**Cookies and Consent in ASP .NET Core**

By Shahed C on January 21, 2019

[2 Replies](https://wakeupandcode.com/cookies-and-consent-in-asp-net-core/#comments)

This is the **third** of a new [series of posts](https://wakeupandcode.com/aspnetcore/#aspnetcore2019) on ASP .NET Core for 2019. In this series, we’ll cover 26 topics over a span of 26 weeks from January through June 2019, titled **A-Z of ASP .NET Core!**

** A – Z of ASP .NET Core!**

**In this Article:**

* C is for Cookies and Consent
* Browser Storage
* Partial Views for your cookie message
* Customizing your message
* Startup Configuration
* References

**C is for Cookies and Consent**

In this article, we’ll continue to look at the (in-progress) [NetLearner](https://wakeupandcode.com/netlearner-asp-net-core-internet-learning-helper/) application, which was generated using one of the standard ASP .NET Core web app project (2.2) templates. Specifically, let’s take a look at how the template makes it very easy for you to store cookies and display a cookie policy.

**NOTE**: *The way cookies are handled in the project templates may change with each new release of ASP .NET Core.*

Unless you’ve been living under a rock in the past year or so, you’ve no doubt noticed all the GDPR-related emails and website popups all over the place. Whether or not you’re required by law to disclose your cookie policies, it’s good practice to reveal it to the end user so that they can choose to accept your cookies (or not).

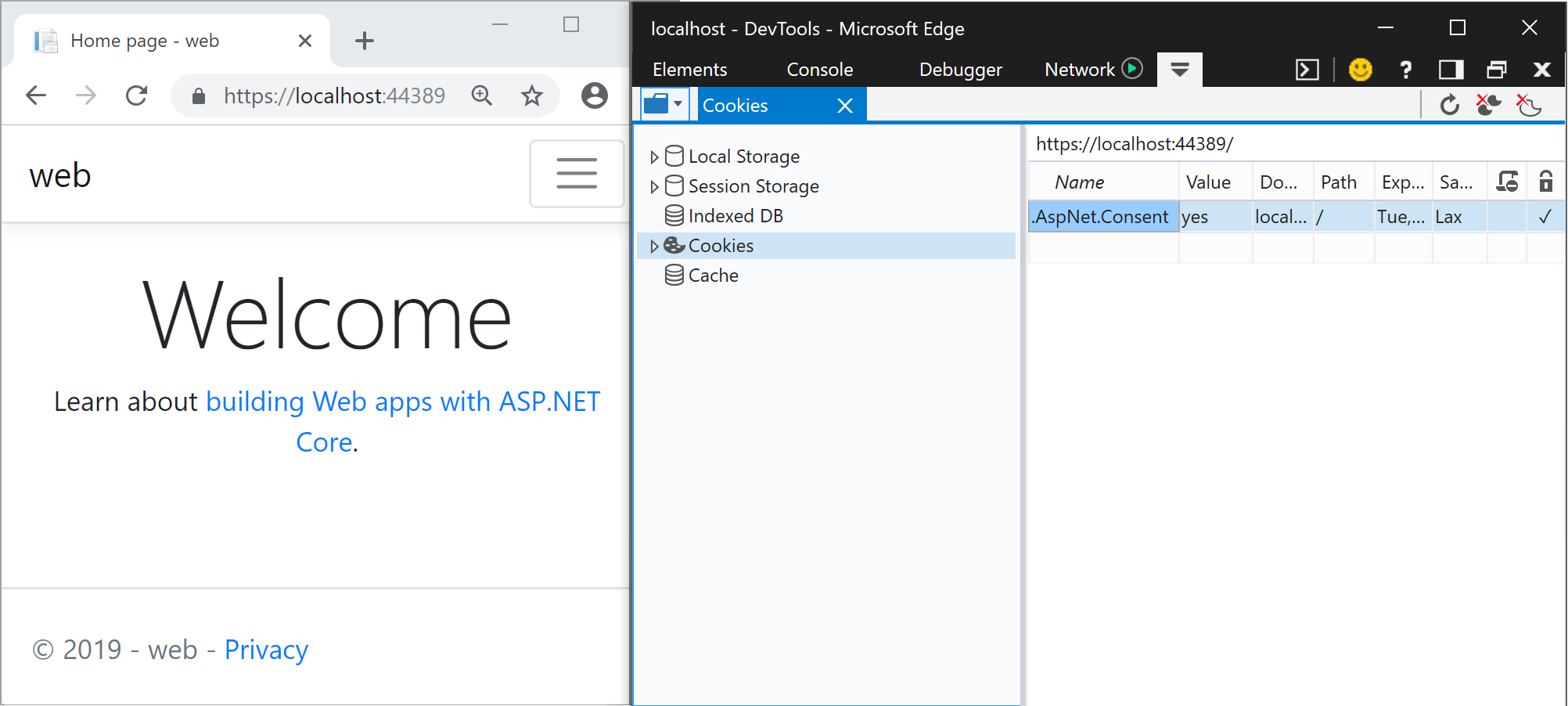
**Browser Storage**

As you probably know, cookies are attached to a specific browser installation and can be deleted by a user at an any time. Some  new developers may not be aware of where these cookies are actually stored.

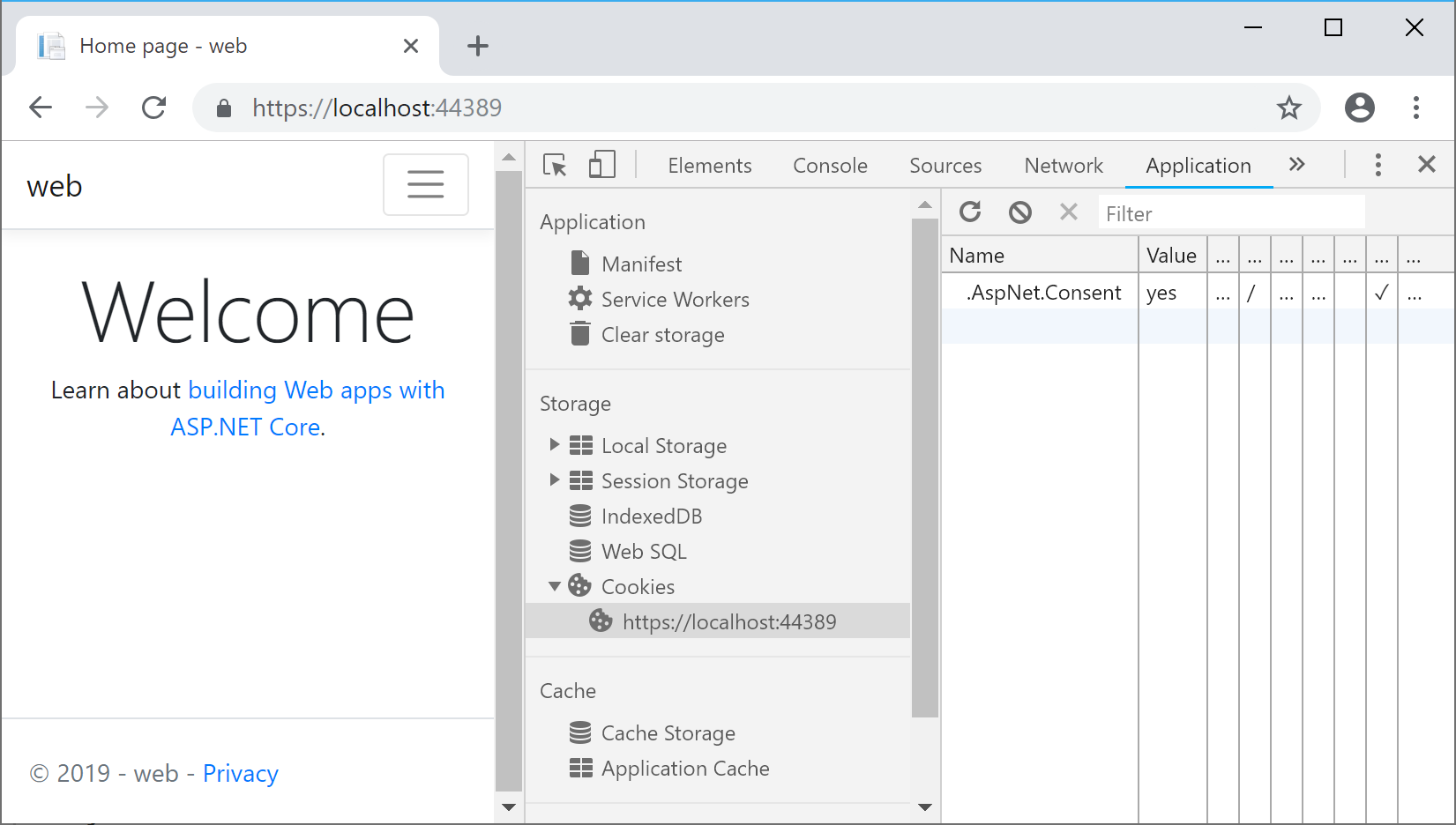
Click F12 in your browser to view the Developer Tools to see cookies grouped by website/domain.

* In Edge/Firefox, expand Cookies under the Storage tab.
* In Chrome, expand Storage | Cookies under the Application tab .

See screenshots below for a couple of examples how AspNet.Consent in stored, along with a boolean Yes/No value:

[](https://wakeupandcode.com/wp-content/uploads/2019/01/cookies-edge.png)

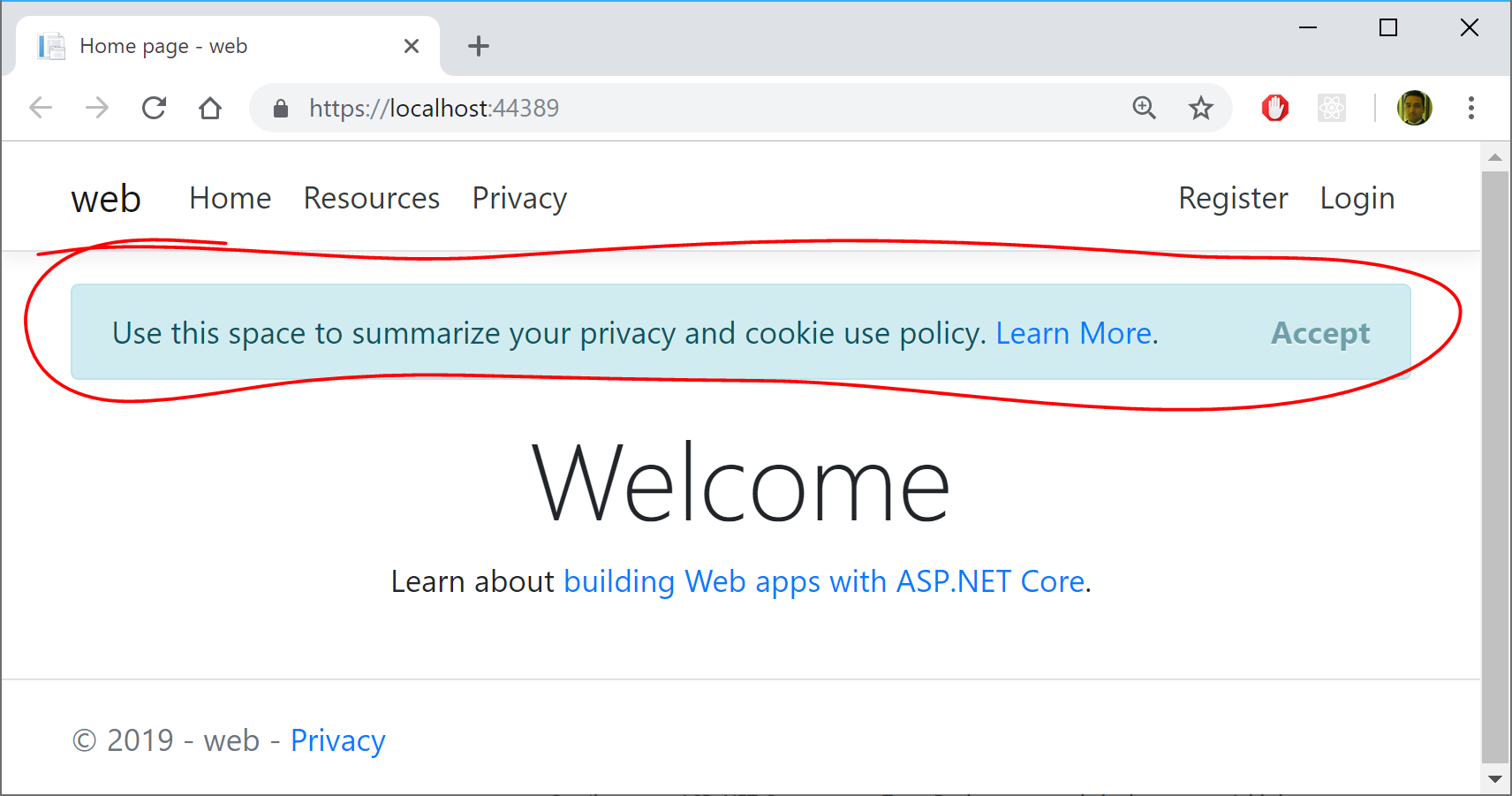
Cookies in Edge

[](https://wakeupandcode.com/wp-content/uploads/2019/01/cookies-chrome.png)

Cookies in Chrome

**Partial Views for your cookie message**

The first time you launch a new template-generated ASP .NET Core web app, you should see a cookie popup that appears on every page that can be dismissed by clicking Accept. Where does it come from? Let’s explore the code to dig in a little further.

[](https://wakeupandcode.com/wp-content/uploads/2019/01/cookie-popup.png)

First, take a look at the [\_CookieConsentPartial.cshtml](https://github.com/shahedc/NetLearner/blob/master/web/Pages/Shared/_CookieConsentPartial.cshtml) partial view, from which CSS class names and [accessibility-friendly role attributes](https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/ARIA_Techniques/Using_the_alert_role) have been removed for brevity in the snippet below. For Razor Pages (in this example), this file should be in the **/Pages/Shared/** folder by default. For MVC, this file should be in the **/Views/Shared/** folder by default.

@using Microsoft.AspNetCore.Http.Features  
  
@{  
 var consentFeature = Context.Features.Get<ITrackingConsentFeature>();  
 var showBanner = !consentFeature?.CanTrack ?? false;  
 var cookieString = consentFeature?.CreateConsentCookie();  
}  
  
@if (showBanner)  
{  
 <div id="cookieConsent">  
 <!-- CUSTOMIZED MESSAGE IN COOKIE POPUP -->  
 <button type="button" data-dismiss="alert" data-cookie-string="@cookieString">  
 <span aria-hidden="true">Accept</span>  
 </button>  
 </div>  
 <script>  
 (function () {  
 var button = document.querySelector("#cookieConsent button[data-cookie-string]");  
 button.addEventListener("click", function (event) {  
 document.cookie = button.dataset.cookieString;  
 }, false);  
 })();  
 </script>  
}

This partial view has a combination of server-side C# code and client-side HTML/CSS/JavaScript code. First, let’s examine the C# code at the very top:

1. The using statement at the top mentions the **Microsoft.AspNetCore.Http.Features** namespace, which is necessary to use **ITrackingConsentFeature.**
2. The local variable **consentFeature** is used to get an instance **ITrackingConsentFeature** (or null if not present).
3. The local variable **showBanner** is used to store the boolean result from the proeperty **consentFeature.CanTrack** to check whether the user has consented or not.
4. The local variable **cookieString** is used to store the “cookie string” value of the created cookie after a quick call to **consentFeature.CreateConsentCookie**().
5. The @**if** block that follows only gets executed if **showBanner** is set to true.

Next, let’s examine the HTML that follows:

1. The **cookieConsent** <div> is used to store and display a customized message for the end user.
2. This <div> also displays an **Accept** <button> that dismisses the popup.
3. The  **data-dismiss** attribute ensures that the modal popup is closed when you click on it. This feature is available because we are using Bootstrap in our project.
4. The data- attribute for “**data-cookie-string**” is set using the server-side variable value for @**cookieString**.

The full value for **cookieString** may look something like this, beginning with the **.AspNet.Consent** boolean value, followed by an expiration date.

.AspNet.Consent=yes; expires=Tue, 21 Jan 2020 01:00:47 GMT; path=/; secure; samesite=lax

Finally, let’s examine the JavaScript that follows within a <**script**> tag:

1. Within the **<script>** tag, an anonymous function is defined and invoked immediately, by ending it with **();** after it’s defined.
2. A **button** variable is defined to represent the HTML button, by using an appropriate **querySelector** to retrieve it from the DOM.
3. An eventListener is added to respond to the button’s onclick event.
4. If accepted, a new cookie is created using the button’s aforementioned **cookieString** value.

To use the partial view in your application, simply insert it into the [\_Layout.cshtml](https://github.com/shahedc/NetLearner/blob/master/web/Pages/Shared/_Layout.cshtml) page that is used by all your pages. The partial view can be inserted above the call to **RenderBody**() as shown below.

<div class="container">  
 <partial name="\_CookieConsentPartial" />  
 <main role="main" class="pb-3">  
 @RenderBody()  
 </main>  
</div>

In an MVC application, the partial view can be inserted the same way, using the <partial> tag helper.

**Customizing your message**

You may have noticed that there is only an Accept option in the default cookie popup generated by the template’s Partial View. This ensures that the only way to store a cookie with the user’s consent is to click Accept in the popup.

You be wondering whether you should also display a Decline option in the cookie popup. But that would be a bad idea, because that would require you to store the user’s “No” response in the cookie itself, thus going against their wishes. If you wish to allow the user to *withdraw* consent at a later time, take a look at the GrantConsent() and WithdrawConsent() methods provided by [ITrackingConsentFeature](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.http.features.itrackingconsentfeature).

But you can still change the message in the cookie popup and your website’s privacy policy. To change the cookie’s displayed message, simply change the text that appears in the \_CookieConsentPartial.cshtml partial view, within the <div> of the client-side HTML. In the excerpt shown in the previous section, this region is identified by the **<!– CUSTOMIZED MESSAGE IN COOKIE POPUP –>** placeholder comment.

<div id="cookieConsent">  
 <!-- CUSTOMIZED MESSAGE IN COOKIE POPUP -->  
 <button type="button" data-dismiss="alert" data-cookie-string="@cookieString">  
 <span aria-hidden="true">Accept</span>  
 </button>  
 </div>

Your message text is also a great place to provide a link to your website’s privacy policy. In the Razor Pages template, the <a> link is generated using a tag helper shown below. The **/Privacy** path points to the [Privacy.cshtml](https://github.com/shahedc/NetLearner/blob/master/web/Pages/Privacy.cshtml) Razor page in the /Pages folder.

<a asp-page="/Privacy">Learn More</a>

In a similar MVC application, you would find the Privacy.cshtml view within the **/Views/Home/** folder, accessible via the Home controller’s Privacy() action method. In the MVC template, the <a> is link is generated using the following tag helper:

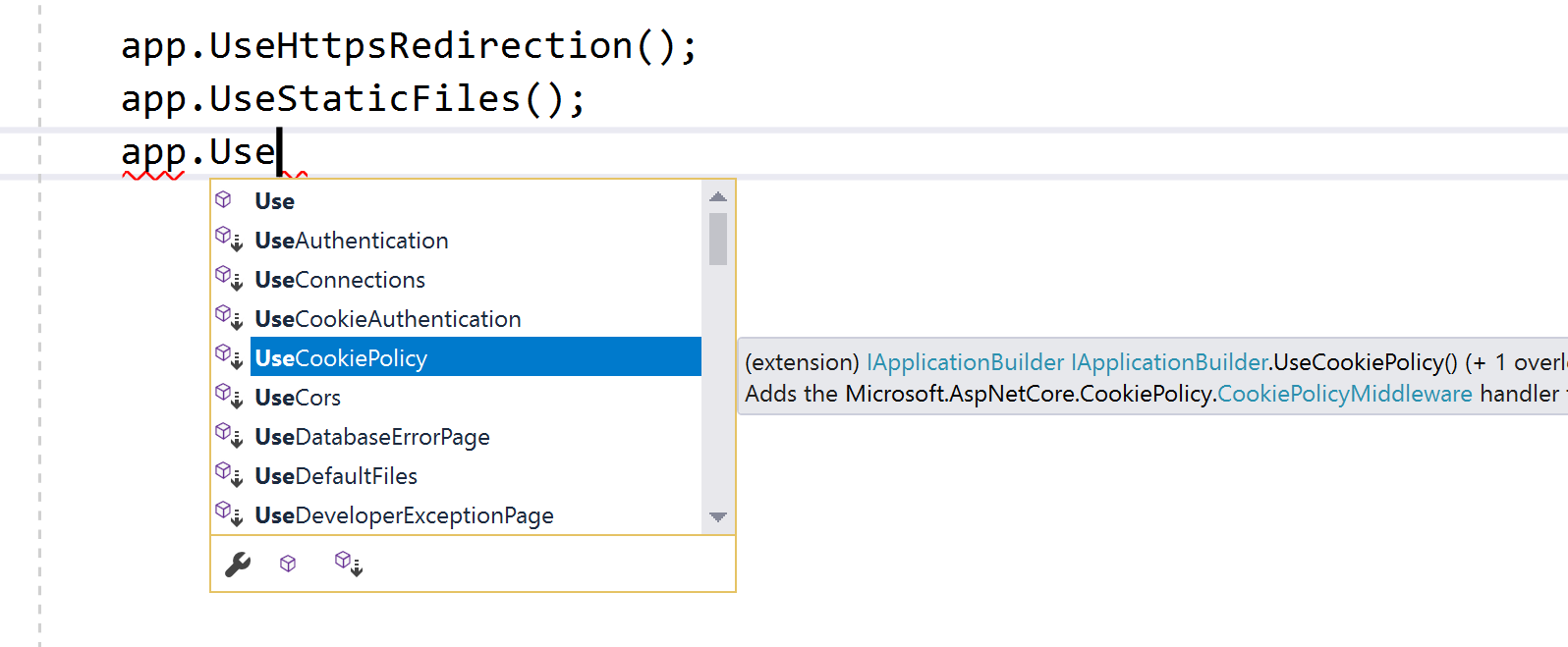
<a asp-area="" asp-controller="Home" asp-action="Privacy">Learn More</a>

**Startup Configuration**

None of the above would be possible without the necessary configuration. The cookie policy can be used by simply calling the extension method **app.UseCookiePolicy**() in the **Configure**() method of your [Startup.cs](https://github.com/shahedc/NetLearner/blob/master/web/Startup.cs) file.

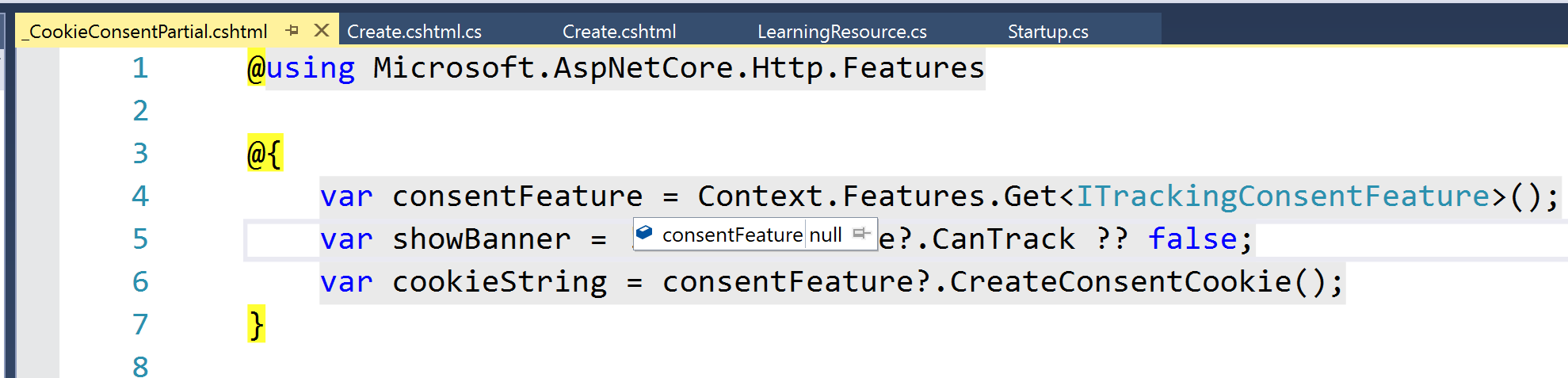
public void Configure(IApplicationBuilder app, IHostingEnvironment env)  
 {  
 ...  
 app.UseCookiePolicy();  
 ...  
 app.UseMvc();  
 }

According to the official documentation, this “*Adds the Microsoft.AspNetCore.CookiePolicy.CookiePolicyMiddleware handler to the specified Microsoft.AspNetCore.Builder.IApplicationBuilder, which enables cookie policy capabilities.*”  
The cool thing about ASP .NET Core middleware is that there are many IApplicationBuilder extension methods for the necessary Middleware components you may need to use. Instead of hunting down each Middleware component, you can simply type **app.Use** in the **Configure**() method to discover what is available for you to use.

[](https://wakeupandcode.com/wp-content/uploads/2019/01/cookies-use-extension.png)

If you remove the call to **app.UseCookiePolicy**(), this will cause the aforementioned **consentFeature** value to be set to null in the C# code of your cookie popup.

var consentFeature = Context.Features.Get<ITrackingConsentFeature>();

[](https://wakeupandcode.com/wp-content/uploads/2019/01/cookie-consentfeature-null.png)

There is also some minimal configuration that happens in the **ConfigureServices**() method which is called *before* the **Configure**() method in your Startup.cs file.

public void ConfigureServices(IServiceCollection services)  
{  
 services.Configure<CookiePolicyOptions>(options =>  
 {  
 // This lambda determines whether user consent for non-essential cookies is needed for a given request.  
 options.CheckConsentNeeded = context => true;  
 options.MinimumSameSitePolicy = SameSiteMode.None;   
 });   
 ...  
}

The above code does a couple of things:

1. As explained by the comment, the lambda (context => true) “determines whether user consent for non-essential cookies is needed for a given request” and then the **CheckConsentNeeded** boolean property for the options object is set to true or false.
2. The property **MinimumSameSitePolicy** is set to SameSiteMode.None, which is an enumerator with the following possible values:

* None = 0
* Lax = 1
* Strict = 2

From the [official documentation](https://docs.microsoft.com/en-us/aspnet/core/security/authentication/cookie) on cookie authentication, *“When set to SameSiteMode.None, the cookie header value isn’t set.* ***Note that Cookie Policy Middleware might overwrite the value that you provide****. To support OAuth authentication, the default value is SameSiteMode.Lax.”* This explains why even though the value is initially set to None in the **ConfigureServices**() method, using the Middleware in the **Configure**() method causes the “samesite” value to be set to “lax” in the cookiestring we observed earlier.

.AspNet.Consent=yes; expires=Tue, 21 Jan 2020 01:00:47 GMT; path=/; secure; samesite=lax

**References**

* General Data Protection Regulation (GDPR) support in ASP.NET Core: <https://docs.microsoft.com/en-us/aspnet/core/security/gdpr>
* Use cookie authentication without ASP.NET Core Identity: <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/cookie>
* ITrackingConsentFeature Interface (Microsoft.AspNetCore.Http.Features): <https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.http.features.itrackingconsentfeature?view=aspnetcore-2.2>
* HTMLElement.dataset: <https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/dataset>
* Using the alert role: <https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/ARIA_Techniques/Using_the_alert_role>
* HTML DOM querySelector() Method: <https://www.w3schools.com/jsref/met_document_queryselector.asp>

This entry was posted in [ASP.NET](https://wakeupandcode.com/category/asp-net/), [Learn](https://wakeupandcode.com/category/learn/), [Web Development](https://wakeupandcode.com/category/web-development/) and tagged [.net](https://wakeupandcode.com/tag/net/), [.NET Core](https://wakeupandcode.com/tag/net-core/), [ASP.NET](https://wakeupandcode.com/tag/asp-net/), [ASP.NET Core](https://wakeupandcode.com/tag/asp-net-core/), [Azure](https://wakeupandcode.com/tag/azure/), [cookies](https://wakeupandcode.com/tag/cookies/), [Visual Studio](https://wakeupandcode.com/tag/visual-studio/), [web development](https://wakeupandcode.com/tag/web-development/) on [January 21, 2019](https://wakeupandcode.com/cookies-and-consent-in-asp-net-core/) by [Shahed C](https://wakeupandcode.com/author/admin/).

**Post navigation**

[← Blazor Full-Stack Web Dev in ASP .NET Core](https://wakeupandcode.com/blazor-full-stack-web-dev-in-asp-net-core/) [Deploying ASP .NET Core to Azure App Service →](https://wakeupandcode.com/deploying-asp-net-core-to-azure-app-service/)

**2 thoughts on “Cookies and Consent in ASP .NET Core”**

1. Pingback: [Dew Drop - January 22, 2019 (#2882) - Morning Dew](https://www.alvinashcraft.com/2019/01/22/dew-drop-january-22-2019-2882/)
2. Pingback: [Middleware in ASP .NET Core | Wake Up And Code!](https://wakeupandcode.com/middleware-in-asp-net-core/)

**Leave a Reply**

This site uses Akismet to reduce spam. [Learn how your comment data is processed](https://akismet.com/privacy/).