ASP.NET Core

Output caching with ASP.NET Core

Miňo Martiniak
.NET Insights

swipe

ASP.NET Core

Register output cache dependencies with the .AddOutputCache() extension to the DI container and register middleware with .UseOutputCache()

Program.cs

```
// Register the output cache dependencies 👇
builder.Services.AddOutputCache();

// ...

// Use the output cache middleware 👇
app.UseOutputCache();
```

Miňo Martiniak
.NET Insights

swipe

ASP.NET Core

Use the .CaheOutput() extension or the [OutputCache] attribute to mark the endpoint for caching. Now the output from your endpoint will be cached.

Program.cs

```
app.MapGet("/products/{id}", async (int id, IProductRepository repository)
    => await repository.GetProductAsync(id)).CacheOutput();

// or 👇

app.MapGet("/products/{id}", [OutputCache] async (int id, IProductRepository repository)
    => await repository.GetProductAsync(id));
```

Miňo Martiniak
.NET Insights

swipe

ASP.NET Core

You can add a base policy for all GET and HEAD endpoints using the .AddBasePolicy(), or use the .AddPolicy() to define a named policy for specific endpoints.

Program.cs

```
builder.Services.AddOutputCache(c ⇒
{
    // Define the default policy for all GET or HEAD requests 👇
    // In this case endpoint doesn't need to be decorated with [OutputCache]
    c.AddBasePolicy(builder ⇒
        builder.Expire(TimeSpan.FromSeconds(30)));

    // Define named policies for specific endpoints 👇
    c.AddPolicy("Expire20", builder ⇒
        builder.Expire(TimeSpan.FromSeconds(20)));
});

app.MapGet("/products", async (IProductRepository repository)
    ⇒ await repository.GetProductsAsync())
    .CacheOutput("Expire20");
```

Miňo Martiniak
.NET Insights

swipe

ASP.NET Core

If you mark your policy with tag, you can clear part of the cache if necessary. Use IOutputCacheStore to invalidate it.

Program.cs

```
builder.Services.AddOutputCache(c ⇒
{
    c.AddPolicy("ProductsPolicy", builder ⇒
        builder.Expire(TimeSpan.FromMinutes(20))
            .Tag("products", "all")); // Add tag 👈
});

app.MapPost("/products",
    async (Product product, IProductRepository repository, IOutputCacheStore cache, CancellationToken token) ⇒
{
    var newProduct = await repository.AddProductAsync(product);
    // Invalidate cache 👇
    await cache.EvictByTagAsync("products", token);

    return newProduct;
});
```

Miňo Martiniak
.NET Insights

swipe

ASP.NET Core

Cache revalidation is supported out of box. Just add ETag to the header and if the client sends the same value in the If-None-Match header the middleware will automatically return 304 Not Modified without body.

Program.cs

```
app.MapGet("/products/{id}", async (
    int id,
    HttpResponse response,
    IProductRepository repository) ⇒
{
    var product = await repository.GetProductAsync(id);

    // Add ETag header 👇
    response.Headers.ETag = $"\"{product.LastModified.Ticks}\"";

    return product;
}).CacheOutput();
```

Miňo Martiniak
.NET Insights

swipe

ASP.NET Core

By default, the cache is stored in the memory. For Redis caching, use the Microsoft.AspNetCore.OutputCaching.StackExchangeRedis package.

Program.cs

```
builder.Services.AddStackExchangeRedisOutputCache(options =>
{
    options.Configuration =
        builder.Configuration.GetConnectionString("MyRedisConStr");
    options.InstanceName = "SampleInstance";
});
```

Miňo Martiniak
.NET Insights

swipe

```
if (you.Enjoyed(this))
{
    // Cache this article in your memory
    you.CacheOutput(TimeSpan.FromDays(100));
    // and...
    you.Like(this);
    you.Share(this);
    // ...keep the knowledge flowing!
}

// Comment:
// Like a good cache, if this article saves you time,
// pass it on — it's how information stays
// 'fresh' in our network!
```

Miňo Martiniak
.NET Insights