29 SEPTEMBER 2018 / DOTNET

Simple ASP.NET Core MVC Razor Setup



With this article I will show how this website, www.cuemon.net, has been configured in ASP.NET Core MVC 2.1.
I will go through it, line by line, where there is a reference to Cuemon related code.

This is the `Startup.cs` file in my project which is invoked by convention by ASP.NET Core.

```
using Cuemon.AspNetCore;
using Cuemon.AspNetCore.Mvc.DependencyInjection;
using Cuemon.AspNetCore.Mvc.Filters.Cacheable;
using Cuemon.Homepage.Razor.TagHelpers;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using WBPA.WellKnownLocations;

namespace Cuemon.Homepage
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        public void ConfigureServices(IServiceCollection services)
        {
            services.AddOptions();
            services.Configure<CdnTagHelperOptions>
(Configuration.GetSection("CdnTagHelperOptions"));
            services.Configure<HttpCacheableOptions>(o =>
            {
                o.Filters.AddEntityTagHeader(etho =>
                {
                    etho.UseEntityTagResponseParser = true;
                });
            });
            services.AddCacheBusting();
            services.AddMvc(o =>
            {
                o.Filters.Add<HttpCacheableFilter>();
            }).SetCompatibilityVersion(CompatibilityVersion.Latest);
        }

        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            app.UseHttpsRedirection();
            app.UseStaticFiles();
            app.UseHostingEnvironmentHeader();
            app.UseCorrelationIdentifierHeader();
            app.UseWellKnownLocations();
            app.UseMvc();
        }
    }
}
```

Breakdown of ConfigureServices(IServiceCollection)

The first thing we do is configuring our services. Personally, I am a big fan of the Options Pattern (either by interface `IOptions<TOptions>` or delegate `Action<TOptions>`), and in this configuration, we are using the former way.

```
services.Configure<CdnTagHelperOptions>(Configuration.GetSection("CdnTagHelperOptions"));
```

What happens here is actually new functionality that will be released in a near future.
We are adding the options necessary for our production environment, that will be injected to or Razor views.

This is what the configuration looks like in `appsettings.Production.json`:

```
{
  "CdnTagHelperOptions": {
    "Scheme": "Relative",
    "BaseUrl": "nblcdn.net"
  }
}
```

And this is what the `_ViewImports.cshtml` looks like:

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@addTagHelper *, cuemon.net
```

On the next line, we are adding the options for `HTTP ETag Header` which will be injected to `HttpCacheableFilter`.
We are allowing dynamic parsing of the body.

```
services.Configure<HttpCacheableOptions>(o =>
{
    o.Filters.AddEntityTagHeader(etho =>
    {
        etho.UseEntityTagResponseParser = true;
    });
});
```

On the next line, we are adding `AssemblyCacheBusting` capabilities using the default shorthand. It basicly juts add a thumbprint to the earlier mentioned `Cdn*TagHelpers`. from the running assembly of the application. Since stylesheets changes rather often, we could consider using a `DynamicCacheBusting` instead.

```
services.AddCacheBusting();
```

On the last line, we are doing the actual configuration of MVC to use the earlier mentioned `HttpCacheableFilter`.

```
services.AddMvc(o =>
{
    o.Filters.Add<HttpCacheableFilter>();
}).SetCompatibilityVersion(CompatibilityVersion.Latest);
```

Breakdown of Configure(IApplicationBuilder, IHostingEnvironment)

Lastly, we are configuring our application to make use of some the features that are not strongly necessary, but can help in your everyday development with ASP.NET Core MVC.

First we add a HTTP header to the response, that reveals the current environment. Default header name is `X-Hosting-Environment`.

```
app.UseHostingEnvironmentHeader();
```

Next, and this is because if work with SOA in my daily work, we add a Correlation-ID to the response. Default header name is `X-Correlation-ID`.

```
app.UseCorrelationIdentifierHeader();
```

Lastly, and this is a sister project to Cuemon, we configure the application to use RFC 5785 that defines a path to `/.well-known/`. We do this as the website is using LetsEncrypt SSL.

```
app.UseWellKnownLocations();
```

Closing Words

This was a walkthrough of the somewhat simple setup of this website. I hope it brought something new for you to enjoy.

Code with passion 🔥