

Tutorial for Class number 9

This exercise intends develop a web application to test the use of **ASP.NET Core security**.

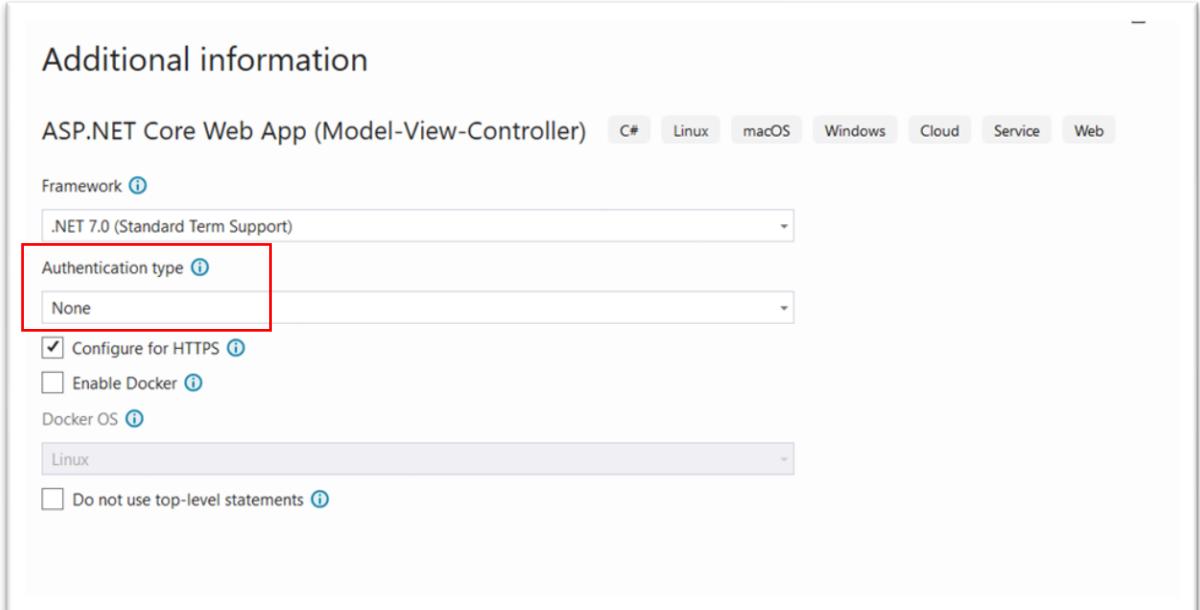
ASP.NET Core provides many tools and libraries to secure your apps including built-in identity providers, but you can use third-party identity services such as Facebook, Twitter, and LinkedIn. With ASP.NET Core, you can easily manage app secrets, which are a way to store and use confidential information without having to expose it in the code.

Authentication vs. Authorization

Authentication is a process in which a user provides credentials that are then compared to those stored in an operating system, database, app or resource. If they match, users authenticate successfully, and can then perform actions that they're authorized for, during an authorization process. The **authorization** refers to the process that determines what a user is allowed to do.

First step – Creating a project with authentication

- Create a new “ASP.NET Core Web App (Model-View-Controller)” project.



In Authentication Type information, select the **Individual Accounts** option. In this configuration, the user information is stored into a local database.

Additional information

ASP.NET Core Web App (Model-View-Controller)

C# Linux macOS Windows Cloud

Framework [?](#)

.NET 7.0 (Standard Term Support)

Authentication type [?](#)

None

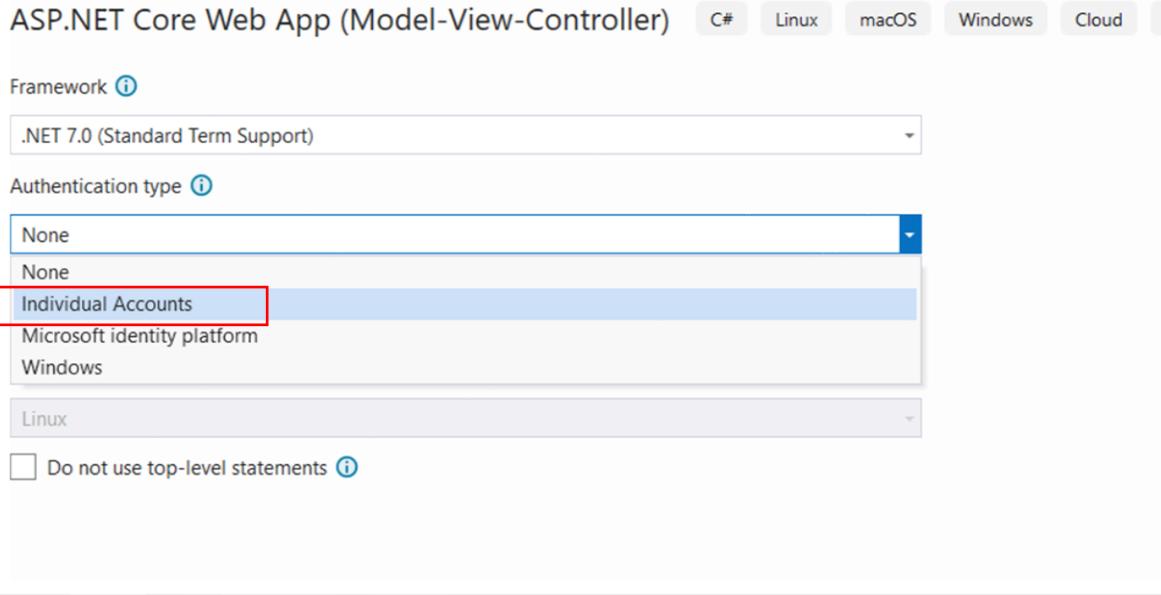
Individual Accounts

Microsoft identity platform

Windows

Linux

Do not use top-level statements [?](#)



- See the generated or modified files:
 - **appsettings.json** with a new connection string. We can change where to save the user database.

```
"ConnectionStrings": {  
    "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=aspnet-Aula09-6e503136-ca9e-4c  
}
```

- in the **Data/Migrations** folder, was created one migration (**CreateIdentitySchema**) with the tables definition.

```
namespace Aula09.Data.Migrations  
{  
    1 reference  
    public partial class CreateIdentitySchema : Migration  
    {  
        0 references  
        protected override void Up(MigrationBuilder migrationBuilder)  
        {  
            migrationBuilder.CreateTable(  
                name: "AspNetRoles",  
                columns: table => new  
                {  
                    Id = table.Column<string>(nullable: false),  
                    Name = table.Column<string>(maxLength: 256, nullable: true),  
                    NormalizedName = table.Column<string>(maxLength: 256, nullable: true),  
                    ConcurrencyStamp = table.Column<string>(nullable: true)  
                },  
                constraints: table =>  
                {  
                    table.PrimaryKey("PK_AspNetRoles", x => x.Id);  
                });  
        }  
    }  
}
```

- and a database context:

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

namespace Aula09.Data
{
    public class ApplicationDbContext : IdentityDbContext
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {
        }
    }
}
```

- in the **Program.cs**

```
// Add services to the container.
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new InvalidOperationException();
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(connectionString));
builder.Services.AddDatabaseDeveloperPageExceptionFilter();

builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = true)
    .AddEntityFrameworkStores<ApplicationDbContext>();

builder.Services.AddControllersWithViews();

var app = builder.Build();

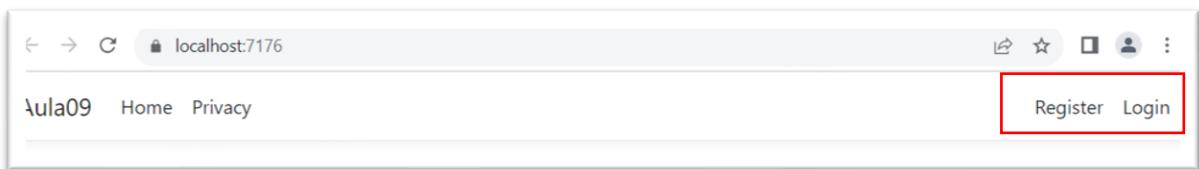
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
else
{
    app.UseHttpsRedirection();
    app.UseStaticFiles();

    app.UseRouting();
    app.UseAuthorization();

    app.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
}
```

- Apply the migrations to initialize the database. Run the following command in the Package Manager Console (PMC):
PM> Update-Database
- Run the application

In the application menu we have a **Register** and **Login** options.



- Register a new user:

A screenshot of the 'Register' page. The page title is 'Register' and the sub-section is 'Create a new account.' There are three input fields: 'Email' (xpto@xpto.pt), 'Password', and 'Confirm password'. Below these fields is a blue 'Register' button. To the right of the input fields, there is a large callout box with a red border containing the text: 'Use another service to register.' and 'In this link we have information what to do to using external logins. (Facebook, Twitter, Google, Microsoft, other)'. A red arrow points from the text 'Password must have at least one uppercase ("A"-“Z").' in the list below the input fields to the 'Password conditions. We can change this' text in the callout box. Another red arrow points from the 'Register' button to the 'Use another service to register.' text.

After the register, the system need a confirmation sending an email.

A screenshot of the 'Register confirmation' page. The page title is 'Register confirmation'. It contains the message: 'This app does not currently have a real email sender registered, see [these docs](#) for how to configure a real email sender. Normally this would be emailed: [Click here to confirm your account](#)'. At the bottom of the page, there is a copyright notice: '© 2023 - Aula09 - Privacy' and a URL: 'https://localhost:7176/Identity/Account/ConfirmEmail?userId=230a89cf-5429-4e3c-8f99-0a7cdf04ec8c&code=Q2ZESjhMQzQrKy92UUiTmpuelZxYUs1OGowZGtUVdp...'. A vertical scrollbar is visible on the right side of the page.

Because we don't have a mail sender registered, we select "Click here to confirm your account" to simulate the email confirmation.

A screenshot of the 'Confirm email' page. The page title is 'Confirm email'. A green message box contains the text: 'Thank you for confirming your email.' In the top right corner of the message box, there is a small 'X' icon.

We can disable this requirement, changing in the **Program.cs**

```
builder.Services.AddDatabaseDeveloperPageExceptionFilter();

builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = false)
    .AddEntityFrameworkStores<ApplicationContext>();

builder.Services.AddControllersWithViews();
```

The next code configures **Identity** with default option values. We must change to our needs.

```
builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = false)
    .AddEntityFrameworkStores<ApplicationContext>();

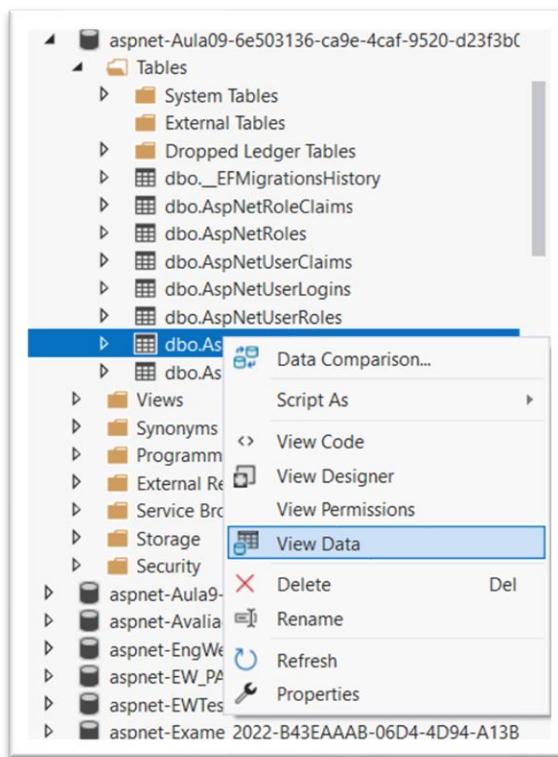
builder.Services.Configure<IdentityOptions>(options =>
{
    //password default settings, we can change this to our needs
    options.Password.RequireDigit= true;
    options.Password.RequireLowercase= true;
    options.Password.RequireNonAlphanumeric = true;
    options.Password.RequireUppercase= true;
    options.Password.RequiredLength = 6;
    options.Password.RequiredUniqueChars= 1;

    // lockout settings
    options.Lockout.DefaultLockoutTimeSpan= TimeSpan.FromMinutes(5);
    options.Lockout.MaxFailedAccessAttempts = 5;
    options.Lockout.AllowedForNewUsers = true;

    // user settings
    options.User.AllowedUserNameCharacters =
        "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-_#";
    options.User.RequireUniqueEmail= true;
});

builder.Services.AddControllersWithViews();
```

See the data stored in the table **AspNetUsers**



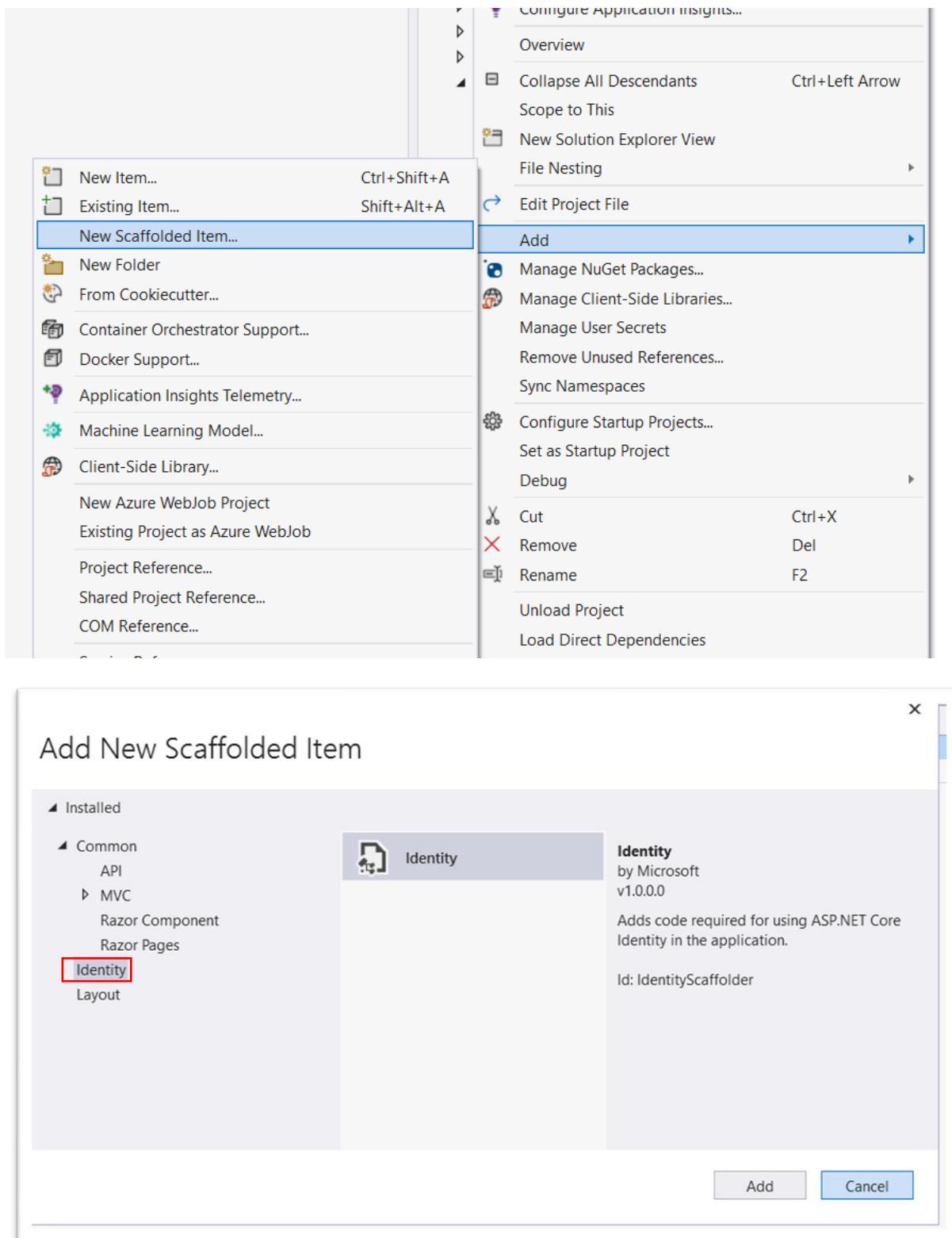
Id	UserName	NormalizedUs...	Email	NormalizedEm...	EmailConfirmed	PasswordHash	SecurityStamp	ConcurrencySt...	PhoneNumber	PhoneNumber
230a89cf-5429-...	xpto@xpto.pt	XPTO@XPTO.PT	xpto@xpto.pt	XPTO@XPTO.PT	True	AQAAAAIAAYa...	YIWOGX2IYAIZ6...	93fd401a-3c42-...	NULL	False
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

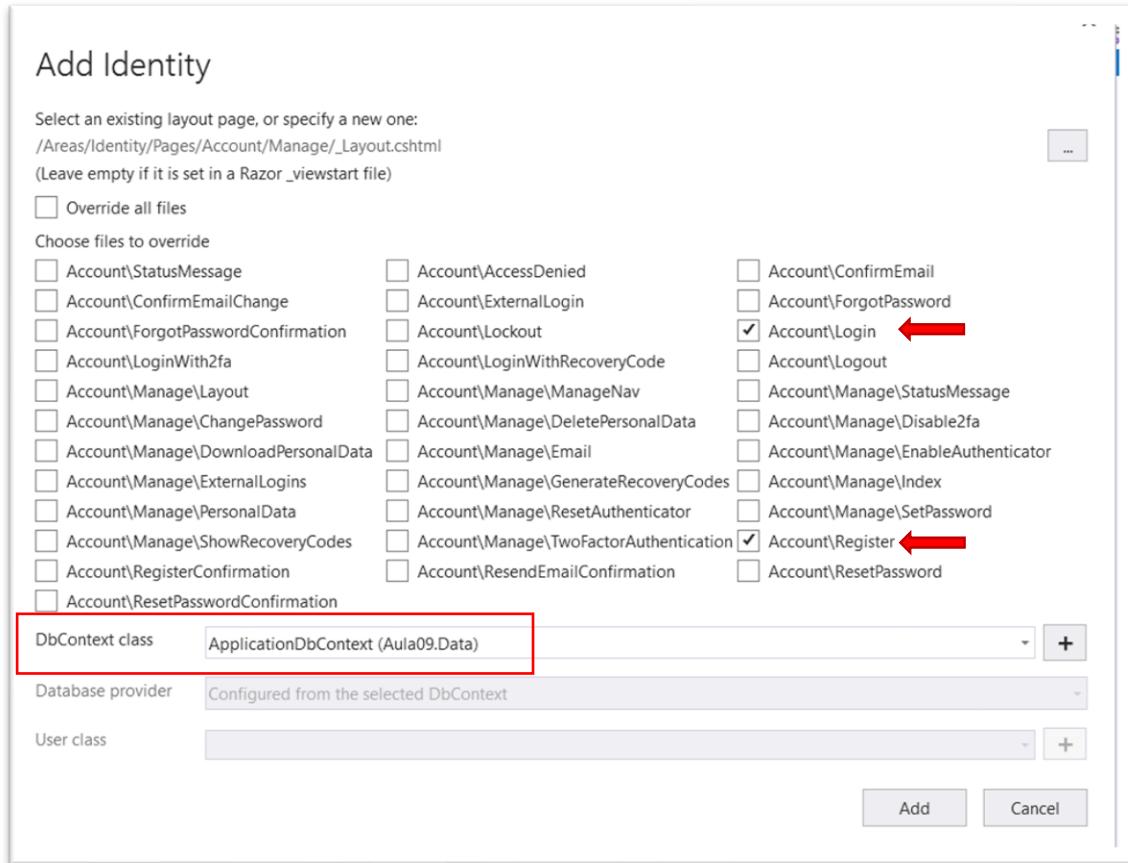
We can verify the data stored for each user. Note that Password is encrypted. By default, the UserName is the Email. In the next steps we will modify this.

- Change authentication views.

These services used to register or login a user in system uses internal code. If we wish do modify then, we first have to pull this code to the project. To access the views and code of the actions associated with authentication, we need to generate that:

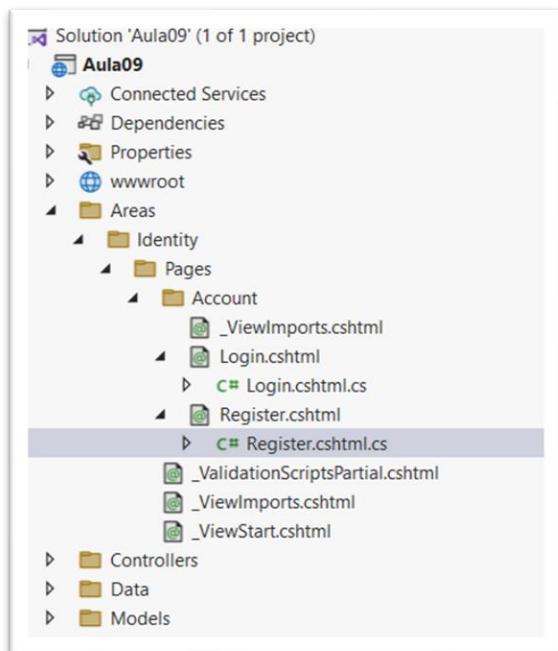
-Right click in project name (Class09) and choose **Add->New Scaffold Item...**





Select the views we want to change (Login, Register)

Expand de folder to see the created files:



These files use the **Blazor** format. The View and code are grouped in two files: **.cshtml** and **.cshtml.cs**

- Insert a Username field in the register process

Open the file **Register.cshtml.cs** and add **UserName** to **InputModel** declaration.

```
public class InputModel
{
    [Required]
    [Display(Name = "User Name")]
    0 references
    public string UserName { get; set; }

    [Required]
    [EmailAddress]
    [Display(Name = "Email")]
    7 references
```

In same file, use this new information when creating an **IdentityUser** instance.

```
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
    returnUrl ??= Url.Content("~/");
    ExternalLogins = await _signInManager.GetExternalAuthenticationSchemesAsync().ToListAsync();
    if (ModelState.IsValid)
    {
        var user = CreateUser();

        await _userStore.SetUserNameAsync(user, Input.UserName, CancellationToken.None);
        await _emailStore.SetEmailAsync(user, Input.Email, CancellationToken.None);
        var result = await _userManager.CreateAsync(user, Input.Password);

        if (result.Succeeded)
        {
            _logger.LogInformation("User created a new account with password.");
            var userId = await _userManager.GetUserIdAsync(user);
            var code = await _userManager.GenerateEmailConfirmationTokenAsync(user);
        }
    }
}
```

And finally, alter de view code to show a **UserName** field.

```
<div class="row">
    <div class="col-md-4">
        <form id="registerForm" asp-route-returnUrl="@Model.ReturnUrl" method="post">
            <h2>Create a new account.</h2>
            <hr />
            <div asp-validation-summary="ModelOnly" class="text-danger" role="alert"></div>
            <div class="form-floating mb-3">
                <input asp-for="Input.UserName" class="form-control" autocomplete="username" aria-required="true" placeholder="username" />
                <label asp-for="Input.UserName">UserName</label>
                <span asp-validation-for="Input.UserName" class="text-danger"></span>
            </div>
            <div class="form-floating mb-3">
                <input asp-for="Input.Email" class="form-control" autocomplete="username" aria-required="true" placeholder="name@example.com" />
                <label asp-for="Input.Email">Email</label>
                <span asp-validation-for="Input.Email" class="text-danger"></span>
            </div>
            <div class="form-floating mb-3">
                <input asp-for="Input.Password" class="form-control" autocomplete="new-password" aria-required="true" placeholder="password" />
                <label asp-for="Input.Password">Password</label>
            </div>
```

Register a new user. After making these changes, we now have to insert the Username, Email and Password fields.

Register

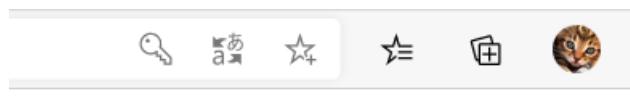
Create a new account.

Use another

User Name	Manelito
Email	Manel@xpto.pt
Password	*****
Confirm password	*****
Register	

There are no exte
[this ASP.NET app](#)

And the result is:



Hello Manelito! Logout

Now we also need modify the **Login** process to accept the **UserName**.

Open **Login.cshtml.cs** file and proceed with these modifications:

```
public class InputModel
{
    [Required]
    [Display(Name = "User Name")]
    4 references
    public string UserName { get; set; }

    [Required]
    [DataType(DataType.Password)]
    4 references
    public string Password { get; set; }

    [Display(Name = "Remember me?")]
    5 references
    public bool RememberMe { get; set; }
}
```

```

public async Task<ActionResult> UnpostAsync(string returnUrl = null)
{
    returnUrl ??= Url.Content("~/");

    ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();

    if (ModelState.IsValid)
    {
        // This doesn't count login failures towards account lockout
        // To enable password failures to trigger account lockout, set lockoutOnFailure: true
        var result = await _signInManager.PasswordSignInAsync(Input.UserName, Input.Password, Input.RememberMe, lockoutOnFailure: false);
        if (result.Succeeded)
        {
            _logger.LogInformation("User logged in.");
            return LocalRedirect(returnUrl);
        }
        if (result.RequiresTwoFactor)
        {
            return RedirectToPage("./LoginWith2fa", new { ReturnUrl = returnUrl, RememberMe = Input.RememberMe });
        }
    }
}

```

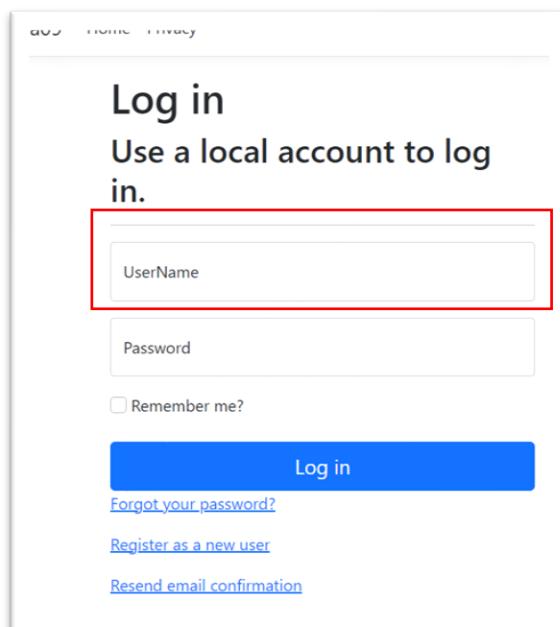
Alter the View (Login.cshtml)

```

<h1>@ViewData["Title"]</h1>
<div class="row">
<div class="col-md-4">
<section>
    <form id="account" method="post">
        <h2>Use a local account to log in.</h2>
        <hr />
        <div asp-validation-summary="ModelOnly" class="text-danger" role="alert"></div>
        <div class="form-floating mb-3">
            <input asp-for="Input.UserName" class="form-control" autocomplete="username" aria-required="true" placeholder="username" />
            <label asp-for="Input.UserName" class="form-label">UserName</label>
            <span asp-validation-for="Input.UserName" class="text-danger"></span>
        </div>
        <div class="form-floating mb-3">
            <input asp-for="Input.Password" class="form-control" autocomplete="current-password" aria-required="true" placeholder="password" />
            <label asp-for="Input.Password" class="form-label">Password</label>
            <span asp-validation-for="Input.Password" class="text-danger"></span>
        </div>
        <div class="checkbox mb-3">
            <label asp-for="Input.RememberMe" class="form-label">
                <input class="form-check-input" asp-for="Input.RememberMe" />
                @Html.DisplayNameFor(m => m.Input.RememberMe)
            </label>
        </div>
    </div>

```

- Test the new login



- See the _Layout.cshtml:

```

<div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
    <ul class="navbar-nav flex-grow-1">
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
        </li>
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
        </li>
    </ul>
    <partial name="_LoginPartial" />
</div>
</nav>
</header>
<div class="container">
    <main role="main" class="pb-3">
        @RenderBody()
    </main>
</div>

```

In here is inserted a partial view _LoginPartial.cshtml

```

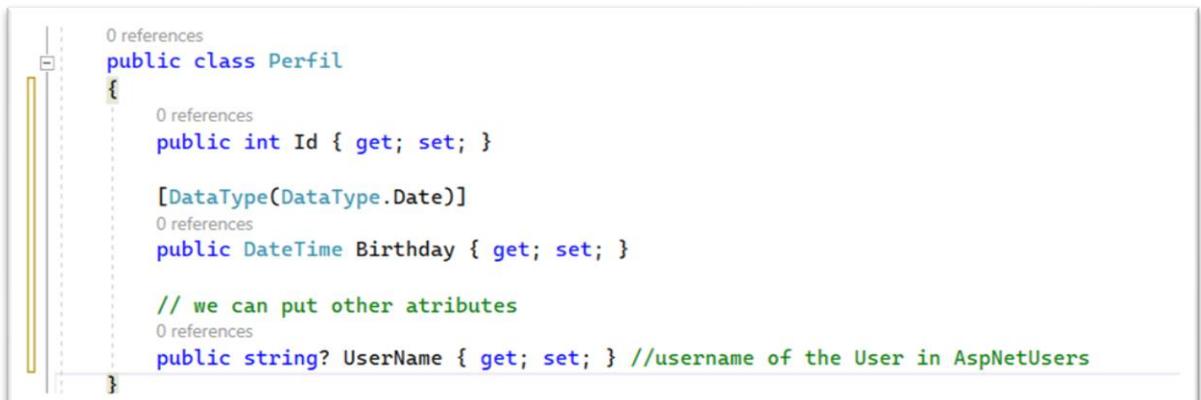
@inject SignInManager<IdentityUser> SignInManager
@inject UserManager<IdentityUser> UserManager

<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-area="Identity" asp-page="/Account/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-area="Identity" asp-page="/Account/Register">Register</a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="Identity" asp-page="/Account/Login">Login</a>
    </li>
}
</ul>

```

- Add other information to the User.
If we need to store any other information of the user profile, we can create another model with the new data.

- Create a new model “Perfil.cs” in **Model** folder.



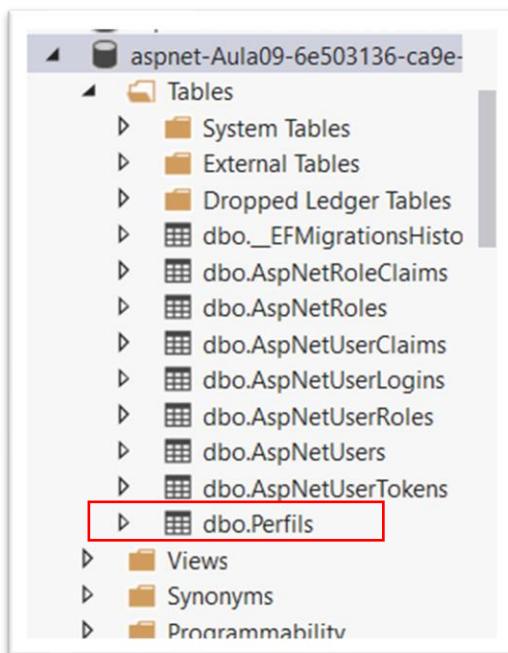
-add a new DbSet to the **ApplicationContext** class.



```
namespace Aula09.Data
{
    public class ApplicationContext : IdentityDbContext
    {
        public ApplicationContext(DbContextOptions<ApplicationContext> options)
            : base(options)
        {
        }

        public DbSet<Perfil> Perfilis { get; set; }
    }
}
```

- Add a new Migration
PM> add-migration addPerfil
- Update the database
PM> update-database
- Confirm the new table in the database



-alter the register page to accept the new information in the **InputModel**.

```
public class InputModel
{
    [Required]
    [Display(Name = "User Name")]
    4 references
    public string UserName { get; set; }

    [DataType(DataType.Date)]
    0 references
    public DateTime Birthday { get; set; }

    [Required]
    [EmailAddress]
    [Display(Name = "Email")]
    6 references
    public string Email { get; set; }
}
```

Inject the **ApplicationDbContext** in the code to access the tables of the database

```
public class RegisterModel : PageModel
{
    private readonly SignInManager<IdentityUser> _signInManager;
    private readonly UserManager<IdentityUser> _userManager;
    private readonly IUserStore<IdentityUser> _userStore;
    private readonly IUserEmailStore<IdentityUser> _emailStore;
    private readonly ILogger<RegisterModel> _logger;
    private readonly IEmailSender _emailSender;
    private readonly ApplicationDbContext _dbContext;

    0 references
    public RegisterModel(
        UserManager<IdentityUser> userManager,
        IUserStore<IdentityUser> userStore,
        SignInManager<IdentityUser> signInManager,
        ILogger<RegisterModel> logger,
        IEmailSender emailSender,
        ApplicationDbContext dbContext)
    {
        _userManager = userManager;
        _userStore = userStore;
        _emailStore = GetEmailStore();
        _signInManager = signInManager;
        _logger = logger;
        _emailSender = emailSender;
        _dbContext = dbContext;
    }
}
```

- If we success registering a new user, do an insert into table Perfil with the other information.

```
await _userStore.SetUserNameAsync(user, Input.UserName, CancellationToken.None);
await _emailStore.SetEmailAsync(user, Input.Email, CancellationToken.None);
var result = await _userManager.CreateAsync(user, Input.Password);

if (result.Succeeded)
{
    _logger.LogInformation("User created a new account with password.");
    Perfil newPerfil = new Perfil { UserName = user.UserName, Birthday = Input.Birthday };
    _dbcontext.Perfil.Add(newPerfil);
    _dbcontext.SaveChanges();

    var userId = await _userManager.GetUserIdAsync(user);
    var code = await _userManager.GenerateEmailConfirmationTokenAsync(user);
    code = WebEncoders.Base64UrlEncode(Encoding.UTF8.GetBytes(code));
    var callbackUrl = Url.Page(
        "/Account/Confirm",
        new { id = userId, code = code },
        Request.Scheme);
}
```

- Alter the view code to insert the new information:

```
<div class="form-floating mb-3">
    <input asp-for="Input.Email" class="form-control" autocomplete="username" aria-required="true" placeholder="name@example.com" />
    <label asp-for="Input.Email">Email</label>
    <span asp-validation-for="Input.Email" class="text-danger"></span>
</div>
<div class="form-floating mb-3">
    <input asp-for="Input.Birthday" class="form-control" autocomplete="birthday" aria-required="true" placeholder="birthday" />
    <label asp-for="Input.Birthday">Birthday</label>
    <span asp-validation-for="Input.Birthday" class="text-danger"></span>
</div>
<div class="form-floating mb-3">
```

- Register a new User:

Create a new account.

UserName
Joselito

Email
jose@xpto.pt

Birthday
06/06/2001

Password

Confirm Password

Register

- Confirm the data in the perfils table

The screenshot shows a database viewer window titled "dbo.Perfils [Data]". The table has four columns: Id, Birthday, and UserName. There are two rows: the first row contains Id=1, Birthday="06/06/2001 00:00:00", and UserName="Joselito"; the second row is a placeholder with Id=NULL, Birthday=NULL, and UserName=NULL. The "Max Rows" dropdown is set to 1000.

	Id	Birthday	UserName
▶	1	06/06/2001 00:00:00	Joselito
*	NULL	NULL	NULL

Second step – Using Authorization

The **authorization** refers to the process that determines what a User is allowed to do.

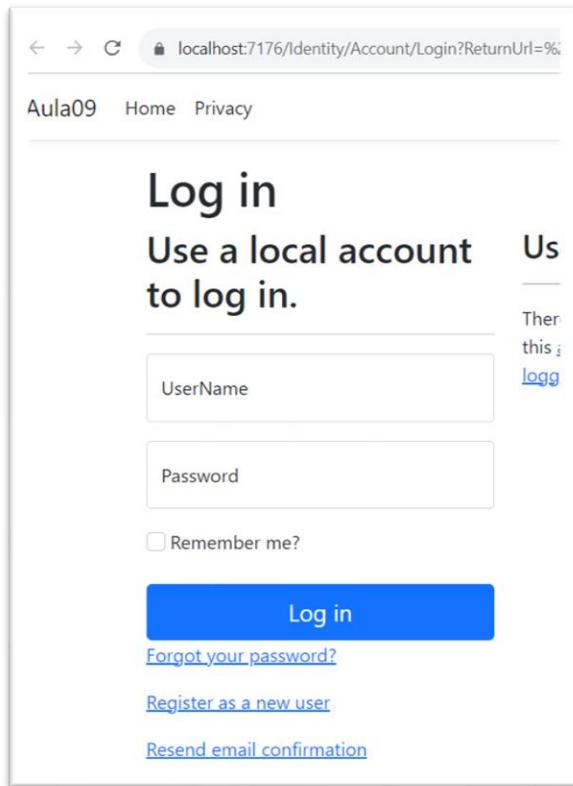
Open the **Home** controller:

The screenshot shows the code for the HomeController.cs file. The [Authorize] filter is highlighted with a red rectangle. The code is as follows:

```
namespace Aula09.Controllers
{
    [Authorize]
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;

        public HomeController(ILogger<HomeController> logger)
        {
            _logger = logger;
        }
    }
}
```

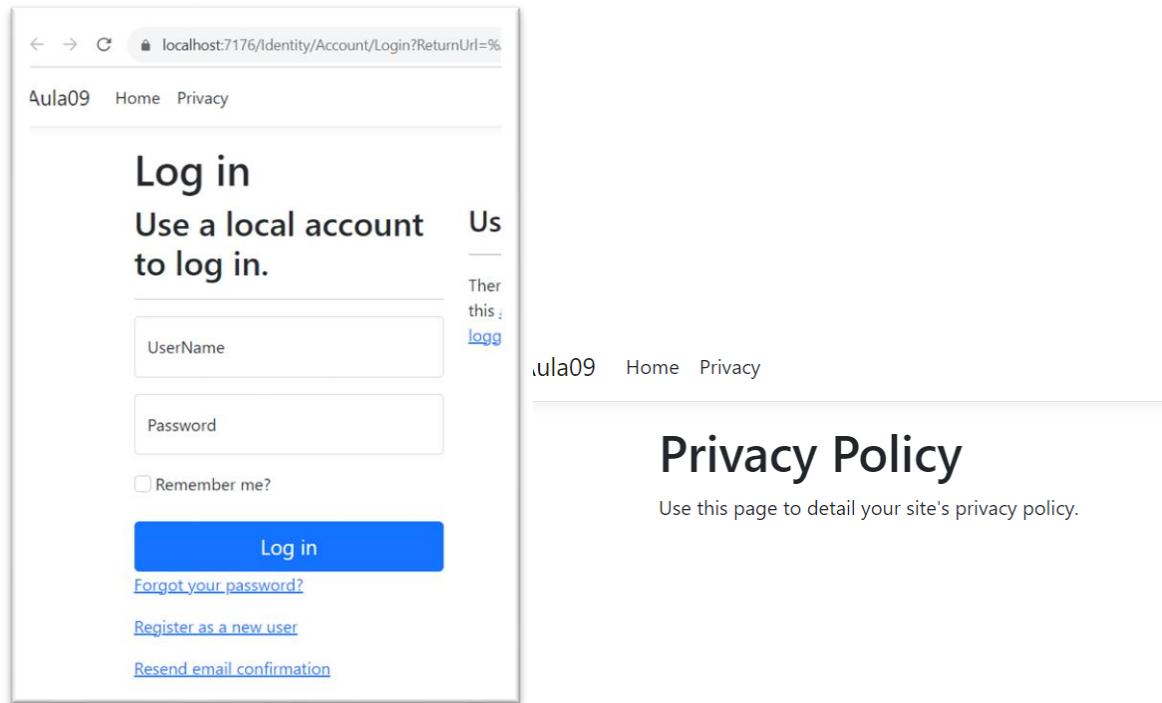
The [Authorize] filter, says that to access this controller (Home) the user needs to be authenticated. If he is not authenticated then is redirect to the login page.



- We can authorize some actions or controllers to be access to anonymous users:

```
[AllowAnonymous]
public IActionResult Privacy()
{
    return View();
}
```

- In this case, the controller is protected, but the action Privacy is not.



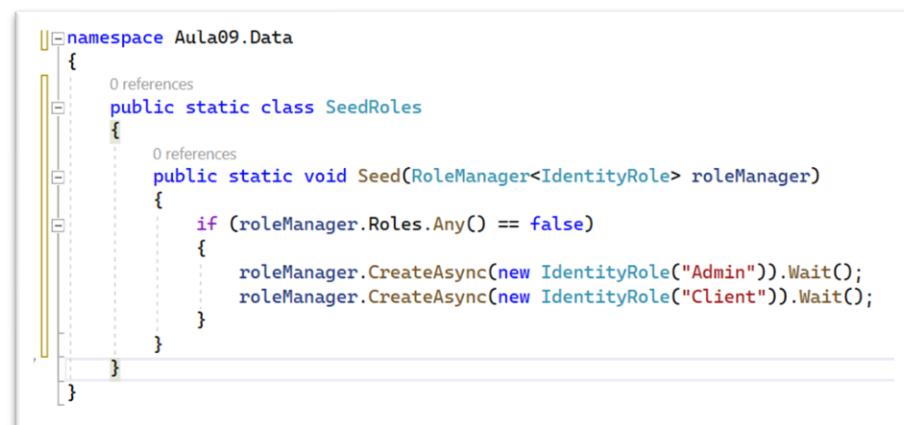
Third step – Using Authorization with Roles

The authorization can be done using Roles

- Add Roles in configuration:
 - Open **Program.cs** file

```
builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedEmail = true)
    .AddRoles<IdentityRole>()
    .AddEntityFrameworkStores<ApplicationContext>();
```

- Create a new classe **SeedRoles.cs** in, the **Data** folder, to seed the predefined Roles:



In the **Program.cs**, call this code:

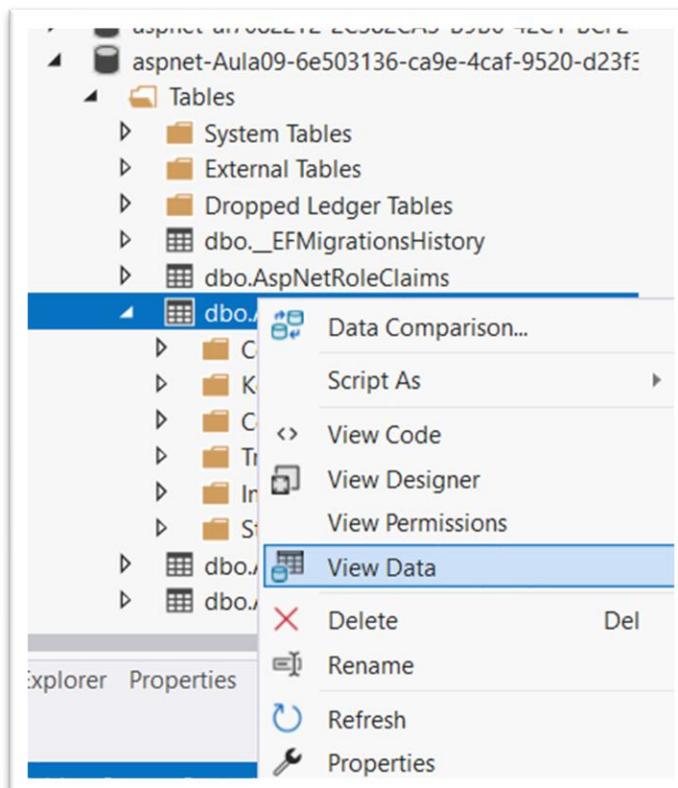
```
builder.Services.AddControllersWithViews();

builder.Services.AddScoped<RoleManager<IdentityRole>>();

var app = builder.Build();

using(var scope = app.Services.CreateScope())
{
    var roleManager=scope.ServiceProvider.GetRequiredService<RoleManager<IdentityRole>>();
    SeedRoles.Seed(roleManager);
}
```

- Run the application and close.
- Verify the existence of the new Roles.



The screenshot shows a grid view titled 'Ibo.AspNetRoles [Data]'. The table has columns: Id, Name, NormalizedNa..., and ConcurrencySt... . There are two visible rows: one for 'Client' (Id: eb-811159ae9619, Name: Client, NormalizedNa...: CLIENT, ConcurrencySt...: NULL) and one for 'Admin' (Id: ec4c8e9f-9f30..., Name: Admin, NormalizedNa...: ADMIN, ConcurrencySt...: NULL). The 'Name' column is highlighted with a red box.

	Id	Name	NormalizedNa...	ConcurrencySt...
>	eb-811159ae9619	Client	CLIENT	NULL
*	ec4c8e9f-9f30...	Admin	ADMIN	NULL
*	NULL	NULL	NULL	NULL

- Alter the Register process to choose an Role:
 - Open Register.cshtml.cs file
 - Inject a RoleManager:

```
public class RegisterModel : PageModel
{
    private readonly SignInManager<IdentityUser> _signInManager;
    private readonly UserManager<IdentityUser> _userManager;
    private readonly IUserStore<IdentityUser> _userStore;
    private readonly IUserEmailStore<IdentityUser> _emailStore;
    private readonly ILogger<RegisterModel> _logger;
    private readonly IEmailSender _emailSender;
    private readonly ApplicationDbContext _dbContext;
    private readonly RoleManager<IdentityRole> _roleManager;

    0 references
    public RegisterModel(
        UserManager<IdentityUser> userManager,
        IUserStore<IdentityUser> userStore,
        SignInManager<IdentityUser> signInManager,
        ILogger<RegisterModel> logger,
        IEmailSender emailSender,
        ApplicationDbContext dbContext,
        RoleManager<IdentityRole> roleManager
    )
    {
        _userManager = userManager;
        _userStore = userStore;
        _emailStore = GetEmailStore();
        _signInManager = signInManager;
        _logger = logger;
        _emailSender = emailSender;
        _dbContext = dbContext;
        _roleManager = roleManager;
    }
}
```

- Add a new property in the InputModel

```
6 references
public string Email { get; set; }

[Required]
0 references
public string Role { get; set; }
```

- Send the existing **Roles** to the view using the **ViewData** structure.

```
public async Task OnGetAsync(string returnUrl = null)
{
    ViewData["roles"] = _roleManager.Roles.ToList();
    returnUrl = returnUrl;
    ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).To
```

- Alter the View

```
</div>
<div class="form-floating mb-3">
    <label asp-for="Input.Role"></label>
    <select asp-for="Input.Role" class="form-control" asp-items='new SelectList(ViewBag.roles, "Name", "Name")'>
        <option value="">-- Select a Role --</option>
    </select>
    <span asp-validation-for="Input.Role" class="text-danger"></span>
</div>
<button id="registerSubmit" type="submit" class="w-100 h-100 btn-primary">Register</button>
```

- Alter de OnPostAsync method

```
if (result.Succeeded)
{
    _logger.LogInformation("User created a new account with password.");
    await _userManager.AddToRoleAsync(user, Input.Role);
    Perfil newPerfil = new Perfil { UserName = user.UserName, Birthday = Input.Birt
    _dbcontext.Perfils.Add(newPerfil);
```

...

```
}
```

// If we got this far, something failed, redisplay form

```
ViewData["roles"] = _roleManager.Roles.ToList();
return Page();
```

- Test register a new user of the Role “Admin”

Register

Create a new account.

User

There
[this A](#)

UserName Administrador	
Email admin@xpto.pt	
Birthday 07/02/2012	<input type="button" value=""/>
Password *****	
Confirm Password *****	
Admin	

Register

- Create a Client user

Register

Create a new account.

User

There
[this A](#)

UserName ClienteManel	
Email Manel@xpto.com	
Birthday 02/12/2020	<input type="button" value=""/>
Password *****	
Confirm Password *****	
Client	

Register

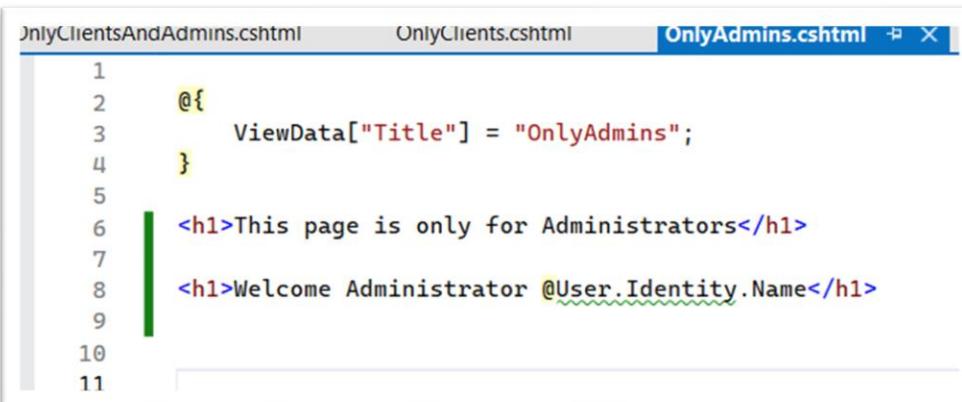
- With the roles created and users of those roles, we can do the protection through the roles:
 - open the Home Control and create the Actions

```
[Authorize(Roles = "Admin")]
0 references
public IActionResult OnlyAdmins()
{
    return View();
}

[Authorize(Roles = "Client")]
0 references
public IActionResult OnlyClients()
{
    return View();
}

[Authorize(Roles = "Admin, Client")]
0 references
public IActionResult OnlyClientsAndAdmins()
{
    return View();
}
```

- Create the correspondents Views



```
OnlyClientsAndAdmins.cshtml      OnlyClients.cshtml      OnlyAdmins.cshtml X
1
2     @{
3         ViewData["Title"] = "OnlyAdmins";
4     }
5
6     <h1>This page is only for Administrators</h1>
7
8     <h1>Welcome Administrator @User.Identity.Name</h1>
9
10
11
```

The screenshot shows a code editor with three tabs: OnlyClientsAndAdmins.cshtml, OnlyClients.cshtml*, and OnlyAdmins.cshtml. The OnlyClients.cshtml tab is active. The code is as follows:

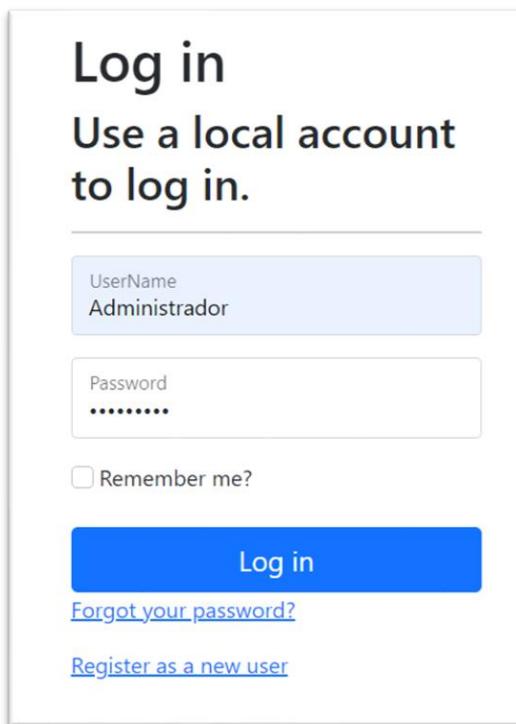
```
1  @{
2      ViewData["Title"] = "OnlyClients";
3  }
4
5  <h1>This page is only for Clients</h1>
6
7  <h1>Welcome Client @User.Identity.Name</h1>
8
9
10
```

The screenshot shows a code editor with four tabs: onlyClientsA...dmins.cshtml, OnlyClients.cshtml*, OnlyAdmins.cshtml, and Register.cshtml. The onlyClientsA...dmins.cshtml tab is active. The code is as follows:

```
1  @{
2      ViewData["Title"] = "OnlyClientsAndAdmins";
3  }
4
5  <h1>This page is only for Administrators and Clients</h1>
6  <h1>
7      Welcome @if (User.IsInRole("Admin"))
8      {
9          <span>Administrator </span>
10     }
11    else
12    {
13        <span>Client </span>
14    }
15    @User.Identity.Name
16
17 </h1>
18
```

A red callout box points to the `@if (User.IsInRole("Admin"))` condition in line 8, containing the text: "See if the User is of the Role "Admin". If not, is "Client"

- Run the application and Login with Administrator account



The screenshot shows a login page with the title "Log in" and the subtitle "Use a local account to log in." It contains two input fields: "UserName" with the value "Administrador" and "Password" with several dots. There is a "Remember me?" checkbox, which is unchecked. A large blue "Log in" button is at the bottom left, with "Forgot your password?" and "Register as a new user" links below it.

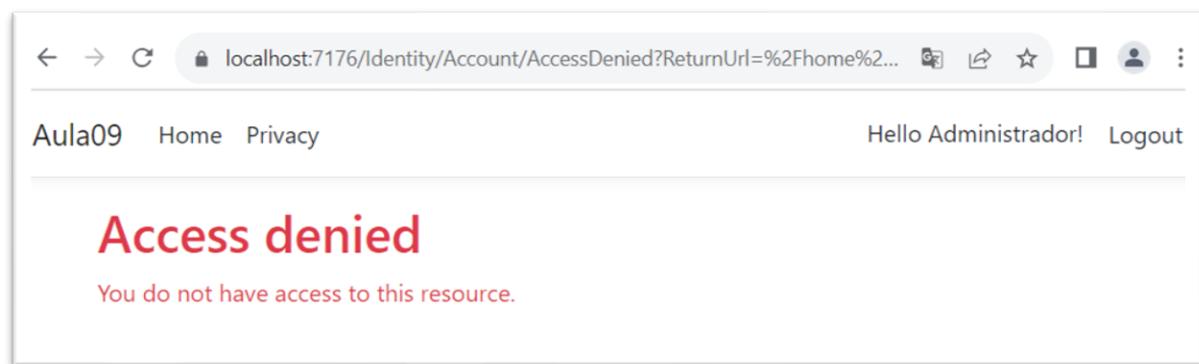
- go to OnlyAdmins action: <https://localhost:44328/home/onlyadmins>.



The screenshot shows a browser window with the URL "localhost:7176/home/onlyadmins". The page content says "This page is only for Administrators" and "Welcome Administrator Administrador". The top navigation bar shows "Aula09 Home Privacy" and the right side shows "Hello Administrador! Logout".

Have access, because the user is in the Role "Admin"

- go to OnlyAdmins action: <https://localhost:44328/home/onlyclients>.



The screenshot shows a browser window with the URL "localhost:7176/Identity/Account/AccessDenied?ReturnUrl=%2Fhome%2F...". The page content says "Access denied" and "You do not have access to this resource.". The top navigation bar shows "Aula09 Home Privacy" and the right side shows "Hello Administrador! Logout".

Do not have access because the user is not a "Client"

- go to OnlyAdmins action: <https://localhost:44328/home/OnlyClientsAndAdmins>.



- Test the application with a user of Role "Client".
- Adapt the Menu to show only the options allowed to the role of the user authenticated.
 - Open the _Layout.cshtml file:

```
<ul class="navbar-nav flex-grow-1">
    @if (User.IsInRole("Client"))
    {
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="OnlyClients">Clients</a>
        </li>
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="OnlyClientsAndAdmins">Clients e Administrors</a>
        </li>
    }
    @if (User.IsInRole("Admin"))
    {
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="OnlyAdmins">Administrator</a>
        </li>
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="OnlyClientsAndAdmins">Clients e Administrors</a>
        </li>
    }
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
    </li>

```