










SESSION 2 - AUTHENTICATION

"Mastering User Authentication Techniques

CREATE A NEW BLAZOR APPLICATION (1 OF 3)

Create a new project

Recent project templates

 Blazor Web App	C#
 Console App	C#
 Console App	F#
 Telerik C# Windows Forms Application	C#
 Class Library	C#
 Windows Forms App (.NET Framework)	C#
 VSIX Project	C#

CREATE A NEW BLAZOR APPLICATION (2 OF 3)

Configure your new project

Blazor Web App

C#

Linux

macOS

Windows

Blazor

Cloud

Web

Project name

AuthenticationService

Location

C:\Users\info\source\repos

...

Solution name ⓘ

AuthenticationService

☐

Place solution and project in the same directory

CREATE A NEW BLAZOR APPLICATION (3 OF 3)

Additional information

Blazor Web App

C#

Linux

macOS

Windows

Blazor

Cloud

Web

Framework [i](#)

.NET 8.0 (Long Term Support)

Authentication type [i](#)

Individual Accounts

☒ Configure for HTTPS [i](#)

Interactive render mode [i](#)

Server

Interactivity location [i](#)

Global

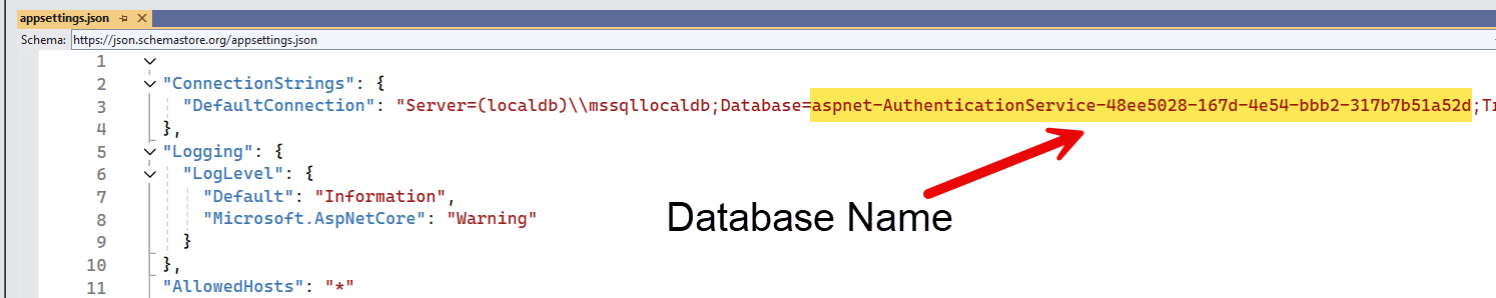
☒ Include sample pages [i](#)

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

☐ Enlist in .NET Aspire orchestration [i](#)

APPSETTINGS.JSON AND DEFAULT CONNECTION

- In Blazor, the “**DefaultConnection**” plays a significant role in authentication, especially when you're using **ASP.NET Core Identity** for managing user authentication and authorization.
- **ASP.NET Core Identity** is a membership system that adds login functionality to ASP.NET Core applications. It helps manage users, passwords, profile data, roles, claims, tokens, email confirmation, and more. Identity uses a database to store all this data, and **DefaultConnection** is often the connection string used to connect to this database.
- When configuring **ASP.NET Core Identity**, the database specified in the **DefaultConnection** is automatically created if it does not exist, ensuring a unique, application-specific data store for user credentials and related security information. This process simplifies initial setup and maintains data integrity by isolating user data in a dedicated database environment.
- **NOTE:** The creation of the database does not occur until the first attempt to log into the application.



```
1  ✓
2  ✓ "ConnectionStrings": {
3    "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=aspnet-AuthenticationService-48ee5028-167d-4e54-bbb2-317b7b51a52d;Trusted_Connection=yes;"
4  },
5  ✓ "Logging": {
6    "LogLevel": {
7      "Default": "Information",
8      "Microsoft.AspNetCore": "Warning"
9    },
10  },
11  "AllowedHosts": "*"

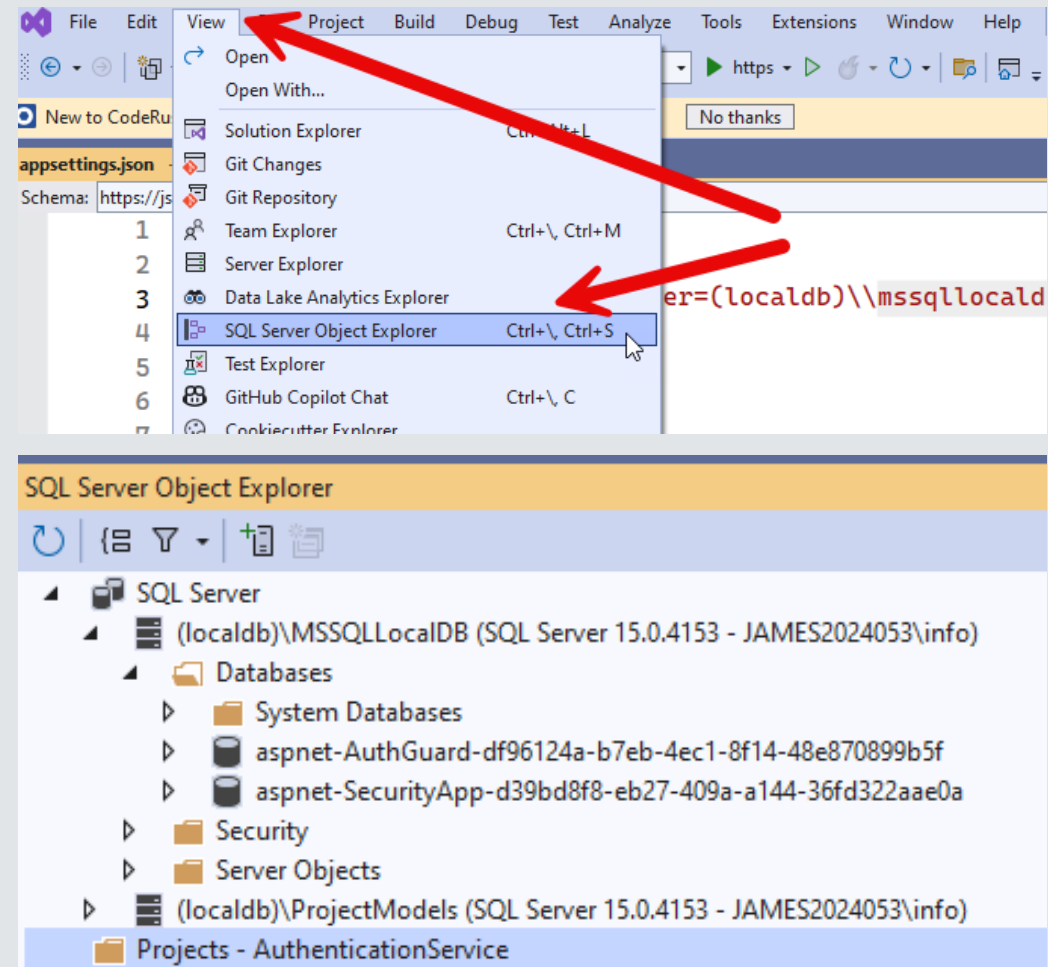
```

Schema: <https://json.schemastore.org/appsettings.json>

Database Name

REVIEWING THE MSSQLLOCALDB DATABASE

1. Open the SQL Server Object Explorer



Note: Currently, there is no database with the same name as specified in the **DefaultConnection** of our **appsettings.json**.

CREATING THE DEFAULT CONNECTION DATABASE

1. **Attempt to Log In:** Please try logging into the application. Note: Registration is necessary eventually, but attempting to log in first will trigger the creation of the database. *This will take a few second to redirect you to the database migrations.*

Log in

Use a local account to log in.

Email
jthompson@nait.ca

Password
.....

☐ Remember me

Log in

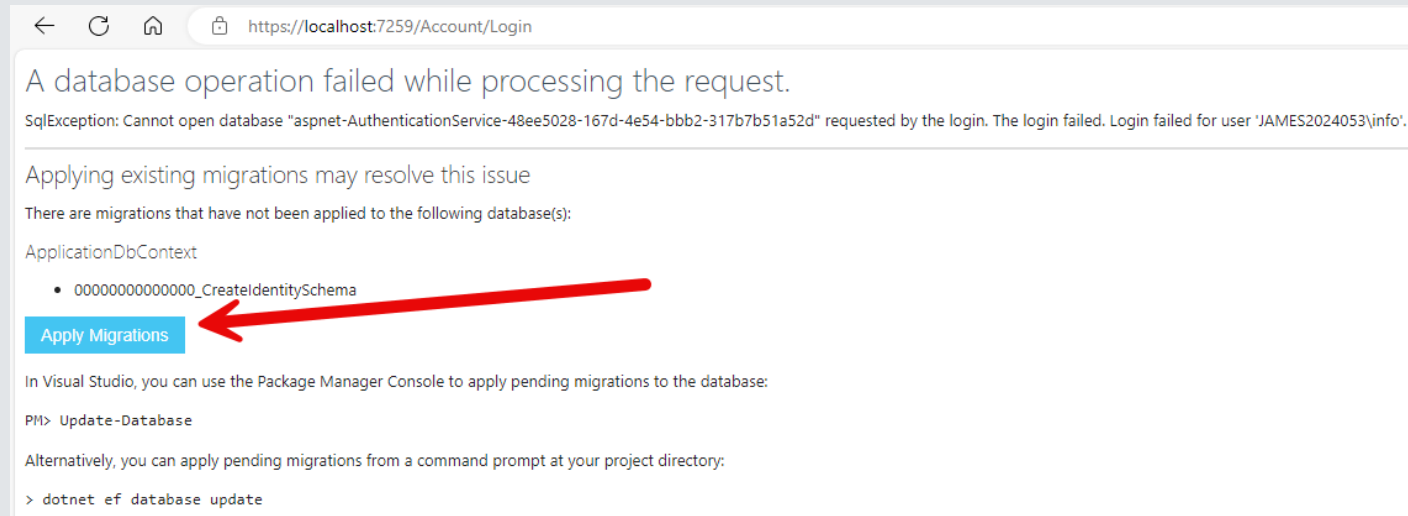
[Forgot your password?](#)

[Register as a new user](#)

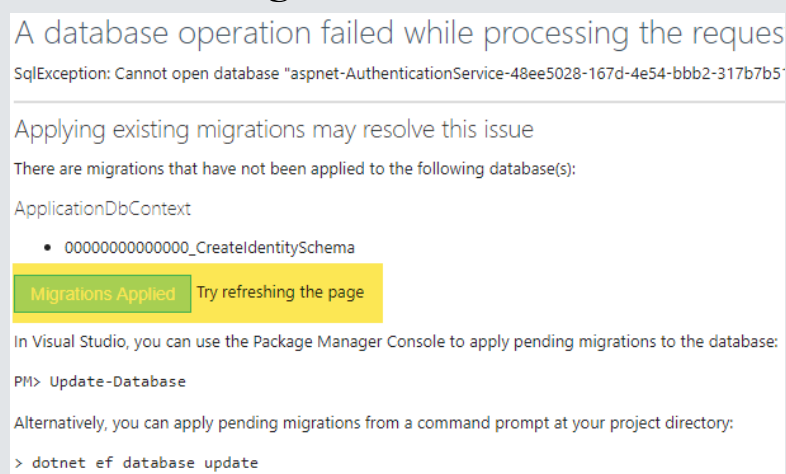
[Resend email confirmation](#)

CREATING THE DEFAULT CONNECTION DATABASE (CONTINUE)

1. Press the “Apply Migrations” to have the application create the default database.

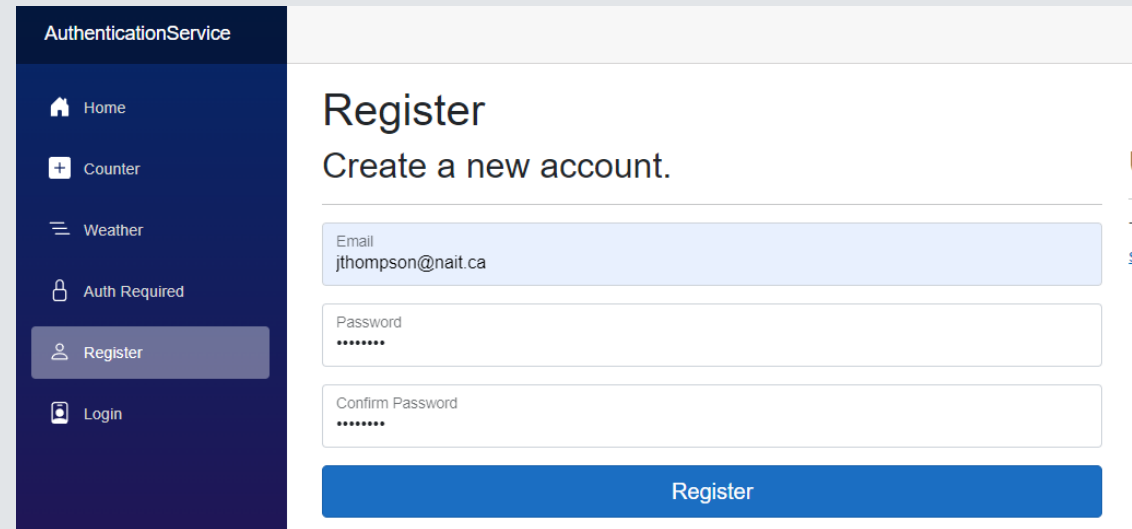


2. After applying the migrations, you can refresh the page. **At this point, you will need to register.**



REGISTERING YOUR ACCOUNT

1. Register your user information.



The screenshot shows a web application interface for registering a new account. On the left is a dark blue sidebar with the title 'AuthenticationService' and a list of navigation links: Home, Counter, Weather, Auth Required, Register (highlighted), and Login. The main content area is white and titled 'Register' with the subtitle 'Create a new account.' It contains three input fields: 'Email' with the value 'jthompson@nait.ca', 'Password' with masked characters, and 'Confirm Password' also with masked characters. A blue 'Register' button is at the bottom right of the form.

2. Register Confirmation. Normally this would involve receiving an email, but we will verify the account by clicking the “confirm your account”

Register confirmation

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](#)



Confirm email

Thank you for confirming your email.

VERIFYING AUTHORIZATION

1. Click on “Auth Required”.
2. This will redirect you to the login screen.
3. Login

AuthenticationService

Home
Counter
Weather
Auth Required 1
Register
Login 2

Log in
Use a local account to log in.

Email 3
jthompson@nait.ca

Password
.....

☐ Remember me

Log in

[Forgot your password?](#)

4. You are now authenticated.

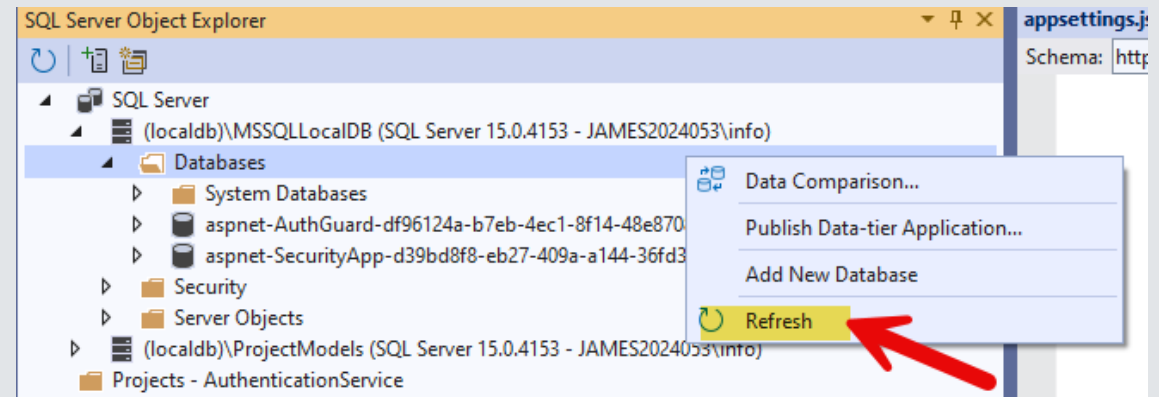
AuthenticationService

Home
Counter
Weather
Auth Required
jthompson@nait.ca
Logout

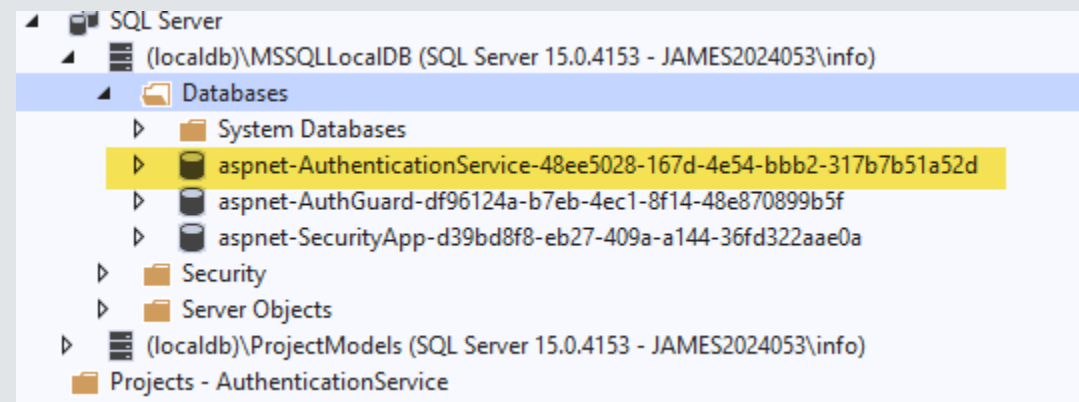
You are authenticated
Hello jthompson@nait.ca!

REFRESH THE DATABASE LIST

1. Close the browser and return to Visual Studio
2. Right-Click on the **Database** folder and click **Refresh**.

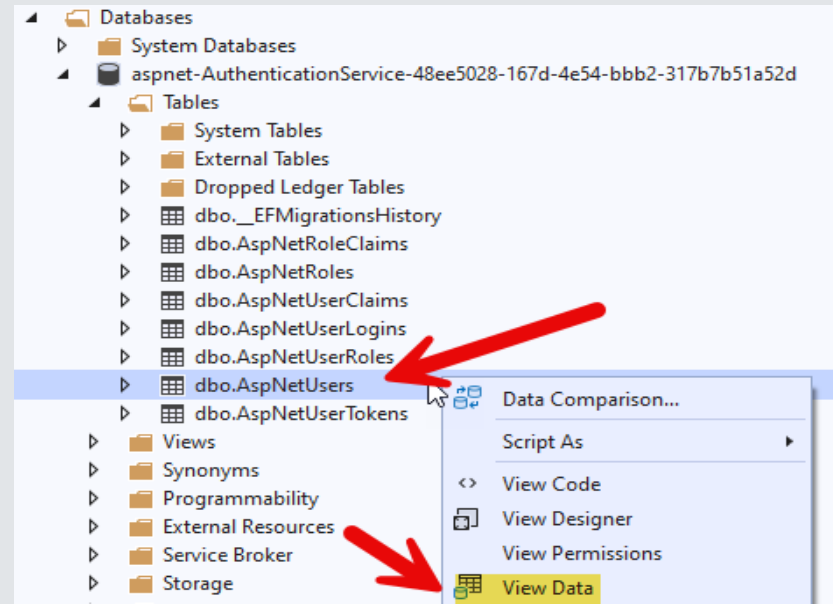


3. You will not see the database name that was listed in the **DefaultConnection** string



REVIEWING THE “ASP NET USERS” TABLE

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

[illegible]

“ASP NET USERS” FIELDS

1. **Id**: A primary key for the user, typically a GUID.
2. **UserName**: The username chosen by the user, used for login purposes.
3. **NormalizedUserName**: A normalized version of the UserName (usually uppercase) used for consistent lookup and comparison.
4. **Email**: The user's email address.
5. **NormalizedEmail**: A normalized version of the Email (usually uppercase) used for consistent lookup and comparison.
6. **EmailConfirmed**: A Boolean flag indicating whether the user's email address has been confirmed.
7. **PasswordHash**: The hash of the user's password. This is stored instead of the actual password for security reasons.
8. **SecurityStamp**: A random value that is used to indicate when a user's credentials have changed.
9. **ConcurrencyStamp**: A stamp used to handle concurrent updates to a user's record.
10. **PhoneNumber**: The user's phone number.
11. **PhoneNumberConfirmed**: A Boolean flag indicating whether the user's phone number has been confirmed.
12. **TwoFactorEnabled**: A Boolean indicating whether two-factor authentication is enabled for the user.
13. **LockoutEnd**: The date and time when the user's lockout ends (if they are currently locked out).
14. **LockoutEnabled**: A Boolean indicating whether the user can be locked out.
15. **AccessFailedCount**: The number of failed login attempts. This is used for implementing lockout functionality.
16. **TwoFactorEnabled**: Indicates whether the user has enabled two-factor authentication.

MANAGING ACCESS CONTROL IN BLAZOR

- In Blazor applications, both `<AuthorizeView>` component and the `[Authorize]` attribute are used to manage access control based on user authentication and authorization.
- `<AuthorizeView>` component is used when you want to control the visibility of certain parts of your UI based on the user's status.
- `[Authorize]` attribute when you need to secure an entire component or page from unauthorized access.

[AUTHORIZE] ATTRIBUTE

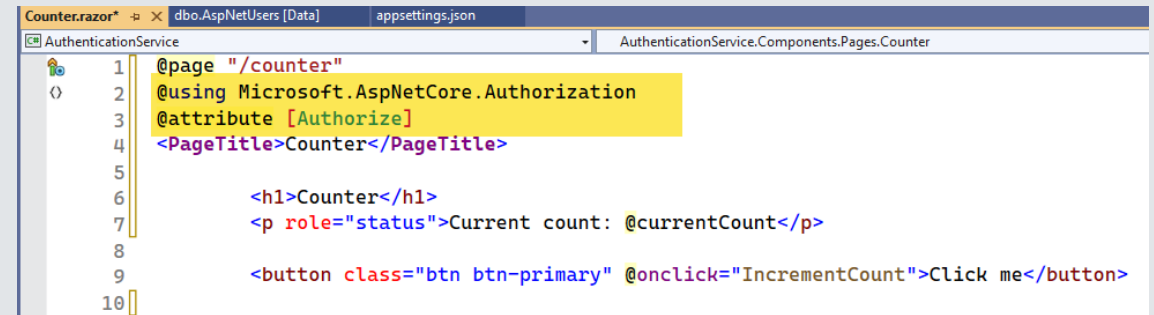
- Usage Context:** The [Authorize] attribute is used on Razor pages or server-side page handlers.

- Purpose:** It is used to enforce authorization policies before the server processes a page or component. If the user does not meet the authorization requirements set by the attribute, the system will not render the component or page and typically redirects the user to a login or access denied page.

- Control Level:** It provides a higher level of security control because it prevents the entire page or component from being processed if the user is not authorized.

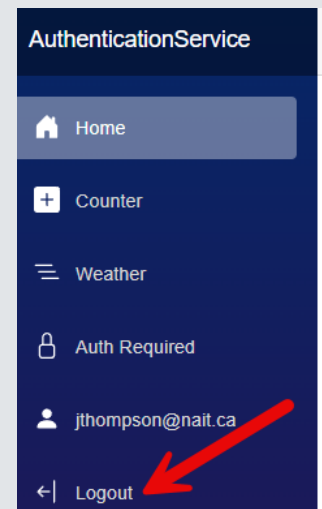
[AUTHORIZE] ATTRIBUTE EXAMPLE

1. Update the Counter.razor page and add the [Authorize] attribute below the @page attribute.
Note: You may have to added the “@using Microsoft.AspNetCore.Authorization”



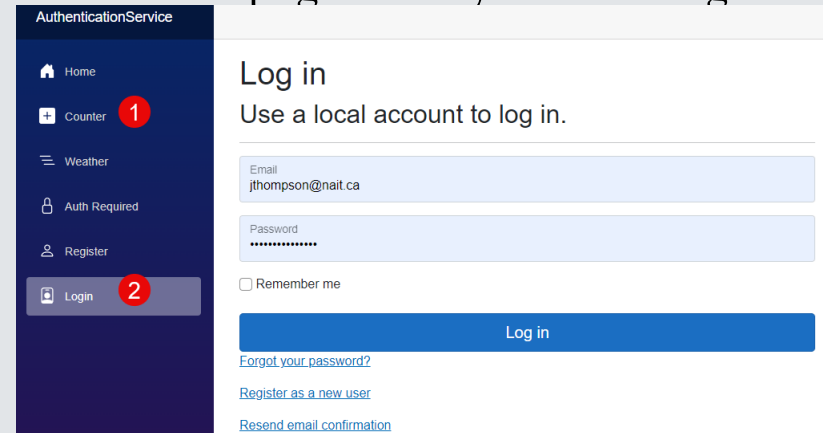
```
1 @page "/"counter"
2 @using Microsoft.AspNetCore.Authorization
3 @attribute [Authorize]
4 <PageTitle>Counter</PageTitle>
5
6 <h1>Counter</h1>
7 <p role="status">Current count: @currentCount</p>
8
9 <button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
10
```

2. Ensure that you are log out of the application.

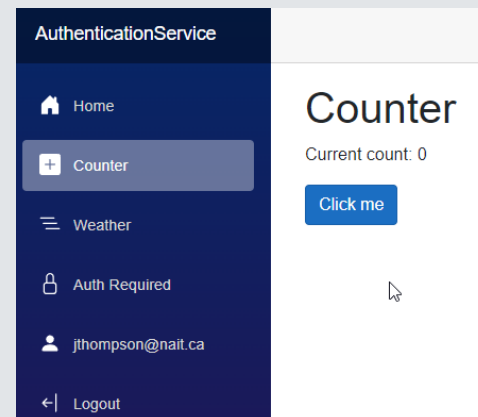


[AUTHORIZE] ATTRIBUTE EXAMPLE (CONTINUE)

1. Click on the “Counter” menu
2. The application will not allow you to access the Blazor page until you have log in.



3. You will now be redirected to the “Counter” page.



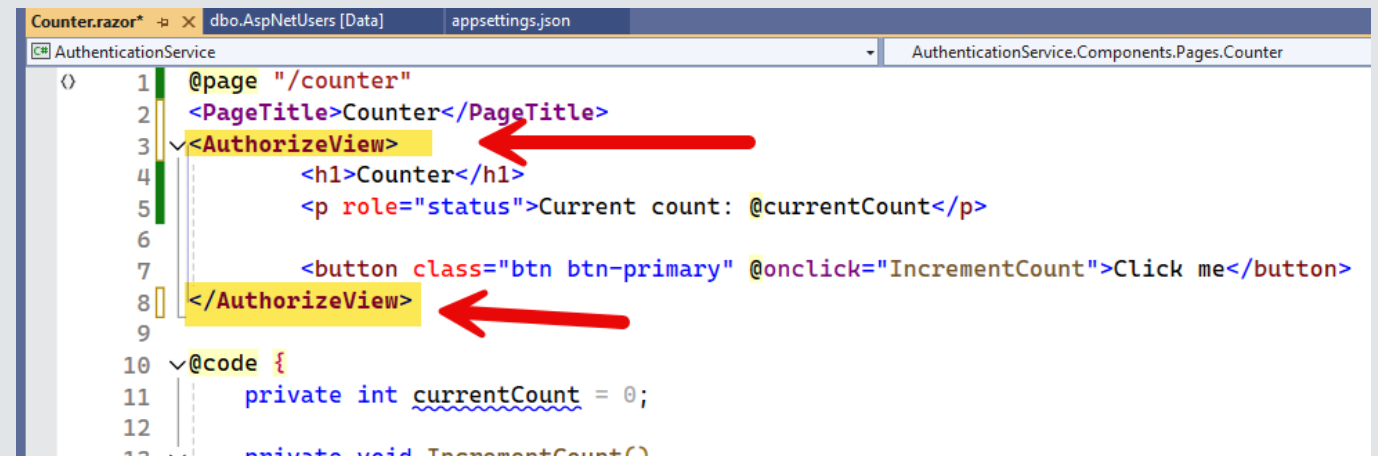
AUTHORIZE VIEW COMPONENT

- **Usage Context:** <AuthorizeView> component is used within Razor components and pages.
- **Purpose:** It allows conditional rendering of UI content based on the user's authentication and authorization status. This means you can show or hide certain UI elements based on whether the user is logged in or whether the user meets certain authorization criteria.
- **Flexibility:** It provides a flexible way to include different UI blocks for authenticated, unauthenticated, or unauthorized users within the same component. For example, you can show a login link to unauthenticated users and a logout link to authenticated users.

USING THE AUTHORIZE VIEW TAG

<AuthorizeView>: This is the main component used for authorization in Blazor. It allows you to display different content based on the user's authentication status.

1. Remove the [Authorize] reference from the “Counter” page.
2. Add the “<AuthorizeView>” tags to the Blazor Page surrounding “Counter” HTML code. NOTE: Do not included either the @page or the @code.



The screenshot shows a code editor with the file 'Counter.razor' open. The code is as follows:

```
1 @page "/counter"
2 <PageTitle>Counter</PageTitle>
3 <AuthorizeView>
4     <h1>Counter</h1>
5     <p role="status">Current count: @currentCount</p>
6
7     <button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
8 </AuthorizeView>
9
10 @code {
11     private int currentCount = 0;
12
13     private void IncrementCount()
```

Two red arrows point to the `<AuthorizeView>` and `</AuthorizeView>` tags, highlighting their placement around the main content of the page. The `@page` and `@code` blocks are not highlighted.

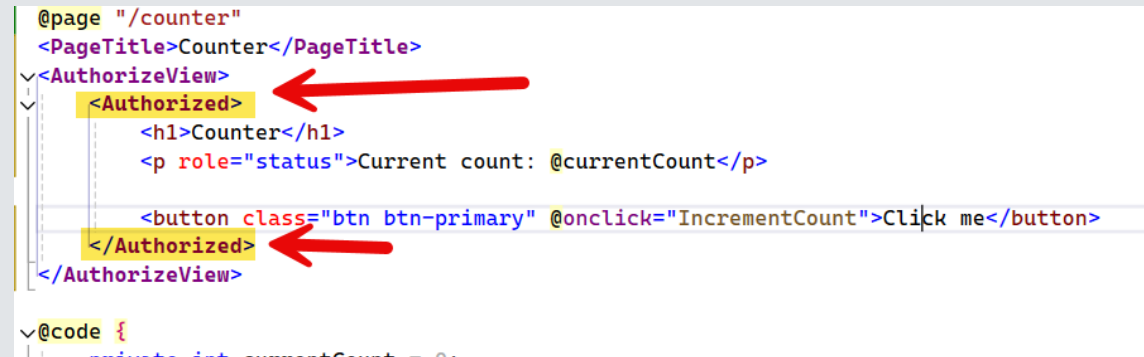
USING THE AUTHORIZED TAG

<Authorized>: This section contains content that will be displayed only to authenticated users.

1. Add the <Authorized> tags to the counter functionality.

```
@page "/counter"
<PageTitle>Counter</PageTitle>
<AuthorizeView>
  <Authorized>
    <h1>Counter</h1>
    <p role="status">Current count: @currentCount</p>
    <button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
  </Authorized>
</AuthorizeView>

@code {
  private int currentCount = 0;
```

A screenshot of a code editor showing the implementation of the <Authorized> tag in a Blazor component. The code is for a page at "/counter". It includes a <PageTitle>Counter</PageTitle>, an <AuthorizeView> tag, and inside it, an <Authorized> tag. The <Authorized> tag contains an <h1>Counter</h1>, a paragraph showing the current count, and a button to increment the count. Red arrows point to the opening and closing <Authorized> tags. Below the <Authorized> tag, the </AuthorizeView> tag is shown. At the bottom, the @code block is partially visible, showing a private integer variable currentCount initialized to 0.

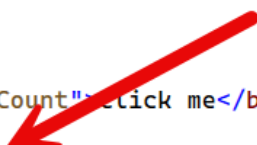
USING THE NOT AUTHORIZED TAG

<NotAuthorized>: This section contains content that will be displayed to users who are not authenticated..

1. Add the <NotAuthorized> tags to show a message indicating that the user is not authorized to view the content.

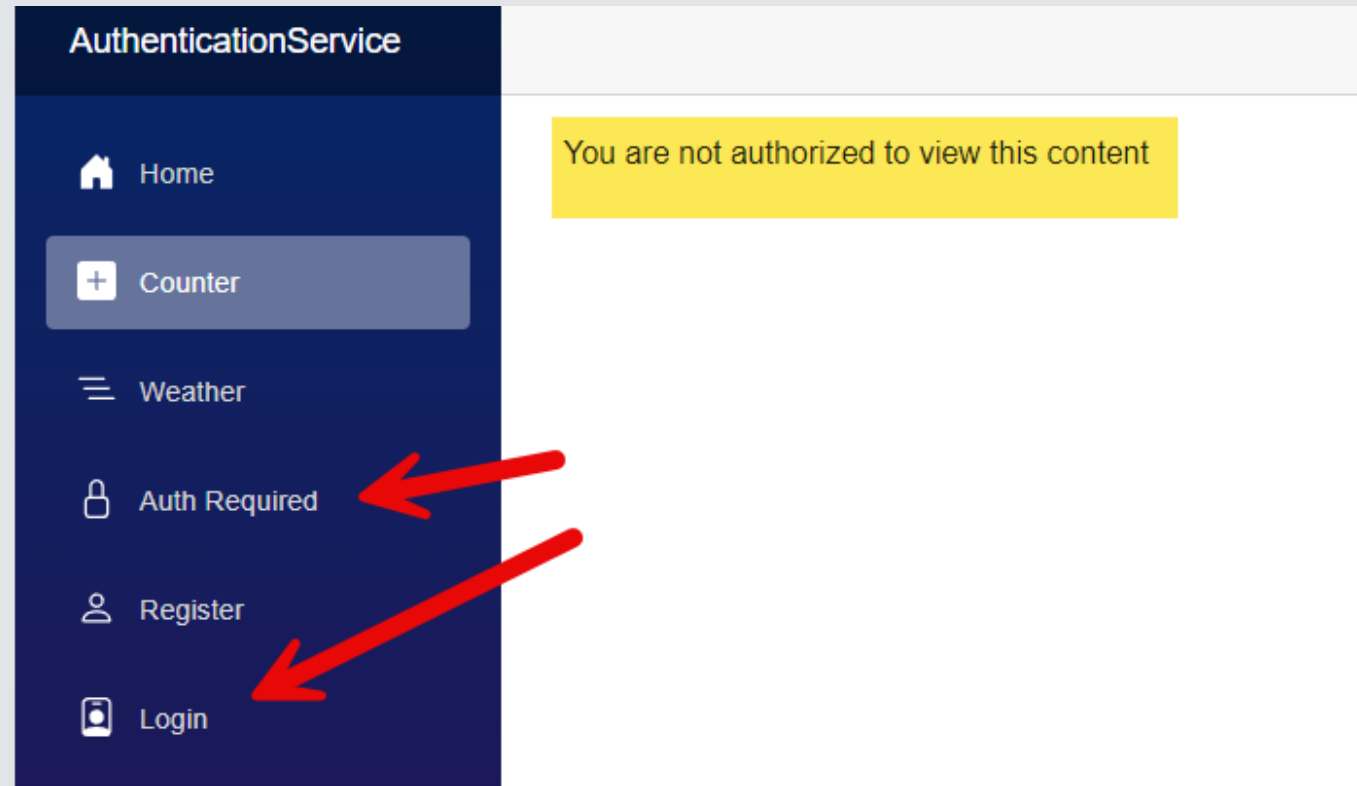
```
@page "/counter"
<PageTitle>Counter</PageTitle>
<AuthorizeView>
  <Authorized>
    <h1>Counter</h1>
    <p role="status">Current count: @currentCount</p>
    <button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
  </Authorized>
  <NotAuthorized>
    <p role="status">You are not authorized to view this content</p>
  </NotAuthorized>
</AuthorizeView>

@code {
    private int currentCount = 0;
```



NOTE: The “role=“status” indicates that the element contains advisory information for the user that is not important enough to justify an alert.

LOADING THE COUNTER PAGE



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

