Identity Rol configuratie

We starten vanaf dit project:

https://github.com/CSharpSyntraWest/IdentityRolConfig

Deze applicatie runt migraties automatisch bij opstart van het project. Dit gebeurt door de extension methode `MigrateDatabase` die wordt aangeroepen door Main methode van Program.cs

```
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().MigrateDatabase().Run();

        }
```

….

ASP.NET Core Identity Implementatie

We installeren eerst de library **Microsoft.AspNetCore.Identity.EntityFrameworkCore** via de NuGet package manager:

Wijzig de `ApplicationContext` class: Zet IdentityDbContext als basis class in plaats van DbContext:

```
using IdentityRolConfig.Models.Configuration;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace IdentityRolConfig.Models
{
    public class ApplicationContext : IdentityDbContext<IdentityUser> // DbContext
    {
        public ApplicationContext(DbContextOptions options) : base(options)
        {
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.ApplyConfiguration(new EmployeeConfiguration());
            base.OnModelCreating(modelBuilder);
        }

        public DbSet<Employee> Employees { get; set; }
    }
}
```

ASP.NET Core Identity Configuratie

De ASP.NET Core Identity kan geregistreerd worden met de volgende extension methoden: `AddIdentityCore<TUser>` en `AddIdentity<TUser, TRole>`.

De `AddIdentityCore` methode voegt de services toe die nodig zijn voor user-management operaties, zoals aanmaken van Users, hashing van wachtwoorden, wachtwoord-validatie,….

Indien je applicatie niet enkel users, maar eveneens Rollen wil ondersteunen, dan moet je de `AddIdentity` methode gebruiken.

Je kan hetzelfde bereiken met de methode `AddIdentityCore` , maar dan moet je manueel nog de services voor beheer van Rollen, SignInManager, Cookies, … registreren.

We gebruiken dus de `AddIdentity` methode:

Open Startup.cs en voeg de volgende lijnen toe aan ConfigureServices :

```
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<ApplicationContext>(opts =>
                opts.UseSqlServer(Configuration.GetConnectionString("sqlConnection")));

            services.AddIdentity<IdentityUser, IdentityRole>()
                    .AddEntityFrameworkStores<ApplicationContext>();

            services.AddControllersWithViews();

        }
```

We gebruiken eveneens de `AddEntityFrameworkStores` methode om de nodige EF Core implementatie van de Identity stores toe te voegen.

Aanmaken van de ASP.NET Core Identity Tabellen

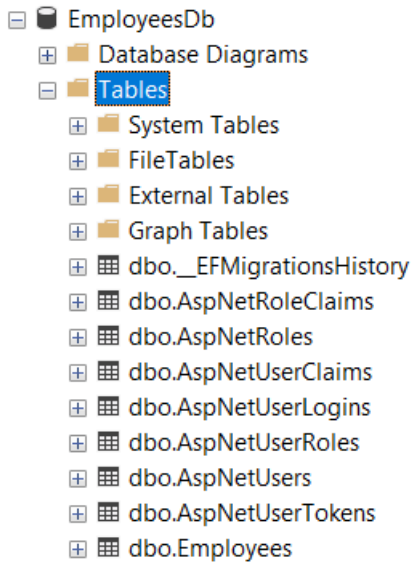
Om de ASPNet tabellen voor users en rollen te stockeren, voegen we eerst een migratie toe via de volgende instructie in de Package Manager Console:

```
PM> Add-Migration CreatingIdentityTables
```

En generatie van de tabellen (uitvoeren van de Up methode van de nieuwe Migration class):

```
PM> Update-Database
```

Controleer in de database dat de AspNet tabellen zijn aangemaakt:



Rollen toevoegen aan de database

Om initieel rollen toe te voegen aan de database, maken we een RoleConfiguration class aan in de folder Configuration:

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace IdentityRolConfig.Models.Configuration
{
    public class RoleConfiguration : IEntityTypeConfiguration<IdentityRole>
    {
        public void Configure(EntityTypeBuilder<IdentityRole> builder)
        {
            builder.HasData(
            new IdentityRole
            {
                Name = "Visitor",
                NormalizedName = "VISITOR"
            },
            new IdentityRole
            {
                Name = "Administrator",
                NormalizedName = "ADMINISTRATOR"
            });
        }
    }
}
```

Open ApplicationContext.cs en voeg de volgende lijn toe aan de `OnModelCreating` methode om de RoleConfiguration aan te roepen:

```
using IdentityRolConfig.Models.Configuration;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace IdentityRolConfig.Models
{
    public class ApplicationContext : IdentityDbContext<IdentityUser> // DbContext
    {
        public ApplicationContext(DbContextOptions options)
        : base(options)
        {
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.ApplyConfiguration(new EmployeeConfiguration());
            modelBuilder.ApplyConfiguration(new RoleConfiguration());
            base.OnModelCreating(modelBuilder);
        }

        public DbSet<Employee> Employees { get; set; }
    }
}
```

Voeg een nieuwe migration toe om de gewijzigde OnModelCreating uit te voeren en de rollen aan de tabel AspNetRoles toe te voegen:

```
PM> Add-Migration InsertedRoles
```

```
PM> Update-Database
```

Controleer in de database dat de 2 volgende rollen aan de tabel AspNetRoles zijn toegevoegd :

```
/****** Script for SelectTopNRows command fro
SELECT TOP (1000) [Id]
      ,[Name]
      ,[NormalizedName]
      ,[ConcurrencyStamp]
  FROM [EmployeesDb].[dbo].[AspNetRoles]
```

Results | Messages

Id	Name	NormalizedName	ConcurrencyStamp
42943528-e9a5-48e2-a181-45c390d1a4fa	Visitor	VISITOR	d03bc84f-c62e-4308-b9cc-8d50173601f6
b77f4ea8-3543-4a65-a760-c1cbbd1a93c0	Administrator	ADMINISTRATOR	15438167-c293-40e4-81c8-844f524a61ee