Implement HTTP Cache (ETag) in ASP.NET Core Web API

Asked 4 years, 11 months ago Active 11 months ago Viewed 20k times

32

10

I am working on ASP.NET Core (ASP.NET 5) Web API application and have to implement HTTP Caching with the help of Entity Tags. Earlier I used CacheCow for the same but it seems it does not support ASP.NET Core as of now. I also didn't find any other relevant library or framework support details for the same.

I can write custom code for the same but before that I want to see if anything is already available. Kindly share if something is already available and what is the better way to implement that.

asp.net-web-api asp.net-core

Share Improve this question Follow

edited Sep 30 '19 at 11:55

Jim Aho

5,597 7 43 69

asked Feb 17 '16 at 13:59

Brij

335 1 3 6

4 according to this etags are implemented for static files if app.UseStaticFiles() is used. – Sam Sippe Jul 13 '16 at 6:37

FYI CacheCow has support ASP.NET Core now. – Drew Sumido Apr 25 '19 at 16:14

6 Answers

Active | Oldest | Votes

39

After a while trying to make it work with middleware I figured out that MVC action filters are actually better suited for this functionality.

```
public class ETagFilter : Attribute, IActionFilter
{
    private readonly int[] _statusCodes;

    public ETagFilter(params int[] statusCodes)
    {
        _statusCodes = statusCodes;
        if (statusCodes.Length == 0) _statusCodes = new[] { 200 };
    }

    public void OnActionExecuting(ActionExecutingContext context)
    {
    }

    public void OnActionExecuted(ActionExecutedContext context)
    {
        if (context.HttpContext.Request.Method == "GET")
        {
            if (_statusCodes.Contains(context.HttpContext.Response.StatusCode))
            {
```

```
                //I just serialize the result to JSON, could do something less costly
                var content = JsonConvert.SerializeObject(context.Result);

                var etag =
ETagGenerator.GetETag(context.HttpContext.Request.Path.ToString(),
Encoding.UTF8.GetBytes(content));

                if (context.HttpContext.Request.Headers.Keys.Contains("If-None-Match")
&& context.HttpContext.Request.Headers["If-None-Match"].ToString() == etag)
                {
                    context.Result = new StatusCodeResult(304);
                }
                context.HttpContext.Response.Headers.Add("ETag", new[] { etag });
            }
        }
    }
}

// Helper class that generates the etag from a key (route) and content (response)
public static class ETagGenerator
{
    public static string GetETag(string key, byte[] contentBytes)
    {
        var keyBytes = Encoding.UTF8.GetBytes(key);
        var combinedBytes = Combine(keyBytes, contentBytes);

        return GenerateETag(combinedBytes);
    }

    private static string GenerateETag(byte[] data)
    {
        using (var md5 = MD5.Create())
        {
            var hash = md5.ComputeHash(data);
            string hex = BitConverter.ToString(hash);
            return hex.Replace("-", "");
        }
    }

    private static byte[] Combine(byte[] a, byte[] b)
    {
        byte[] c = new byte[a.Length + b.Length];
        Buffer.BlockCopy(a, 0, c, 0, a.Length);
        Buffer.BlockCopy(b, 0, c, a.Length, b.Length);
        return c;
    }
}
```

And then use it on the actions or controllers you want as an attribute:

```
[HttpGet("data")]
[ETagFilter(200)]
public async Task<IActionResult> GetDataFromApi()
{
}
```

The important distinction between Middleware and Filters is that your middleware can run before and after MVC middlware and can only work with HttpContext. Also once MVC starts sending the response back to the client it's too late to make any changes to it.

Filters on the other hand are a part of MVC middleware. They have access to the MVC context, with which in this case it's simpler to implement this functionality. More on Filters and their pipeline in MVC.

Share Improve this answer Follow

edited Mar 9 '18 at 7:06

answered Nov 6 '16 at 22:00

erikbozic

1,347 1 14 19

just a note to people that might be thinking of using this answer for web pages, (instead of an api). it doesn't appear to account for changes to view files. – jimasp Feb 26 '17 at 20:16

Doesn't executing this action AFTER the endpoint has executed negated the major benefit of not having to serialise the result? i.e. one benefit of detecting that you can return a 304 is that you can immediately stop any further server processing and just return 304. – oatsoda Nov 16 '20 at 12:47

@oatsoda yes, you're right. The main benefit here is that we don't waste clients bandwith. But the server still does all the processing it would do otherwise. Ideally the server would cache responses in some key-value structure, where key would be the etag. I have done something like that, but it's not a completely generic solution. If I manage to do it in a way that's generic enough for a SO anwser I might post it here. – erikbozic Nov 16 '20 at 13:03

@erikbozic Yeah, I guess there are two parts 1) time taken to read/calc current etag - and then 2) time taken to serialise response. I was thinking that you'd at least want to prevent #2 as it's redundant if you return 304! #1 isn't strictly redundant but could be optimised! – oatsoda Nov 16 '20 at 14:44

@oatsoda just to clarify: this implementation relies on the serialized response to generate the etag. That's why it's **after** the endpoint executed. You have to generate it before you can decide to return 304. If there was actual server-side caching involved then you could check the etag value sent in the request with the cached one before executing the endpoint and just return the 304 if they match. But the server-side cache is very specific to the application (is the key just the url, additional tenant-id header, cached per user, ...?) – erikbozic Nov 16 '20 at 15:23

5

Building on Eric's answer, I would use an interface that could be implemented on an entity to support entity tagging. In the filter you would only add the ETag if the action is returning a entity with this interface.

This allows you to be more selective about what entities get tagged and allows you have each entity control how its tag is generated. This would be much more efficient than serializing everything and creating a hash. It also eliminates the need to check the status code. It could be safely and easily added as a global filter since you are "opting-in" to the functionality by implementing the interface on your model class.

```
public interface IGenerateETag
{
    string GenerateETag();
}

[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method, AllowMultiple =
false)]
public class ETagFilterAttribute : Attribute, IActionFilter
{
    public void OnActionExecuting(ActionExecutingContext context)
    {
```

```
    }

    public void OnActionExecuted(ActionExecutedContext context)
    {
        var request = context.HttpContext.Request;
        var response = context.HttpContext.Response;

        if (request.Method == "GET" &&
            context.Result is ObjectResult obj &&
            obj.Value is IGenerateETag entity)
        {
            string etag = entity.GenerateETag();

            // Value should be in quotes according to the spec
            if (!etag.EndsWith("\""))
                etag = "\"" + etag +"\"";

            string ifNoneMatch = request.Headers["If-None-Match"];

            if (ifNoneMatch == etag)
            {
                context.Result = new StatusCodeResult(304);
            }

            context.HttpContext.Response.Headers.Add("ETag", etag);
        }
    }
}
```

Share Improve this answer Follow

edited Oct 27 '17 at 13:49

answered Oct 27 '17 at 13:44

sectrean
10.7k 6 26 31

Hey, Could you show us how `GenerateETag` will works here? and should I need send json data to it? – Divyang Desai Oct 15 '19 at 6:47

1

Here's a more extensive version for MVC Views (tested with asp.net core 1.1):

```
using System;
using System.IO;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Http.Extensions;
using Microsoft.Net.Http.Headers;

namespace WebApplication9.Middleware
{
    // This code is mostly here to generate the ETag from the response body and set 304 as
required,
    // but it also adds the default maxage (for client) and s-maxage (for a caching proxy
like Varnish) to the cache-control in the response
    //
    // note that controller actions can override this middleware behaviour as needed with
[ResponseCache] attribute
    //
    // (There is actually a Microsoft Middleware for response caching - called
```

```
"ResponseCachingMiddleware",
    // but it looks like you still have to generate the ETag yourself, which makes the MS
Middleware kinda pointless in its current 1.1.0 form)
    //
    public class ResponseCacheMiddleware
    {
        private readonly RequestDelegate _next;
        // todo load these from appsettings
        const bool ResponseCachingEnabled = true;
        const int ActionMaxAgeDefault = 600; // client cache time
        const int ActionSharedMaxAgeDefault = 259200; // caching proxy cache time
        const string ErrorPath = "/Home/Error";

        public ResponseCacheMiddleware(RequestDelegate next)
        {
            _next = next;
        }

        ..
```

Share Improve this answer Follow

edited Mar 7 '20 at 12:11

answered Mar 25 '17 at 14:19

jimasp
882 7 24

0

I am using a middleware that works fine for me.

It adds HttpCache headers to responses (Cache-Control, Expires, ETag, Last-Modified), and implements cache expiration & validation models.

You can find it on nuget.org as a package called **Marvin.Cache.Headers**.

You could find more information from its Github home page:
https://github.com/KevinDockx/HttpCacheHeaders

Share Improve this answer Follow

edited Jul 12 '17 at 16:16

answered Jul 11 '17 at 16:20

Jeremy.F
222 2 10

1 Link-only answers are generally frowned upon on Stack Overflow. In time it is possible for links to atrophy and become unavailable, meaning that your answer is useless to users in the future. It would be best if you could provide the general details of your answer in your actual post, citing your link as a reference. – herrbischoff Jul 11 '17 at 16:42

@herrbischoff, I added more details to my answer, hope it is better now. – Jeremy.F Jul 12 '17 at 16:21

0

I found an alternative solution which is "closer" to the web api controller method - so you can decide per method which ETag to set...

See my response here: How to use ETag in Web API using action filter along with HttpResponseMessage

Share Improve this answer Follow

answered Nov 29 '17 at 8:22

James Joyce
1,354 14 16

-2

As an addendum to Erik Božič's answer I found that the HttpContext object was not reporting back the StatusCode correctly when inheriting from ActionFilterAttribute, and applied controller-wide. HttpContext.Response.StatusCode was always 200, indicating it was probably not set by this point in the pipeline. I was instead able to grab the StatusCode from ActionExecutedContext context.Result.StatusCode.

Share Improve this answer Follow

edited Aug 11 '17 at 17:04

answered Aug 11 '17 at 16:58

cagefree
7 3

An addendum is better suited as a comment. – ToothlessRebel Aug 11 '17 at 17:34

@ToothlessRebel tried that first, too little rep :(– cagefree Aug 11 '17 at 17:39

That is a not a valid reason to circumvent the system in place and post a comment as an answer. – ToothlessRebel Aug 24 '17 at 2:23

@cagefree Did you have this issue only when you applied to the entire controller, and not the individual rest endpoints? I will edit Erik's answer to feature your answer if you give me this information. – William Bernting Jan 23 '18 at 7:49

Although somewhat late (and on a different version of asp.net core), I tried this now and works fine for me. Maybe there something else in your request pipeline that affects this? Can you still repro? My working example is just a new API project with the code I posted. – erikbozic Jul 28 '18 at 8:43