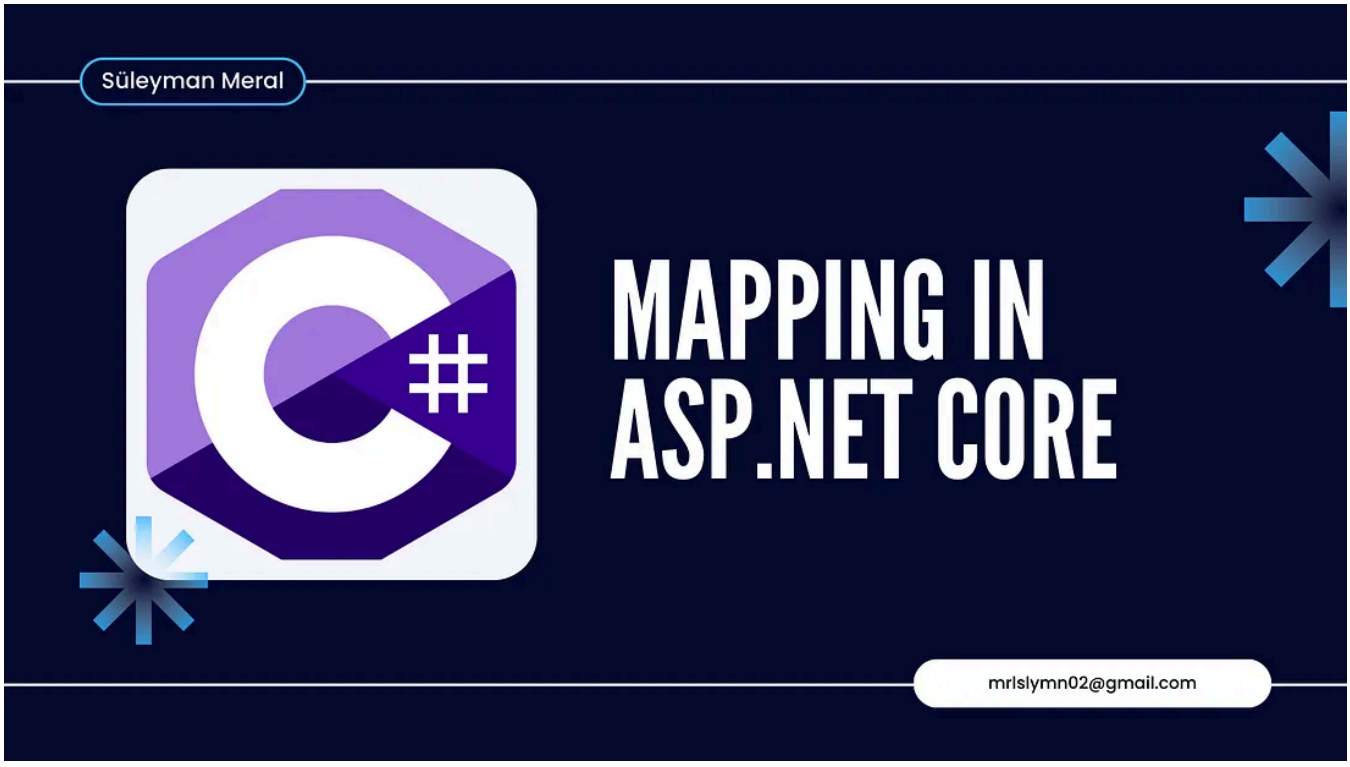
ASP.Net Core Mapping İşlemi — Automapper Kullanımı

5 min read · Apr 12, 2025

Süleyman Meral

Share More



Bu yazımda, yazılım geliştirme sürecinde sıkça karşılaşılan ve özellikle **Backend Developer** olmayı hedefleyenler için büyük önem taşıyan **mapping (eşleme)** kavramına değineceğim. Aynı zamanda bir örnek yaparak bu işlemi pekiştireceğiz.

Mapping İşlemi Nedir ?

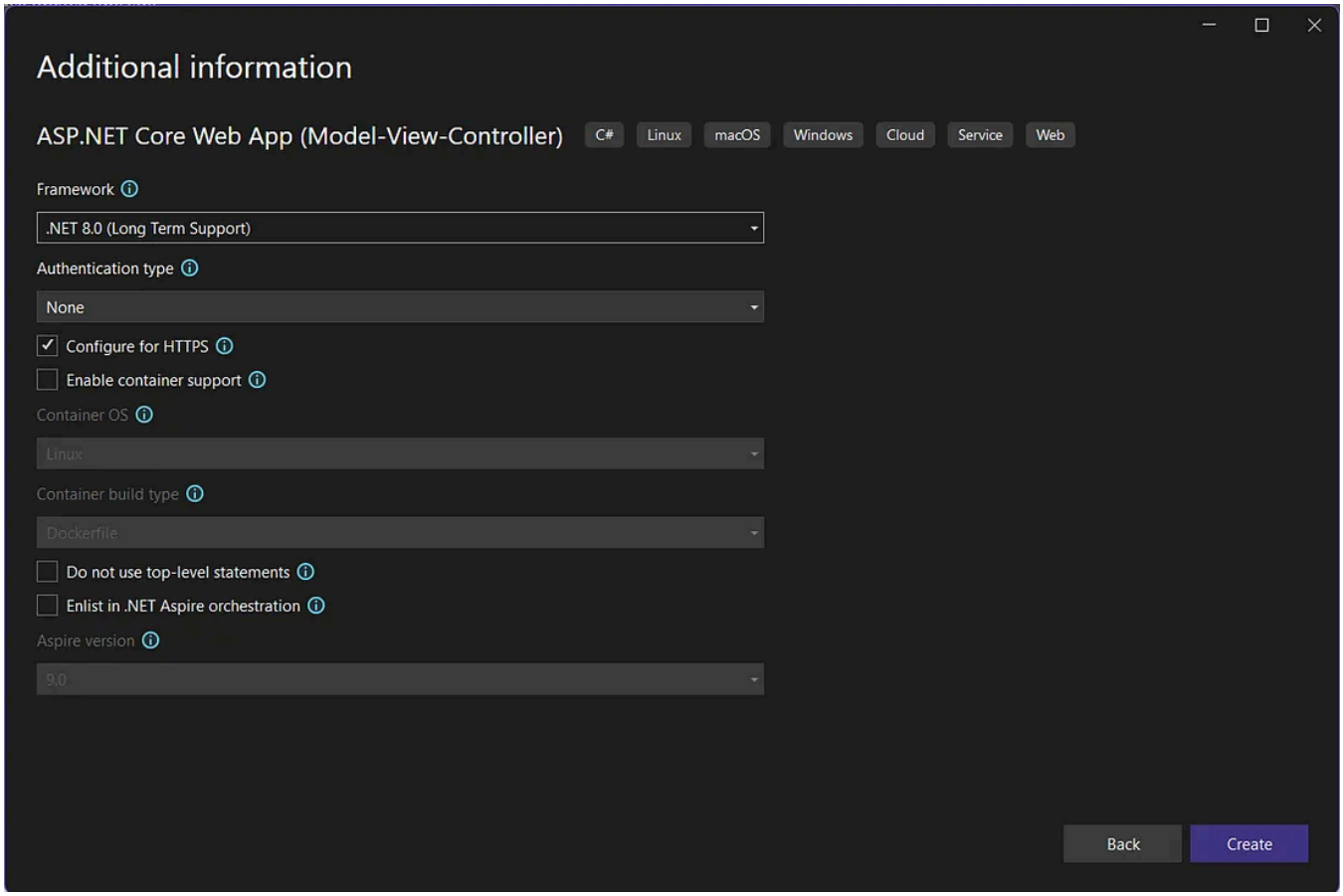
Mapping, yazılım geliştirme sürecinde bir veriyi bir yapıdan başka bir yapıya dönüştürme işlemidir. Özellikle katmanlı mimarilerde; veri tabanından gelen modellerin (entity), kullanıcıya sunulacak veri transfer nesnelerine (DTO — Data Transfer Object) dönüştürülmesi veya tam tersi işlemler sırasında kullanılır. Bu

süreç, katmanlar arası bağımlılığı azaltarak uygulamanın daha sürdürülebilir, okunabilir ve yönetilebilir olmasını sağlar. Mapping işlemi genellikle manuel olarak kodlanabileceği gibi, AutoMapper gibi kütüphaneler kullanılarak da

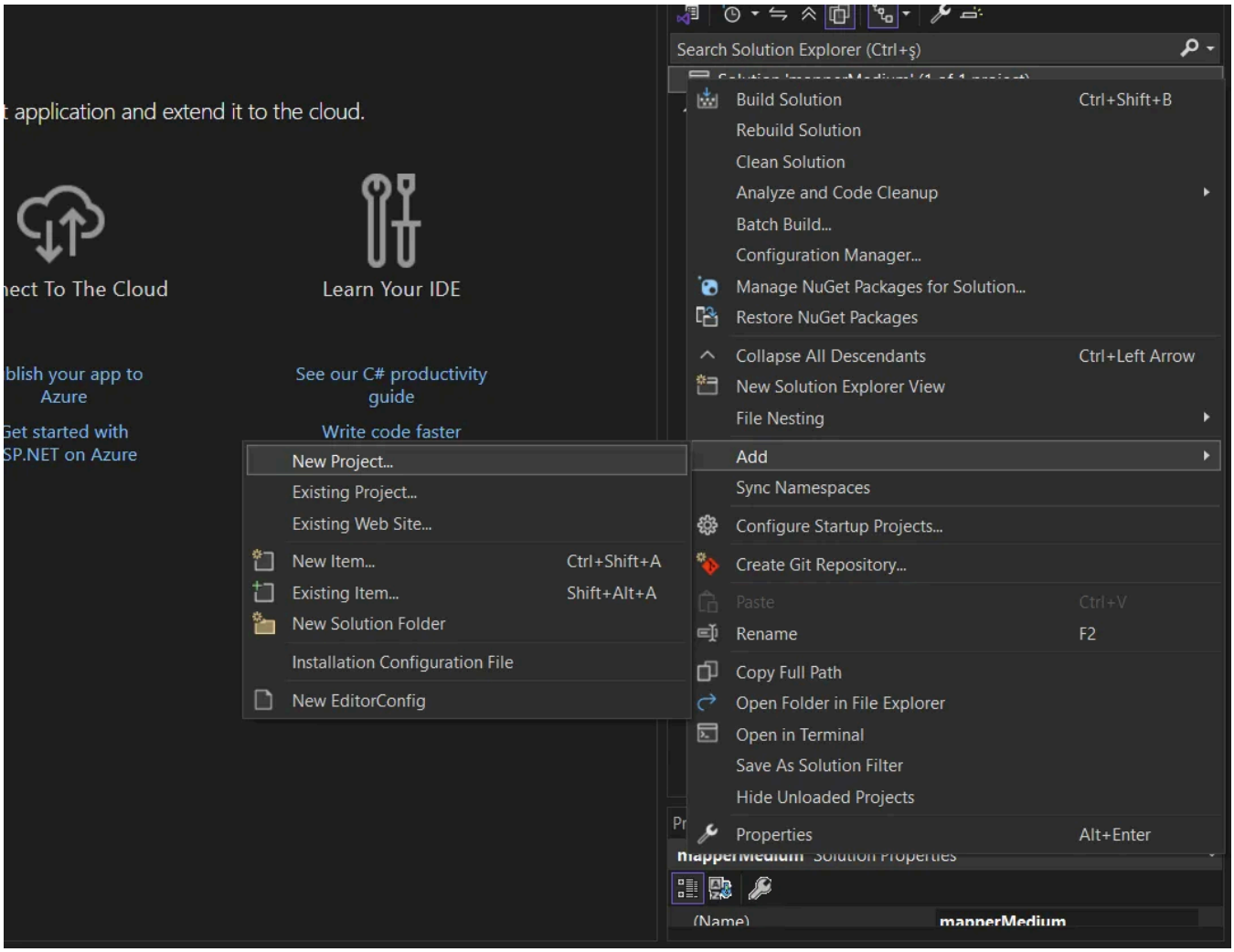
Mapping, yazılım mimarisinde veri tutarlılığını ve katmanlar arası ayrımı sağlamak için kritik bir rol oynar. Özellikle büyük ve ölçeklenebilir projelerde, veritabanı modellerini doğrudan istemcilere sunmak hem güvenlik hem de esneklik açısından ciddi riskler doğurur. Bu noktada mapping, veritabanından alınan verilerin istemciye uygun formatta ve güvenli şekilde sunulmasını mümkün kılar. Aynı zamanda domain modelleri, DTO'lar ve ViewModel'ler arasında açık bir ayrım oluşturarak uygulamanın bakımını kolaylaştırır ve test edilebilirliğini artırır. Mapping sayesinde her katman kendi sorumluluğu dahilinde çalışır; bu da yazılımda temiz mimari ve SOLID prensiplerine daha uygun bir yapı sağlar.

Ufak bir teorik bilgi verdikten sonra örneğimizle birlikte konumuzu pekiştirelim.

Visual Studio üzerinden bir Asp.Net Core projesi oluşturalım.

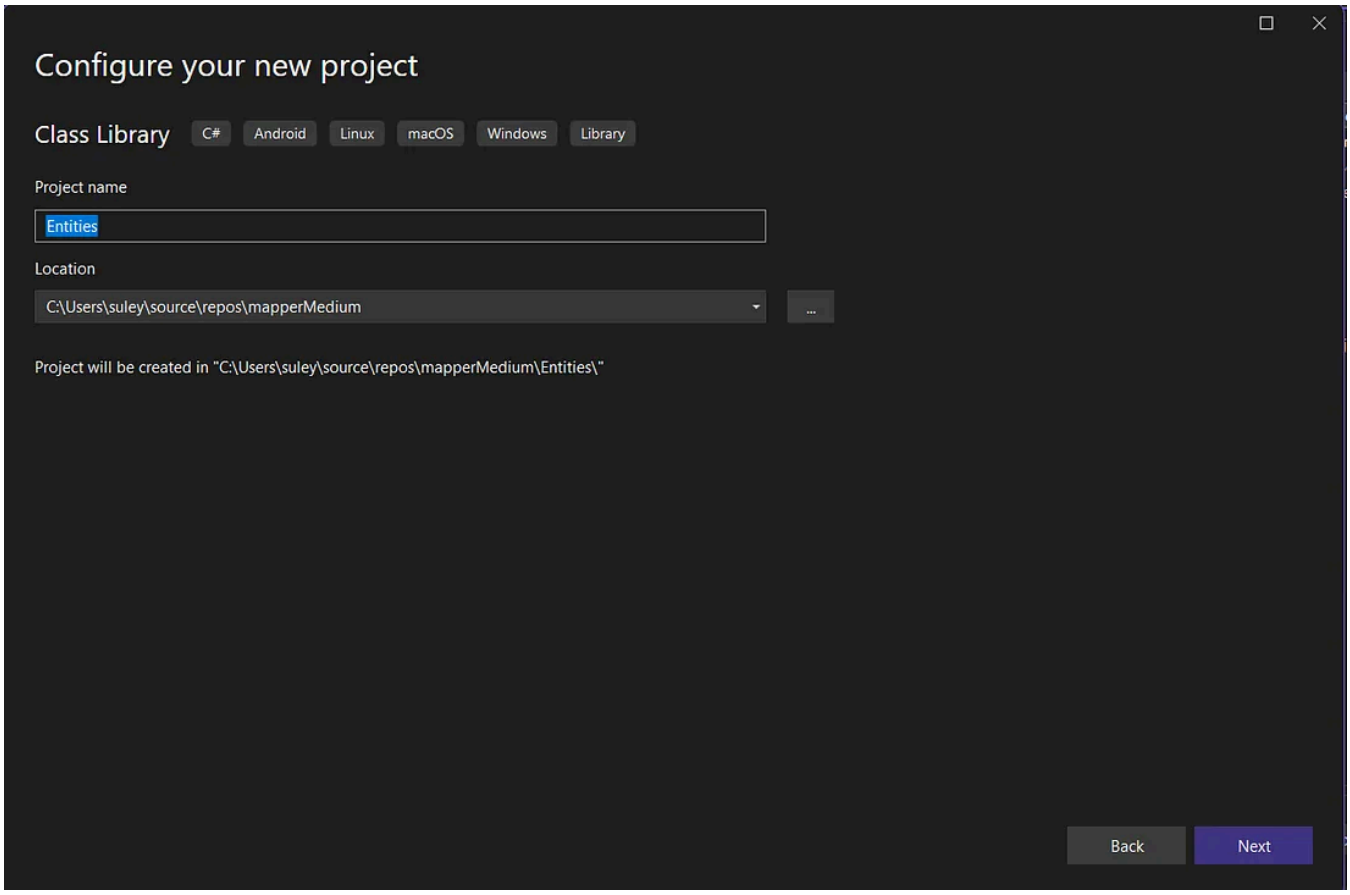


Framework olarak son framework versiyondan bir öncekini seçmek genelde daha iyi bir tercih olacaktır. Bu yüzden long term support olan .Net 8.0 ile projemi oluşturuyorum. .Net 9 kullansaydık da bir problem olmazdı.

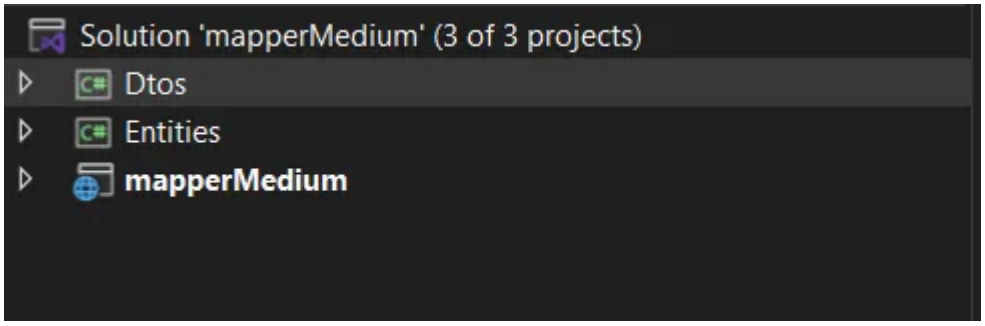


Temiz ve anlaşılır bir mimari kurmak için entity ve dto'larımızı klasörde tutmak yerine ayrı katmanlar olarak projemize dahil edeceğiz.

Solution'a sağ tıklayıp yeni bir proje ekliyoruz.



Entitylerimizi tutması için bir Entities Class Librarysi oluşturuyoruz . Aynı işlemi Dtos katmanı için de yapıyoruz.



mapperMedium projesini Sunum katmanı olarak varsayabiliriz. Fakat isterseniz bir Presentation Layer'da oluşturabilirsiniz. Görüldüğü üzere 3 katmandan oluşan bir projemiz var. Şimdi örnek olarak bir entity oluşturalım. Adı Product olsun.

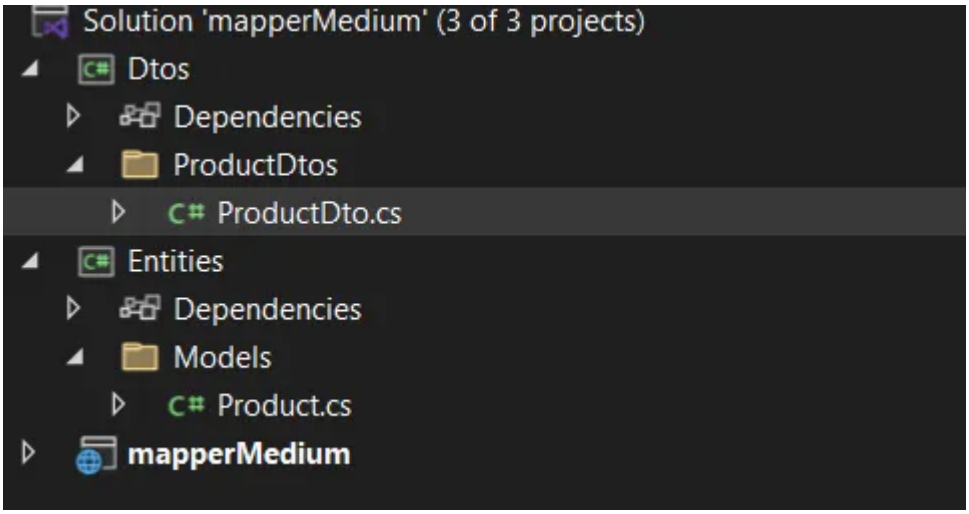
Product entity classımızın içerisine 3 adet property girelim.

```
public class Product
{
    public int Id { get; set; }
    public string? Name { get; set; }
    public decimal Price { get; set; }
}
```

Entityler aslında veri tabanındaki tabloları teslim etmektedir. Entity classlarındaki propertyler ise tablolardaki sütunları temsil etmektedir.

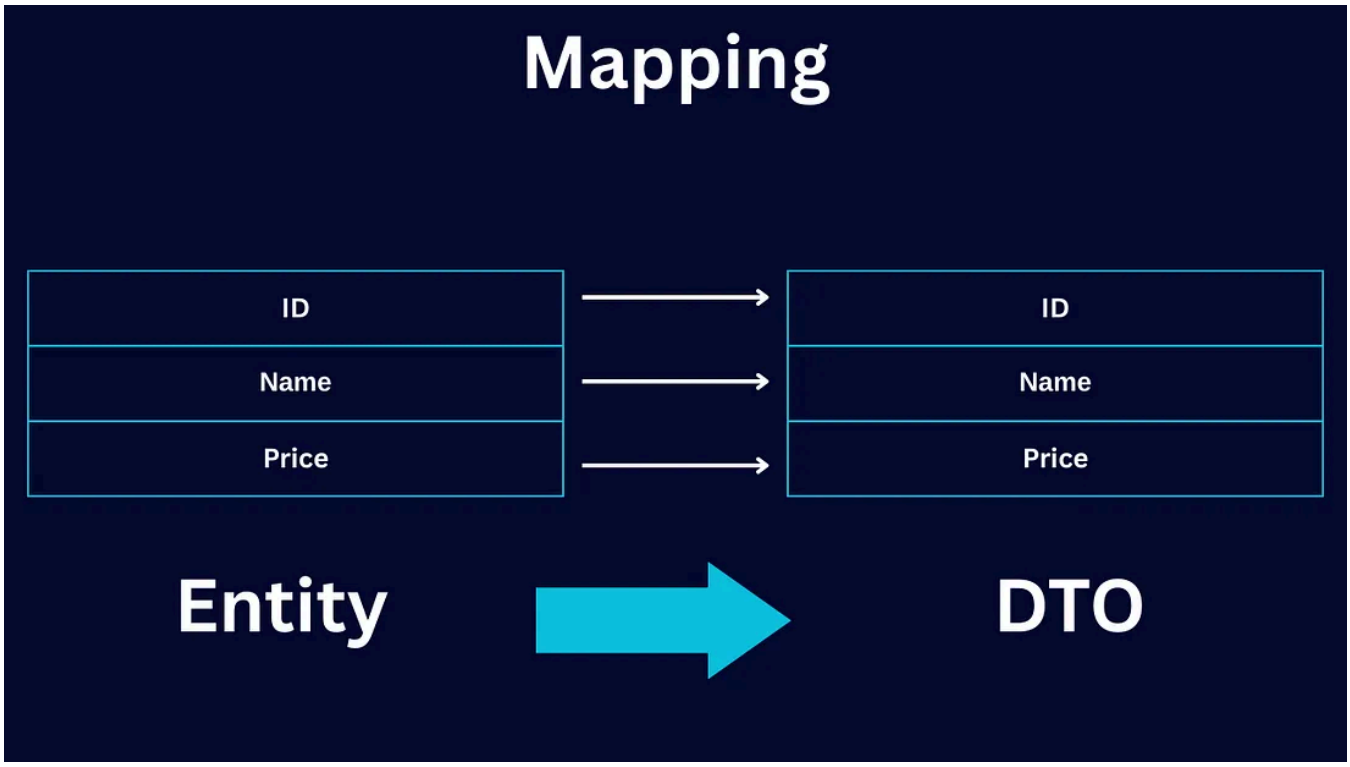
Örneğimizin uzamaması adına bir veri tabanı bağlantısı yapmayacağız. Mock Data kullanarak bir veri örneği ekleyeceğiz.

Şimdi ise productDto classımızı oluşturalım.



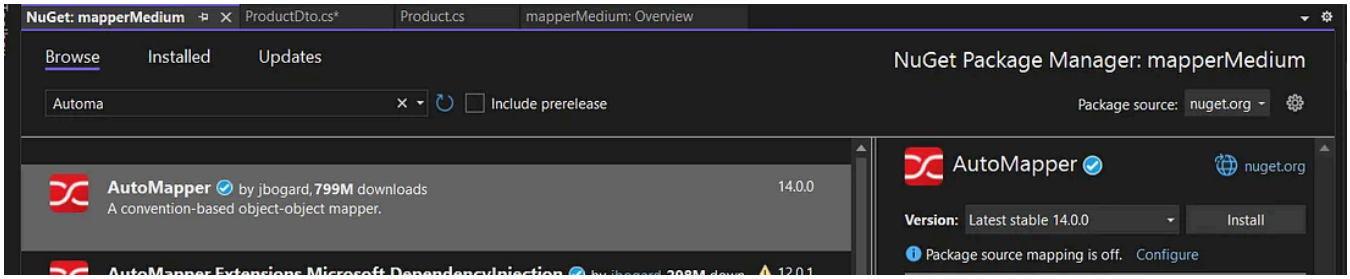
ProductDto genellikle read işlemi için kullanılır. Biz de öyle yapacağız. CreateProductDto , UpdateProductDto, DeleteProductDto şeklinde eklemeler yapılabilir. Her Dto sadece işine yarayan propertyleri alabilir. Örneğin DeleteProductDto için sadece Id kullanılabilir. Bunu da ek bir bilgi olarak ekleyelim.

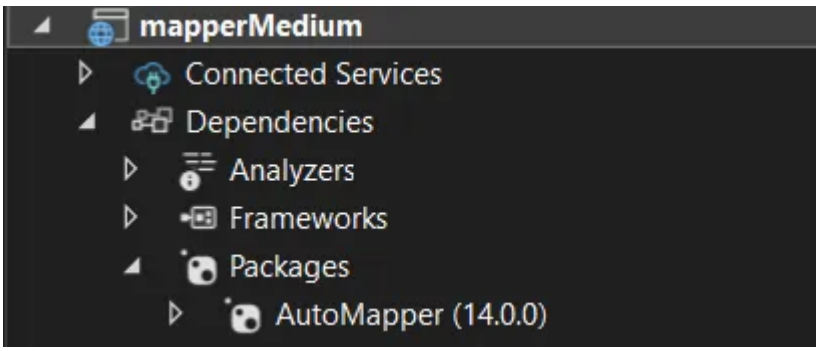
Dto Ve Entity katmanlarımızı oluşturduk. Bizim asıl amacımız bu iki obje arasında bir eşleşme yapmak.



Bu işlemin tam tersi de gerçekleşebilir.

Automapper kütüphanemizi kullanacağımızı söylemiştik. Sunum Katmanımıza Nuget Package üzerinden Automapper kütüphanesini ekleyelim.



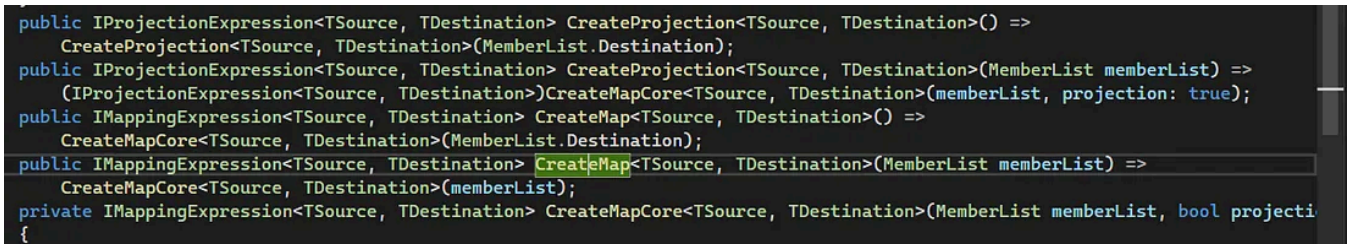


Dependencies üzerinden kütüphanenin yüklenip yüklenmediğini kontrol edebiliriz. Görüldüğü üzere kütüphanemiz yüklenmiş durumda.

Şimdi bir Mapping klasörü oluşturalım ve mapping işlemi için gerekli olan profile classını oluşturalım.

```
using AutoMapper;

namespace mapperMedium.Mapping
{
    public class ProductMappingProfile:Profile
    {

    }
}
```

Bu classımız Profile classından miras alacak. Bu class automapper ile projemize dahil olan bir class. O yüzden projenizde Profile isimli bir class varsa adını değiştirmeniz daha iyi olacaktır.



Miras aldığımız Profile classını incelersek içinde “CreateMap” methodunun olduğunu görebiliriz. Mapleme işlemi için bu fonksiyonu kullanacağız.

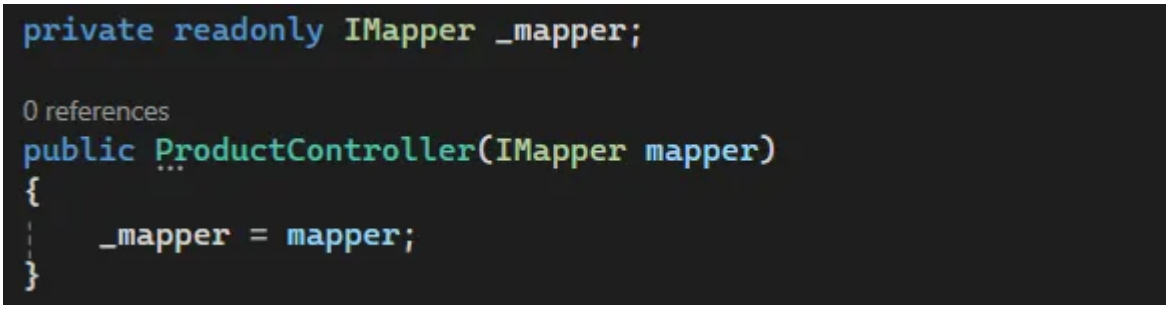
Bildiğiniz üzere miras alınan sınıflara ait methodlar türettiğimiz classlarda da kullanılabilir. Bunu da ek bir bilgi olarak ekleyelim.

Bir ProductController oluşturalım. Ve mock data olarak bir datayı görelim.

```
0 references
public class ProductController : Controller
{
    0 references
    public IActionResult Index()
    {
        return View();
    }

    [HttpGet]

    0 references
    public IActionResult GetProduct()
    {
        var product = new Product
        {
            Id = 53,
            Name = "Phone",
            Price = 50000
        };

        return Ok(product);

        }
    }
}
```

Test için sahte bir veri oluşturduk ve bunu return ettik. Bu verinin veri tabanından gelen bir veri olduğunu varsayalım. Kullanıcıya direkt olarak veri tabanından gelen nesneyi sunuyoruz. Bunun önüne geçmek için product nesnesini productDto ya çevirip bu nesneyi döndüreceğiz.

Bu işlemi de mapper ile yapacağız.



```
private readonly IMapper _mapper;

0 references
public ProductController(IMapper mapper)
{
    _mapper = mapper;
}
```

Burada dependency injection işlemi uygulayarak mapper servisini kullanmayı sağlıyoruz. readonly olarak işaretlenmesinin sebebi sadece ctor içinde atanabilmesidir.

```
[HttpGet]

0 references
public IActionResult GetProduct()
{
    var product = new Product
    {
        Id = 53,
        Name = "Phone",
        Price = 50000
    };


    var productDto = _mapper.Map<ProductDto>(product);
    return Ok(productDto);

}
```

Map fonksiyonunda önce Varış nesnesini sonra kaynak nesneyi yazıyoruz.

Tabi bu işlemin gerçekleşebilmesi için Profile classımızda bu işlem için bir Map oluşturmalıyız.

```
1 reference
public class ProductMappingProfile : Profile
{
    0 references
    public ProductMappingProfile()
    {
        CreateMap<Product, ProductDto>();
    }


}
```

Map işlemini de gerçekleştirdik. Şimdi Program.cs üzerinden AutoMapper için servis kaydını yapalım.

```
builder.Services.AddControllersWithViews();
builder.Services.AddAutoMapper(typeof(Program));
```

Tek satır kod ile bu şekilde kaydı yapabilmekteyiz.

Pretty-print

{"id":53,"name":"Phone","price":50000}

GetProduct Sayfasını çalıştırdığımızda geçici verimiz bu şekilde gözüküyor. Fakat bu veri direkt olarak Product nesnesinden değil productDto nesnesinden geliyor.

Mapping işlemimiz basit olarak bu şekilde gerçekleşmektedir.

Peki şöyle bir soru aklımıza takılabilir. ProductMappingProfile sınıfını controllerda hiç kullanmadık bu Mapper bu classı nasıl tanıdı?

Bunun sebebi

```
builder.Services.AddAutoMapper(typeof(Program));
```

Bu servis kaydının Program içerisinde Profile sınıfından class alan tüm Profile sınıflarını eklemesidir.

`AddAutoMapper(typeof(Program))` , `Program.cs` 'in bulunduğu assembly'deki tüm `Profile` sınıflarını tarar ve AutoMapper'a tanıtır.

Kendimce mapping işleminden ve AutoMapper kütüphanesinden bahsetmeye çalıştım. Bu konuyu ayırca youtube kanalımda anlattım. İlgilenenler için linki bırakıyorum 👇

Yazılım | Softare Development | Software Engineering

Edit profile

Written by Süleyman Meral

6 followers · 6 following

Backend Developer(.NET Core)

No responses yet

Süleyman Meral

What are your thoughts?

More from Süleyman Meral