**Section05 Authentication and Authorization Extra**

**Notes:-**

**1-there are by default three layers in the authorization layer**

**A-Authorization Requirement**

**B-Authorization Policy**

**C-Authorization Handler**

**Steps:-**

**1-we can apply authorization check on custom action or with view as below**

**public class HomeController : Controller{**

**public IAuthorizationService \_authorizationService { get; }**

**public HomeController(IAuthorizationService authorizationService){**

**\_authorizationService = authorizationService;}**

**//to check authorization in custom action**

**public async Task<IActionResult> DoStuff(){**

**//we are doning stuff here**

**//it will create Authorization Builder**

**var builder = new AuthorizationPolicyBuilder("Schema");**

**//it will create Authorization Policy**

**var customPolicy = builder.RequireClaim("Hello").Build();**

**var authResult = await \_authorizationService.AuthorizeAsync(User, customPolicy);**

**if (authResult.Succeeded){}**

**return View("Index");}}**

**2-on the view Index we check authorization also as below**

**<h1>Hello World</h1>**

**@\* Simple access quick checks \*@**

**@if (User.Identity.IsAuthenticated){<h1>User is authenticated</h1>}**

**else{<h1>User is not authenticated</h1>}**

**@\*to check authorization in View\*@**

**@using Microsoft.AspNetCore.Authorization**

**@inject IAuthorizationService authorizationService**

**@if ((await authorizationService.AuthorizeAsync(User, "Claim.DoB")).Succeeded)**

**{<h1>User has DOB Claim</h1>}**

**Authorization Operation:-**

**We can apply custom handler to check three arguments**

**User**

**Requirement**

**Resources**

**Steps:-**

**1-we create class CookieJarAuthorizationHandler as below**

**using Microsoft.AspNetCore.Authorization;**

**using Microsoft.AspNetCore.Authorization.Infrastructure;**

**using System.Threading.Tasks;**

**namespace Basics.Authorization{**

**//we can control the requirement passed through the handler**

**public class CookieJarAuthorizationHandler : AuthorizationHandler<OperationAuthorizationRequirement, CookieJar>{**

**protected override Task HandleRequirementAsync(**

**AuthorizationHandlerContext context,**

**OperationAuthorizationRequirement requirement, CookieJar cookieJar){**

**if (requirement.Name == CookieJarOperations.Look){**

**if (context.User.Identity.IsAuthenticated){**

**context.Succeed(requirement);}}**

**else if (requirement.Name == CookieJarOperations.ComeNear){**

**if (context.User.HasClaim("Friend", "Good")){context.Succeed(requirement);}}**

**return Task.CompletedTask;}}**

**//its requirement that we can access to it directly**

**public static class CookieJarAuthOperations{**

**public static OperationAuthorizationRequirement Open = new OperationAuthorizationRequirement{Name = CookieJarOperations.Open};}**

**//its operations that represnt the allowed claims to access**

**public static class CookieJarOperations{**

**public static string Open = "Open";**

**public static string TakeCookie = "TakeCookie";**

**public static string ComeNear = "ComeNear";**

**public static string Look = "Look";}**

**//this class represent the resource that passed**

**public class CookieJar{public string Name { get; set; }}}**

**2-on startup.cs we inject the handler as below**

**services.AddScoped<IAuthorizationHandler, CookieJarAuthorizationHandler>();**

**IClaims Transformations Interface**

**Notes:-**

**1-We can apply IClaim transformation that hit on each request after the user is authenticated**

**using Microsoft.AspNetCore.Authentication;**

**using System.Linq;**

**using System.Security.Claims;**

**using System.Threading.Tasks;**

**namespace Basics.Transformer{**

**//every time is authenticate the user it will fire**

**public class ClaimsTransformer : IClaimsTransformation{**

**public Task<ClaimsPrincipal> TransformAsync(ClaimsPrincipal principal){**

**var hasFriendClaims = principal.Claims.Any(x => x.Type == "Friend");**

**if (hasFriendClaims){**

**((ClaimsIdentity)principal.Identity).AddClaim(new Claim("Friend", "Bad"));}**

**return Task.FromResult(principal);}}}**

**2-we apply inject the ClaimsTransformter in the startup.cs**

**//inject the claims transformation**

**services.AddScoped<IClaimsTransformation, ClaimsTransformer>();**

**Authorization Policy Provider**

**Notes:-**

**1-authorization policy provider: is container or resolver for the policies injected**

**We can use it to register large number of policies instead of put all these policies in hard coded in the startup.cs with using of IAuthorizationPolicyProvider**

**Example: -**

**if we have custom attribute that used to generate different policy like time period employee, we have to use IAuthorizationPolicyProvider to handle any policy entered dynamically**

**2-The Authorization Requirement is the collection of data which can be used to evaluate the user principal and the Authorization handler contains an evaluation mechanism for properties of requirement. One requirement may be associated with multiple handlers.**

**Example: -**

**Here, I have created the requirement for minimum time spent for the organization and created the handler that calculates no of days for an employee by subtracting today date from claim "DateOfJoing" and if the result is greater than or equal to supplied data then the user is authorized to access.**

**Steps:-**

**1-we create new class**