**Api Projesi Notlar**

Startup dosyası hakkında olan video. Start up donsyasında 2 önemli fonkisyon bulunur;

ConfigureServices ve Configure metodlarıdır. ConfigureServices’de servisler implemente edilir.

Configure ise; little verileri implemente edilen metod, katmanları eklediğimiz yer.

Belli bir zaman sonra startup dosyası şişmeye başlıcak onun için;sade olması için , ConfigureServices ve Configure metodları içindeki yapıları, extension metod içine taşıyo olucaz.

Controller içinde aciton metodları dışında yardımcı metodlar bulundurulmamalı

Hataları merkezi bir metoddan alınmalı, global olmalı, action metodlar içinde try catch kullanılmamalı.zor durumda kalırsan action içinde kullanılabilirsin

DTO nedir; dto dönülen modelin birebir aynısı olan ama içerisinde sadece client’ın görmesi gereken property’ler olan sınıftır.bu sınıflara DTO adı veriyoruz. Aynı zaman client’tan data isteğimiz zaman entity tanımlamak yerine bu DTO nesnelerini kullanıyoruz.

Seed dosyası nedir, veritabanı oluşturulduğunda default değerler olur, bu default değerleri seed dosyası üzerinden yöneticez

Product, kategori sınıfına bağlı olacağı için, kategori sınıfına referans vermemiz lazım, bunun için public virtual Category Category {get;set;} dedik, böylece category sınıfna referans verdik.Neden virtual dedik çünkü entityframewrok category üzeriden inherit kullanarak tracking yapıcak yani değişiklikleri izliycek ,yani category üzerinde dağişiklikleri hafızasında tutabilsin.

//category'de bire çok ilişki olacağı için. bir categorynin birden çok ürün olacağı için collection tanımladım

//category ilk oluşunca bir tane boş bir colletion oluşturmak için

public Category()

{

Products = new Collection<Product>();

}

IRepository Generic İnterface

Repository pattern ne iş yapar? Veriye tek bir kanaldan erişmeyi sağlar,ayrıca create-delete-update gibi crud işlemlerinin merkezi bir yerden tek seferde oluşturulmasını sağlıyor, yani her tablo için create,delete,update yazmamıza gerek kalmıycak

//generic repository olacağı için TEntity yaptık sadece T'de yazabilirdik

Yazdığımız bu metodların implementasyonunu Data katmanında yapıcaz, şu anda sadece interface’leri yazıyoruz(şu anda sözleşmelerimiz,contractlarımız var).

Repository ve Service arasındaki fark; Repository sadece veritabanıyla ilgili işlemleri gerçekleştirir.Ama serviste örneğin product ile ilgili iç bir metod oluşturmak istediğimizde örneğin barcodun doğru olup olmadığını kontrol etmek istediğimizde iç metod yazabiliriz, bu kontrol için veritabanıyla işimiz yok örneğin; //bool ControlInnerBarcode(Product product); veritabanıyla ilgili olmayan işlem için örnek

İnterface İmplementasyonu ne demek; interface’i uyguluycak class’lar anlamına geliyor

IUnitOfWork interface’nin oluşturulması. IUnitOfWork interface nedir;entity tarafında yapılan delete- update-insert gibi işlemlerin veritabanına yansıyıp yansımamasını bize bırakmasıdır

AppDbContect, sql server tarafınfaki veritabanına karşılık gelmektedir.Tablolara karşılık gelen ise DbSet’lerdir

Productseed.cs dosyası nedir; default dataların oluşmasını sağlar

Repository class’ı oluşturma videosu. Repository veritabanı ile ilgili işlemlerinin yapıldığı yerdir.irepository’lerin implemente edildiği yerdir.

Task senkron programlamadaki void’e karşılık gelir

Startup.cs dosyasının güncellenmesi videosu; şu anda DbContext’imiz hangi veritabanını kullanacağını bilmiyor, o yüzden bununla ilgili sevisin eklenmesini gerçekleştiricez.

Migration nedir; entity tarafındaki değişikliklerin veritabanına yansımasıdır veya başka açıklama ile entity tarafındaki yapıların sqlserver tarafında tablo olarak oluşmasını sağlamaktır.

Dbcontext’i nerede oluşturuyoruz Data’da oluşturuyoruz

Migration nerede gerçekleştirilir? AppDbContext neredeyse orada gerçekleştirilir projemizde AppDbContext data katmanında.

add-migration Initial klasörünün oluşması için kullanıyoruz, bu komutu yazdıktan sonra migration klasöürü içinde initial.cs adında cs dosyası oluşuyor, bu cs dosyası içinde 2 tane metod oluşuyor,

up ve down metodları. Up metodu şu anlama geliyor; veritabanında oluşacak nesneler burada otomatik şekilde tanımlanıyor bu işlemi gerçekleştiriyor up metodu.Yani data katmanındaki configuration klasörü içindeki categoryconfiguration.cs içindekiler migration tarafında işleniyor

down metodu ise; initial migrationu geri almak istersek down metodu çalışır,yani tabloları siler

Update-Database dersek; veritabanında oluşturur.

Not: migration bir tool’dur

Api direk servis katmanı ile haberleşicek; yani api içindeki controller içerisindeki metodlar direk service katmanı üzeriden haberleşicek

hatırlarsak repositroy tarafında savechange metodları kullanmamıştık ,Veritabanına yansıma işlemini service katmanı mı yapıyor, sorumlu oluyor

async ve await birbirinden ayrılmaz, beraber kullanılır.bu ikisi asenkron programlamanın 2 önemli keywordu’dür

public class CategoryService : Service<Category>, ICategoryService//

Service<Category> bu generic olarak category aldığını gösterir

Startup.cs dosyasının güncellenmesi; hangi interface’e karşılık hangi class’tan nesne üretilicek belirtmemiz lazım, bunu gerçekleştiricez

CategoryController dosyamız

Bu controller insert,update,delete gibi endpointleri içericek olan controllerdır.

Not hatırlatma: core mimarimiz , içerisinde model(entity) ve interface’lerin olduğu temel projemizdi. Data mimarimizde; migration, seed, repository(core’daki interfaceleri implemente eden),Dbcontext vardı. Service mimarimizde; service(core’daki IServiceleri implemente eden),Business Codes vardı.

Not 2: api sadece service katmanı ile haberleşicek.yani product üzerinde bir işlem yapıcaksak sadece sadece categoryservice’i implemente edicez. product içinde aynısı geçerli

Not 3:

//bu return'de önceden return Ok(categories); vardı ama best pratice için üstüdekini kullandık

//çünkü best pratice açısından hiçbir zaman entity'leri dönmemek gerekiyor, bu yüzden data transfer objesi oluşturduk.client’a bazı propertyleri göstermek isteyebiliriz,bunun için dto ile kapsülleme yapıyoruz

//entity'leri dto'lara dönüştürmek için auto mapper kütüphanesini kullandık

CategoryDto.cs class’ının oluşturulması

Automapper kurulması ve yüklenmesi;

Mapping klasörünü neden oluşturduk; hangi nesnenin hangi nesneye dönüşmesini belirlemek için oluşturduk

CategoriesController’daki GetById action metodunun oluşturulması

CategoriesController’daki Save action metodunun oluşturulması

[HttpPost]

public async Task<IActionResult> Save(CategoryDto categoryDto)//client’dan da dto olarak alıyoruz

{

var newCategory = await categoryService.AddAsync(\_mapper.Map<Category>(categoryDto));

return Created(string.Empty, \_mapper.Map<CategoryDto>(newCategory));

}

CategoriesController’daki Update action metodunun oluşturulması

[HttpPut] kullanıyoruz

//best practice için update de bişey dönmüyoruz 204 durum kodu gömüyoruz

CategoriesController’daki delete action metodunun oluşturulması

[HttpDelete("{id}")] kullandık

CategoriesController daki son action, son endpoint; GetWithProductsById; bu metod ile hem bu id’ye ait categori dönülecek hem bu kategoriye ait product’lar varsa dönülecek

Not:controller.cs dosyasında dto dönüştürme işlemi yapabilmek için Mapping klasöründe de bunu belirtmemiz gerekir.

ProductController ve GetById action metodunun oluşturulması

ProductController ve Save/Update/Remove action metdolarının oluşturulması

ProductController’daki GetWithByCategoryId action metodunun oluşturulması, bu metod ile şunu sağlıycaz; product’ı dönerken aynı zaman bu product’ın category’sini de dönmüş olucaz

Person adında bir entity oluşturalım, nerde oluşturucaz, core katmanında oluşturucaz.Core katmanında Person adında entity oluşturduktan sonra bu entity için Data katmanında Dbset olarak eklememiz gerekiyor, bu ekleme işlemini AppDbContext.cs sınıfında yapıyoruz.yaptığımız tanımlama şöyle=>public DbSet<Person> Persons { get; set; }

Sonra Configuration yapıyoruz,aslında ayrı bir configuration yapmak gerekir ama AppDbSet içinde yaptık Person için. Şimdi person için migration yapmalıyız, bunun için Package Manager Console’a Add Migration Person yazıyoruz => enter. Sonra Update-database diyoruz

Not hatırlatma; core katmanında sadece model ve interface’ler bulunmaktadır.

PersonController ve action metodlarının oluşturulması

Not, person için IService’i kullanıcaz

Not hatırlatma; migration’ı ve Configuration’u data katmanında yapıyoruz

Validationfilter, kod tekrarını azaltmak için kullanıyoruz.

Validation için, Startup dosyasında düzenleme yapıyoruz.neden validation için kendimiz filter yazıyoruz çünkü dönen hataların tek tipte olmasını ve aynı json formatında olmasını isityoruz, böylece best practice’e uygun şekilde yapmış oluyoruz.

Client’lara dönüş yaparken DTO şeklinde dönüyorduk, onun içiç Api katmanındaki DTO klasöründe ErrorDto.cs isimli cs dosyası oluşturuyoruz.

Şimdi filter yazmak için api katmanında yeni bir klasör oluşturuyoruz Filter adında.Bu klasör altında VlidationFilter.cs adında cs dosyası oluşturuyoruz.Filter fonksiyonunda OnActionExecuting olarak seçtik neden Request ilgili action’a geldiği zaman müdahale etmek istiyoruz, yani ilk anda müdahale etmek istiyoruz onun için bunu seçtik. Örneğin result bittiğinde müdahale etmek isteyebilirdik o zaman OnActionExecuting değil farklı fonksiyon kullanmamız gerekirdi.

Şu şekilde çalışacak; ErrorDto.cs’deki public List<String> Errors { get; set; }ile Errors listesi dolucak ve ValidationFilter.cs’deki Add ile dolucak.şimdi filter kullanılmaya hazır bunu nerede kullanıcaz örneğin ProductController’da kullanıcaz, onun için ValidationFilter’i ProductController’a ekliycez

Özellikle delete ve update komutlarında ilgili id’de ürün var mı yok mu kontrolü için bu filter kodluyoruz.bunun öncelikle Api katmanındaki Filter klasöründe NotFoundFilter adında bir cs dosyası oluşturuyoruz.

NotFoundFilter’i ValidationFilter gibi tanımlayabilirdik ama NotFoundFilter yapıcı metodunda “id” alıyor onun için Startup.cs dosyasına services.AddScoped<NotFoundFilter>(); ekledik(şöylede açıklayabiliriz neden eklediğimizi çünkü bu filter’in yapıcı metodunda interface implemente ettiğmizden dolayı ).yani filterlarımız bir id alıyorsa, bunu mutlaka startup tarafında AddScoped olarak ekliyoruz ve bir action metod üzerinde kullanabiliyoruz

Not: ValidationFilter’in global olarak tüm controller’lara eklenmesini istiyorsak,startup tarafına geliyoruz burada services.AddControllers(); bulunuyor bunu değiştiriyoruz.

services.AddControllers(o =>

{

o.Filters.Add(new ValidationFilter());

});//değiştirdiğimiz hali

Bset Practice açısında UpdateDto adında yeni bir Dto oluştumaktır ama şimdi ProductDto üzerinden gidicez.Extension metod nedir; Net framework tarafında var olan, objelerin üzerine extra yazdığımız metodlardır.(ok olanların hepsi extension metod) Extension’lar mutlaka static olmalı