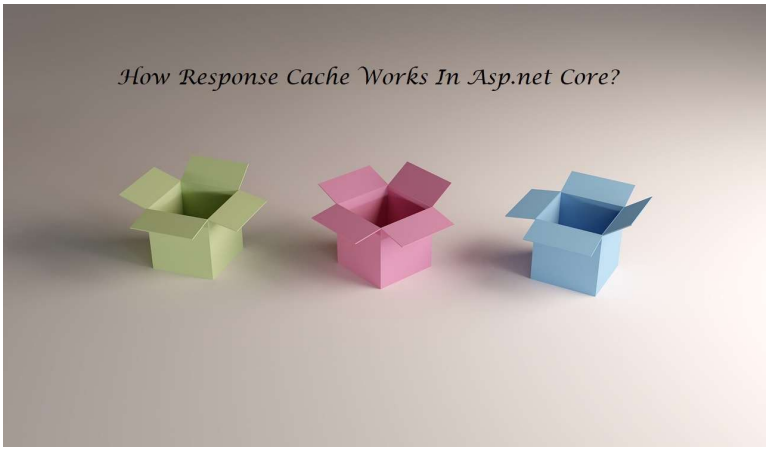
Succeed online.

Our helpful guides are available in English and Arabic to get your bus
idea online.

GoDaddy™ Shop

How Response Caching Works In Asp.Net Core

January 19, 2020



SHARE

Labels

Asp.Net Core

Asp.net core 3.1

Dotnet-Technology

What Is Response Caching?:

Response Caching means storing of response output and using stored response until it's under it's the expiration time.

Response Caching approach cuts down some requests to the server and also reduces some workload on the server.

Response Caching Headers:

Response Caching carried out by the few Http based headers information between client and server.
Main Response Caching Headers are like below

- Cache-Control
- Pragma
- Vary

Cache-Control Header:

Cache-Control header is the main header type for the response caching. Cache-Control will be decorated with the following directives.

- **public -** this directive indicates any cache may store the response.
- **private -** this directive allows to store response with respect to a single user and can't be stored with shared cache stores.
- **max-age -** this directive represents a time to hold a response in the cache.
- **no-cache -** this directive represents no storing of response and always fetch the fresh response from server
- **no-store -** cache can't store the request or response.

Succeed online.
Ad Our helpful guides are a
English and Arabic to get y
GoDaddy™
Shop now

Pragma Header:

Pragma Header used for Http/1.0 request with no-cache directive value. Pragma gets ignored if response decorated with Cache-Control

Vary Header:

Vary Header fields must match with the original request-response and the new request.

ResponseCache Attribute:

Asp.Net Core provided ResponseCache Attribute to enable response cache on MVC/Web API/Razor Pages. This attribute needs to be populated with respective input parameters to generate the cache headers. The following are few input parameters for the cache attribute.

- **Duration -** Gets or sets the duration in seconds for which response is cached. This sets "max-age" in "cache-control" header.
- **Location -** Gets or sets the location where the data from a particular URL must be cached. It accepts three types of parameters. If Location decorated with "ResponsCacheLocation.Any" works like response cached in both proxies and client and sets "cache-control" header to the public. If Location decorated with "ResponseCacheLocation.Client" works like response cached only in the client and sets "cache-control" header to private. if Location decorated with "ResponseCacheLocation.None" works like "cache-control" and "Pragma" headers set to "no-cache".
- **NoStore -** Gets or sets the value which determines whether the data should store or not. When NoStore decorates with 'true' value, it sets the "cache-control" header to "no-store" and ignores the "Location" parameter for values other than "None" and ignores "Duration" as well.

- **VaryByHeader -** Gets or sets the value for the "Vary" response header.
- **VaryByQueryKeys -** Gets or sets the query keys to "Vary".

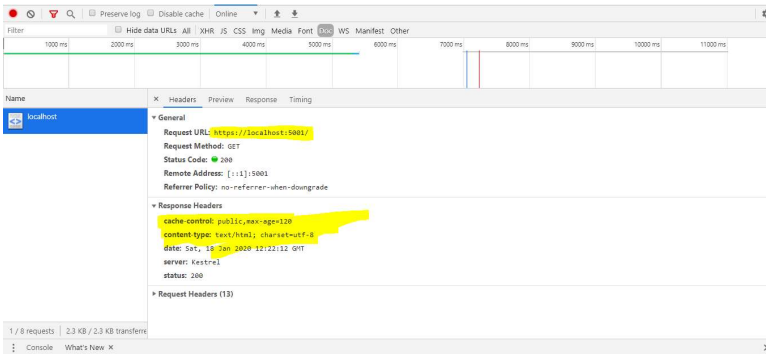
Let's start testing the response cache on MVC endpoint using ResponseCache Attribute as below.

HomeController.cs:

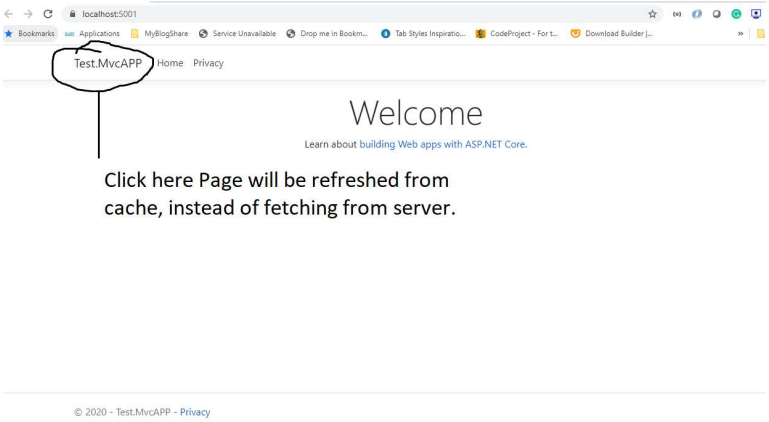
```
[ResponseCache(Duration =120, Location = ResponseCac
public IActionResult Index()
{
 return View();
}
```

Above controller decorated with ResponseCache Attribute. Cache duration set to 120 secs, Location set to ResponseCache.Any which results in "public" value for the "cache-control" header.

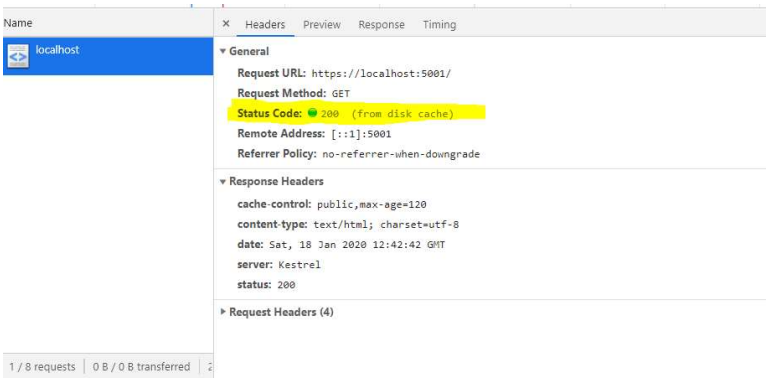
Now run the application and open developer console and then check cache headers are added to response headers as below.



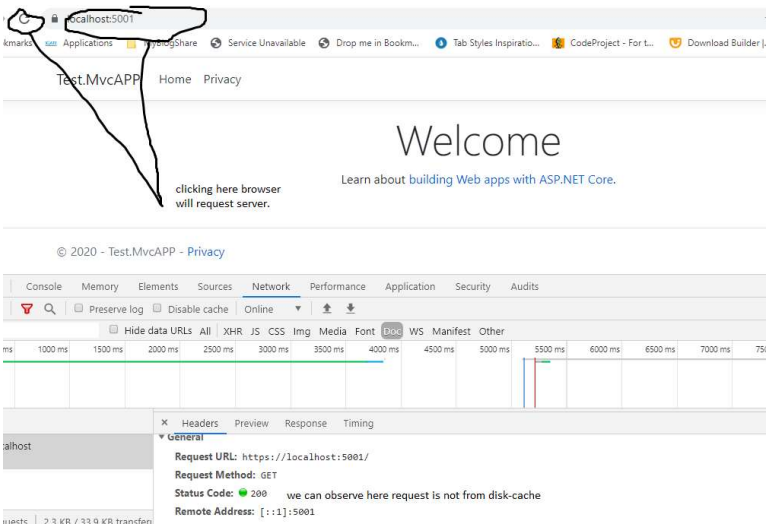
Now from the picture above "max-age=120" means the browser after receiving the first request with "cache-control" header value with "public" stores the data for 120 secs. So from the next request of the same page from the website will be loads from the disk cache as shown below



After clicking on the home page link from the website as shown in the above picture, we can observe page gets reloaded from the cache as below.



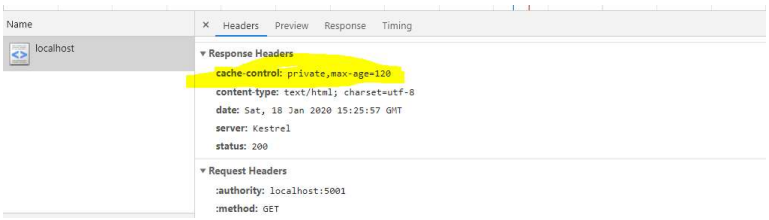
While Testing response caching always click on links within the websites because clicking Href on the website browser will check the previous response headers if they are cached so it will load the page from the cache if headers matched. If we reload the page or clicking the enter button browser search bar browser will request the page from the server.



Now update the ResponseCache Attribute location parameter with a value "ResponseCache.Client" as below

```
[ResponseCache(Duration =120, Location = ResponseCac
```

Now observe the response header where the "cache-control" value set to "private" as below.



Now update the ResponseCache Attribute location parameter with a value "ResponseCache.None" as below.

```
[ResponseCache(Duration =120, Location = ResponseCac
```

Now observe the response header where the "cache-control" value set to "no-cache" as below.

Now update the ResponseCache Attribute "NoStore" parameter with a value "true". By setting "true" to "NoStore" will override the "cache-control" value of the "Location " parameter as below.

```
[ResponseCache(Duration =120, Location = ResponseCac
```

Now observe the response header where the "cache-control" value set to "no-store" as below.

public vs private vs no-cache values of cache-control:

What is a proxy server? - Internet Service Providers may maintain some multiple numbers of proxy servers were to maintain and cache the data to fastly serve the information to their customers.

- **cache-control: public -** caching of response both in proxy servers and in client machines(browser).
- **cache-control: private -** caching of response only in the client machine.
- **cache-control: no-cache -** no caching, every request will be freshly fetched from the server.

VaryByHeader In ResponseCache Attribute:

VaryByHeader in Asp.Net Core represents the Http "Vary" Header. The "Vary" header uses the current request-response header value to be matched with future request header. If headers match future request-response fetch from the cache and if not match fetch response freshly from the server.

The best example to test Http "Vary" Header by populating it with "User-Agent". "User-agent" gives information like the type of client request to the server. In general "User-Agent" will be different for the device to device.

Example of how User-Agent works with response caching?

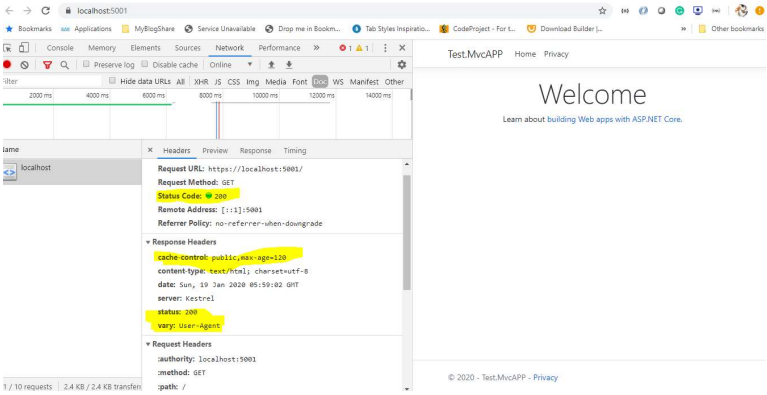
- In a website configure home page by enabling response cache attribute with "VaryByHeader" value set to "User-Agent".
- Now from desktop access, the home page will be rendered with response header with "Vary". After that click on the home page link to refresh the page since "User-Agent" cache enabled and headers match response will be loaded from the cache.
- Now open developers' tools and select the setting to mobile device options and then click on the home page

link on the website. Now the device type is changed so the "User-Agent" value will be changed so now the browser will send a fresh request to the server instead of loading a response from the cache.

Now let's do VaryByHeader practically. Update the ResponseCache Attribute "VarByHeader" parameter with the value "User-Agent" as below.

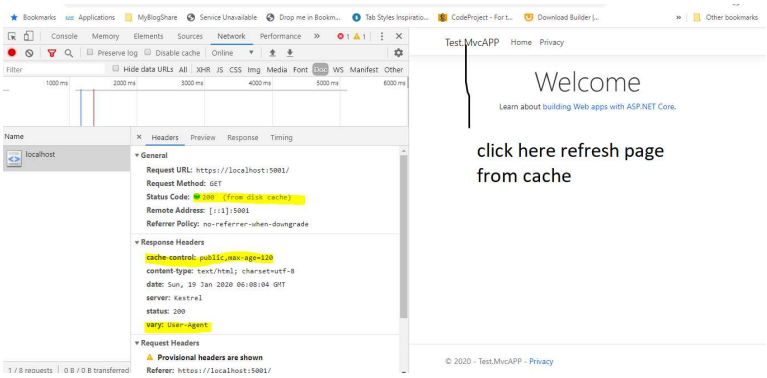
```
[ResponseCache(Duration =120,VaryByHeader = "User-Ag
```

Now run the application in desktop mode and the "Vary" response header will have value as "User-Agent" as below.

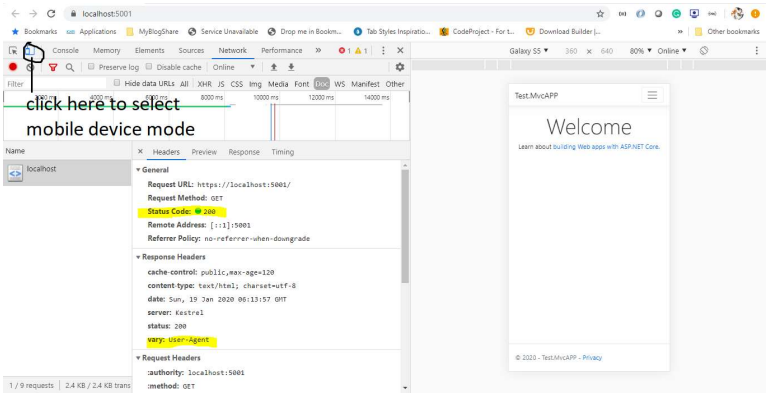


Request User-Agent looks as below.

Now click refresh the page by clicking the home link on the website in desktop device mode. Since user-agent won't change response will be rendered from the cache as below



Now from the developers' tool change settings to mobile devices and click on the home page link. Since device mode changed "User-agent" value will be changed, so browser request data from the server instead of cache as below.



We can observe user agent value will be changed as below.

شركة ايديال ستاندرد تطلق رحلة إلى 6
مدن مختلفة حول العالم في تجربة رقمية...
مع ايديال ستاندرد ستسافر حول العالم أونلاين Ad
وستكسب هدية أحد منتجاتنا عند التسجيل أونلاين.
Ideal Standard
معرفة المزيد

Cache Profile:

ResponseCache Attribute input parameters may be the same for almost all the action methods in an application. Asp.Net Core provides options like profile cache where all the parameter options are configured startup class with a name to that configuration, that profile name can be used inside the ResponseCache Attribute which removes duplication parameters setting in it.

```
public void ConfigureServices(IServiceCollection ser
{
 services.AddControllersWithViews(options => {
  options.CacheProfiles.Add("Profile1", new Microsof
```

```
  {
   Duration = 120,
   Location = Microsoft.AspNetCore.Mvc.ResponseCache
   NoStore = false
  });
  options.CacheProfiles.Add("Profile2", new Microsof
  {
   VaryByHeader = "User-Agent",
   Duration = 60,
   Location = Microsoft.AspNetCore.Mvc.ResponseCache
   NoStore = false
  });
 });
}
```

Here we have two different cache profiles with their respective settings. We can observe all the parameters we generally configured in ResponseCache Attribute are configured here inside the CacheProfile object."Profile2" cache settings represent caching based on VaryByHeader.

Now update the ResponseCache Attributes with its respective profile names as below.

```
[ResponseCache(CacheProfileName ="Profile1")]
public IActionResult Index()
{
 return View();
}

[ResponseCache(CacheProfileName = "Profile2")]
public IActionResult Test()
{
 return View();
}
```

Custom Middleware To Add Cache:

We can add response cache by writing middleware but this implementation by default adds response cache to any page on the application.

Startup.cs:

```
app.Use(async (context, next) =>
{
 context.Response.GetTypedHeaders().CacheControl =
  new Microsoft.Net.Http.Headers.CacheControlHeaderV
  {
   Public = true,
   MaxAge = TimeSpan.FromSeconds(60),
  };
 await next();
});
```

Here we can observe on every request, response context adding "CacheControl" object.

HomeController.cs:

```
public IActionResult Index()
{
  return View();
}
```

Combination Of Custom Cache Middleware And ResponseCache Attribute:

Yes, we can use both of them in our Asp.Net Core applications. But ResponseCache Attribute settings override the middleware settings. So it will help like globally every page will have cache information from the custom middleware and if we want to override those settings to any of the endpoint then we can easily override by implementing ResponseCache Attribute on the particular action method. For example "MaxAge" set to 180 secs on custom middleware and if we configure ResponseCache Attribute with "Duration"

set to 60 secs then a response will be cached for 60secs only.

Support Me!

Buy Me A Coffee PayPal Me

Wrapping Up:

Hopefully, this article will help to understand Response Cache implementation in the Dotnet core. I will love to have your feedback, suggestions, and better techniques in the comments section.

Refer:

Redis Cache Implementation In .NetCore Api(Redis Instance Docker)

Follow Me:

Facebook Twitter Pinterest

LABELS: ASP.NET CORE , ASP.NET CORE 3.1 , DOTNET-TECHNOLOGY **SHARE**

Comments

Enter your comment...

The fun way to learn phonics

A five-level series with animated chants and stories that will make ki
to learn.

Compass Publishing Learn M

The fun way to learn phonics

A five-level series with animated chants and stories that will make ki
to learn.

Compass Publishing Learn M

Popular posts from this blog

Endpoint Routing In Asp.Net Core

January 12, 2020

How Routing Works In Core 2.1 And Below Versions?:

In Asp.Net Core routing is configured

using app.UseRouter() or app.UseMvc() middleware.

app.UseMvc(routes => { routes.MapRoute(n ...

SHARE 1 COMMENT READ MORE

Asp.Net Core MVC Form Validation Techniques

December 08, 2019

Introduction: Form validations in any applications are like assures that a valid data is storing on servers. All programing frameworks have their own individual implementations for form validations. In Do ...

SHARE POST A COMMENT READ MORE

.NET Core MVC Application File Upload To Physical Location With Buffered Technique

February 19, 2020

Buffering Technique In File Upload: The server will use its Memory(RAM) or Disk Storage to save the files on receiving a file upload request from the client. Usage of Memory(RAM) or Disk depends on ...

SHARE POST A COMMENT READ MORE

Ionic Picker Sample Code In Angular

January 05, 2020

Introduction: Ionic Picker(ion-picker) is a popup slides up from the bottom of the device screen, which contains rows with selectable column separated items. The main building block of ion-picke ...

SHARE 5 COMMENTS READ MORE

Blazor WebAssembly Custom Authentication From Scratch

October 21, 2020

In this article, we are going to explore and implement custom authentication from the scratch. In this sample, we will use JWT authentication for user authentication. Main Building Blocks Of Bla ...

SHARE 1 COMMENT READ MORE

GraphQL API Integration In Asp.Net Core Application

November 30, 2019

Introduction: GraphQL is a query language for your API and a server-side runtime for executing queries by using a type system you define for your data. GraphQL can be integrated into any framew ...

SHARE 2 COMMENTS READ MORE

ASP.NET Core Web API Versioning

November 03, 2019

Introduction: An iteration and evolutionary changes of an ASP.NET Core Web API is handled by Versioning. Versioning of an API gives confidence to the clients which consumes API for a long time. Any ch ...

SHARE POST A COMMENT READ MORE

.Net Core HttpClient JSON Extension Methods Using System.Net.Http.Json Package

June 30, 2020

.Net Core 3.0 onwards Microsoft brought up a new package called System.Net.Http.Json. This new package provides JSON extension methods for HttpClient. These JSON extension methods ...

SHARE 1 COMMENT READ MORE

Blazor Server CRUD Operations

March 07, 2020

Introduction: Blazor Server is a web framework to develop server-side single-page applications. Blazor is made up of components with the combinations on C#, Html, CSS. Blazor Server is production ...

SHARE 1 COMMENT READ MORE

→

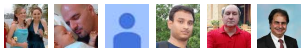
About Me

Naveen Bommidi

Web Technology learner

VISIT PROFILE

المتابعون (6)



متابعة

Archive

Labels

Translate

اختيار اللغة ▼

Part-3 Steps To Implement Google Authentication Into Existing Blazor WebAssembly Standalone Application
Part-2 Adding New User Into Database And Generating Custom JWT Token - (Blazor WebAssembly Standalone App Google Authentication)
Part-1 Blazor WebAssembly Standalone Application User Login Using Google Account
A Blazor WebAssembly CRUD Sample Using MudBlazor Components
A Small Guide On NestJS Queues

Report Abuse

Powered by Blogger

© Naveen Bommidi, Tech Seeker, 2019 - 2021