Building middleware

Write a middleware that sets the current culture based on a query string value

1. Start with the application you created in [Lab 2](2. Introduction to ASP.NET Core.md), or just create a new empty ASP.NET Core application

2. Open `Startup.cs`

3. Create an inline middleware that runs **before** the hello world delegate that sets the culture for the current request from the query string:

```
public void Configure(IApplicationBuilder app)
{
    app.Use((context, next) =>
    {
        var cultureQuery = context.Request.Query["culture"];
        if (!string.IsNullOrWhiteSpace(cultureQuery))
        {
            var culture = new CultureInfo(cultureQuery);

            CultureInfo.CurrentCulture = culture;
            CultureInfo.CurrentUICulture = culture;
        }

        // Call the next delegate/middleware in the pipeline
        return next();
    });

    app.Run(async (context) =>
    {
        await context.Response.WriteAsync(
            $"Hello {CultureInfo.CurrentCulture.DisplayName}");
    });
}
```

4. Run the app now and set the culture via the query string, e.g. http://localhost/?culture=no

Move the middleware to its own type

1. Create a new class in the application `RequestCultureMiddleware`

2. Add a constructor that takes a parameter `RequestDelegate next` and assigns it to a private field `private readonly RequestDelegate _next`

3. Add a method `public Task Invoke(HttpContext context)`

4. Copy the code from the inline middleware delegate in the application's `Startup.cs` file to the `Invoke` method you just created and fix the `next` method name

5. Your middleware class should now look something like this:

```
public class RequestCultureMiddleware
{
    private readonly RequestDelegate _next;
```

```
    public RequestCultureMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    public Task Invoke(HttpContext context)
    {
        var cultureQuery = context.Request.Query["culture"];
        if (!string.IsNullOrWhiteSpace(cultureQuery))
        {
            var culture = new CultureInfo(cultureQuery);

            CultureInfo.CurrentCulture = culture;
            CultureInfo.CurrentUICulture = culture;
        }

        return _next(context);
    }
}
```

6. At the bottom of the file, add a class that exposes the middleware via an extension method on `IApplicationBuilder`.

```
public static class RequestCultureMiddlewareExtensions
{
    public static IApplicationBuilder UseRequestCulture(
        this IApplicationBuilder builder)
    {
        return builder.UseMiddleware<RequestCultureMiddleware>();
    }
}
```

7. Back in the application's `Startup.cs` file, delete the inline middleware delegate

8. Add your new middleware class to the HTTP pipeline:

```
app.UseRequestCulture();
```

9. Run the application again and see that the middleware is now running as a class

Adding options to middleware

1. Create a class called `RequestCultureOptions.cs` with a `CultureInfo` property called DefaultCulture.

```
public class RequestCultureOptions
{
    public CultureInfo DefaultCulture { get; set; }
}
```

2. Add an overload to `UseRequestCulture` that takes those options and passes them into the `UseMiddleware<RequestCultureMiddleware>` call.

```
public static IApplicationBuilder UseRequestCulture(
    this IApplicationBuilder builder)
{
    return builder.UseRequestCulture(new RequestCultureOptions());
}

public static IApplicationBuilder UseRequestCulture(
    this IApplicationBuilder builder,
    RequestCultureOptions options)
```

```
{
    return builder.UseMiddleware<RequestCultureMiddleware>(options);
}
```

3. Change the `RequestCultureMiddleware` constructor to take the `RequestCultureOptions`.

```
public class RequestCultureMiddleware
{
    private readonly RequestDelegate _next;
    private readonly RequestCultureOptions _options;

    public RequestCultureMiddleware(RequestDelegate next,
        RequestCultureOptions options)
    {
        _next = next;
        _options = options;
    }
    ...
}
```

4. Change the `Invoke` method of the middleware to use the DefaultCulture from options if none specified on the query string

```
public Task Invoke(HttpContext httpContext)
{
    CultureInfo requestCulture = null;

    var cultureQuery = httpContext.Request.Query["culture"];
    if (!string.IsNullOrWhiteSpace(cultureQuery))
    {
        requestCulture = new CultureInfo(cultureQuery);
    }
    else
    {
        requestCulture = _options.DefaultCulture;
    }

    if (requestCulture != null)
    {
        CultureInfo.CurrentCulture = requestCulture;
        CultureInfo.CurrentUICulture = requestCulture;
    }

    return _next(httpContext);
}
```

5. Set the fallback culture in `Startup.cs` `Configure` method to some default value:

```
app.UseRequestCulture(new RequestCultureOptions
{
    DefaultCulture = new CultureInfo("en-GB")
});
```

6. Run the application again and see the default culture when no query string is specified matches the one configured.

Read request culture configuration from a file

1. Add a constructor to the application's `Startup.cs`

2. Create a new `Configuration` object in the constructor and assign it to a new private class field `IConfiguration _configuration`

3. Add a reference to the `Microsoft.Extensions.Configuration.Json` package in the application's `project.json` file

4. Back in the `Startup.cs`, add a call to `.AddJsonFile("config.json")` immediately after the creation of the `Configuration` object (inline, chained method). It should now look like the following:

```
public class Startup
{
    private readonly IConfiguration _configuration;

    public Startup()
    {
        var configuration = new ConfigurationBuilder()
            .AddJsonFile("config.json")
            .Build();

        _configuration = configuration;
    }
    ...
}
```

5. Add a new JSON file to the project called `config.json`

6. Add a new key/value pair to the `config.json` file: `"culture": "en-US"`

7. Change the code in `Startup.cs` to set the default culture using the configuration system:

```
app.UseRequestCulture(new RequestCultureOptions
{
    DefaultCulture = new CultureInfo(_configuration["culture"] ?? "en-GB")
});
```

1. Run the application and the default culture should be set from the configuration file.
2. Change the culture in the `config.json` file and refresh the page (without changing any other code). Note that the message hasn't changed as the configuration was only read when the application was started.
3. Go back to Visual Studio and touch and save the `Startup.cs` file to force the process to restart
4. Go back to the browser now and refresh the page and it should show the updated message

Flowing options from dependency injection system to middleware

1. Add the `Microsoft.Extensions.Options` package to `project.json`.

2. Change the `RequestCultureMiddleware` constructor to take `IOptions<RequestCultureOptions>` instead of `RequestCultureOptions`:

```
public RequestCultureMiddleware(RequestDelegate next,
    IOptions<RequestCultureOptions> options)
{
    _next = next;
    _options = options.Value;
}
```

3. Change the `UseRequestCulture` extension methods to both call `UseMiddleware<RequestCultureMiddleware>`. The overload taking `RequestCultureOptions` should wrap it in an `IOptions<RequestCultureOptions>` by calling `Options.Create(options)`:

```
public static class RequestCultureMiddlewareExtensions
{
    public static IApplicationBuilder UseRequestCulture(
```

```
        this IApplicationBuilder builder)
    {
        return builder.UseMiddleware<RequestCultureMiddleware>();
    }

    public static IApplicationBuilder UseRequestCulture(
        this IApplicationBuilder builder,
        RequestCultureOptions options)
    {
        return builder.UseMiddleware<RequestCultureMiddleware>(
            Options.Create(options));
    }
}
```

4. In `Startup.cs` change the `UseRequestCulture` middleware to not take any arguments:

```
app.UseRequestCulture();
```

5. In `Startup.cs` add a `ConfigureServices(ISeviceCollection services)` method and add a line that configures the culture using the `services.Configure<RequestCultureOptions>` method:

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<RequestCultureOptions>(options =>
    {
        options.DefaultCulture
            = new CultureInfo(_configuration["culture"] ?? "en-GB");
    });
}
```

6. Run the application and see that options are now being configured from the dependency injection system.