//Install

Install-Package Microsoft.AspNetCore.Authentication.JwtBearer

//appSetting

"Jwt": {

"Key": "your-very-long-and-random-secret-key-that-is-at-least-32-bytes-long\",",

"Issuer": "your\_issuer\_here",

"Audience": "your\_audience\_here"

}

// program.cs

// Note 1 Before var app = builder.Build();

// Register TokenGenerator as a singleton or transient service

builder.Services.AddSingleton<TokenGenerator>(); // or .AddTransient<TokenGenerator>()

// Retrieve JWT settings from configuration

var jwtSettings = builder.Configuration.GetSection("Jwt");

var key = jwtSettings.GetValue<string>("Key");

var issuer = jwtSettings.GetValue<string>("Issuer");

var audience = jwtSettings.GetValue<string>("Audience");

// Ensure values are not null

if (string.IsNullOrEmpty(key) || string.IsNullOrEmpty(issuer) || string.IsNullOrEmpty(audience))

{

throw new InvalidOperationException("JWT settings are not properly configured.");

}

// Add JWT Authentication

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options =>

{

var jwtSettings = builder.Configuration.GetSection("Jwt");

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = jwtSettings["Issuer"],

ValidAudience = jwtSettings["Audience"],

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSettings["Key"]))

};

});

builder.Services.AddAuthorization(options =>

{

options.AddPolicy("Admin", policy => policy.RequireRole("Admin"));

});

// Configure the HTTP request pipeline.

app.UseAuthentication();

app.UseAuthorization();

// Implement the TokenGenerator class

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

public class TokenGenerator

{

private readonly IConfiguration \_configuration;

public TokenGenerator(IConfiguration configuration)

{

\_configuration = configuration;

}

public string GenerateToken(string username, IList<string> roles)

{

var jwtSettings = \_configuration.GetSection("Jwt");

var key = jwtSettings.GetValue<string>("Key");

var claims = new List<Claim>

{

new Claim(ClaimTypes.Name, username)

};

claims.AddRange(roles.Select(role => new Claim(ClaimTypes.Role, role)));

var keyBytes = Encoding.UTF8.GetBytes(key);

var signingKey = new SymmetricSecurityKey(keyBytes);

var creds = new SigningCredentials(signingKey, SecurityAlgorithms.HmacSha256);

var token = new JwtSecurityToken(

issuer: jwtSettings.GetValue<string>("Issuer"),

audience: jwtSettings.GetValue<string>("Audience"),

claims: claims,

expires: DateTime.Now.AddMinutes(30),

signingCredentials: creds);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

// USer Controller add

public UserController(MyDbContext context, TokenGenerator tokenGenerator)

{

\_context = context;

\_tokenGenerator = tokenGenerator;

}

and in the user login action add

var token = \_tokenGenerator.GenerateToken(user.Username, roles);

return Ok(new { Token = token });

in VS Code

when login add this localStorage.setItem('jwtToken', result.token);

then when i fetch api that Authentication add this in the header

var response = await fetch(urlSpecific,{

headers: {

'Authorization': `Bearer ${token}`

}

Js:

1. var roles = \_context.UserRoles.Where(x => x.UserId == user.UserId).Select(ur => ur.Role).ToList();

var token = localStorage.getItem('jwtToken') // var token =localStorage.getItem('jwtToken') var response = await fetch(url, { method: "GET", headers: { 'Authorization': Bearer ${token} }, });