Understand Basics Of Authentication In Blazor Server App

c-sharpcorner.com/article/understand-basic-of-authentication-in-blazor-server-app

Introduction

Authentication and authorization are the most common requirements of most applications. Authentication is a process of validating users and Authorization is a process of validating access right of users for accessing application resources. Blazor uses the ASP.NET core security model to provide authentication and authorization. Both Blazor server app and client app (WebAssembly) have different security scenarios as Blazor server app uses server resource to provide authorization, and Blazor client app (WebAssembly) runs on the client; hence authorization is only determined which UI option can be accessible by the user.

Authentication

Server-side Blazor uses ASP.NET Core authentication mechanisms. The Server-side Blazor uses SignalR for real-time connection between the server and UI. So, SignalR handles the authentication on established connection. There is an option available to enable authentication for the Blazor app when you create the application.

To enable authentication for Blazor server app, click on “Change” under “Authentication section and select “Individual User Accounts” option and then click on “Ok” button when you create a new Blazor server app.

ASP.NET Core Identity membership provides user interface for login and registration functionality and Blazor server app uses ASP.NET Core authentication mechanisms, hence these user interfaces are also available with Blazor server app. The identity membership also provides external login validation such Google, Microsoft, Facebook etc. Here, login credential is validated against the third-party authenticator.

When you run the Blazor server app, you can find Register and Login option at top right corner. The identity server use SQL server database by default. The Visual studio template generates a default connection string "appsettings.json" file.

When you click on "Register" link, it opens the default registration page UI. The registration and login page provided by default with identity membership middleware. They are not Blazor components but razor components. Using the following code in ConfigureServices method, the default pages are injected.

```
public void ConfigureServices(IServiceCollection services)
{
    ...
    services.AddDefaultIdentity<IdentityUser>()
    ...
}
```

You can create custom pages (or components) for registration and login and inject into the identity middleware.

When you access membership pages for the first time and the database is not setup, the system asks for migration. The migration generates various tables and stored procedures related to user and role management.

AuthenticationStateProvider service

Blazor server app provides AuthenticationStateProvider service that helps us to find-out user’s authentication state data from HttpContext.User. This service is used by CascadingAuthenticationState and AuthorizeView component to get the authentication states. You can use this service in your code, but it is not recommended as it does not notify automatically when authentication state data gets changed.

In the following example code, AuthenticationStateProvider retrieves the user from ASP.net core 's HttpContext.User and check authentication state of user. If the user is authenticated, it fetches user claims:

```
@page "/"
@using Microsoft.AspNetCore.Components.Authorization
@inject AuthenticationStateProvider AuthenticationStateProvider
<h1>Hello, world!</h1>
Welcome to your new app.
<p>
   @userAuthenticated
</p>
@code{
   string userAuthenticated;
   protected override async Task OnInitializedAsync()
   {
      var authState = await AuthenticationStateProvider.GetAuthenticationStateAsync();
      var user = authState.User;
      if (user.Identity.IsAuthenticated)
      {
         userAuthenticated = $"{ user.Identity.Name} is authenticated.";
      }
      else
      {
         userAuthenticated = "The user is NOT authenticated.";
      }
   }
}
...
...
```

Using user.Identity.IsAuthenticated property, you can verify whether user is authenticated (if it is set to true) or not. The Blazor server app runs on the server hence it is able to fetch current user information from the server but it is not possible with Blazor client (WebAssembly) app.

Custom AuthenticationStateProvider

The AuthenticationStateProvider service is a built-in service in the Blazor server app that helps you to obtain the authentication state data from HttpContext.User. You can also create custom AuthenticationStateProvider. There is a rare scenario where you need to create custom AuthenticationStateProvider. Also, it is not recommended as it added security vulnerabilities to your app.

```
using Microsoft.AspNetCore.Components.Authorization;
using System.Security.Claims;
using System.Threading.Tasks;
namespace AuthenticationBlazorApp.Provider
{
    public class CustomAuthenticationStateProvider : AuthenticationStateProvider
    {
        public override Task<AuthenticationState> GetAuthenticationStateAsync()
        {
            var identity = new ClaimsIdentity(new[]
            {
            new Claim(ClaimTypes.Name, "testUser"),
            }, "TestUser authentication type");
            var user = new ClaimsPrincipal(identity);
            return Task.FromResult(new AuthenticationState(user));
        }
    }
}
```

There are two easy steps to Implement a custom AuthenticationStateProvider.

Step 1

Create class which inherits from AuthenticationStateProvider abstract class and implements GetAuthenticationStateAsync abstract method

Step 2

Register the custom authentication state provider in ConfigureServices method of startup class

```
public void ConfigureServices(IServiceCollection services)
{
    ...
    ...
    services.AddScoped<AuthenticationStateProvider, CustomAuthenticationStateProvider>();
    services.AddRazorPages();
    ...
    ...
}
```

In the above code, custom authentication state providers authenticate all users using "testUser" users.

You can use NotifyAuthenticationStateChanged method of AuthenticationStateProvider base class to notify all the consumers of authentication state such as AuthorizeView component.

AuthorizeView Component

The AuthorizeView is Blazor's built-in component that is able to show page content based on user 's authentication state. This component also supports policy-based authorization and role-based authorization. This component is very useful when you want to show page content based on the role, policy, or authentication status of the user. It uses AuthenticationStateProvider to know the user authentication state.

This component provides Authorized, and NotAuthorized render fragments. The Authorized fragment renders when the user is authenticated, and NotAuthorized fragment renders when the user is unauthenticated. Both fragments accept other interactive components.

```
<AuthorizeView>
    <Authorized>
        <h4>Hello, @context.User.Identity.Name!</h4>
        <p>This content is only visible if user is authenticated.</p>
    </Authorized>
    <NotAuthorized>
        <p>Please signed in.</p>
    </NotAuthorized>
</AuthorizeView>
```

The AuthorizeView provides Authorizing render fragment that is used to display content when authentication is in progress.

Summary

The Blazor server app uses ASP.NET Core authentication mechanisms. Using AuthenticationStateProvider service, you can find user 's authentication state. This service is used by CascadingAuthenticationState and AuthorizeView Blazor component. You can use AuthorizeView component to render page content based on user 's authentication state.

Next Recommended Article Localization In Blazor App Using Microsoft.JSInterop