APPLICATIONS SECURITY

ASP.Net Core Identity

(Authentication & Authorization)

Content

- Creating an Identity
- Authenticating an Identity
- Authorizing an Identity (Claims, Roles, Policy)

Creating an Identity

- Namespace : Microsoft.AspNetCore.Identity
- The UserManager class of Microsoft.AspNetCore.Identity namespace helps to manage Identity users stored in the database.
- The generic version of this class is UserManager<T> where T is the class chosen to represent users.
  - UserManager<IdentityUser> userManager
- Use this class to perform CRUD operations for the users.

AppSec

UserManager<T> class members

Name	Description
Users	This property returns a sequence containing the users stored in the Identity database.
FindByIdAsync(id)	This method queries the database for the user object with the specified ID.
CreateAsync(user, password)	This method register a user in Identity with the specified password.
UpdateAsync(user)	This method modifies an existing user in the Identity database.
DeleteAsync(user)	This method removes the specified user from the Identity database.
AddToRoleAsync(user, name)	Adds a user to a role
RemoveFromRoleAsync(user, name)	Removes a user from a role
GetRolesAsync(user)	Gives the names of the roles in which the user is a member
IsInRoleAsync(user, name)	Returns true is the user is a member of a specified role, else returns false

Creating user in Identity

(Add reference)

```
using Microsoft.AspNetCore.Identity;
```

(Create a user Class)

```
public class Rmodel {

  [Required]
  public string Name { get; set; }

  [Required]
  [RegularExpression("^[a-zA-Z0-9_\\.-]+@([a-zA-Z0-9-]+\\.)+[a-zA-Z]{2,6}$", ErrorMessage = "E-mail is not valid")]
  public string Email { get; set; }

  [Required]
  public string Password { get; set; }
}
```

(Prepare the Identity User data)

```
var user = new IdentityUser()
{
        UserName = RModel.Name,
        Email = RModel.Email
};
```

(Add User into Database)

```
IdentityResult result = await userManager.CreateAsync(user, RModel.Password);

if (result.Succeeded)
{
return RedirectToPage("Index");
}
else {}
```

User created in AspNetUsers table

Program.cs | dbo.AspNetUsers [Data] | dbo.AspNetUserTokens [Data] | Register.cs

Max Rows: 1000

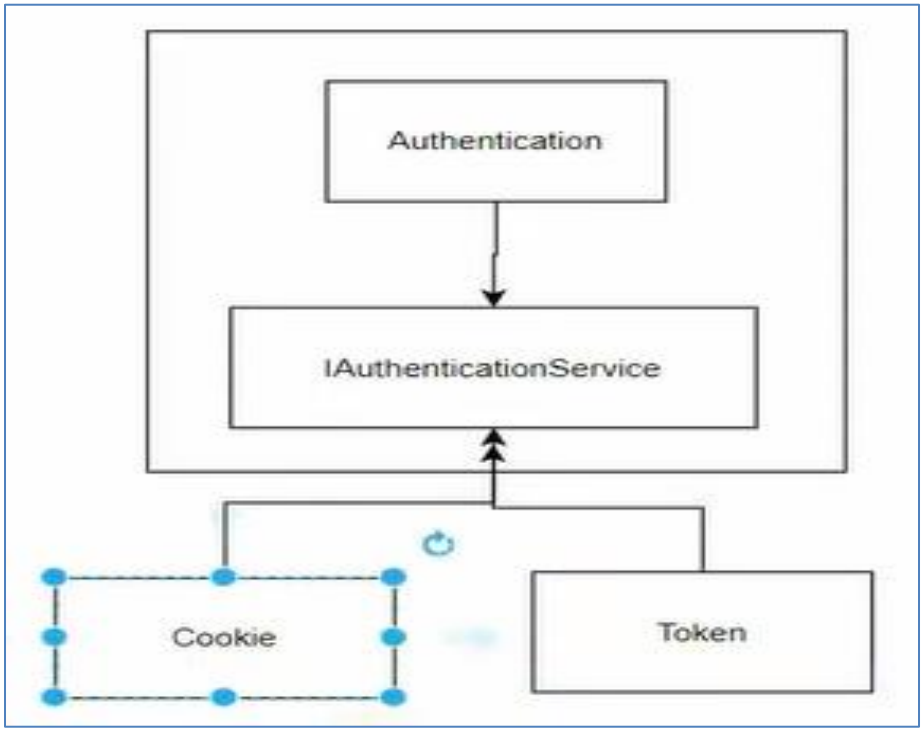
Id	UserName	NormalizedUs...	Email	NormalizedEm...	EmailConfir...	PasswordHash
14d96a13-e602-...	c@c.com	C@C.COM	c@c.com	C@C.COM	False	2jexjUMOn79Q==
2577eec9-745b-...	b@b.com	B@B.COM	b@b.com	B@B.COM	False	AQAAAAEAAC...
8c88b9f9-108c-...	Abc@123.com	ABC@123.COM	Abc@123.com	ABC@123.COM	False	AQAAAAEAAC...
90ebe82f-e1dc-...	aaa@aaa.com	AAA@AAA.COM	aaa@aaa.com	AAA@AAA.COM	False	AQAAAAEAAC...
NULL	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*

Authenticating an Identity

- As discussed previously, in order to use the Authentication feature in .net Core, developers will need to add support (3 steps) for ASP.NET Core Identity.
- There are 2 methods of authentication in ASP.NET Core Identity.
  - Cookie vs Token

Authentication methods

- builder.Services.AddAuthentication(..).AddCookie (..)
- builder.Services.AddAuthentication(..).AddFacebook(..)
- builder.Services.AddAuthentication(..).AddGoogle(..)



App Cookie configuration

- To make any changes to the behaviour of the cookie, you may make code changes to ConfigureApplicationCookie() method of the Startup.cs or Program.cs

```
services.ConfigureApplicationCookie(options =>
{
options.Cookie.Expiration = TimeSpan.FromDays(150);
options.Cookie.HttpOnly = true;
options.LoginPath = "/Account/Login";
options.LogoutPath = "/Account/Logout";
options.SlidingExpiration = true;
});
```

App Cookie settings

- **Cookie.Name** : name of the application cookie
- **Cookie.HttpOnly** : it specify that cookie is accessible from client side scripts or not
- **ExpireTimeSpan** : it specify that the time that authentication ticket stored in the cookie and remain valid.
- **LoginPath** : Login page path
- **LogoutPath** : Logout page path
- **AccessDeniedPath** : it defined the path on which user will redirected if authorization is fail
- **SlidingExpiration** : This is Boolean property. When it set to true, new cookie will be created when current cookie is more than halfway through the expiration window.

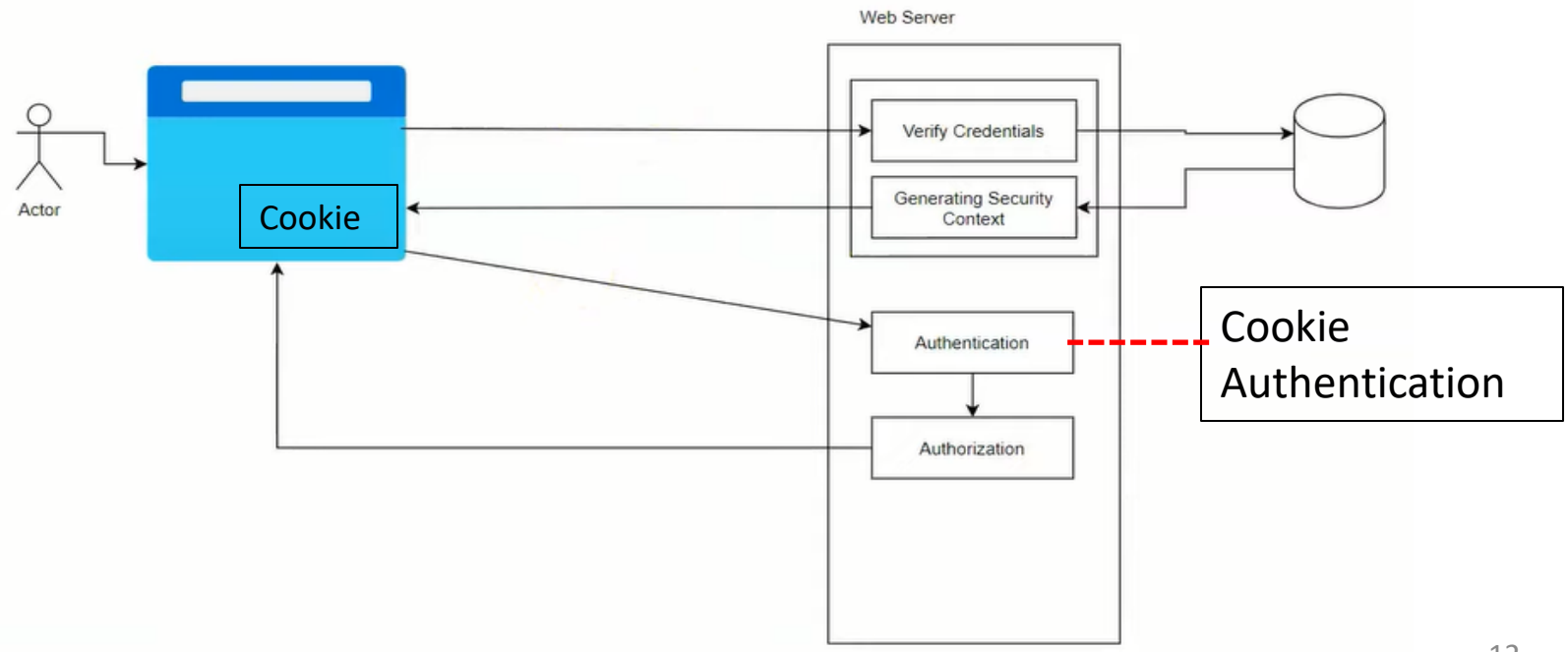
SignInManager classes

- The SignInManager is responsible for Authenticating a user:
  - signing in
  - signing out
  - Issue authentication cookie

- SignInManager defines methods related to the authentication of users.
  - PasswordSignInAsync method accepts a username and a password and returns a sign-in result with a property indicating whether the authentication attempt was successful.

SignInAsync

- If the user credential is successfully verified, the SignInAsync method creates an encrypted cookie (together with the security context) and adds it to the current response.
- The Current response is returned to User. The User browser reads the cookie from the response and stores it securely. When user sends another request to the server, the browser sends the cookie along with the request.



Identity in Razor Class Library

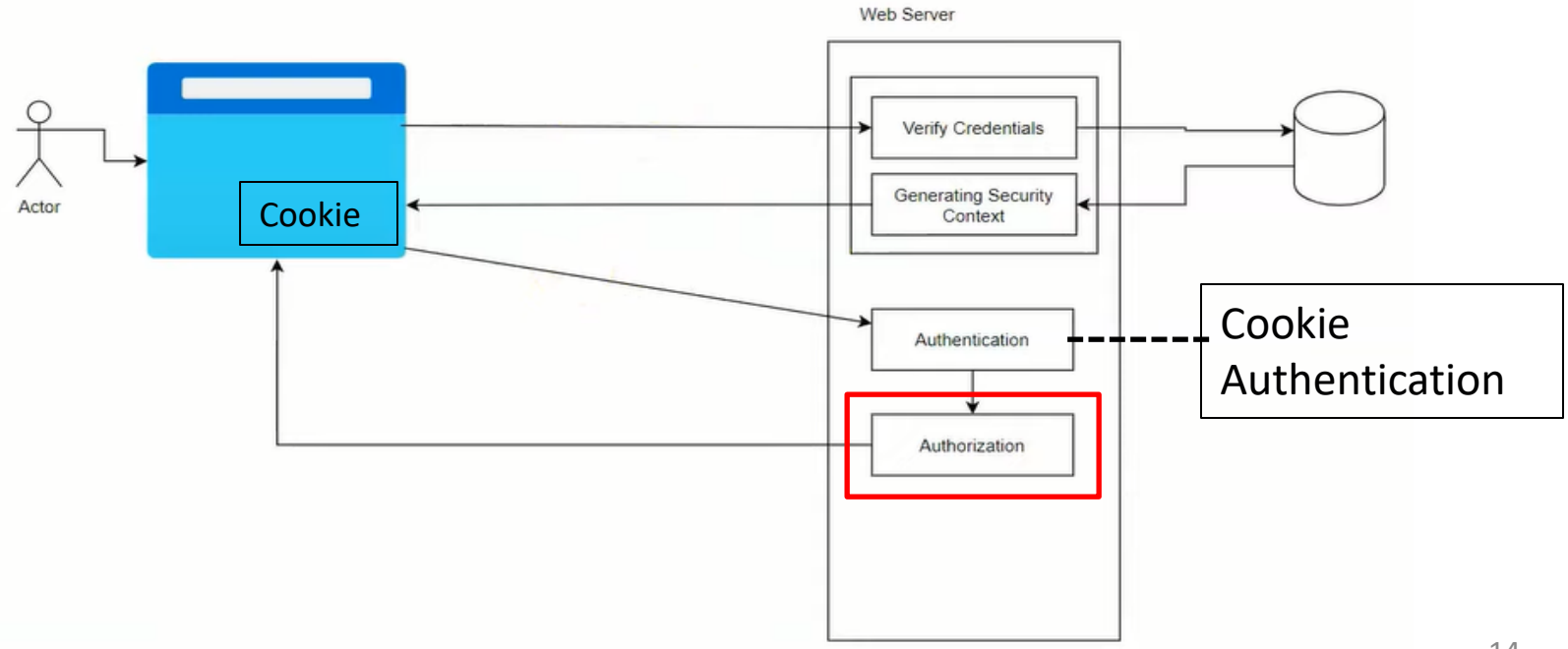
- Identity is included in a Razor class library with all the views that it usually needs to support identity in an application.

```
@{
    if (SignInManager.IsSignedIn(User))
    {
      • Make Index Page visible
      • Make Privacy Page visible
      • Make Logout feature visible
      • Display User Name
    }
    else {
      • Make Login feature visible
      • Make Register feature visible
    }
}
```

sample.cshtml

Authorizing an Identity

- After successful login to the application, authorization mechanism checks whether login user has privileges to access the application resource.



Types of Authorization in Identity

- Role-based
  - Roles are commonly used to create fine-grained authorization policies that differentiate between different signed-in users.
- Claims-based
  - Claims are a general-purpose approach to describing any data that is known about a user and allow custom data to be added to the Identity user store.

AppSec

Roles

```
string[] roleNames = { "Administrator", "GroupUser", "User", "Guest" };
foreach (var roleName in roleNames)
 {
 var roleExist = await RoleManager.RoleExistsAsync(roleName);
 if (!roleExist)
 {
 roleResult = await RoleManager.CreateAsync(new IdentityRole(roleName));
 }
```

Create Roles

```
user = new IdentityUser()
 {
        UserName = “Timothy.W@gmail.com",
        Email = “Timothy.W@gmail.com",
 };
 await UserManager.CreateAsync(user, “Tim@123");
 }
 await UserManager.AddToRoleAsync(user, "Administrator");
```

Assigning Identity to Roles

Membership table - Roles

```
select * from [dbo].[AspNetRoles]
select * from [dbo].[AspNetUsers]
select * from [dbo].[AspNetUserRoles]
```

100 %

Results | Messages

	Id	Name	NormalizedName	ConcurrencyStamp
1	181f88f1-1b5e-4dbe-9140-5c2f0d888ee9	Administrator	ADMINISTRATOR	aef294fe-85d6-4e90-8679-19811b86c
2	66c5d7c1-d7a2-42b1-b20a-eea2fd455806	GroupUser	GROUPUSER	b888ea98-04dc-40e4-ab58-0499009b
3	670eb6b5-091c-4290-a4d3-1e2b08968555	Guest	GUEST	78ef9551-1c3d-4205-b9ec-0f55d4c65
4	8ed68715-acf5-46ba-b226-f7289cc2c72f	User	USER	85f56083-ced0-466d-b90a-26e12964

	Id	UserName	NormalizedUserName	Email
1	0eec0d6b-0753-4a34-82aa-070d6df4e5c6	Timothy.W@gmail.com	Timothy.W@gmail.com	Timothy.W@gmail.com

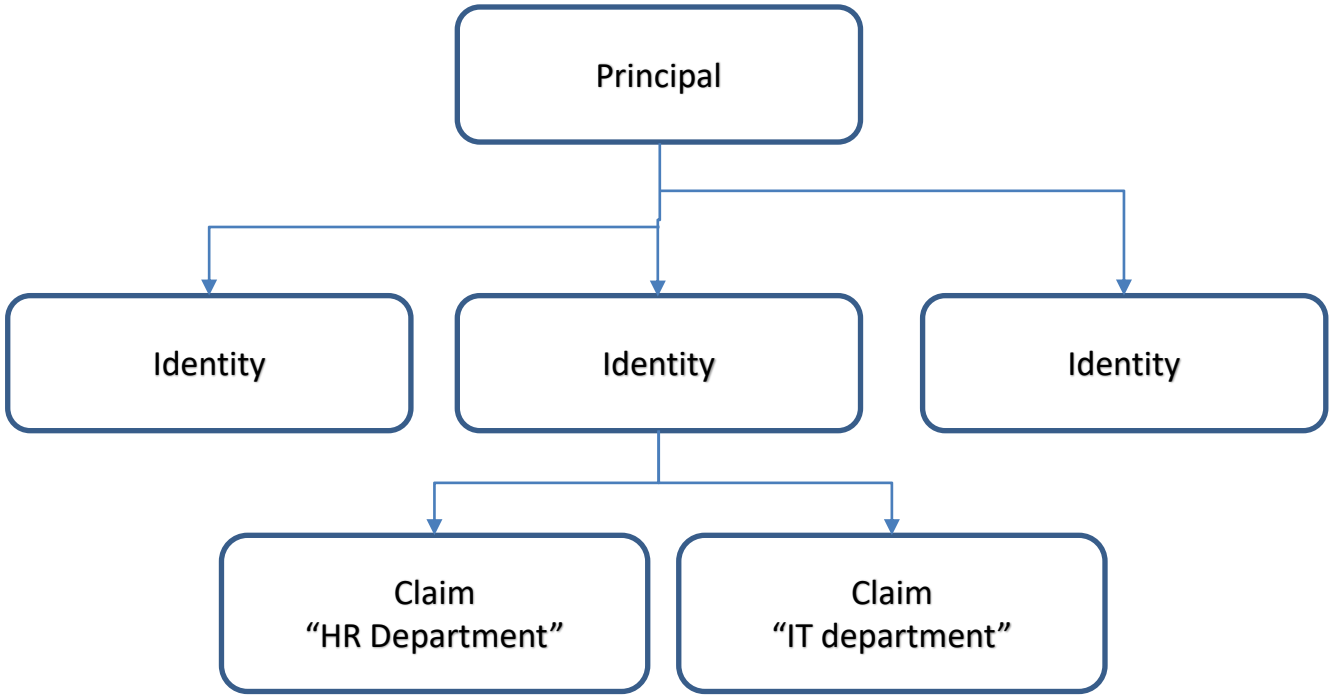
	UserId	RoleId
1	0eec0d6b-0753-4a34-82aa-070d6df4e5c6	181f88f1-1b5e-4dbe-9140-5c2f0d888ee9
2	74a03b7e-58b5-432a-ba96-c9d5fa83ef99	66c5d7c1-d7a2-42b1-b20a-eea2fd455806
3	ceb8c7bb-9936-48d1-8581-603658f2f2eb	670eb6b5-091c-4290-a4d3-1e2b089685...
4	62154c2c-eea0-4521-94e5-690b32b588...	8ed68715-acf5-46ba-b226-f7289cc2c72f

Setting the page requirements

- Authorize the pages using the following examples:

```
[Authorize(Roles = "GroupUser")]
[Authorize(Roles = "User")]
 public IActionResult MultipleRoleAccess1()
 { … }
```

```
[Authorize(Roles = "GroupUser, User")]
public IActionResult MultipleRoleAccess1()
 { … }
```


Principal
Identity
Identity
Identity
Claim
“HR Department”
Claim
“IT department”