**Learning Pure ASP.NET Core Security**

**Step 1**

Create Blank Solution

LearningPureASP.NETCoreSecurity.sln

Add: New Solution Folder:

Step 1

Create Folder:

Step\_01

In ‘Step 1’ Solution Folder:

Add: New Project… (In Step\_01 Folder)

ASP.NET Core Empty

Project Name: Client

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Fix Files!

Simplify ‘launchSettings.json’ file in ‘Properties’ folder and change the default port to 5000.

{

"profiles": {

"https": {

"launchBrowser": true,

"dotnetRunMessages": true,

"commandName": "Project",

"applicationUrl": "https://localhost:5000",

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

}

}

}

**Step 2**

Create Folder: Pages

Create Pages in ‘Pages’ Folder:

Index

MySecuredPage

Create Folder: ‘Account’

Create Pages in ‘Account’ Folder:

Login

Logout

Register

AccessDenied

Create Folder: Controllers

Create Controllers in ‘Controllers’ Folder:

‘MyApiController’

Create Some Actions:

Get

Post

‘MyController’

Create some Actions:

Index

Create View

MySecuredAction

Create View

Update Index.cshtml (Razor Page) and add some menu items and JavaScript code.

**Step 3**

In ‘Controllers’ Folder:

In ‘MyController.cs’ File:

Secure ‘MySecuredAction’ action by ‘Authorize’ attribute.

In ‘MyApiController.cs’ File:

Secure ‘Post’ action by ‘Authorize’ attribute.

In ‘Pages’ Folder:

Secure Some Pages with ‘Authorize’ attribute:

MySecuredPage

Account\Logout

Account\AccessDenied

**نکته مهم:** در ASP.NET Core MVC و ASP.NET Core Web API می‌توانیم Authorize Attribute را هم برای Controller و هم برای تک تک Action ها قرار دهیم، ولی در ASP.NET Core Razor Pages، فقط می‌توانیم این Attribute را برای Class قرار داده و امکان قرار دادن آن برای Handler ها وجود ندارد!

In Step 3, Client 🡪 Set as startup project

Run the application and check all end points!

با اجرای برنامه و در صورتی که بخواهیم وارد صفحاتی بشویم که برای Action و یا Handler آن‌ها از Attribute ای به نام Authorize استفاده کرده باشیم، با خطای ذیل مواجه می‌شویم!



برای حل این مشکل و بنابر توصیه مندرج در خطای فوق، باید UseAuthorization را بعد از UseRouting و قبل از هرگونه UseEndpoints قرار دهیم!

ولی ابتدا می‌خواهیم به اشتباه، دستور UseAuthorization را قبل از UseRouting قرار دهیم!

مجددا برنامه را اجرا کرده و سعی می‌کنیم که وارد صفحه MySecuredPage شویم.

مجددا، دقیقا همان پیغام خطای فوق را خواهیم گرفت!

حال دستور UseAuthorization را بعد از دستور UseRouting قرار می‌دهیم و مجددا برنامه را اجرا کرده و وارد صفحه MySecuredPage می‌شویم.

این بار با خطای دیگری مواجه می‌شویم! خطایی که اعلام می‌کند که ما هیچ‌گونه Authentication Scheme هایی را مشخص نکرده‌ایم!



برای حل این مشکل، قبل از دستور builder.Build() از دستور AddAuthentication و به شکل ذیل استفاده می‌کنیم:

builder.Services

.AddAuthentication();

اگر مجددا برنامه را اجرا نماییم، مجددا با همان خطای قبلی مواجه می‌شویم! چرا که هنوز Default Scheme مورد نظر را انتخاب نکرده‌ایم!

لذا برای حل این مشکل، یکی از دستورات ذیل را می‌نویسیم:

builder.Services

.AddAuthentication(defaultScheme: Microsoft.AspNetCore

.Authentication.Cookies.CookieAuthenticationDefaults.AuthenticationScheme)

.AddCookie()

;

OR

builder.Services

.AddAuthentication()

.AddCookie()

;

باید دقت داشته باشیم که اگر به جای دستورات فوق، هر یک از دو دستور ذیل را بنویسیم، مجددا با همان خطای قبل مواجه خواهیم شد:

builder.Services

.AddAuthentication(defaultScheme: "Cookies")

;

OR

builder.Services

.AddAuthentication(defaultScheme: Microsoft.AspNetCore

.Authentication.Cookies.CookieAuthenticationDefaults.AuthenticationScheme)

;

نکته: به پارامتر ورودی Login باید دقت داشته باشیم! زمانی که مستقیما وارد صفحه Login می‌شویم، این پارامتر null می‌باشد، ولی زمانی که Login نکرده باشیم و بخواهیم به صفحه‌ای برویم که Authorize Attribute داشته باشد، سیستم به طور خودکار ما را به صفحه Login، اصطلاحا Redirect می‌کنم و پارامتر ReturnUrl مشخص می‌کند که کاربر می‌خواسته وارد چه صفحه‌ای شود!

**Step 4**

**قسمت اول**

در صفحه (Page) اول یعنی Index.cshtml کد ذیل را می‌نویسیم:

@if (User is not null && User.Identity is not null && User.Identity.IsAuthenticated)

{

<b>Welcome @(User.Identity.Name)!</b>

<hr />

}

یک ViewModel مناسب برای Login ایجاد می‌کنیم:

Create Folder: ‘ViewModels’

Create Folder: ‘Account’

Create File: ‘LoginViewModel.cs’

در فایل Program.cs نیز، دستور ذیل را می‌نویسیم و باید دقت داشته باشیم که ترتیب و مکان نوشتن آن بسیار اهمیت دارد:

// Where do you want to go?

app.UseRouting();

// New - Who are you?

app.UseAuthentication();

// What do you allowed to access?

app.UseAuthorization();

// let me lead you to that endpoint!

app.MapRazorPages();

حال صفحه Login را چه از نظر ظاهر و چه از نظر Code تکمیل می‌کنیم:

In Folder: ‘Pages’

In Folder: ‘Account’

Update Files:

Login.cshtml

Login.cshtml.cs

برنامه را اجرا می‌کنیم و با موفقیت Login می‌کنیم، ولی در کمال تعجب زمانی که وارد صفحه MySecuredPage می‌شویم، مجددا ما را به صفحه Login اصطلاحا Redirect می‌کند! و اگر از طریق Inspect و قسمت Application وارد قسمت Cookie شویم، خواهیم دید که هیچ Cookie ایجاد نشده است! و در ضمن در صفحه اول نیز عبارت Welcome ظاهر نمی‌شود!

**قسمت دوم**

علت این مشکل آن است که ما در زمان خلق شیء مربوط به Identity، هیچ Scheme ای را مشخص نکرده‌ایم! البته این مشکل از دات نت ۳.۱ به بعد به دلیل تغییرات به‌وجود آمده است، لذا کد ذیل را فایل Login.cshtml.cs ویرایش می‌کنیم:

//var identity =

// new System.Security.Claims.ClaimsIdentity(claims: claims);

var identity =

new System.Security.Claims.ClaimsIdentity

(claims: claims, authenticationType: Microsoft.AspNetCore

.Authentication.Cookies.CookieAuthenticationDefaults.AuthenticationScheme);

حال مجددا برنامه را اجرا کرده، Login می‌کنیم و مجددا وارد صفحه MySecuredPage می‌شویم، و خواهید دید که دیگر مشکلی وجود ندارد! و علاوه بر این فایل Cookie نیز به درستی ساخته شده است!

در جاهای دیگری نیز می‌توانیم به صراحت Scheme را اعلام نماییم، ولی این‌که جای دیگری معرفی نکرده‌ایم و سیستم به درستی کار می‌کند آن است که Scheme ذیل Scheme پیش‌فرض می‌باشد:

Microsoft.AspNetCore.Authentication.Cookies

.CookieAuthenticationDefaults.AuthenticationScheme

با Login کردن، یک فایل Cookie ایجاد می‌شود و با Logout کردن، این فایل Cookie حذف می‌گردد.

**قسمت سوم**

با توجه به این‌که ممکن است به هر دلیلی بخواهیم Scheme پیش‌فرض را تغییر داده و یا تغییر نام دهیم و ممکن است که دچار مشکل شویم، ابتدا در پوشه‌ای به نام Infrastructure، پوشه‌ای به نام Security ایجاد کرده و در داخل آن کلاسی Static‌ای به نام Constants ایجاد کرده و در داخل کلاس یک Const به نام DefaultScheme ایجاد می‌کنیم و سپس در همه جا به صراحت اعلام می‌کنیم که می‌خواهیم با این Scheme کار کنیم:

In Program.cs:

builder.Services

.AddAuthentication(defaultScheme:

Infrastructure.Authentication.DefaultScheme)

.AddCookie(authenticationScheme:

Infrastructure.Authentication.DefaultScheme)

;

In Login.cshtml.cs:

var identity =

new System.Security.Claims.ClaimsIdentity(claims: claims,

authenticationType: Infrastructure.Authentication.DefaultScheme);

await HttpContext.SignInAsync(scheme: Infrastructure

.Authentication.DefaultScheme, principal: claimsPrincipal);

حال اگر در فایل Constants.cs، مقدار DefaultScheme را به مثلا Googooli تغییر دهیم، باز هم، همه چیز به خوبی کار می‌کند!

**Step 5**

در صفحه اول، منوها را به گونه‌ای اصلاح می‌کنیم که گزینه‌هایی که نیاز به Login دارند، تا وقتی که هنوز Login نکرده‌ایم ظاهر نشوند!

**ایجاد یک صفحه به نام Admin Page**

این صفحه را به گونه‌ای ایجاد می‌کنیم که برای روئت آن توسط کاربر، نه تنها کاربر باید Login کرده باشد! بلکه باید نقش Admin هم داشته باشد!

در صفحه Login، گزینه Remember Me را نیز فعال می‌کنیم.