Simple JWT Authentication Setup in ASP.NET Core

This guide explains how to set up a minimal JWT-based authentication mechanism in an ASP.NET Core Web API application.   
We will cover the necessary package installations, configuration files, and code implementation.

# 1. Install Required NuGet Packages

Run the following commands in the server project directory to install the necessary packages:

dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer --version 6.0.25  
dotnet add package System.IdentityModel.Tokens.Jwt

# 2. Configuration in appsettings.json

Update or create the appsettings.json file with the following content:

{  
 "Jwt": {  
 "Key": "your\_very\_long\_secret\_key\_here\_which\_is\_secure",  
 "Issuer": "your-app-name-or-domain",  
 "Audience": "your-app-users"  
 },  
 "Logging": {  
 "LogLevel": {  
 "Default": "Information",  
 "Microsoft.AspNetCore": "Warning"  
 }  
 },  
 "AllowedHosts": "\*"  
}

# 3. Modify Program.cs

Ensure your Program.cs configures JWT authentication correctly before building the app:

using Microsoft.AspNetCore.Authentication.JwtBearer;  
using Microsoft.IdentityModel.Tokens;  
using System.Text;  
  
var builder = WebApplication.CreateBuilder(args);  
  
var jwtKey = builder.Configuration["Jwt:Key"];  
var jwtIssuer = builder.Configuration["Jwt:Issuer"];  
var jwtAudience = builder.Configuration["Jwt:Audience"];  
  
// Add services to the container.  
  
builder.Services.AddControllers();  
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle  
builder.Services.AddEndpointsApiExplorer();  
builder.Services.AddSwaggerGen();  
  
builder.Services.AddSignalR(); // Adds SignalR to the app's service container  
  
  
builder.Services.AddAuthentication(options =>  
{  
 options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;  
 options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;  
})  
.AddJwtBearer(options =>  
{  
 options.TokenValidationParameters = new TokenValidationParameters  
 {  
 ValidateIssuer = true,  
 ValidateAudience = true,  
 ValidateLifetime = true,  
 ValidateIssuerSigningKey = true,  
 ValidIssuer = jwtIssuer,  
 ValidAudience = jwtAudience,  
 IssuerSigningKey = new SymmetricSecurityKey(Encoding.ASCII.GetBytes(jwtKey)),  
 };  
  
 // Allow JWT via query string for SignalR  
 options.Events = new JwtBearerEvents  
 {  
 OnMessageReceived = context =>  
 {  
 var accessToken = context.Request.Query["access\_token"];  
 var path = context.HttpContext.Request.Path;  
 if (!string.IsNullOrEmpty(accessToken) && path.StartsWithSegments("/hub/cache"))  
 {  
 context.Token = accessToken;  
 }  
 return Task.CompletedTask;  
 }  
 };  
});  
  
var app = builder.Build();  
  
// Configure the HTTP request pipeline.  
if (app.Environment.IsDevelopment())  
{  
 app.UseSwagger();  
 app.UseSwaggerUI();  
}  
  
app.UseHttpsRedirection();  
  
app.UseAuthentication();  
  
  
app.UseAuthorization();  
  
app.MapControllers();  
  
app.MapHub<CacheHub>("/hub/cache"); // Maps the CacheHub to the route "/hub/cache"  
  
app.Run("http://localhost:5000"); // Starts the server and listens only on HTTP at port 5000 (disables HTTPS) http://localhost:5000/hub/cache

# 4. Create LoginModel.cs

Add the following model in the 'Models' folder:

namespace server.Models;  
  
public class LoginModel  
{  
 public string? Username { get; set; }  
 public string? Password { get; set; }  
}

# 5. Implement AuthController.cs

Add the following controller to the Controllers folder. It handles authentication and token generation:

using Microsoft.AspNetCore.Mvc;  
using Microsoft.IdentityModel.Tokens;  
using System.IdentityModel.Tokens.Jwt;  
using System.Security.Claims;  
using System.Text;  
using server.Models;  
  
namespace server.Controllers  
{  
 [Route("api/[controller]")]  
 [ApiController]  
 public class AuthController : ControllerBase  
 {  
 private readonly IConfiguration \_configuration;  
  
 public AuthController(IConfiguration configuration)  
 {  
 \_configuration = configuration;  
 }  
  
 [HttpPost("authenticate")]  
 public IActionResult Authenticate([FromBody] LoginModel model)  
 {  
 if (model.Username != "user" || model.Password != "pass")  
 {  
 return Unauthorized(new { success = false, error = "Invalid credentials" });  
 }  
  
 var token = GenerateJwtToken(model.Username);  
 return Ok(new { token });  
 }  
  
 private string GenerateJwtToken(string username)  
 {  
 var key = Encoding.UTF8.GetBytes(\_configuration["Jwt:Key"] ?? throw new InvalidOperationException("JWT Key is not configured."));  
 var tokenHandler = new JwtSecurityTokenHandler();  
  
 var tokenDescriptor = new SecurityTokenDescriptor  
 {  
 Subject = new ClaimsIdentity(new[] {  
 new Claim("cid", Guid.NewGuid().ToString())  
 }),  
 Expires = DateTime.UtcNow.AddHours(1),  
 SigningCredentials = new SigningCredentials(  
 new SymmetricSecurityKey(key),  
 SecurityAlgorithms.HmacSha256Signature  
 ),  
 Issuer = \_configuration["Jwt:Issuer"]  
 };  
  
 var token = tokenHandler.CreateToken(tokenDescriptor);  
 return tokenHandler.WriteToken(token);  
 }  
 }  
}

# Notes

- The JWT key must be at least 256 bits long. Use a secure and randomly generated key.  
- The Issuer and Audience should match the values expected by the consumer of the JWT.  
- Place the LoginModel in a separate Models folder for structure.  
- The AuthController can go into your existing Controllers folder.