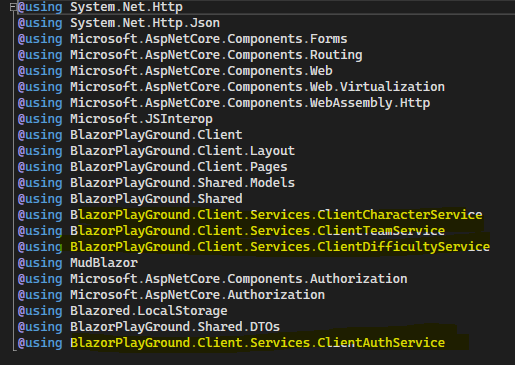
**SERVICE**

- Class (inherits from Interface)

- Interface

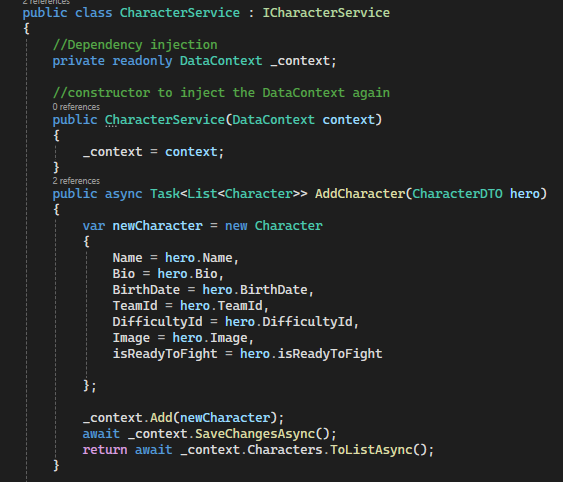
- Needs to be registered in the program.cs of Client and Server

- Import in Client side



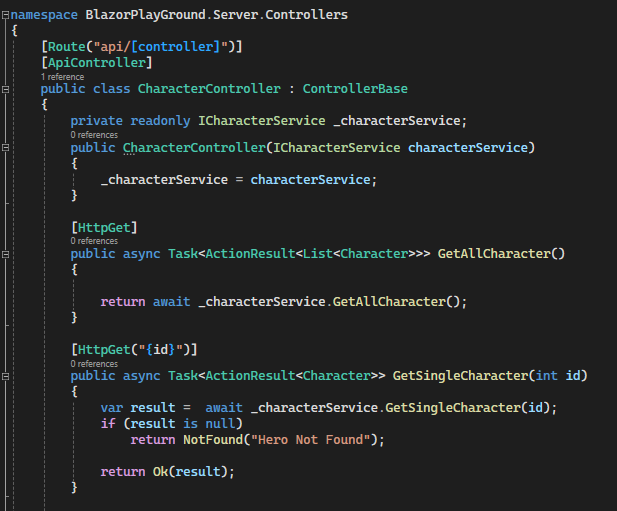
The Server service contains the main logic of the system. (Accessing the database)

- Dependency Inject the DataContext to access the database



Controller or API - forwarding the request to Server Service. (Forwarding the request by end point)

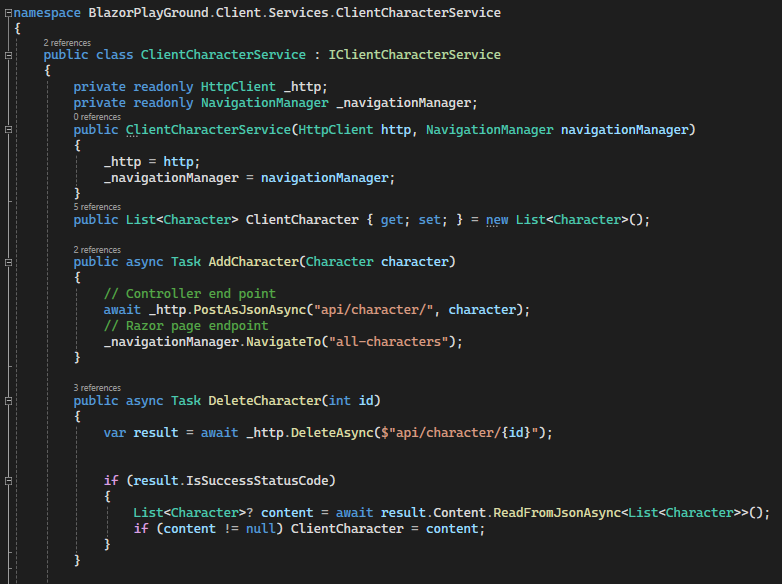
- Inject the service by using Dependency Injection to access the logic from the server service.



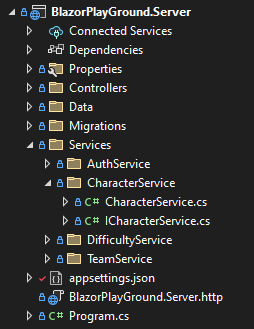
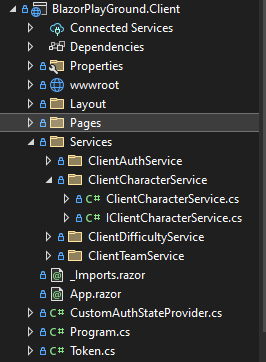
The Client service - sending request by using API that can request to Server service.

- Inject HttpClient for sending HTTP request and receiving HTTP responses from a source.

- Inject NavigationManager for navigation



**Server Client**

** **

**ENTITY FRAMEWORK (DATABASE)**

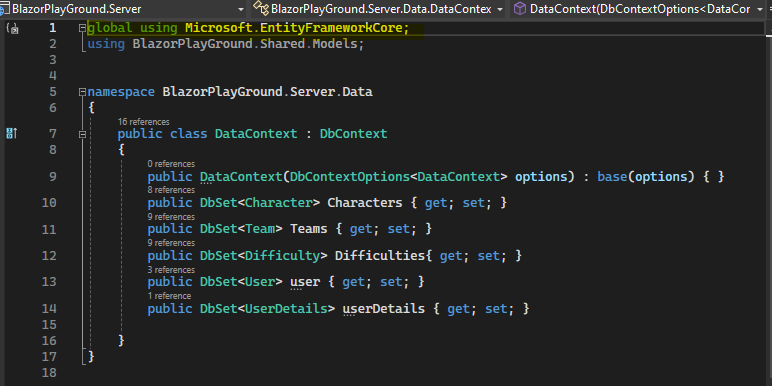
Nuget Packages (install)

- Microsoft.EntityFrameworkCore

- Microsoft.EntityFrameworkCore.Design

- Microsoft.EntityFrameworkCore.SqlServer

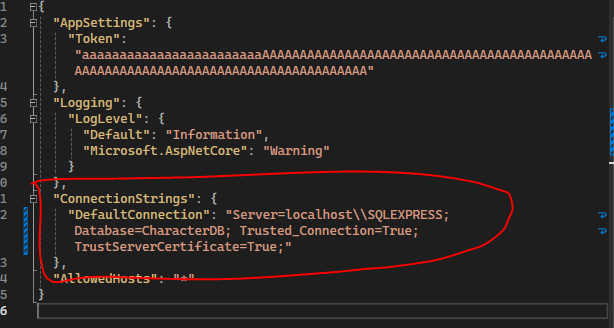
Entities (Models)



AppSettings: Modify the “DefaultConnection” for security purposes

"ConnectionStrings": {

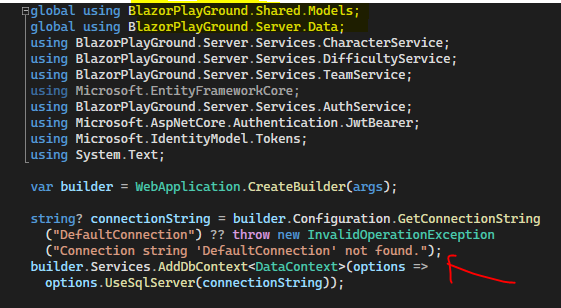
"DefaultConnection": "Server=localhost\\SQLEXPRESS; Database=CharacterDB; Trusted\_Connection=True; TrustServerCertificate=True;"

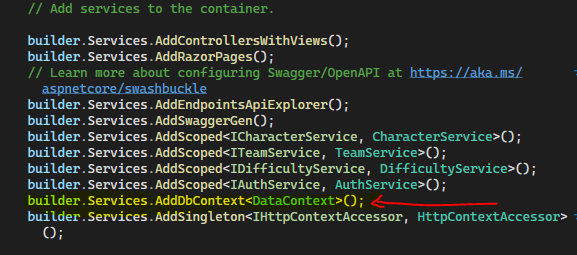


Register Db Context

string? connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new InvalidOperationException("Connection string 'DefaultConnection' not found.");

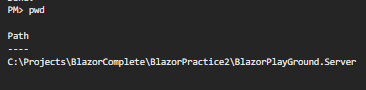
builder.Services.AddDbContext<DataContext>(options => options.UseSqlServer(connectionString));





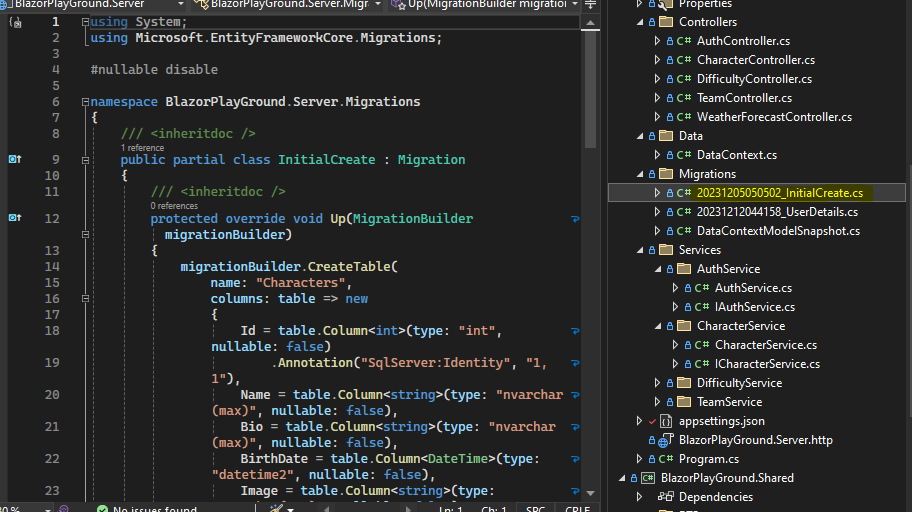
Create Migration using Package Manager Console

- first navigate to Server





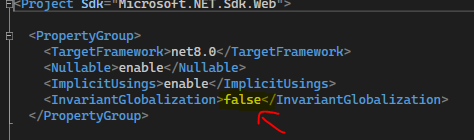
After migrating, check the Migrations file content



After checking, run the command “ dotnet ef database update “

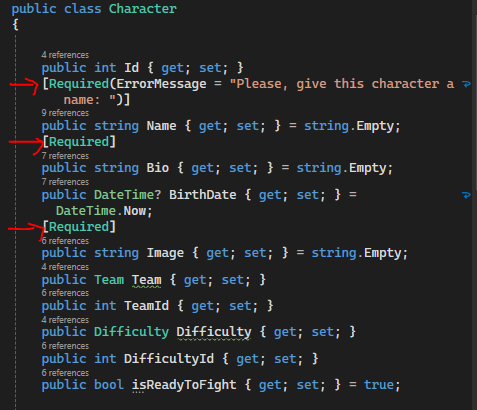


Modify this from server. (make it false)



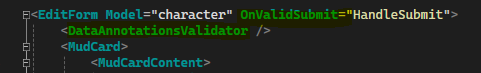
**EDITFORM with ALL Components & Validation**

Validation in Model

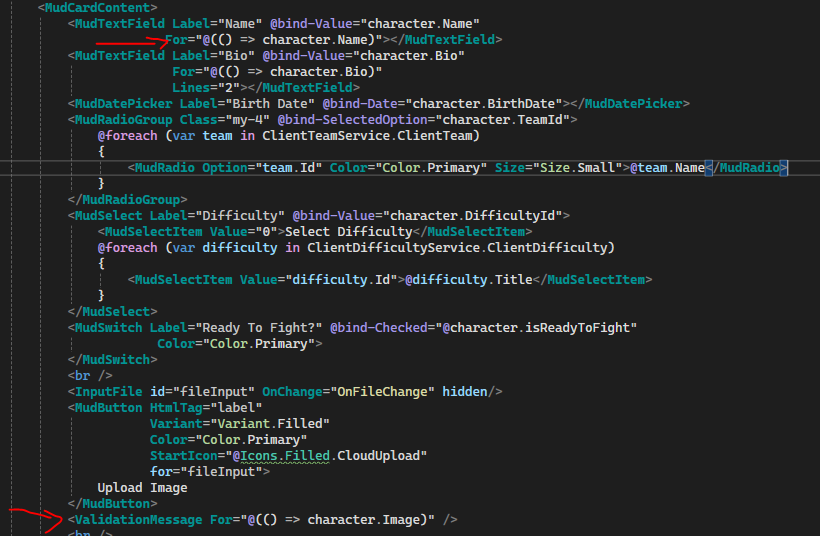


Insert DataAnnotationValidator inside the EditForm.

Make sure it’s using **OnvalidSubmit**

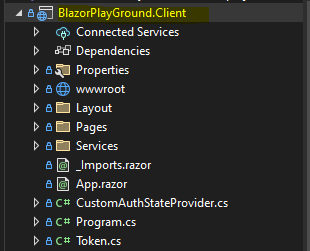


If the component already has the **For** method, it does not need a **ValidationMessage For** anymore.

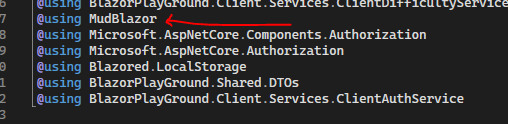


**MudBlazor**

Install MudBlazor in Client side



Imports

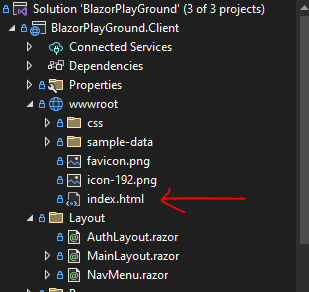


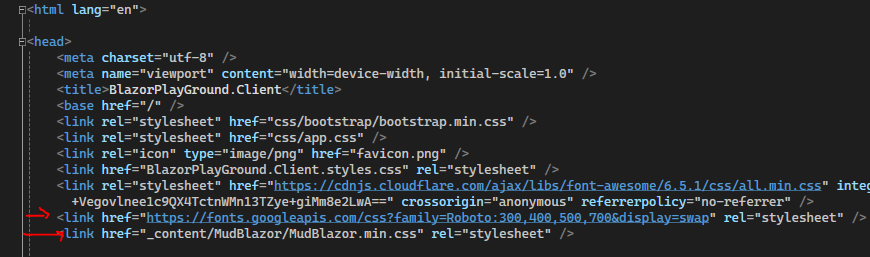
Add t o index.html

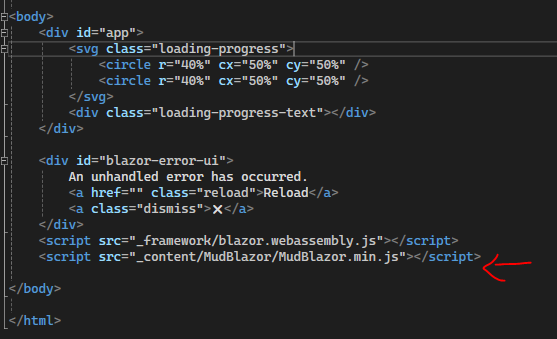
<link href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700&display=swap" rel="stylesheet" />

<link href="\_content/MudBlazor/MudBlazor.min.css" rel="stylesheet" />

<script src="\_content/MudBlazor/MudBlazor.min.js"></script>







Register to program.cs

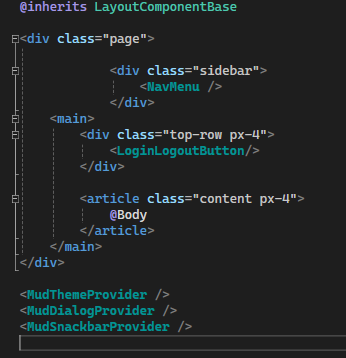


Add Components on MainLayout.razor

<**MudThemeProvider** />

<**MudDialogProvider** />

<**MudSnackbarProvider** />



**Authentication and Authorization**

Client

- Install Microsoft.AspNetCore.Components.Authorization

- Install Blazored.LocalStorage

Add to program.cs

global using Microsoft.AspNetCore.Components.Authorization;

builder.Services.AddScoped<AuthenticationStateProvider, CustomAuthStateProvider>();

builder.Services.AddAuthorizationCore();

builder.Services.AddBlazoredLocalStorage();+

global using Blazored.LocalStorage;

Imports.razor

@using Microsoft.AspNetCore.Components.Authorization

@using Blazored.LocalStorage

Inject in pages

@inject HttpClient Http

@inject AuthenticationStateProvider AuthStateProvider

@inject ILocalStorageService LocalStorage

@inject IClientCharacterService ClientCharacterService

@inject IClientTeamService ClientTeamService

@inject IClientDifficultyService ClientDifficultyService

@inject IClientAuthService ClientAuthService

@inject NavigationManager NavigationManager

@layout AuthLayout

@inject ISnackbar Snackbar

App.razor

@inject AuthenticationStateProvider AuthStateProvider

@inject IClientAuthService ClientAuthService

@inject NavigationManager NavigationManager

<**CascadingAuthenticationState**>

<**Router** **AppAssembly**="@typeof(App).Assembly">

<**Found** **Context**="routeData">

<**AuthorizeRouteView** **RouteData**="@routeData" **DefaultLayout**="@typeof(MainLayout)">

<**NotAuthorized**>

Sorry, but you are not allowed to see this.

</**NotAuthorized**>

</**AuthorizeRouteView**>

<**FocusOnNavigate** **RouteData**="@routeData" **Selector**="h1" />

</**Found**>

<**NotFound**>

<**PageTitle**>Not found</**PageTitle**>

<**LayoutView** **Layout**="@typeof(MainLayout)">

<p role="alert">Sorry, there's nothing at this address.</p>

</**LayoutView**>

</**NotFound**>

</**Router**>

</**CascadingAuthenticationState**>

@code{

protected override async Task OnInitializedAsync()

{

//LOGOUT IF TOKEN EXPIRE OR INVALID

if (string.IsNullOrEmpty(ClientAuthService.token.value))

{

AuthenticationState token\_auth\_state = await AuthStateProvider.GetAuthenticationStateAsync();

if (token\_auth\_state.User.Identity != null)

{

if (token\_auth\_state.User.Identity.IsAuthenticated != true)

{

NavigationManager.NavigateTo("login");

}

}

else

{

NavigationManager.NavigateTo("login");

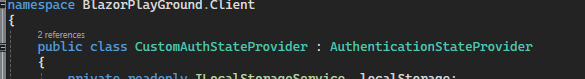
}

}

}

}

Create a CustomAuthStateProvider class in Client that inherits from AuthenticationStateProvider



private readonly ILocalStorageService \_localStorage;

private readonly HttpClient \_http;

public CustomAuthStateProvider(ILocalStorageService localStorage, HttpClient http)

{

\_localStorage = localStorage;

\_http = http;

}

public override async Task<AuthenticationState> GetAuthenticationStateAsync()

{

string token = await \_localStorage.GetItemAsStringAsync("token");

// no token

var identity = new ClaimsIdentity();

\_http.DefaultRequestHeaders.Authorization = null;

// has token

if (!string.IsNullOrEmpty(token))

{

identity = new ClaimsIdentity(ParseClaimsFromJwt(token), "jwt");

\_http.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token.Replace("\"", ""));

}

var user = new ClaimsPrincipal(identity);

var state = new AuthenticationState(user);

NotifyAuthenticationStateChanged(Task.FromResult(state));

return state;

}

public static IEnumerable<Claim> ParseClaimsFromJwt(string jwt)

{

var payload = jwt.Split('.')[1];

var jsonBytes = ParseBase64WithoutPadding(payload);

var keyValuePairs = JsonSerializer.Deserialize<Dictionary<string, object>>(jsonBytes);

return keyValuePairs.Select(kvp => new Claim(kvp.Key, kvp.Value.ToString()));

}

private static byte[] ParseBase64WithoutPadding(string base64)

{

switch (base64.Length % 4)

{

case 2: base64 += "=="; break;

case 3: base64 += "="; break;

}

return Convert.FromBase64String(base64);

}

LoginLogoutButton.Razor

@inject NavigationManager NavigationManager

@inject ILocalStorageService LocalStorage

@inject AuthenticationStateProvider AuthStateProvider

@inject ISnackbar Snackbar

<**MudButton** **OnClick**="Logout"

**Class**="mt-4"

**Color**="Color.Error"

**Variant**="Variant.Filled">Logout</**MudButton**>

@code {

void Login()

{

NavigationManager.NavigateTo("login");

}

async Task Logout()

{

await LocalStorage.RemoveItemAsync("token");

await AuthStateProvider.GetAuthenticationStateAsync();

NavigationManager.NavigateTo("login");

Snackbar.Add(

"Logout Successfully!",

Severity.Success,

config =>

{

config.ShowTransitionDuration = 200;

config.HideTransitionDuration = 400;

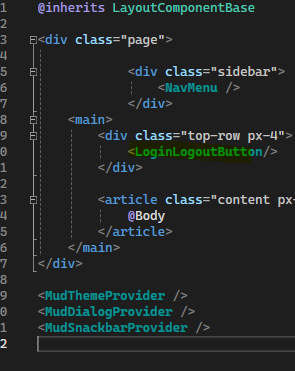
config.VisibleStateDuration = 2500;

});

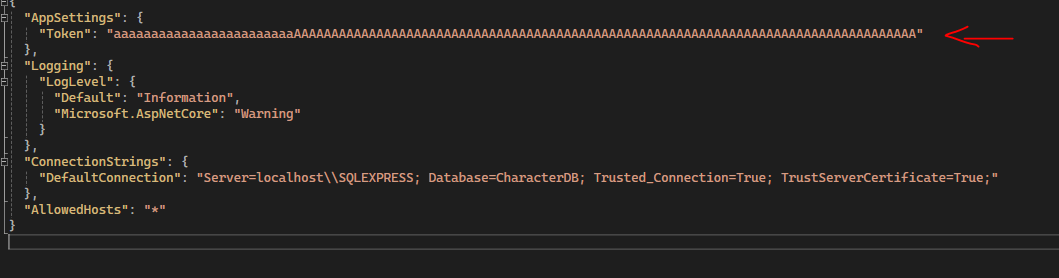
}

}

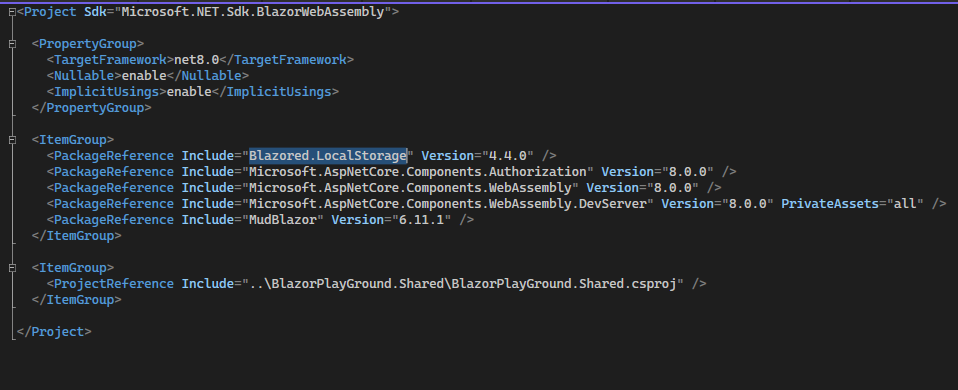
Add it to MainLayout



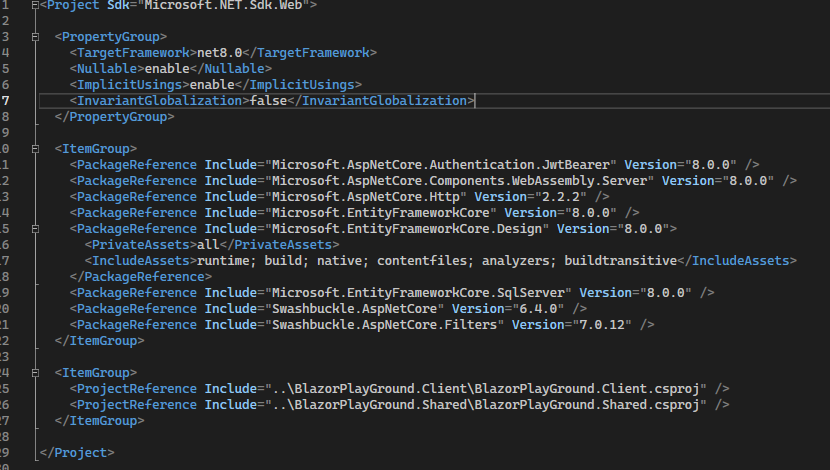
Add to AppSettings



CLIENT



SERVER



Register Microsoft.AspNetCore.Authentication.JwtBearer

Program.Cs

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuerSigningKey = true,

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8

.GetBytes(builder.Configuration.GetSection("AppSettings:Token").Value)),

ValidateIssuer = false,

ValidateAudience = false

};

})

;