



## Session 3 - Roles

Because even in the digital world,  
someone's got to be the boss!

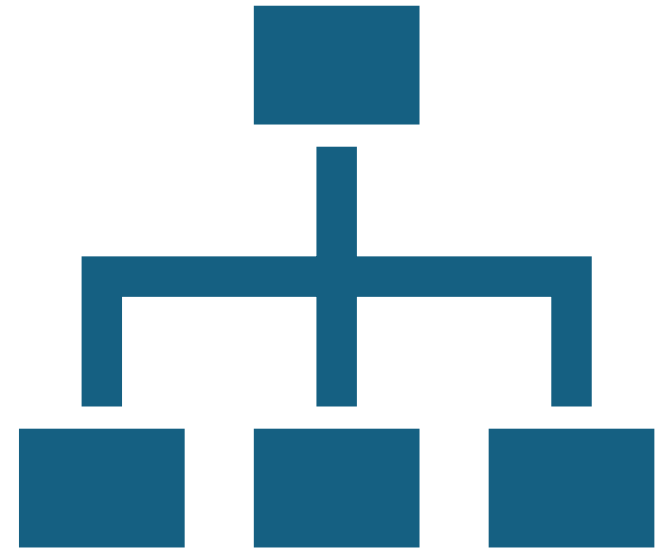
# Roles

- Roles in an application represent a way to group users based on their permissions and responsibilities. By assigning roles to users, you can control their access to various parts of the application and ensure they have the appropriate level of privileges.



# Examples of Roles

- **Admin**
  - Can manage users, roles, and settings.
  - Has full access to all features and data within the application.
- **Manager**
  - Can oversee team projects and manage resources.
  - Has access to detailed reports and performance metrics, but limited administrative settings.
- **User**
  - Can use the main features of the application.
  - Has access to personal data and basic functionalities but restricted from administrative tasks.



# Introduction to Roles and User Management

- In the context of an ASP.NET Core application, roles play a crucial role in managing user permissions and access levels. By defining roles and assigning them to users, you can efficiently control who has access to different parts of your application and what actions they can perform. The following three tables are fundamental to implementing this role-based access control:

# AspNetRoles

- Description:** This table stores information about the roles in the application. Each role has a unique identifier, a name, and a normalized name for efficient lookups.
- Used for:** Creating and storing roles such as "Admin" and "Manager".



# AspNetUsers

- Description:** This table stores information about the users in the application. Each user has a unique identifier, username, email, and other related information.

- Used for:** Creating and storing user accounts such as `admin@bb.cc` and `manager@bb.cc`.



# AspNetUserRoles

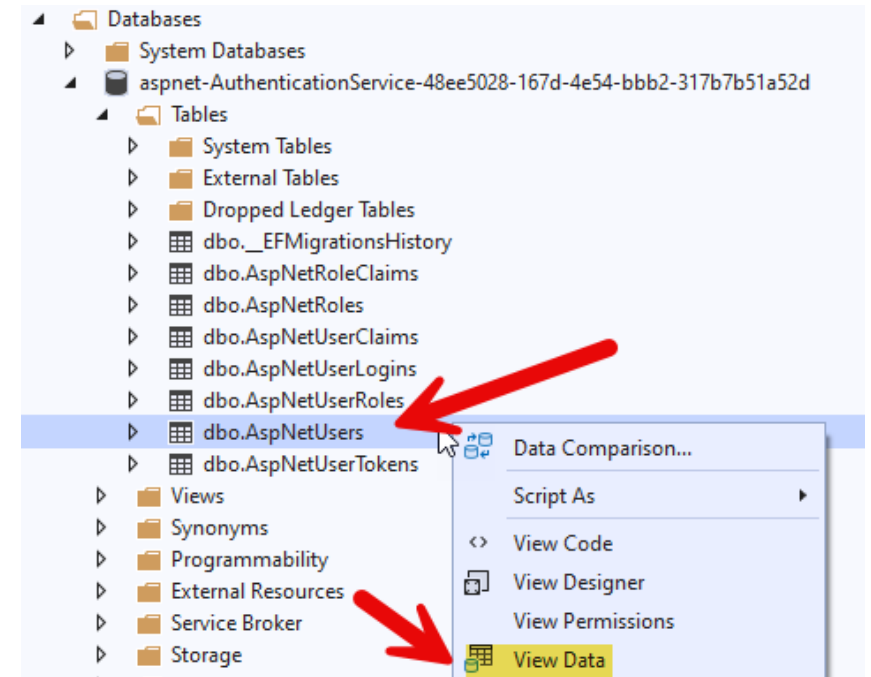
- Description:** This table establishes the many-to-many relationship between users and roles. It links users to the roles they are assigned to.

- Used for:** Assigning roles to users by linking the user's ID with the role's ID.



# Reviewing ASP Roles Tables

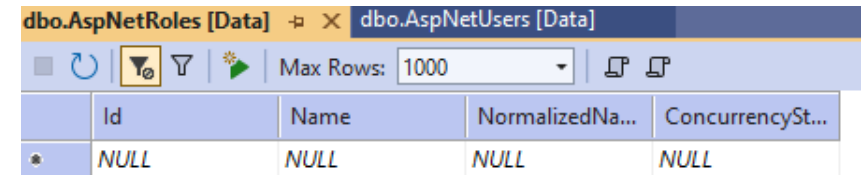
1. Open the Authentication Service database and select theAspNetUsers table.
2. Right-Click on the **AspNetUser table** and click **View Data**.

[illegible]



# Reviewing ASP Roles Tables (Continue)

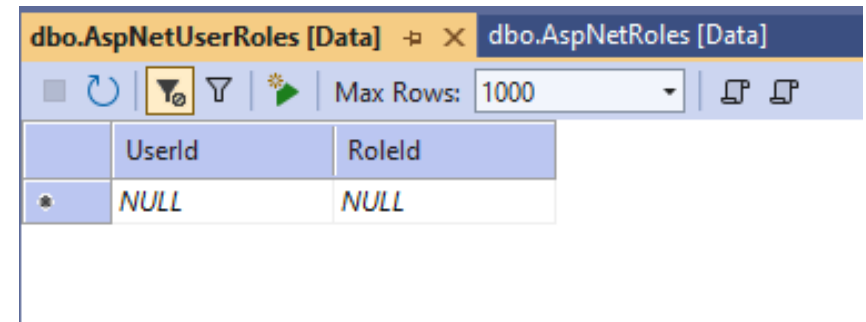
1. Right-Click on the **AspNetRoles** table and click **View Data**.



The screenshot shows the SQL Server Enterprise Manager interface. The top pane displays the table 'dbo.AspNetRoles [Data]'. The bottom pane shows a single row with all NULL values. The table has four columns: Id, Name, NormalizedName, and ConcurrencyStamp.

	Id	Name	NormalizedNa...	ConcurrencySt...
*	NULL	NULL	NULL	NULL

2. Right-Click on the **AspNetUserRoles** table and click **View Data**.



The screenshot shows the SQL Server Enterprise Manager interface. The top pane displays the table 'dbo.AspNetUserRoles [Data]'. The bottom pane shows a single row with NULL values for both UserId and RoleId. The table has two columns: UserId and RoleId.

	UserId	RoleId
*	NULL	NULL

# Creating Admin & Manager Roles

Create 2 new user account with the following information (admin and manager) with a default password of “dmit2018@NAIT”.

Admin: [admin@bb.cc](mailto:admin@bb.cc)

Manager: [manager@bb.cc](mailto:manager@bb.cc)

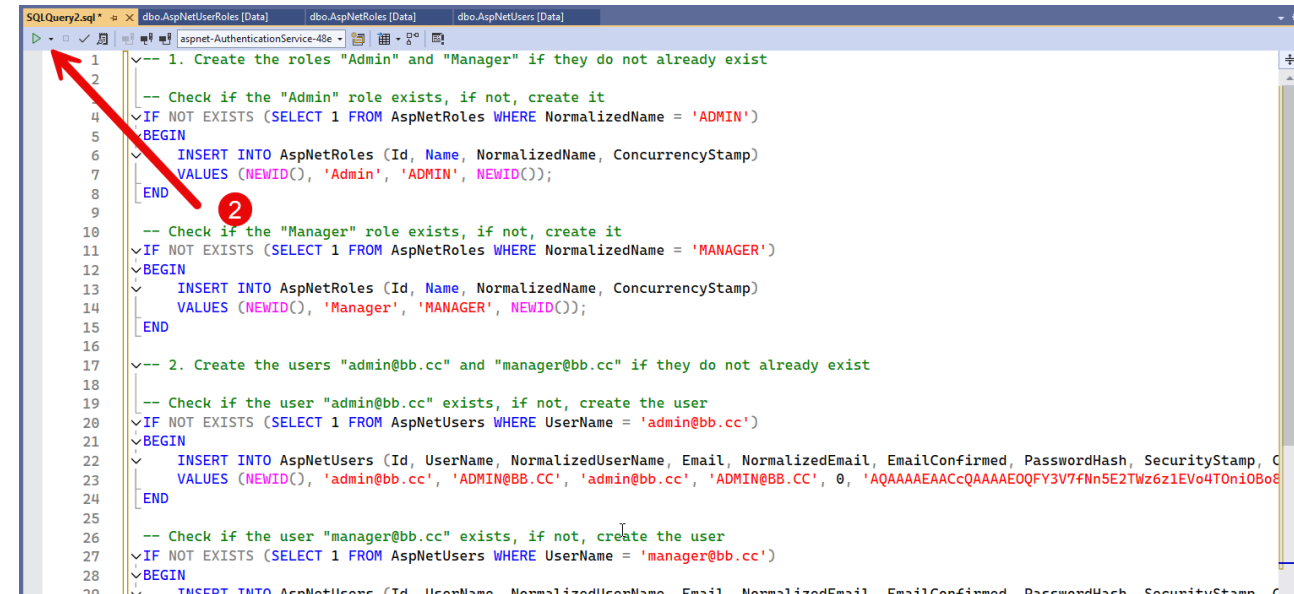
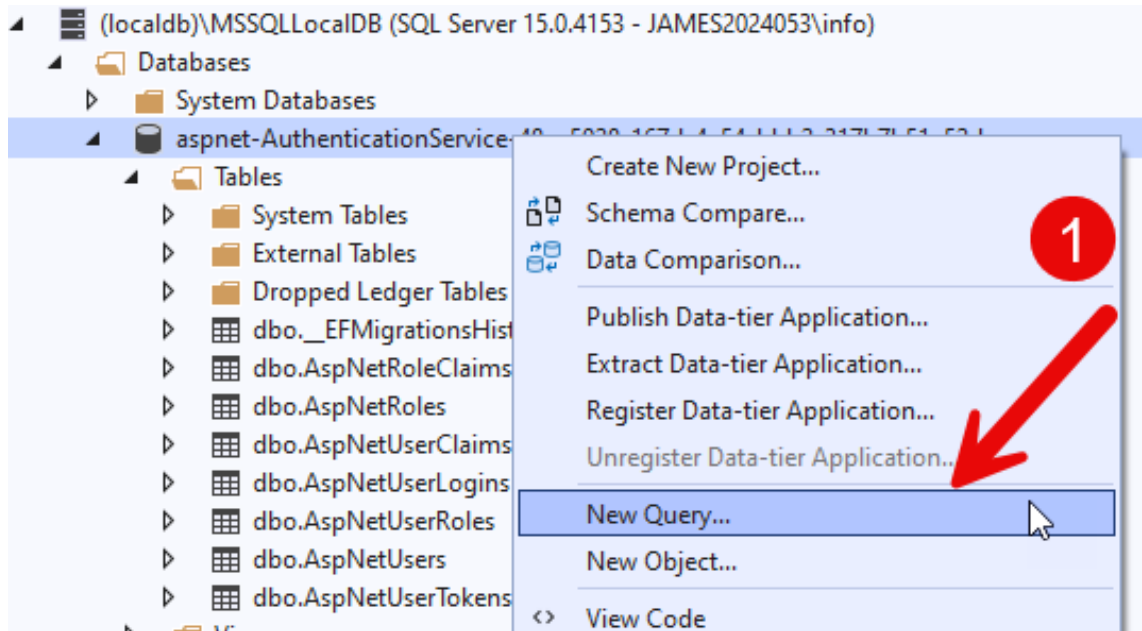
## Register

Create a new account.

Register

# Running the Role Script

1. Right click on the database and select “New Query”
2. Paste the content of the SQL script into the editor and click the “Run” button.



dbo.AspNetUsers [Data] SQLQuery2.sql *							
	Id	UserName	NormalizedUs...	Email	NormalizedEm...	EmailConfirmed	PasswordHa
	1E7BED0B-8684...	manager@bb.cc	MANAGER@BB...	manager@bb.cc	MANAGER@BB...	True	AQAAAAEA/
	8e5-0471a021f69e	jthompson@na...	JTHOMPSON@...	jthompson@na...	JTHOMPSON@...	True	AQAAAAIAA
	A0D4BF7D-DCB...	admin@bb.cc	ADMIN@BB.CC	admin@bb.cc	ADMIN@BB.CC	True	AQAAAAEA/
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

dbo.AspNetRoles [Data] dbo.AspNetUsers [Data] SQLQuery2.sql *				
	Id	Name	NormalizedNa...	ConcurrencySt...
	669AE368-96E3...	Manager	MANAGER	D0CB6806-AB2...
	95DA36B1-FC6...	Admin	ADMIN	5DB6AD58-6AD...
▶*	NULL	NULL	NULL	NULL

dbo.AspNetUserRoles [Data] dbo.AspNetRoles [Data]		
	UserId	RoleId
	1E7BED0B-8684-4EB...	669AE368-96E3-4C2A-AB4...
	A0D4BF7D-DCB2-44...	95DA36B1-FC61-4660-9C6...
▶*	NULL	NULL

## Review the Asp Tables

NOTE: A user can belong to multiple roles

# Update the Program.cs

- You need to update the “Program.cs” file to make use of the roles.

```
builder.Services.AddIdentityCore<ApplicationUser>(setupAction: options => options.SignIn.RequireConfirmedAccount = true)
    .AddRoles<IdentityRole>() // Enables role-based authorization and adds role-related services.
    .AddEntityFrameworkStores<ApplicationDbContext>()
    .AddSignInManager()
    .AddDefaultTokenProviders()
    .AddRoleManager<RoleManager<IdentityRole>>(); // Adds the role manager service for managing roles.
```

# Apply Roles to Pages


- We are going to update the functionality of the “Weather” page by doing the following.
  1. Add the following tags to the Weather page <AuthorizeView>, <NotAuthorized> and <Authorized> and demonstrate role access.
  2. Add function to the menu items so only a “Manager” or “Admin” can see the “Weather” page.
  3. Add a role to the button so that only the “Admin” will be able to see it.
  4. Refactor the “Weather” page by adding a “Refresh” button and moving the existing code to a new method call “RefreshWeather()”.

# Adding AuthenticationStateProvider Tags to Weather Page

```
gram.cs Weather.razor X AuthenticationService.Components.Pages.Weather
@page "/weather"
@inject AuthenticationStateProvider AuthenticationStateProvider

<PageTitle>Weather</PageTitle>

<h1>Weather</h1>
```



# Adding “Authorize View” With Roles and “Not Authorized” Tags to Weather Page

```
@page "/weather"
@inject AuthenticationStateProvider AuthenticationStateProvider

<PageTitle>Weather</PageTitle>

<h1>Weather</h1>
<AuthorizeView Roles="Admin, Manager"> 1
    <NotAuthorized> 2
        <!-- Content for users not in the Admin/Manager role -->
        <p>You do not have access to this content. Please contact an admin for assistance.</p>
    </NotAuthorized>
```



# Adding “Authorized” Tags to Weather Page

```
<AuthorizeView Roles="Admin, Manager">
  <NotAuthorized>
    <!-- Content for users not in the Admin/Manager role -->
    <p>You do not have access to this content. Please contact an admin for assistance.</p>
  </NotAuthorized>

  <Authorized> 1
    <p>This component demonstrates showing data.</p>

    @if (forecasts == null)
    {
```

# Adding “Authorized” Tags to Weather Page

```
<AuthorizeView Roles="Admin, Manager">
  <NotAuthorized>
    <!-- Content for users not in the Admin/Manager role -->
    <p>You do not have access to this content. Please contact an admin for assistance.</p>
  </NotAuthorized>

  <Authorized> 1
    <p>This component demonstrates showing data.</p>

    @if (forecasts == null)
    {
```

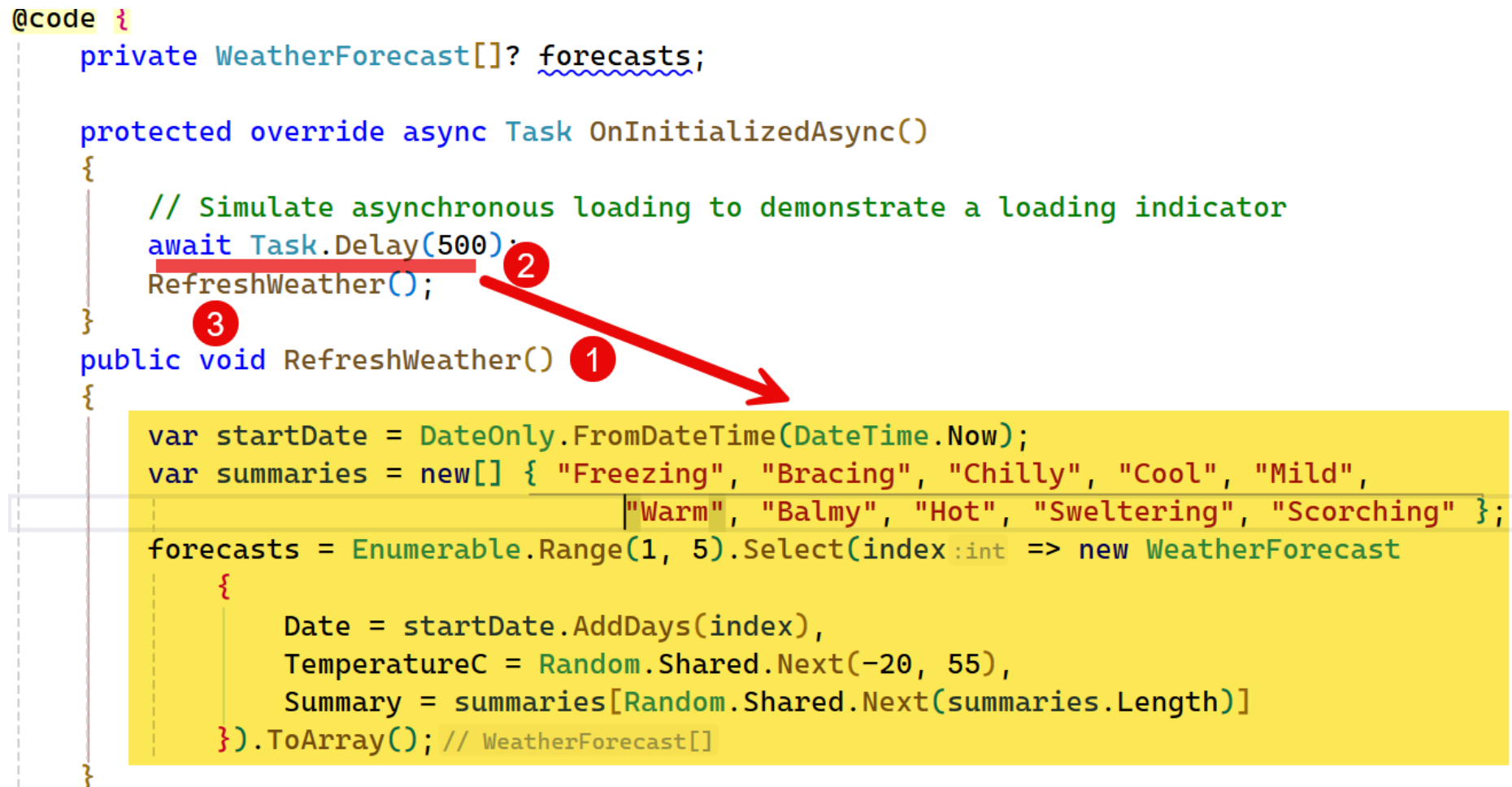
# Add Refresh Button with Roles and Closing Tags

```
        </tbody>
    </table>
}
1 @if (AuthenticationStateProvider.GetAuthenticationStateAsync() // Task<AuthenticationState>
    .Result.User.Identity.IsAuthenticated // bool
    && AuthenticationStateProvider.GetAuthenticationStateAsync() // Task<AuthenticationState>
    .Result.User.IsInRole("Admin"))
{
    <button onclick="@RefreshWeather">Refresh</button>
}
</Authorized>
2
</AuthorizeView>
3
```

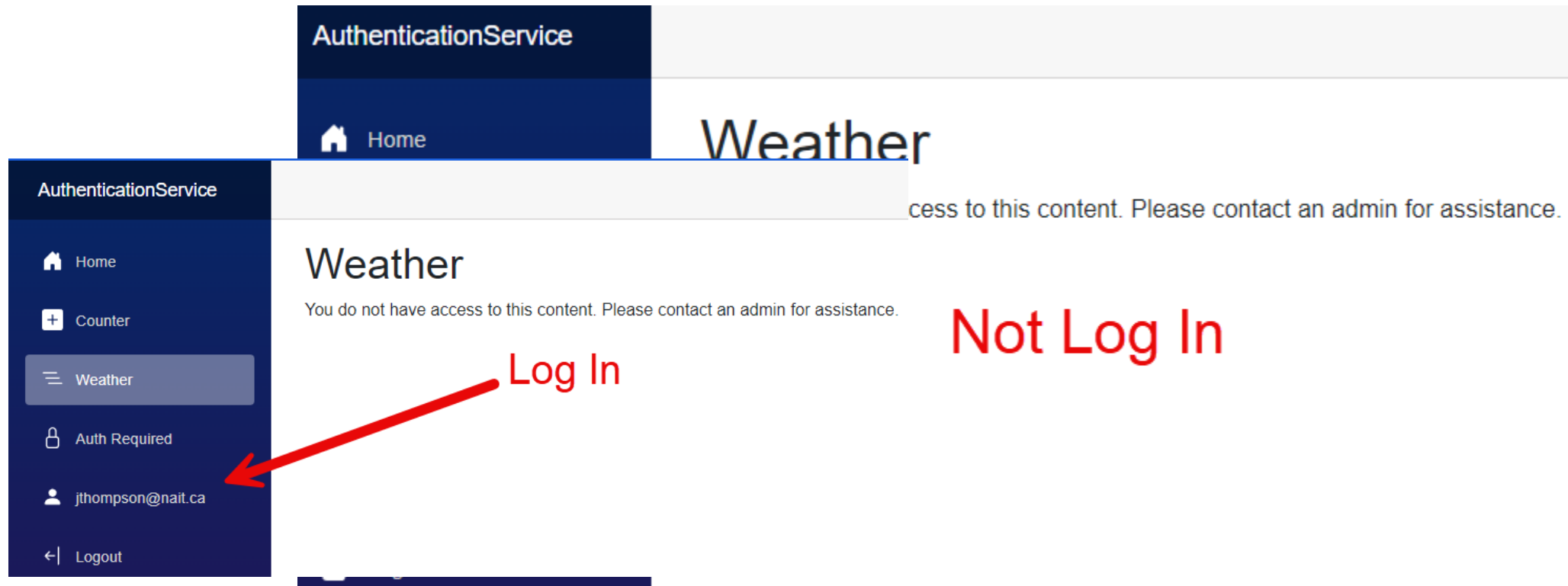
## Add Refresh Weather Method and Refactor OnInitializedAsync()

1. Add a new method call RefreshWeather()
2. Move existing code excluding “await Task.Delay(500)” to the RefreshWeather() method.
3. Add code to call the RefreshWeather() method on initialized.

```
@code {  
    private WeatherForecast[]? forecasts;  
  
    protected override async Task OnInitializedAsync()  
    {  
        // Simulate asynchronous loading to demonstrate a loading indicator  
        await Task.Delay(500);  
        RefreshWeather();  
    }  
  
    public void RefreshWeather()  
    {  
        var startDate = DateOnly.FromDateTime(DateTime.Now);  
        var summaries = new[] { "Freezing", "Bracing", "Chilly", "Cool", "Mild",  
                                "Warm", "Balmy", "Hot", "Sweltering", "Scorching" };  
        forecasts = Enumerable.Range(1, 5).Select(index:int => new WeatherForecast  
        {  
            Date = startDate.AddDays(index),  
            TemperatureC = Random.Shared.Next(-20, 55),  
            Summary = summaries[Random.Shared.Next(summaries.Length)]  
        }).ToArray(); // WeatherForecast[]  
    }  
}
```



# Non-Role User



# Manager Role

AuthenticationService

Home

Counter

Weather

Auth Required

manager@bb.cc

Logout

## Weather

This component demonstrates showing data.

Date	Temp. (C)
7/26/2024	6
7/27/2024	14
7/28/2024	6
7/29/2024	46
7/30/2024	36

No Refresh Button

# Admin Role

AuthenticationService

Home

Counter

Weather

Auth Required

admin@bb.cc

Logout

## Weather

This component demonstrates showing data.

Date	Temp. (C)
7/26/2024	40
7/27/2024	9
7/28/2024	-3
7/29/2024	43
7/30/2024	13

Refresh

# User Identity Claims

- Claims are key-value pairs that describe the user
- Issued by the identity provider during authentication
- Stored in the ClaimsPrincipal object in .NET
- Common claims include:
  - Name (username)
  - Email
  - Role
  - NameIdentifier (user ID)
- Used for personalization, access control, and auditing



The role names come from the `AspNetRoles.Name` field (e.g., Admin, Manager).


Roles are added to the user's claims as `ClaimTypes.Role` and used for authorization checks like `[Authorize(Roles = "Admin")]`.

# Displaying Claims Information

- User Id:
  - This is the unique user ID from **`AspNetUsers.Id`** (usually a GUID).
  - It's never an email or username.
- Username:
  - This is usually the `UserName` field from **`AspNetUsers.UserName`**.
  - Could be an email if email was used as the username during registration.
  - Otherwise, it could be something like `james123`, etc.
- Email:
  - This is the actual email address from **`AspNetUsers.Email`**.
- Role(s):
  - This refers to the user's assigned roles stored in the `AspNetUserRoles` table.
  - The role names come from the `AspNetRoles.Name` field (e.g., Admin, Manager).
  - Roles are added to the user's claims as `ClaimTypes.Role` and used for authorization checks like `[Authorize(Roles = "Admin")]`.

# Adding Placeholders for Claims Information

```
</AuthorizeView>  
@code {  
    private WeatherForecast[]? forecasts;  
    private string? userId;  
    private string? userName;  
    private string? email;  
    private List<string> roles = new();  
    protected override async Task OnInitializedAsync()  
}
```



# Getting Claim Information 1/2

```
var authState = await AuthenticationStateProvider.GetAuthenticationStateAsync();  
var user = authState.User;
```

```
if (user.Identity is not null && user.Identity.IsAuthenticated)  
{  
    // This is the unique user ID from AspNetUsers.Id (usually a GUID).  
    // It's never an email or username.  
    userId = user.FindFirst(ClaimTypes.NameIdentifier)?.Value;  
  
    // This is usually the UserName field from AspNetUsers.UserName  
    // Could be an email if email was used as the username during registration.  
    // Otherwise, it could be something like yourName123, etc.  
    userName = user.Identity.Name; // or user.FindFirst(ClaimTypes.Name)?.Value
```

# Getting Claim Information 2/2

```
userName = user.Identity.Name; // or user.FindFirst(ClaimTypes.Name)?.Value
```

```
// This is the actual email address fromAspNetUsers.Email  
email = user.FindFirst(ClaimTypes.Email)?.Value;
```

```
// Get all role claims  
roles = user.FindAll(ClaimTypes.Role).Select(r => r.Value).ToList();
```

```
}
```

```
// Simulate asynchronous loading to demonstrate a loading indicator
```

# Updating Blazor UI

```
<PageTitle>Weather</PageTitle>
```

```
<h1>Weather</h1>
```

```
<p>UserID: @userId</p>
```

```
<p>User Name: @userName</p>
```

```
<p>User Email: @email</p>
```

```
<p>User Role(s): @(roles.Any() ? string.Join(", ", roles) : "None")</p>
```

```
<AuthorizeView Roles="Admin, Manager">
```

# User with no Roles

## Weather

UserID: 7f97736b-5f76-41cc-a742-1e139cba7781

User Name: jthompson@nait.ca

User Email: jthompson@nait.ca

User Role(s): None



You do not have access to this content. Please contact an admin for assistance

# User with Roles

## Weather

UserID: b519a65f-6935-473b-b442-5f0bf0b58c73

User Name: admin@bb.cc

User Email: admin@bb.cc

User Role(s): Admin



This component demonstrates showing data.

# Statement Regarding Slide Accuracy and Potential Revisions

- Please note that the content of these PowerPoint slides is accurate to the best of my knowledge at the time of presentation. However, as new research and developments emerge, or to enhance the learning experience, these slides may be subject to updates or revisions in the future. I encourage you to stay engaged with the course materials and any announcements for the most current information

