the following parameters: • max-age • max-stale† • min-fresh • must-revalidate • no-cache • no-store • only-if-cached • private • public • s-maxage • proxy-revalidate‡ †If no limit is specified to `max-stale`, the middleware takes no action. ‡`proxy-revalidate` has the same effect as `must-revalidate`. For more information, see RFC 7231: Request Cache-Control Directives .
Pragma	A `Pragma: no-cache` header in the request produces the same effect as `Cache-Control: no-cache`. This header is overridden by the relevant directives in the `Cache-Control` header, if present. Considered for backward compatibility with HTTP/1.0.
Set-Cookie	The response isn't cached if the header exists. Any middleware in the request processing pipeline that sets one or more cookies prevents the Response Caching Middleware from caching the response (for example, the cookie-based TempData provider).

Header	Details
Vary	The `Vary` header is used to vary the cached response by another header. For example, cache responses by encoding by including the `Vary: Accept-Encoding` header, which caches responses for requests with headers `Accept-Encoding: gzip` and `Accept-Encoding: text/plain` separately. A response with a header value of `*` is never stored.
Expires	A response deemed stale by this header isn't stored or retrieved unless overridden by other `Cache-Control` headers.
If-None-Match	The full response is served from cache if the value isn't `*` and the `ETag` of the response doesn't match any of the values provided. Otherwise, a 304 (Not Modified) response is served.
If-Modified-Since	If the `If-None-Match` header isn't present, a full response is served from cache if the cached response date is newer than the value provided. Otherwise, a *304 - Not Modified* response is served.
Date	When serving from cache, the `Date` header is set by the middleware if it wasn't provided on the original response.
Content-Length	When serving from cache, the `Content-Length` header is set by the middleware if it wasn't provided on the original response.
Age	The `Age` header sent in the original response is ignored. The middleware computes a new value when serving a cached response.

Caching respects request Cache-Control directives

The middleware respects the rules of the [HTTP 1.1 Caching specification](#) . The rules require a cache to honor a valid `Cache-Control` header sent by the client. Under the specification, a client can make requests with a `no-cache` header value and force the server to generate a new response for every request. Currently, there's no developer control over this caching behavior when using the middleware because the middleware adheres to the official caching specification.

For more control over caching behavior, explore other caching features of ASP.NET Core. See the following topics:

- [Cache in-memory in ASP.NET Core](#)

- Distributed caching in ASP.NET Core
- Cache Tag Helper in ASP.NET Core MVC
- Distributed Cache Tag Helper in ASP.NET Core

Troubleshooting

If caching behavior isn't as expected, confirm that responses are cacheable and capable of being served from the cache. Examine the request's incoming headers and the response's outgoing headers. Enable logging to help with debugging.

When testing and troubleshooting caching behavior, a browser may set request headers that affect caching in undesirable ways. For example, a browser may set the `Cache-Control` header to `no-cache` or `max-age=0` when refreshing a page. The following tools can explicitly set request headers and are preferred for testing caching:

- Fiddler
- Postman

Conditions for caching

- The request must result in a server response with a 200 (OK) status code.
- The request method must be GET or HEAD.
- In `Startup.Configure`, Response Caching Middleware must be placed before middleware that require caching. For more information, see ASP.NET Core Middleware.
- The `Authorization` header must not be present.
- `Cache-Control` header parameters must be valid, and the response must be marked `public` and not marked `private`.
- The `Pragma: no-cache` header must not be present if the `Cache-Control` header isn't present, as the `Cache-Control` header overrides the `Pragma` header when present.
- The `Set-Cookie` header must not be present.
- `Vary` header parameters must be valid and not equal to `*`.
- The `Content-Length` header value (if set) must match the size of the response body.
- The IHttpSendFileFeature isn't used.
- The response must not be stale as specified by the `Expires` header and the `max-age` and `s-maxage` cache directives.
- Response buffering must be successful. The size of the response must be smaller than the configured or default SizeLimit. The body size of the response must be smaller

than the configured or default MaximumBodySize.

- The response must be cacheable according to the RFC 7234 specifications. For example, the `no-store` directive must not exist in request or response header fields. See *Section 3: Storing Responses in Caches* of RFC 7234 for details.

ⓘ **Note**

The Antiforgery system for generating secure tokens to prevent Cross-Site Request Forgery (CSRF) attacks sets the `Cache-Control` and `Pragma` headers to `no-cache` so that responses aren't cached. For information on how to disable antiforgery tokens for HTML form elements, see **Prevent Cross-Site Request Forgery (XSRF/CSRF) attacks in ASP.NET Core**.

Additional resources

- App startup in ASP.NET Core
- ASP.NET Core Middleware
- Cache in-memory in ASP.NET Core
- Distributed caching in ASP.NET Core
- Detect changes with change tokens in ASP.NET Core
- Response caching in ASP.NET Core
- Cache Tag Helper in ASP.NET Core MVC
- Distributed Cache Tag Helper in ASP.NET Core

Is this page helpful?

👍 Yes   👎 No