Why do so many articles online not include the ‘using directives’ for programming dependencies when they suggest hot fixes or code snippets? The answer to this question is the reason why I am writing this article.

This guide will walk you through how to create a JwtBearer token with a post request & how to verify the authenticity of said token via get request. It is important to note that we are using both EntityAuthentication as well as JwtBearer & we MUST specify which is being used & when.  
 **\* THIS GUIDE IS FOR ASP.NET CORE 2.1 \***  
The biggest issue we faced when attempting to implement the JWT bearer token was that we attempted to use OpenIDdict which, if you ask me, has far too many dependencies that do not hook seamless functionality into your project**. \* WE ASSUME YOU ARE STARTING FROM A SCRATCH MVC PROJECT \*  
We Credit Tahir Naushad for his creation of the actual token creation code:** <https://github.com/TahirNaushad/Fiver.Security.Bearer> **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* NuGet Packeges: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
microsoft.aspnetcore.authentication.jwtbearer v 2.20  
Microsoft.EntityFrameworkCore.Tools v 2.2.6  
Microsoft.AspNetCore.All v 2.1.6   
X.PagedList.Mvc.Core v 7.9.1**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* NuGet Packeges: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. **Replace all the code in your Startup.CS file with the code below**

using System;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using TestTokenApp.Data;

using TestTokenApp.Models;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.DependencyInjection.Extensions;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Authentication.Cookies;

using Microsoft.IdentityModel.Tokens;

using System.Net;

using TestTokenApp;

namespace TestTokenApp

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<ApplicationDbContext>(options =>

{

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"));

});

services.AddDbContext<TestTokenAppContext>(options =>

{

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"));

});

services.AddIdentity<IdentityUser, IdentityRole>()

.AddEntityFrameworkStores<ApplicationDbContext>();

services.ConfigureApplicationCookie(config =>

{

config.Events = new CookieAuthenticationEvents

{

OnRedirectToLogin = ctx => {

if (ctx.Request.Path.StartsWithSegments("/api"))

{

ctx.Response.StatusCode = (int)HttpStatusCode.Unauthorized;

}

else

{

ctx.Response.Redirect(ctx.RedirectUri);

}

return Task.FromResult(0);

}

};

});

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddCookie(cfg => cfg.SlidingExpiration = true)

.AddJwtBearer(options => {

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = "Fiver.Security.Bearer",

ValidAudience = "Fiver.Security.Bearer",

IssuerSigningKey = JwtSecurityKey.Create("fiver-secret-key")

};

options.Events = new JwtBearerEvents

{

OnAuthenticationFailed = context =>

{

Console.WriteLine("OnAuthenticationFailed: " + context.Exception.Message);

return Task.CompletedTask;

},

OnTokenValidated = context =>

{

Console.WriteLine("OnTokenValidated: " + context.SecurityToken);

return Task.CompletedTask;

}

};

});

services.AddAuthorization(options =>

{

options.AddPolicy("Member",

policy => policy.RequireClaim("MembershipId"));

});

// Add application services.

services.AddMvc();

// services.TryAddSingleton<IHttpContextAccessor, HttpContextAccessor>();

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

if (env.IsDevelopment())

{

// app.UseBrowserLink();

app.UseDeveloperExceptionPage();

app.UseDatabaseErrorPage();

}

else

{

app.UseExceptionHandler("/Home/Error");

}

app.UseStaticFiles();

app.UseAuthentication();

app.UseMvc(routes =>

{

routes.MapRoute(

name: "default",

template: "{controller=Home}/{action=Index}/{id?}");

});

//app.UseMvcWithDefaultRoute();

}

}

}

**We will explain this later.**

1. **In your solution create a new folder called ‘Helpers’**
2. **Create a new class file inside of that folder called ‘JwtHelpers.cs’**
3. **Put this content into the file:**

using Microsoft.IdentityModel.Tokens;

using System;

using System.Collections.Generic;

using System.IdentityModel.Tokens.Jwt;

using System.Linq;

using System.Security.Claims;

using System.Text;

using System.Threading.Tasks;

namespace TestTokenApp

{

public class JwtLoginInputModel

{

public string Username { get; set; }

public string Password { get; set; }

}

public static class JwtSecurityKey

{

public static SymmetricSecurityKey Create(string secret)

{

return new SymmetricSecurityKey(Encoding.ASCII.GetBytes(secret));

}

}

public sealed class JwtTokenBuilder

{

private SecurityKey securityKey = null;

private string subject = "";

private string issuer = "";

private string audience = "";

private Dictionary<string, string> claims = new Dictionary<string, string>();

private int expiryInMinutes = 5;

public JwtTokenBuilder AddSecurityKey(SecurityKey securityKey)

{

this.securityKey = securityKey;

return this;

}

public JwtTokenBuilder AddSubject(string subject)

{

this.subject = subject;

return this;

}

public JwtTokenBuilder AddIssuer(string issuer)

{

this.issuer = issuer;

return this;

}

public JwtTokenBuilder AddAudience(string audience)

{

this.audience = audience;

return this;

}

public JwtTokenBuilder AddClaim(string type, string value)

{

this.claims.Add(type, value);

return this;

}

public JwtTokenBuilder AddClaims(Dictionary<string, string> claims)

{

this.claims.Union(claims);

return this;

}

public JwtTokenBuilder AddExpiry(int expiryInMinutes)

{

this.expiryInMinutes = expiryInMinutes;

return this;

}

public JwtToken Build()

{

EnsureArguments();

var claims = new List<Claim>

{

new Claim(JwtRegisteredClaimNames.Sub, this.subject),

new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())

}

.Union(this.claims.Select(item => new Claim(item.Key, item.Value)));

var token = new JwtSecurityToken(

issuer: this.issuer,

audience: this.audience,

claims: claims,

expires: DateTime.UtcNow.AddMinutes(expiryInMinutes),

signingCredentials: new SigningCredentials(

this.securityKey,

SecurityAlgorithms.HmacSha256));

return new JwtToken(token);

}

#region " private "

private void EnsureArguments()

{

if (this.securityKey == null)

throw new ArgumentNullException("Security Key");

if (string.IsNullOrEmpty(this.subject))

throw new ArgumentNullException("Subject");

if (string.IsNullOrEmpty(this.issuer))

throw new ArgumentNullException("Issuer");

if (string.IsNullOrEmpty(this.audience))

throw new ArgumentNullException("Audience");

}

#endregion

}

public sealed class JwtToken

{

private JwtSecurityToken token;

internal JwtToken(JwtSecurityToken token)

{

this.token = token;

}

public DateTime ValidTo => token.ValidTo;

public string Value => new JwtSecurityTokenHandler().WriteToken(this.token);

}

}

1. **Now, make another folder inside of your controllers folder called ‘Api’**
2. **Create 2 class files inside of this folder : ‘ApiLogin.cs’ & ‘LocalAdminPasswordController.cs’**

**Put this inside of ‘LocalAdminPasswordController.cs’:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Security.Cryptography;

using System.Text;

using System.Threading.Tasks;

using TestTokenApp.Models;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Microsoft.AspNetCore.Authorization;

using TestTokenApp.Data;

using Microsoft.AspNetCore.Authentication.JwtBearer;

namespace TestTokenApp.Controllers

{

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme, Policy = "Member")]

[Route("api/[controller]")]

[ApiController]

public class LocalAdminPasswordController : Controller

{

private readonly TestTokenAppContext \_context;

public LocalAdminPasswordController(TestTokenAppContext context)

{

\_context = context;

}

public virtual DbSet<AspNetUserTokens> AspNetUserTokens { get; set; }

// GET: api/LocalAdminPassword/string The string is the hostname

[HttpGet("{hostname}", Name = "Get")]

public string Get(string hostname)

{

string rIp = Request.HttpContext.Connection.RemoteIpAddress.ToString();

LocalAdminPassword lap;

string password = CreateRandomPassword(20); // password better not have any of these "%"

int count = \_context.LocalAdminPasswords.Where(q => q.HostName == hostname).Count();

if (count > 0)

{

// TODO: update the password

lap = \_context.LocalAdminPasswords.Where(q => q.HostName == hostname).First();

lap.Password = password;

lap.RemoteIpAddress = rIp;

\_context.LocalAdminPasswords.Update(lap);

}

else

{

// TODO: insert the host & password

lap = new LocalAdminPassword();

lap.HostName = hostname;

lap.Password = password;

lap.RemoteIpAddress = rIp;

\_context.LocalAdminPasswords.Add(lap);

}

\_context.SaveChanges();

return password;

}

private static string CreateRandomPassword(int passwordLength)

{

string allowedChars = "abcdefghijkmnopqrstuvwxyzABCDEFGHJKLMNOPQRSTUVWXYZ0123456789!@$?\_-";

char[] chars = new char[passwordLength];

Random rd = new Random();

for (int i = 0; i < passwordLength; i++)

{

chars[i] = allowedChars[rd.Next(0, allowedChars.Length)];

}

return new string(chars);

}

}

}

**Put this into ‘ApiLogin.cs’:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authorization;

namespace TestTokenApp.Controllers.API

{

public class ApiLogin

{

}

}

1. **inside of the ‘controllers’ folder make a new class called ‘AuthorizationController.cs’**

Put this into ‘AuthorizationController.cs’:

using TestTokenApp.Models;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using System.Threading.Tasks;

namespace TestTokenApp.Controllers

{

[Route("token")]

[AllowAnonymous]

public class TokenController : Controller

{

private readonly SignInManager<IdentityUser> \_signInManager;

private readonly UserManager<IdentityUser> \_userManager;

public TokenController(

SignInManager<IdentityUser> signInManager,

UserManager<IdentityUser> userManager)

{

\_signInManager = signInManager;

\_userManager = userManager;

}

[HttpPost]

public async Task<IActionResult> Create([FromBody]JwtLoginInputModel inputModel)

{

var user = await \_userManager.FindByNameAsync(inputModel.Username);

if (user == null)

{

return Unauthorized();

}

// Validate the username/password parameters and ensure the account is not locked out.

var result = await \_signInManager.CheckPasswordSignInAsync(user, inputModel.Password, lockoutOnFailure: true);

if (!result.Succeeded)

{

return Unauthorized();

}

var token = new JwtTokenBuilder()

.AddSecurityKey(JwtSecurityKey.Create("fiver-secret-key"))

.AddSubject("james bond")

.AddIssuer("Fiver.Security.Bearer")

.AddAudience("Fiver.Security.Bearer")

.AddClaim("MembershipId", "111")

.AddExpiry(1)

.Build();

//return Ok(token);

return Ok(token.Value);

}

}

public class AuthorizationController : Controller

{

}

}

1. **In the Models folder, create 4 new files:**

**ApplicationUser.cs | AspNetUserToken.cs | LocalAdminPassword.cs | TestTokenAppContext.cs**

**Put this code inside ApplicationUser.cs:**using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Identity;

namespace TestTokenApp.Models

{

public class ApplicationUser : IdentityUser

{

}

}

**Put this code inside AspNetUserToken.cs:**using System;

using System.Collections.Generic;

// added these using statements below

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace TestTokenApp.Models

{

public partial class AspNetUserTokens

{

public string UserId { get; set; }

public string LoginProvider { get; set; }

public string Name { get; set; }

public string Value { get; set; }

}

}

**Put this code inside LocalAdminPassword.cs:**

using System;

using System.Collections.Generic;

// added these using statements below

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using TestTokenApp.Models;

namespace TestTokenApp.Models

{

public partial class LocalAdminPassword

{

// added these lines below

public int LocalAdminPasswordId { get; set; }

[Column("HostName")]

[StringLength(128)]

public string HostName { get; set; }

[Column("Password")]

[StringLength(128)]

public string Password { get; set; }

[Column("RemoteIpAddress")]

[StringLength(128)]

public string RemoteIpAddress { get; set; }

}

}

**Put this code inside TestTokenAppContext.cs:**

using System;

// added these using statements

using TestTokenApp.Models;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata;

namespace TestTokenApp.Data

{

public partial class TestTokenAppContext : DbContext

{

//public AcumenAdminContext() {

//}

public TestTokenAppContext(DbContextOptions<TestTokenAppContext> options)

: base(options)

{

}

public virtual DbSet<LocalAdminPassword> LocalAdminPasswords { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

/\*the optionsBuilder should match your database connection string from appsettings.json\*/

{

if (!optionsBuilder.IsConfigured)

{

optionsBuilder.UseSqlite("Server=(localdb)\\mssqllocaldb;Database=aspnet-TestTokenApp-C089B35D-37D5-4F35-A2C8-5AB786500773;Trusted\_Connection=True;MultipleActiveResultSets=true");

}

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

}

}

}

1. **Make sure your ApplicationDbContext.cs file is using Microsoft.EntityFrameworkCore;**
2. **In the folder ‘Data ‘ open the ApplicationDbContext.cs file**

**Add this code below inside the ApplicationDbContext.cs file:**

using System;

using System.Collections.Generic;

using System.Text;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

namespace TestTokenApp.Data

{

public class ApplicationDbContext : IdentityDbContext

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

}

}

**Now, it is important to note a few things;  
Everywhere you see ‘TestTokenApp’, or ‘TestTokenAppContext’, is a reference to an interface from either the model, data, services, Controller, or context related to the namespace of YOUR project.**

**The point of this code is to generate a random admin password for a specified local machine. The string will be around 20 characters and is constrained to specific characters for readability. Note there is not a view for this controller.**

**This all happens on the backend of the code and we can access this by using power shell.**

1. **Create a new folder inside of your solution called ‘Powershell Script Examples’**

**Create a new text file inside of this ‘Powershell Script Examples’ folder:**

$response = Invoke-WebRequest -UseBasicParsing https://localhost:44321/token -Method POST -ContentType "application/json" -Body "{ 'username':’timmy’, 'password':'Turner123!' }"

$bearer\_token = $response.content

$bearer\_token

$headers = @{Authorization = "Bearer $bearer\_token"}

$headers

$url = "https://localhost:44321/api/localadminpassword/$env:computername"

$url

$response2 = Invoke-RestMethod -UseBasicParsing $url -ContentType "application/json" -Method GET -Headers $headers

$response2

**This is a script that you would run in power shell that should allow you to see a valid token being created if you have a valid username & password. If not, it should return a 401 error (unauthorized). Be sure to change the port # in the script and whatever your app route is for the ‘LocalAdminPassword’ controller.**