

Inleiding

We gaan nu een project maken voor The Belgian Beer Company waarbij gebruik gaan maken van Authenticatie met JWT Bearer tokens een standaard om API verkeer te beveiligen. Daarnaast gaan we nu niet meer gebruik maken van SQLite maar van een Microsoft SQL Server database.

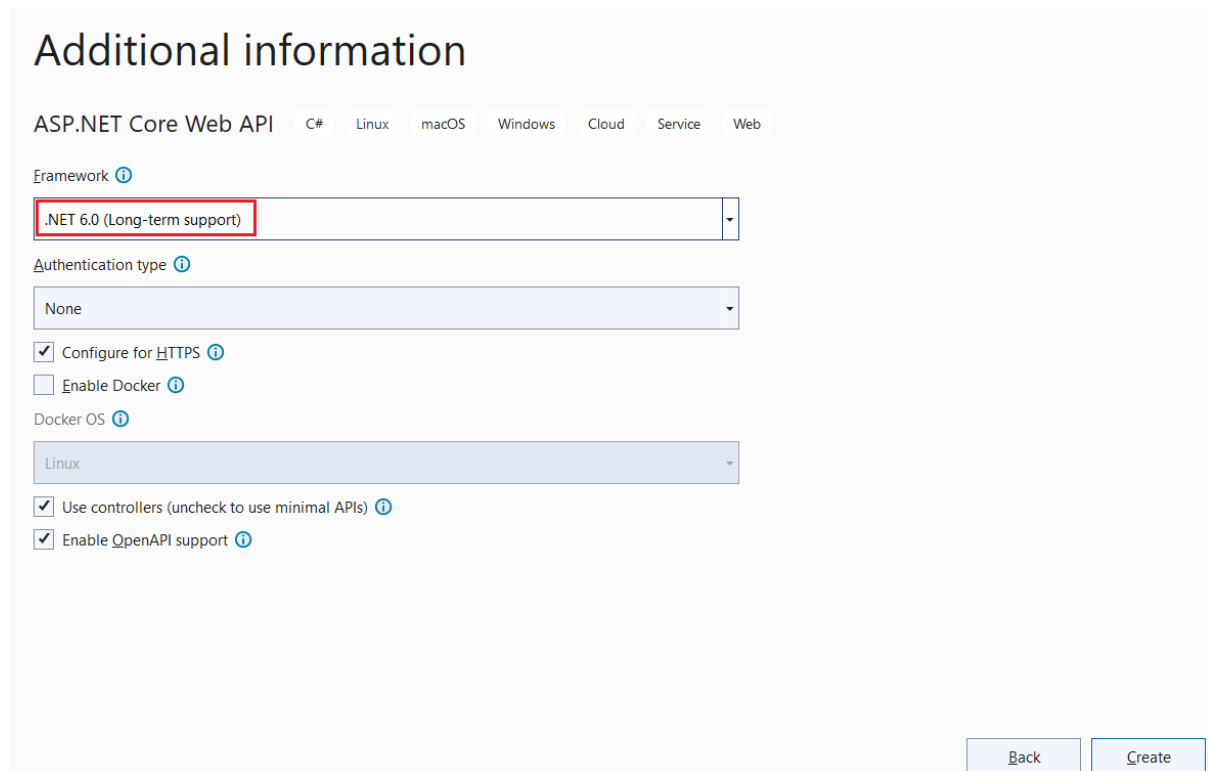
In een volgend project gaan we hetzelfde doen maar dan met MySQL als backend database.

Stappen

Maak een nieuw project :

Met gebruik van het **ASP.NET Core Web API** template

Gebruik None bij Authentication type. (Dit gaan we handmatig toevoegen.)



Additional information

ASP.NET Core Web API C# Linux macOS Windows Cloud Service Web

Framework ⓘ

.NET 6.0 (Long-term support)

Authentication type ⓘ

None

☒ Configure for HTTPS ⓘ

☐ Enable Docker ⓘ

Docker OS ⓘ

Linux

☒ Use controllers (unchecked to use minimal APIs) ⓘ

☒ Enable OpenAPI support ⓘ

Back Create

Doe de volgende acties in de package manager Console, dit om sQL server en Identity framework te kunnen gaan gebruiken in de code:

Install-Package Microsoft.EntityFrameworkCore.SqlServer

Install-Package Microsoft.EntityFrameworkCore.Tools

Install-Package Microsoft.AspNetCore.Identity.EntityFrameworkCore

Voeg toe aan je appsettings.json de volgende regels toe

```
"ConnectionStrings": {  
  "TBBCompany": "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=JWTAuthDB;Integrated  
Security=True;ApplicationIntent=ReadWrite;MultiSubnetFailover=False"  
},  
"JWT": {  
  "ValidAudience": "http://localhost:4200",  
  "ValidIssuer": "http://localhost:5000",  
  "Secret": "JWTAuthenticationHIGHsecuredPasswordVVVp10H7Xzyr"  
}
```

Dit zijn de connectiestring voor de db en JWT gegevens.

We moeten nu een aantal bestanden maken die je kunt vinden op :

<https://www.c-sharpcorner.com/article/jwt-authentication-and-authorization-in-net-6-0-with-identity-framework/> of in de github omgeving die hier bij hoort in het mapje Auth .

Het gaat om:

ApplicationDbContext.cs (een standaard context die erft van IdentityDbContext)

UserRoles.cs (een simpele class waarin de rollen worden gedefinieerd)

RegisterModel.cs (Het model wat de API gaat gebruiken voor datauitwisseling bij registreren).

LoginModel.cs (Het model wat de API gaat gebruiken voor datauitwisseling bij inloggen).

Response.cs (De reactie die de API gaat geven)

AuthenticateController.cs : een basis controller die alle requests afhandelt.

In de **Program.cs** moet het volgende opgenomen worden:

```
ConfigurationManager configuration = builder.Configuration;  
  
// Add services to the container.  
  
// For Entity Framework  
builder.Services.AddDbContext<ApplicationDbContext>(options =>  
options.UseSqlServer(configuration.GetConnectionString("ConnStr")));  
  
// For Identity  
builder.Services.AddIdentity<IdentityUser, IdentityRole>()  
    .AddEntityFrameworkStores<ApplicationDbContext>()  
    .AddDefaultTokenProviders();  
  
// Adding Authentication  
builder.Services.AddAuthentication(options =>  
{  
    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
```

```

        options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
        options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
    })

    // Adding Jwt Bearer
    .AddJwtBearer(options =>
    {
        options.SaveToken = true;
        options.RequireHttpsMetadata = false;
        options.TokenValidationParameters = new TokenValidationParameters()
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidAudience = configuration["JWT:ValidAudience"],
            ValidIssuer = configuration["JWT:ValidIssuer"],
            IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration["JWT:Secret"]))
        }
    });
});

```

Hier gebeuren een paar dingen :

1. het koppelen van de DB (met connectionstring),
2. Het koppelen van Identity service aan de DbContext.
3. Authenticatie toevoegen volgens JWT Bearer principe.
4. JWT Bearer instellen.

Aanmaken van de Database:

We gaan nu de database genereren met alle Identity tabellen (Users / Roles etc.)

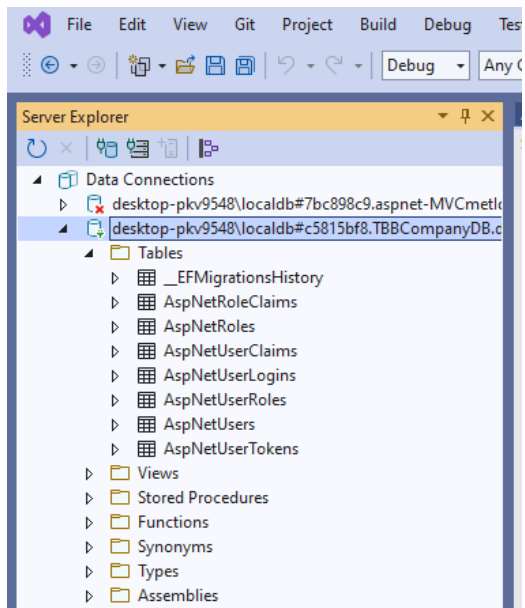
Ga hiervoor naar de package manager console en type in :

Add-Migration InitialRevision

Dit werkt of lukt alleen als er verder geen type fouten of compile errors in je code zitten. Hij genereert nu code voor de migrations. Met het volgende commando wordt deze code ook echt uitgevoerd op je database:

Update-database

In je Server Explorer kun je nu de aangemaakte Database zien:



Je kunt aan deze database nu ook je eigen tabellen toevoegen en scaffolden (Database First) of je eigen classes maken en dmv migrations toe gaan voegen aan de database (Code First).

In de Github vind je een bier.sql die je kunt executen op deze database (dat kan via DBeaver of direct vanuit visual studio => Server Explorer => rechtermuisklik New Query.

Met het scaffold commando wordt weer een dbContext en een Class Bier aangemaakt

```
Scaffold-DbContext name=ConnectionStrings:TBBCompany  
Microsoft.EntityFrameworkCore.SqlServer -t bier -OutputDir Models
```

Ik vind het mooier om alles in 1 DbContext te stoppen dus ik ga ze samenvoegen. Dit moet helaas handmatig.

Ik pas dus de gegenereerde OnModelCreating toe aan mijn bestaande ApplicationDbContext en ik voeg de DbSet toe:

```
public virtual DbSet<Bier> Biers { get; set; } = null!;
```

De controller pak ik uit het voorgaande project, let op dat je de naam nu ook aanpast naar ApplicationDbContext

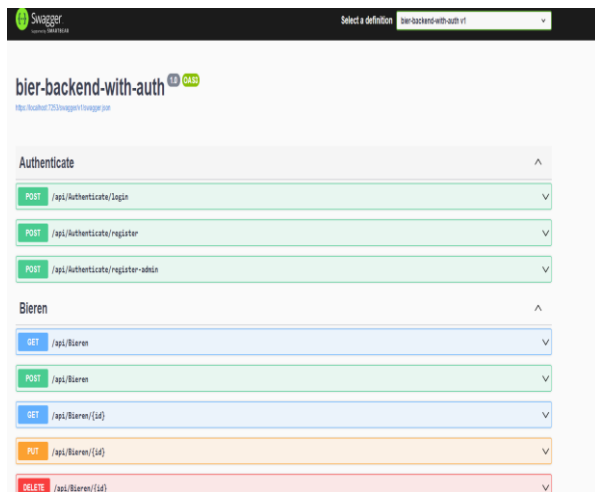
Bovenaan in de Controller kun je nu ook [Authorize] zetten om de beveiliging op deze hele controller te zetten, of je kunt de Authorize per action aangeven, daarnaast kun je de authorize ook per rol aangeven met : [Authorize(Roles = "Admin, User")]

Zie meer over combinaties van rollen en ook anonieme acces : <https://docs.microsoft.com/en-us/aspnet/core/security/authorization/roles?view=aspnetcore-6.0>

Ik geef de Get van alle bieren nog een [AllowAnonymous] zodat iedereen daar altijd bij mag.

```
// GET: api/Bieren  
[HttpGet]  
[AllowAnonymous]  
0 references  
public async Task<ActionResult<IEnumerable<Bier>>> GetBiers()  
{  
    return await _context.Biers.ToListAsync();  
}  
  
// GET: api/Bieren/5  
[HttpGet("{id}")]  
0 references  
public async Task<ActionResult<Bier>> GetBier(int id)  
{  
    var bier = await _context.Biers.FindAsync(id);
```

Test nu je API maar , om de bearer token toe te voegen moet je een tool als POSTMAN gebruiken, andere calls lukken prima met Swagger in je browser:



Succes!