Asp.NET Core 2.0 WebApi JWT Authentication with Identity & MySQL

Özgür GÜL Follow
Oct 24, 2017 · 3 min read

Recently I was configuring JWT authentication using Asp.NET Core 2.0 but with the latest update from 1.0 to 2.0, there was no tutorial or documentation, so I'm sharing.
In this post we will use **Entity Framework Core** with **MySQL**, and Identity with **JWT**. So, it will be a little long post.
If you don't know what is JWT, check this introduction.

Source

Source project is available at github: https://github.com/jatarga/WebApiJwt

. . .

Let's create a new project for our **WebApiJwt** example project:

```
# mkdir WebApiJwt
# cd WebApiJwt
# dotnet new webapi
```

Our project will be created in a few sec.
First, we will start with connecting **MySQL** to our application, but before that open the project using your preferred IDE, I'll use **Rider** since I'm on a Mac OS.

Step 1

Create a database named **webapijwt** in MySQL.

Step 2

Add Entity Framework Core and MySQL dependencies, our new .csproj file will look like this:

```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>netcoreapp2.0</TargetFramework>
    <UserSecretsId>aspnet-WebApiJwt-9EB56A08-D5EE-4EEF-B339-DEE0A5CA6277</UserSecretsId>
  </PropertyGroup>
  <ItemGroup>
    <Folder Include="wwwroot\" />
  </ItemGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.0" />
    <PackageReference Include="MySql.Data.Core" Version="7.0.4-IR-191" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="2.0.0" PrivateAsse
    <PackageReference Include="Microsoft.EntityFrameworkCore.Design" Version="2.0.0" />
    <PackageReference Include="Pomelo.EntityFrameworkCore.MySql" Version="2.0.0-rc-*" />
```

```
    <DotNetCliToolReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Tools" Version="
  </ItemGroup>
</Project>
```

acw_add_deps.csproj hosted with ♥ by GitHub view raw

Step 3

Create a directory named **Entities** in our project and create **ApplicationDbContext.cs** file in it:

```
namespace WebApiJwt.Entities
{
    public class ApplicationDbContext : IdentityDbContext
    {
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseMySql(GetConnectionString());
        }

        private static string GetConnectionString()
        {
            const string databaseName = "webapijwt";
            const string databaseUser = "";
            const string databasePass = "";

            return $"Server=localhost;" +
                   $"database={databaseName};" +
                   $"uid={databaseUser};" +
                   $"pwd={databasePass};" +
                   $"pooling=true;";
        }
    }
}
```

acw_add_dbcontext.cs hosted with ♥ by GitHub view raw

This basically extends **IdentityDbContext** and we don't have to create manually necessary tables in our database.

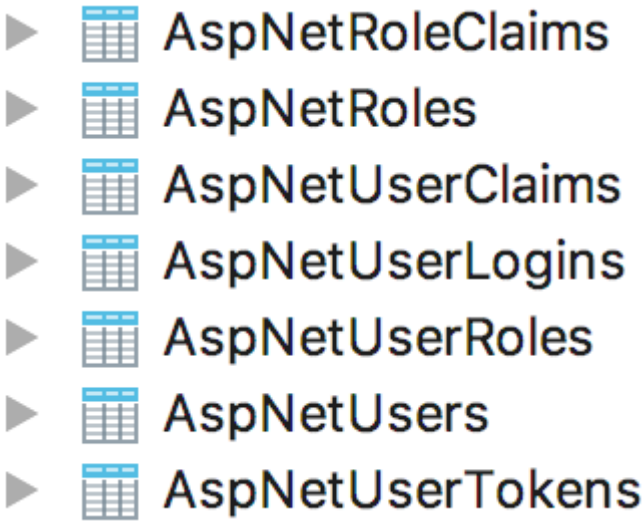
Step 4

Configure our **ApplicationDbContext** in **Startup.cs** file, it will look like this:

```
using System;
using System.Collections.Generic;
using System.IdentityModel.Tokens.Jwt;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Identity;
```

```
using Microsoft.Extensions.Options;
using Microsoft.IdentityModel.Tokens;
using WebApiJwt.Entities;

namespace WebApiJwt
{
    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 conta
        public void ConfigureServices(IServiceCollection services)
        {
            // ===== Add our DbContext ========
            services.AddDbContext<ApplicationDbContext>();

            // ===== Add Identity ========
            services.AddIdentity<IdentityUser, IdentityRole>()
                .AddEntityFrameworkStores<ApplicationDbContext>()
                .AddDefaultTokenProviders();

            // ===== Add MVC ========
            services.AddMvc();
        }

        // This method gets called by the runtime. Use this method to configure the HTTP reques
        public void Configure(
            IApplicationBuilder app,
            IHostingEnvironment env,
            ApplicationDbContext dbContext
        )
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

            app.UseMvc();

            // ===== Create tables ======
```

```
62  }
```



Step 5

Now our Identity should work. Let's configure JWT authentication

In ConfigureServices() method, add jwt stuff after adding identity, so new Startup file is:

```
1   using System;
2   using System.Collections.Generic;
3   using System.IdentityModel.Tokens.Jwt;
4   using System.Linq;
5   using System.Text;
6   using System.Threading.Tasks;
7   using Microsoft.AspNetCore.Authentication.JwtBearer;
8   using Microsoft.AspNetCore.Builder;
9   using Microsoft.AspNetCore.Hosting;
10  using Microsoft.AspNetCore.Identity;
11  using Microsoft.Extensions.Configuration;
12  using Microsoft.Extensions.DependencyInjection;
13  using Microsoft.Extensions.Logging;
14  using Microsoft.Extensions.Options;
15  using Microsoft.IdentityModel.Tokens;
16  using WebApiJwt.Entities;
17
18  namespace WebApiJwt
19  {
20      public class Startup
21      {
22          public Startup(IConfiguration configuration)
23          {
24              Configuration = configuration;
25          }
26
27          public IConfiguration Configuration { get; }
```

```
31          {
32              // ===== Add our DbContext ========
33              services.AddDbContext<ApplicationDbContext>();
34
35              // ===== Add Identity ========
36              services.AddIdentity<IdentityUser, IdentityRole>()
37                  .AddEntityFrameworkStores<ApplicationDbContext>()
38                  .AddDefaultTokenProviders();
39
40              // ===== Add Jwt Authentication ========
41              JwtSecurityTokenHandler.DefaultInboundClaimTypeMap.Clear(); // => remove default cl
42              services
43                  .AddAuthentication(options =>
44                  {
45                      options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
46                      options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
47                      options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
48
49                  })
50                  .AddJwtBearer(cfg =>
51                  {
52                      cfg.RequireHttpsMetadata = false;
53                      cfg.SaveToken = true;
54                      cfg.TokenValidationParameters = new TokenValidationParameters
55                      {
56                          ValidIssuer = Configuration["JwtIssuer"],
57                          ValidAudience = Configuration["JwtIssuer"],
58                          IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(Conf
59                          ClockSkew = TimeSpan.Zero // remove delay of token when expire
60                      };
61                  });
62
63              // ===== Add MVC ========
64              services.AddMvc();
65          }
66
67          // This method gets called by the runtime. Use this method to configure the HTTP reques
68          public void Configure(
69              IApplicationBuilder app,
70              IHostingEnvironment env,
71              ApplicationDbContext dbContext
72          )
73          {
74              if (env.IsDevelopment())
```

```
78
79              // ===== Use Authentication ======
80              app.UseAuthentication();
81              app.UseMvc();
82
83              // ===== Create tables ======
84              dbContext.Database.EnsureCreated();
85          }
86      }
87  }
```

these key & values to **appsettings.json**:

```
{
  "JwtKey": "SOME_RANDOM_KEY_DO_NOT_SHARE",
  "JwtIssuer": "http://yourdomain.com",
  "JwtExpireDays": 30
}
```

Step 6

Create a controller named **AccountController** for authentication that will contain **/Account/Login** and **/Account/Register** endpoints. It will produce JWT tokens using our **GenerateJwtToken(…)** method when login and register operation succeed:

```
1   using System;
2   using System.Collections.Generic;
3   using System.ComponentModel.DataAnnotations;
4   using System.IdentityModel.Tokens.Jwt;
5   using System.Linq;
6   using System.Security.Claims;
7   using System.Text;
8   using System.Threading.Tasks;
9   using Microsoft.AspNetCore.Identity;
10  using Microsoft.AspNetCore.Mvc;
11  using Microsoft.Extensions.Configuration;
12  using Microsoft.IdentityModel.Tokens;
13
14  namespace WebApiJwt.Controllers
15  {
16      [Route("[controller]/[action]")]
17      public class AccountController : Controller
18      {
19          private readonly SignInManager<IdentityUser> _signInManager;
20          private readonly UserManager<IdentityUser> _userManager;
21          private readonly IConfiguration _configuration;
22
```

```
26              IConfiguration configuration
27              )
28          {
29              _userManager = userManager;
30              _signInManager = signInManager;
31              _configuration =     configuration;
32          }
33
34          [HttpPost]
35          public async Task<object> Login([FromBody] LoginDto model)
36          {
37              var result = await _signInManager.PasswordSignInAsync(model.Email, model.Password,
38
39              if (result.Succeeded)
40              {
41                  var appUser = _userManager.Users.SingleOrDefault(r => r.Email == model.Email);
42                  return await GenerateJwtToken(model.Email, appUser);
43              }
44
45              throw new ApplicationException("INVALID_LOGIN_ATTEMPT");
46          }
47
48          [HttpPost]
49          public async Task<object> Register([FromBody] RegisterDto model)
50          {
51              var user = new IdentityUser
52              {
53                  UserName = model.Email,
54                  Email = model.Email
55              };
56              var result = await _userManager.CreateAsync(user, model.Password);
57
58              if (result.Succeeded)
59              {
60                  await _signInManager.SignInAsync(user, false);
61                  return await GenerateJwtToken(model.Email, user);
62              }
63
64              throw new ApplicationException("UNKNOWN_ERROR");
65          }
66
67          private async Task<object> GenerateJwtToken(string email, IdentityUser user)
68          {
69              var claims = new List<Claim>
70              {
```

```
            };

            var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["JwtKey"]
            var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
            var expires = DateTime.Now.AddDays(Convert.ToDouble(_configuration["JwtExpireDays'

            var token = new JwtSecurityToken(
                _configuration["JwtIssuer"],
                _configuration["JwtIssuer"],
                claims,
                expires: expires,
                signingCredentials: creds
            );

            return new JwtSecurityTokenHandler().WriteToken(token);
        }

        public class LoginDto
        {
            [Required]
            public string Email { get; set; }

            [Required]
            public string Password { get; set; }

        }

        public class RegisterDto
        {
            [Required]
            public string Email { get; set; }

            [Required]
            [StringLength(100, ErrorMessage = "PASSWORD_MIN_LENGTH", MinimumLength = 6)]
            public string Password { get; set; }
        }
    }
}
```

```
curl -X POST \
  http://localhost:5000/Account/Register \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/json' \
```

```
8  }'
```

acw_register_curl hosted with ♥ by GitHub view raw

Now, it should response some thing like that:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzd
WIiOiJtZUBvemd1ci5kayIsImp0aSI6ImMwMTgx
MmQ4LTI3MjktNGJhYS04YWQwLTk1ZTI4Yj
gzNzc1NCIsImh0dHA6Ly9zY2hlbWFzLnhtbHN
vYXAub3JnL3dzLzIwMDUvMDUvaWRlbnRpd
HkvY2xhaW1zL25hbWVpZGVudGlmaWVyIjoiZ
Dc2MTRiNzEtN2MyOS00OTk3LTlmODUtND
NkYzlmMDI2NzZlIiwiZXhwIjoxNTExNDIwNT
Q3LCJpc3MiOiJodHRwOi8veW91cmRvbWFpbi5
jb20iLCJhdWQiOiJodHRwOi8veW91cmRvbWF
pbi5jb20ifQ.v8YLTMTUraD7KqoHTskvcg9X_z
H5WdWkcpGuHHeqYKM
```

The returned token should be stored by your client application and will send all requests with HTTP header Authorization:

```
Authorization: Bearer eyJhbGciOiJI...
```

This is up to you how you store your token. For example in Android you may save it in **SharedPreferences** and assign to HTTP requests or you can use **localStorage** with web.

Step 8

Create a protected are for only signed in users using **Authorize** attribute:

```
[Authorize]
[HttpGet]
public async Task<object> Protected()
{
    return "Protected area";
}
```

When you do a GET request without correct token, you will get a HTTP 401 error. But if you do a correct request, it will work as expected:

```
1  curl -X GET \
2    http://localhost:5000/Account/Protected \
3    -H 'authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJtZUBvemd1ci5kayIsImp
4    -H 'cache-control: no-cache' \
5    -H 'content-type: application/json'
```

Conclusion

In this tutorial we configured Entity Framework Core with **Identity** and added JWT Authentication using Asp.NET Core 2.0 Web Api. I also used dependency injection for example when creating **AccountController.** If you don't know what it's, check this tutorial by Microsoft:

Dependency Injection in ASP.NET Core

By Steve Smith and Scott Addie ASP.NET Core is designed from the ground up to support and leverage dependency injection...

docs.microsoft.com

Aspnetcore Jwt Entity Framework Core Web Development

WRITTEN BY

Özgür GÜL

Software Engineer

Follow

See responses (48)

More From Medium

Related reads Related reads Related reads