

Output caching with ASP.NET Core



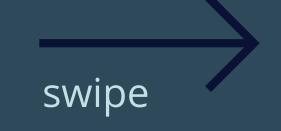




Register output cache dependencies with the .AddOutputCache() extension to the DI container and register middleware with .UseOutputCache()

```
// Register the output cache dependencies huilder.Services.AddOutputCache();
// ...
// Use the output cache middleware happ.UseOutputCache();
```







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





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.







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







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.

```
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();
```







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

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





```
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!
```



