Key learnings for this project

Pascal Lüscher

December 29, 2019

Contents

1 Entity Framework

1.1 Packages

Package	Description
Npgsql.EntityFrameworkCore.PostgreSQL	For postgresql db access
Microsoft.EntityFrameworkCore.Design	For migrations

1.2 CLI

Command	Description
dotnet ef migrations add MigrationName	Create a migration
dotnet ef migrations remove	Delete the latest migration
dotnet ef database update	Apply the migration(s)
dotnet ef database update MigrationName	Revert the migration

1.3 Configuration

Add the dbcontext to the service collection

```
services.AddDbContext<DoenerOrderContext>(options =>
{
  options.UseNpgsql(
    Configuration.GetConnectionString("DefaultConnection")
  );
});
```

1.4 Relations

For a many to many relationship, a join-class is needed. In this example I used the join-class ProductIngredient for the Product $\leftrightarrow$ Ingredient relationship.

```
public class ProductIngredient
{
    public int ProductId { get; set; }
    public Product Product { get; set; }

    public int IngredientId { get; set; }
    public Ingredient Ingredient { get; set; }
}
```

In the dbcontext you need to configure the composite primary key

```
protected override void OnModelCreating(ModelBuilder builder)
{
  base.OnModelCreating(builder);
  builder.Entity<ProductIngredient>()
    .HasKey(t => new {t.ProductId, t.IngredientId});
}
```

2 Authentication & Authorization

2.1 Packages

Package	Description
Microsoft.AspNetCore.Identity	For authentity items
Microsoft.AspNetcore.Identity.EntityFrameworkCore	For IdentityDbContext
Microsoft.AspNetcore.Authentication.JwtBearer	For JWT access token

2.2 Identity Setup

Create a Model for the IdentityUser and IdentityRole (this is useful if I need to overwrite stuff / add fields etc.)

```
public class User : IdentityUser<int>
{
  [PersonalData]
  public string Firstname { get; set; }

  [PersonalData]
  public string Lastname { get; set; }
}

public class Role : IdentityRole<int>
{

}
```

Make the DbContext either derrive from IdentityDbContext or make a new dbContext for Identity.

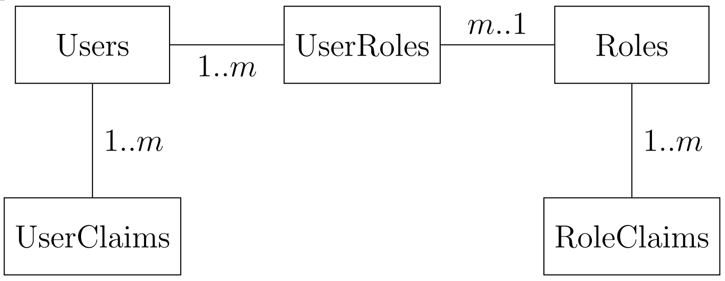
```
public class DoenerOrderContext : IdentityDbContext<User,
   Role, int>
{
  public DoenerOrderContext(DbContextOptions options) : base(
     options) { }
  ...
}
```

Add the Identity to the services (in `configureServices`)

```
services.AddIdentity<User, Role>(options =>
{
  options.User.RequireUniqueEmail = true;
  options.Password.RequireDigit = true;
})
.AddEntityFrameworkStores<DoenerOrderContext>();
```

2.3 Identity Usage

The identity framework can be used for several different authorization methods. For an easier understanding what is going on, the following erd should help :



The authorization can be role-based or claim-based.

The role-based authorization uses checks if the `Users` is in a `Roles`. To check that simply add the `[Authorize(Roles = "RoleName"])]` annotation to a controller method. If the method should be accessible for multiple roles, separate them with a comma: `[Authorize(Roles = "Role1,Role2")]`. If more than one role is required, stack the `Authrize` Annotations.

```
[Authorize(Roles = "Role1")]
[Authorize(Roles = "Role2")]
public async Task<IActionResult> DoSomething() {...}
```

For an in-depth view about role-based authorization see the official documentation.

The claim based authorization uses the `UserClaims` and `RoleClaims` assigned to a user to check if he can access the method. I will focus on permission based authorization using the Claims. For an in-depth view about claim-based authorization see the official documentation. I added a helper class `CustomClaimTypes` very similar to `System.Security.Claims.ClaimTypes` with constants

```
public static class CustomClaimTypes
{
  private const string ClaimTypeNamespace
    = "http://isitar.ch";
  public const string Permission
    = ClaimTypeNamespace + "/permission";
}
```

To keep permissions consistent and easy to manage, I also created the helper class `ClaimPermissions` where I store all my permission strings.

```
public static class ClaimPermission
{
  public const string CreateUser = "User.Create";
  public const string DeleteUser = "User.Delete;
  ...
}
```

To recognize the authorization in a `Authorize` annotation I need to configure the Authorization in the `ConfigureServices` method in the `Startup` class.

```
services.AddAuthorization(options =>
{
  foreach (var prop in typeof(ClaimPermission).GetFields(
      BindingFlags.Public | BindingFlags.Static |
      BindingFlags.FlattenHierarchy)
  )
  {
    var propertyValue = prop.GetValue(null).ToString();
    options.AddPolicy(propertyValue, policy =>
      policy.RequireClaim(CustomClaimTypes.Permission,
          propertyValue));

  }
});
```

This adds all the permission claims as a `Policy` and it can be checked in the controllers and methods via the `Authorize` attribute

```
[Authroize(Policy = ClaimPermission.CreateUser)]
public async Task<IActionResult> DoSomething() {...}
```

2.4 JWT Authentication

To add JWT support, the Authentiaction needs to be added to the services:

```
services.AddAuthentication(config =>
  {
    config.DefaultAuthenticateScheme =
      JwtBearerDefaults.AuthenticationScheme;
    config.DefaultChallengeScheme =
      JwtBearerDefaults.AuthenticationScheme;
  })
  .AddCookie(options => {
    options.SlidingExpiration = true;
  })
  .AddJwtBearer(JwtBearerDefaults.AuthenticationScheme,
    options =>
    {
      var jwtSettings =
        Configuration.GetSection("JwtSettings");
      options.TokenValidationParameters = new
         TokenValidationParameters()
      {
        ValidateIssuer = true,
        ValidIssuer = jwtSettings["Issuer"],
        ValidateAudience = true,
        ValidAudience = jwtSettings["Audience"],
        IssuerSigningKey = new SymmetricSecurityKey(
          Encoding.UTF8.GetBytes(jwtSettings["Secret"])
        )
      };
    }
  );
```

In the appsettings the following Keys need to be present:

```
"JwtSettings": {
  "Issuer": "isitar.ch",
  "Audience": "DoenerUser",
  "Secret": "SomeVeryLongSecretStringThatIsTotallyRandom"
}
```

To generate the JWT-Token a new JwtSecuritytoken needs to be created.

```
[AllowAnonymous]
public async Task<IActionResult> GetTokenAsync([FromBody]
   LoginViewModel loginViewModel)
{
  if (!ModelState.IsValid)
  {
    return BadRequest(ModelState);
  }

  var user = await userManager.FindByNameAsync(loginViewModel
     .Username);
  var res = await userManager.CheckPasswordAsync(user,
     loginViewModel.Password);
  if (!res)
  {
    return BadRequest();
  }

  var jwtSettings = configuration.GetSection("JwtSettings");
  var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(
     jwtSettings["Secret"]));
  var singingCreds = new SigningCredentials(key,
     SecurityAlgorithms.HmacSha256);
  var expiry = DateTime.UtcNow.AddMinutes(30);
  var claims = await GetValidClaims(user);

  var token = new JwtSecurityToken(
    jwtSettings["Issuer"],
    jwtSettings["Audience"],
    claims,
    expires: expiry,
    signingCredentials: singingCreds
  );

  return Created("", new
    {
    token = new JwtSecurityTokenHandler().WriteToken(token)
    }
  );
}
```

The `GetValidClaims` method is explained in 2.5

2.5 JWT Authorization

When authorizing with the JWT-Token, ASP.NET assumes the claims are stored in the token. An easy thing to do is add all claims the user possesses into the token. For this I added a helper method in the `LoginController`.

```
private async Task<IEnumerable<Claim>> GetValidClaims(User
   user)
{
  IdentityOptions identityOptions = new IdentityOptions();
  var claims = new List<Claim>
  {
    new Claim(JwtRegisteredClaimNames.Sub, user.UserName),
    new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().
       ToString()),
    new Claim(JwtRegisteredClaimNames.Iat, DateTimeOffset.
       UtcNow.ToUnixTimeSeconds().ToString(),
    ClaimValueTypes.Integer64),
    new Claim(identityOptions.ClaimsIdentity.UserIdClaimType,
        user.Id.ToString()),
    new Claim(identityOptions.ClaimsIdentity.
       UserNameClaimType, user.UserName),
  };
  var userClaims = await userManager.GetClaimsAsync(user);
  claims.AddRange(userClaims);
  var userRoles = await userManager.GetRolesAsync(user);
  foreach (var userRole in userRoles)
  {
    claims.Add(new Claim(ClaimTypes.Role, userRole));
    var role = await roleManager.FindByNameAsync(userRole);
    if (role != null)
    {
      var roleClaims = await roleManager.GetClaimsAsync(role)
          ;
      claims.AddRange(roleClaims);
    }
  }

  return claims;
}
```

3 Swagger

3.1 Packages

Package	Description
Swashbuckle.AspNetcore	For Swagger setup and doc generation

3.2 Setup

To setup the swagger doc the generation needs to be added to the services.

```
services.AddSwaggerGen(c =>
{
  c.SwaggerDoc("v1", new OpenApiInfo
  {
    Title = "Doener Order Api",
    Version = "v1",
    Description = "Simple doener order app",
  });
}
```

The swagger json and swagger ui need to be exposed, so in the `Configure` method the following snipped needs to be added

```
app.UseSwagger();
app.UseSwaggerUI(c =>
{
  c.SwaggerEndpoint("/swagger/v1/swagger.json", "Doener Order
     Api");
  c.RoutePrefix = "swagger";
});
```

With this the swagger ui is accessible under url.tld/swagger.

To add the documentation from the methods, the `csproj` file needs to be edited so the documentation is exported in an xml file:

```
<PropertyGroup>
  <GenerateDocumentationFile>true</GenerateDocumentationFile>
  <NoWarn>$(NoWarn);1591</NoWarn>
</PropertyGroup>
```

The NoWarn removes the warnings for methods that have no documentation. Further the swagger generation (`services.AddSwaggerGen`) needs to be extendes by the following snippet:

```
var xmlFile = $"{Assembly.GetExecutingAssembly().GetName().
    Name}.xml";
var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile)
    ;
c.IncludeXmlComments(xmlPath);
```

To add jwt security to the swagger gen (used to authorize in swagger ui) the following snippet needs to be added in the swagger generation (`services.AddSwaggerGen`):

```
c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
{
  Description =@"JWT Authorization header using the Bearer
      scheme.
    \r\n\r\n Enter 'Bearer' [space] and then your token in
        the text input below.
    \r\n\r\nExample: 'Bearer 12345abcdef'",
  Name = "Authorization",
  In = ParameterLocation.Header,
  Type = SecuritySchemeType.ApiKey,
  Scheme = "Bearer",
});
c.AddSecurityRequirement(new OpenApiSecurityRequirement()
{
  {
    new OpenApiSecurityScheme
    {
      Reference = new OpenApiReference
      {
        Type = ReferenceType.SecurityScheme,
        Id = "Bearer"
      },
      Scheme = "oauth2",
      Name = "Bearer",
      In = ParameterLocation.Header,
    },
    new List<string>()
  }
});
```