JWT Authentication Setup for SignalR Server (C#)

# 📦 Purpose

This guide explains how to implement basic JWT authentication in your SignalR server. It ensures that each client request is verified via token-based authentication.

# 📚 Prerequisites

- .NET 6 SDK  
- Visual Studio Code or Visual Studio  
- Basic knowledge of C#  
- A working SignalR server project

# 🧱 Required NuGet Package

Run this in your terminal to install the required authentication package:

dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer --version 6.0.25

dotnet add package System.IdentityModel.Tokens.Jwt

# 🗂️ File Structure and Responsibilities

## 1. Program.cs

This file sets up the application configuration.  
- Adds authentication middleware with JWT Bearer.  
- Sets validation parameters like issuer, audience, signing key.  
- Enables token support for SignalR via query strings.  
- Adds `app.UseAuthentication()` before `app.UseAuthorization()` to ensure requests are authenticated.

Here is the full content of your `Program.cs` file:

using Microsoft.AspNetCore.Authentication.JwtBearer;  
using Microsoft.IdentityModel.Tokens;  
using System.Text;  
  
var builder = WebApplication.CreateBuilder(args);  
  
// 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  
  
var app = builder.Build();  
  
// Configure the HTTP request pipeline.  
if (app.Environment.IsDevelopment())  
{  
 app.UseSwagger();  
 app.UseSwaggerUI();  
}  
  
app.UseHttpsRedirection();  
  
  
builder.Services.AddAuthentication(options =>  
{  
 options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;  
 options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;  
})  
.AddJwtBearer(options =>  
{  
 options.TokenValidationParameters = new TokenValidationParameters  
 {  
 ValidateIssuer = false,  
 ValidateAudience = false,  
 ValidateLifetime = true,  
 ValidateIssuerSigningKey = true,  
 IssuerSigningKey = new SymmetricSecurityKey(Encoding.ASCII.GetBytes("supersecretkey1234567890")),  
 };  
  
 // 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;  
 }  
 };  
});  
  
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

## 2. LoginModel.cs

This is a simple model representing the login credentials. It has two properties: `Username` and `Password`. It's used to deserialize the incoming JSON body from the client.

Here is the content of `LoginModel.cs`:

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

## 3. AuthController.cs

This controller handles the authentication logic.  
- Accepts login credentials.  
- If valid, generates a JWT token containing client ID, issuer, and timestamps.  
- Returns the token to the client.  
  
The token must be sent as a query parameter `access\_token` when connecting to the SignalR hub.

Here is the content of `AuthController.cs`:

using Microsoft.AspNetCore.Mvc;  
using server.Models;  
  
using System.IdentityModel.Tokens.Jwt;  
using System.Security.Claims;  
using Microsoft.IdentityModel.Tokens;  
using System.Text;  
  
namespace server.Controllers;  
  
[ApiController]  
[Route("api/[controller]")]  
public class AuthController : ControllerBase  
{  
 [HttpPost("authenticate")]  
 public IActionResult Authenticate([FromBody] LoginModel model)  
 {  
 // Simple hardcoded check  
 if (model.Username == "user" && model.Password == "pass")  
 {  
 var token = GenerateJwtToken(model.Username);  
 return Ok(new { success = true, data = new { token } });  
 }  
  
 return Unauthorized(new { success = false, error = "Invalid credentials" });  
 }  
  
 private string GenerateJwtToken(string username)  
 {  
 var tokenHandler = new JwtSecurityTokenHandler();  
 var key = Encoding.ASCII.GetBytes("supersecretkey1234567890"); // move to config in prod  
  
 var tokenDescriptor = new SecurityTokenDescriptor  
 {  
 Subject = new ClaimsIdentity(new[]  
 {  
 new Claim("cid", Guid.NewGuid().ToString()),  
 new Claim(JwtRegisteredClaimNames.Iss, Guid.NewGuid().ToString()),  
 new Claim(JwtRegisteredClaimNames.Iat, DateTimeOffset.UtcNow.ToUnixTimeSeconds().ToString(), ClaimValueTypes.Integer64)  
 }),  
 Expires = DateTime.UtcNow.AddHours(1),  
 SigningCredentials = new SigningCredentials(  
 new SymmetricSecurityKey(key),  
 SecurityAlgorithms.HmacSha256Signature)  
 };  
  
 var token = tokenHandler.CreateToken(tokenDescriptor);  
 return tokenHandler.WriteToken(token);  
 }  
}

# 🧪 Testing Instructions

1. Run your server:  
 dotnet run  
  
2. Send a POST request to authenticate:  
 Endpoint: http://localhost:5000/api/authenticate  
 Body: { "username": "user", "password": "pass" }  
  
3. Copy the returned token.  
  
4. Connect your SignalR client with the token:  
 Example URL: http://localhost:5000/hub/cache?access\_token=YOUR\_TOKEN  
  
5. You can now send authorized requests from the client.

# 🔐 Notes

- Do NOT use hardcoded secrets in production.  
- Move the secret key and token lifetime to configuration files or environment variables.  
- For secure SignalR usage in production, use HTTPS and proper token storage.