Authentication, Authorization & OAuth

Day 5

Agenda for Day 5

- ASP.NET Core Identity & Authentication Basics
- Cookie-Based Authentication (for MVC)
- JWT Authentication (for Web API)
- Authorization Mechanisms (Role-Based, Policy-Based, User-Specific)
- OAuth 2.0 & External Logins
- Hands-on Demo & Lab Assignment

Authentication vs. Authorization (Review)

- **Authentication**: Verifying who a user is ("Are you who you say you are?").
- Examples: Username/password, biometrics, tokens.
- **Authorization**: Determining what an authenticated user is allowed to do ("What are you allowed to access/do?").
- Examples: Accessing specific resources, performing certain actions (e.g., delete product).

ASP.NET Core Identity

- **Purpose**: A comprehensive membership system for ASP.NET Core applications.
- **Features**: User registration, login, logout, password hashing, two-factor authentication, external logins, role management.
- **Integration**: Built on Entity Framework Core, provides default UI (Razor Pages).

ASP.NET Core Identity

`IdentityUser`

- Represents a user in the system.
- **Properties**:
  - `Id`, `UserName`, `Email`, `PasswordHash`, etc.
- **Location**:
  - `Microsoft.AspNetCore.Identity.IdentityUser`

`IdentityRole` Roles

- Groups of permissions. Users are assigned roles to grant permissions.
- **Purpose**: To apply a common set of permissions to a group of users.
- **Example Roles**: `"Admin"`, `"Editor"`, `"Viewer"`.
- **Properties**:
  - `Id ,Name`
- **Location**:
  - `Microsoft.AspNetCore.Identity`

Claims

- Key-value pairs describing the user's attributes or specific permissions
- **Examples**:
  - `"email": "user@example.com"`
  - `"department": "Engineering"`
  - `"role": "Admin"` (automatically added when assigning roles)
  - `"CanEditPosts": "true"` (Specific Permission)
- **Location**:
  - `System.Security.Claims`

```
// Using claims for authorization
[Authorize(Policy = "CanEditPosts")]
public IActionResult EditPost()
{
    return View();
}
```

`UserManager<TUser>`

- This handles all user-related operations:-
  - Creating new users
  - Updating user information
  - Managing passwords
  - Assigning roles to users

`SignInManager<TUser>`

- This manages the login/logout process:
  - Authenticating users
  - Creating authentication cookies
  - Handling "Remember Me" functionality
  - Managing lockouts

RoleManager<TRole>

- This handles role-related operations:
  - Creating new roles
  - Deleting roles
  - Managing role permissions

Setup

- 1. Add Identity services to the container (Program.cs)
- `AddDefaultIdentity()`
  - `Microsoft.AspNetCore.Identity.EntityFrameworkCore`
  - `Microsoft.AspNetCore.Identity.UI`
  - Simple configuration (no roles)
  - Includes UI (MVC only)
  - Must have `_LoginPartial.cshtml`
  - Add `app.MapRazorPages();` in program.cs

```
builder.Services.AddDefaultIdentity<IdentityUser>(options =>
options.SignIn.RequireConfirmedAccount = true)
    .AddRoles<IdentityRole>() // Optional: Add support for roles
    .AddEntityFrameworkStores<ApplicationDbContext>() // Specify your DbContext
    .AddDefaultUI();
```

Setup

- AddIdentity()
  - Microsoft.AspNetCore.Identity.EntityFrameworkCore
  - Microsoft.AspNetCore.Authentication.JwtBearer (web API)
  - Full feature
  - MVC or WebAPI

```
builder.Services.AddIdentityCore<IdentityUser>(option=>
{
    option.Password.RequireDigit = false;
    option.User.RequireUniqueEmail = true;
})
    .AddEntityFrameworkStores<CompanyDbContext>()
    .AddApiEndpoints();  // Adds /register, /login endpoints
// Add Authentication (e.g., JWT)
builder.Services.AddAuthentication().AddJwtBearer();
// Add authorization
builder.Services.AddAuthorization();
```

Setup

- 2. Configure the **DbContext** for Identity

```
// load connection string from appsettings.json
var connectionString =
builder.Configuration.GetConnectionString("DefaultConnection")

// This assumes ApplicationDbContext inherits from IdentityDbContext<IdentityUser>
 builder.Services.AddDbContext<CompanyDbContext>(options =>
     options.UseSqlServer(connectionString));
```

```
public class CompanyDbContext:IdentityDbContext<IdentityUser>
{

       protected override void OnModelCreating(ModelBuilder modelBuilder)
       {
              base.OnModelCreating(modelBuilder);// must for IdentityDbContext
```

Setup

- 3. Add authentication middleware to the request pipeline

```
// This must be placed before UseAuthorization()
app.UseAuthentication();
app.UseAuthorization();
```

- 4. map identity razor pages

```
app.MapRazorPages()
 .WithStaticAssets();
```

Setup

- 5. add scaffold Identity
- Or add `_LoginPartial.cshtml` in views/shared
- 6. add `<partial name="_LoginPartial" />` in _Layout

Setup

- Example of a custom **`IdentityUser`** (Models folder)

```
public class ApplicationUser:IdentityUser
    {
        // Add any custom properties here, e.g.,
        public string FirstName { get; set; }
        public string LastName { get; set; }
    }
}
```

Setup

- Example of a custom **`ApplicationDbContext`** (Data folder)

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext>
options)
        : base(options)
    {
    }

    // You can add other DbSets for your application data here
}
```

Cookie-Based Authentication (for MVC)

- **Mechanism**: After successful login, the server issues an encrypted cookie to the client.
- **How it works**:
  - 1. User submits credentials to MVC app.
  - 2. MVC app authenticates user (via `UserManager`, `SignInManager`).
  - 3. If successful, MVC app issues an authentication cookie.
  - 4. Browser automatically sends this cookie with subsequent requests.
    5. MVC app validates the cookie to identify the user.
  - **`[Authorize]` attribute**: Applied to MVC controllers/actions to restrict access.

```
[Authorize] // Requires authentication for all actions in this controller
public class HomeController : Controller
{
  public IActionResult Index()
  {
       // This action requires an authenticated user
  }
 [AllowAnonymous] // Allows unauthenticated access to this specific action
 public IActionResult About()
 {
       // This action can be accessed by anyone
 }
 [Authorize(Roles = "Admin")] // Requires the user to be in the 'Admin' role
 public IActionResult AdminDashboard()
 {
  // Only users with the 'Admin' role can access this
 }
[Authorize(Policy = "RequireManagerRole")] // Requires a specific authorization policy
 public IActionResult ManagerReport()
 {
        // Only users satisfying the 'RequireManagerRole' policy can access this
  }
 }
```

Account Controller

```
public class AccountController : Controller
{
    private readonly UserManager<IdentityUser> _userManager;
    private readonly SignInManager<IdentityUser> _signInManager;
    public AccountController(UserManager<IdentityUser> userManager,
        SignInManager<IdentityUser> signInManager)
    {
        _userManager = userManager;
        _signInManager = signInManager;
    }
    public IActionResult Index()
    {
        return View();
    }
```

Account Controller Register Action

```
[HttpGet]
public IActionResult Register()
{
    return View();
}

[HttpPost]
public async Task<IActionResult> Register(RegisterModel model)
{
    if (!ModelState.IsValid)
    {
        return View(model);
    }
    var user = new IdentityUser { Email = model.Email, UserName = model.Email };
    var result = await _userManager.CreateAsync(user, model.Password);
    if (result.Succeeded)
    {
        return RedirectToAction("Index", "Home");
    }
    foreach (var error in result.Errors)
    {
        ModelState.AddModelError("", error.Description);
    }

    return View(model);
}
```

Account Controller Login Action

```
[HttpGet]
public IActionResult Login(string? ReturnUrl)
{
    ViewBag.ReturnUrl = ReturnUrl;
    return View();
}

[HttpPost]
public async Task<IActionResult> Login(LoginModel model, string? ReturnUrl)
{
    if (!ModelState.IsValid)
    {
        return View(model);
    }
var result = await _signInManager.PasswordSignInAsync(model.Email, model.Password,
        isPersistent: false, lockoutOnFailure: false);
    if (result.Succeeded)
    {
        if (ReturnUrl == null)
            return RedirectToAction("Index", "Home");

            return LocalRedirect(ReturnUrl);
    }
    return View(model);

}
```

Account Controller Logout Action

```
public IActionResult Logout()
{
    _signInManager.SignOutAsync();
    return RedirectToAction("Index", "Home");
}
```

Seeding Data

```
public static class SeedData
{
    public static async Task Initialize(IServiceProvider serviceProvider)
    {
        var userManager = serviceProvider.GetRequiredService<UserManager<IdentityUser>>();
        var roleManager = serviceProvider.GetRequiredService<RoleManager<IdentityRole>>();
        string[] roleNames = { "Admin", "User" };
        // create roles if they do not exist
        foreach (var roleName in roleNames)
        {
            // Check if the role already exists
            if (!await roleManager.RoleExistsAsync(roleName))
            {
                // Create the role
                await roleManager.CreateAsync(new IdentityRole(roleName));
            }
        }
        var adminUser =
            new IdentityUser
            {
                UserName = "admin@hotmail.com",
                Email = "admin@hotmail.com",
                EmailConfirmed = true
            };
        string adminPassword = "Admin@123";
        // Check if the users already exist
        var admin = await userManager.FindByEmailAsync(adminUser.Email);
        if (admin == null)
        {
            var createAdmin = await userManager.CreateAsync(adminUser, adminPassword);
            if (createAdmin.Succeeded)
            {
                // Assign Admin role to the user
                await userManager.AddToRoleAsync(adminUser, "Admin");
            }
        }
        var normalUser =
            new IdentityUser
            {
                UserName = "user@hotmail.com",
                Email = "user@hotmail.com",
                EmailConfirmed = true
            };

        string userPassword = "User@123";
        var user = await userManager.FindByEmailAsync(normalUser.Email);
        if (user == null)
        {
            var createUser = await userManager.CreateAsync(normalUser, userPassword);
            if (createUser.Succeeded)
            {
                // Assign User role to the user
                await userManager.AddToRoleAsync(normalUser, "User");
            }
        }
    }
}
```

_PartialLogin.cshtml

```
@using Microsoft.AspNetCore.Identity

@inject SignInManager<IdentityUser> SignInManager
@inject UserManager<IdentityUser> UserManager

<ul class="navbar-nav">
    @if (SignInManager.IsSignedIn(User))
    {
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="Identity" asp-page="/Account/Manage/Index"
title="Manage">Hello @User.Identity?.Name!</a>
        </li>
        <li class="nav-item">
            <form class="form-inline" asp-controller="Auth" asp-action="Logout" asp-route-
returnUrl="@Url.Action("Index", "Home", new { area = "" })">
                <button type="submit" class="nav-link btn btn-link text-dark">Logout</button>
            </form>
        </li>
    }
    else
    {
        <li class="nav-item">
            <a class="nav-link text-dark" asp-controller="Auth" asp-action="Register">Register</a>
        </li>
        <li class="nav-item">
            <a class="nav-link text-dark" asp-controller="Auth" asp-action="Login">Login</a>
        </li>
    }
</ul>
```