**SaveChangeAsync Interceptor**

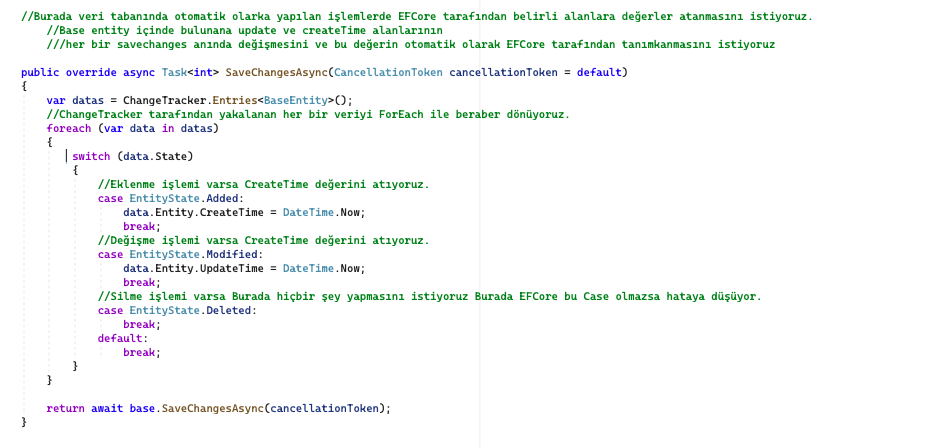
Burada amacımız bizim Entitylerimiz içinde bulunan BaseEntity’den gelen CreateTime ve UpdateTime değerlerinin tamamının bizim tarafımızdan değil o an o işlem olduğu anda gerçekleşmesini sağlamak.

metin, ekran görüntüsü, yazı tipi, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Görselde görüldüğü gibi bir veri insert edilirken bir T anında bu değerin oluşturulmasını isteyebiliriz. Örnek olarak bizim projemizde bulunan CreateTime ve değerlerini verebiliriz.  
  
Şimdi BaseEntity dosyamızı düzenleyelim.

|  |
| --- |
| using System;  namespace OnionArch.Domain.Entities.Common  {  public class BaseEntity  {  public Guid ID { get; set; }  public DateTime CreateTime { get; set; }  public DateTime UpdateTime { get; set; }  }  } |

Şimdi amacımız her bir veri ekleme ve düzenleme işleminde veri tabanında bulunan CreateTime ve UpdateTime alanlarını EFCore tarafından otomatik olarak eklenecek düzenlenecek ve güncellenecek hale getirmek.  
  
Bunun için yapmamız gereken ,   
**OnionArch.Persistance** altında bulunan Context yapımızda birkaç adet Configürasyon yapmamız gerekmekte bunun için ilk olarak SaveChangesAsync methodunu override yaparak bu işlemi gerçekleştirebiliriz.  
  


Şimdi artık ekleme çıkarma işlemlerini yaparken hiçbir şekilde kullanıcının yada geliştiricinin bu değerleri atamasına gerek yok .   
  
Şimdi yeni bir veri oluşturma ve bir veriyi düzenleme işlemlerimize geçiş yapalım.

Öncelikle OnionArch.WebApi katmanına geçiş yapalım yani bizim Presentation katmanımıza.

Burada Controller yapımızın içinde Update ve Create işlemleri için geliştirdiğimiz kodlarımıza tekrar göz gezdirelim.  
  
metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Görselde göreceğimiz üzerine biz giriş değerleri için Product Entity’sini direkt olarak Swager içinde tanımladık.  
Swager içinde hiçbir zaman Gelecek olan tip Entityl olarak verilmez !

Bunun için ViewModel denen kavram kullanılır .

**ViewModel**

OOP yapılanmasında bir modelin kullanıcıyla etkileşimi neticesinde kullanılan/doldurulan memberlarını temsil eden ve süreçte ilgili model yerine kullanılan/taşınan/transfer edilen/post edilen vs. bir nesnedir.  
  
Yani daha basit bir anlatımla EFCore’da bulunan Entitylerde kullanıcı yada geliştiricilerin Veri Tabanında müdahale edeceği alanları içeren ve bu Objeden oluşmuş verileri hedef objeye çevirirken geliştiriciler için velinimet bir yapıdır.

OnionArch.Application Katmanımıza geçelim.

* Bir adet klasör oluşturalım klasörümüzün adını View\_Models olarak adlandıralım.
* Daha sonra projemizde bulunan bütün Entityler için burada Modeller oluşturalım örnek olarak biz burada Product Entity değerimize göre bir ViewModel oluşturalım.
* İhtiyaca göre Create ve Update işlemlerinde farklı farklı senaryolar olabileceğinden dolayı burada farklı farklı işlemler için farklı farklı ViewModeller oluşturmak her zaman daha mantıklıdır. (Best Practice açısından)

metin, yazı tipi, çizgi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu **metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

Şimdi ViewModelimizi oluşturduğumuza göre Create işlemi ile burada örneklerimize göz atalım.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Update fonksiyonumuzu geliştirelim.

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Validation işlemleri

Uygulamaları geliştiriken validasyon işlemlerini hem Frontend hem Backend olacak şekilde iki taraftada uygulamamız gerekmekte. Şimdi backend kısmında validasyonların nasıl çalıştığına göz atacağız.  
İlk olarak ;

* Şimdi burada fluentValidation adında bir kütüphane kullanacağız bu sayede validation işlemlerini dinamik olarak gerçekleştirebileceğiz.
* Burada ViewModel üzerinden uygulama yapacağız . (EFCore sayesinde Entity üzerinden’de validation işlemi yapabiliyorduk)
* Aplication katmanında bu işlemi gerçekleştiriyoruz.
* Application katmanı altında yeni bir klasör oluşturalım ve adını Validators olarak adlandıralım.
* Şimdi projemize kurmamız gereken 3 adet kütüphane söz konusu bu kütüphaneleri öncelikle Application katmanında kurmamız gerekmekte.Aşağıda bulunan görselde ilk 3 kütüphaneyi projemize kuralım.

metin, yazılım, multimedya yazılımı, bilgisayar simgesi içeren bir resim

Açıklama otomatik olarak oluşturuldu

* Şimdi Oluşturduğumuz klasörde şöyle bir mantık ile ilerleyeceğiz kardeşim sen bu validator’u ne için oluşturuyorsun?   
   -ViewModel için değil mi ?

Peki Bu ViewModel hangi Entitiy için   
 -Product => Hemen bir tane Klasör açalım ve ismini Product\_Validator olarak ayarlayalım  
 Peki hangi işlem için ?   
 -Create. => O zaman bir class oluşturalım adını CreateProductValidator olarak isimlendirelim.

O zaman dosyamızı ViewModel\_Create\_Product\_Validator olarak adlandıralım.

Şimdi dosyamızı oluşturduk.

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu  
  
Burada istenen şartları biz belirledik ve kullanıcın eğer bu bilgileri doldurmadan bize bir şekilde bir post işlemi sağlarsa, bu tarz hatalar almasını sağlamak amacımız.  
  
Her zaman olduğu gibi burada da yapmamız gereken işlem bu Validation işlemlerimizi bizim IOC’de tanımlamak.

Burada işimiz aslında biraz daha kolay burada bu Validation işlemleri Controller yapısı üzerinde yapıldığı için ,  
Daha önce .NetCore tarafından tanımlanmış olan

builder.Services.AddControllers();

Yapısına doğrudan bu kütüphaneyi eklememiz yeterli olacaktır. metin, ekran görüntüsü, yazı tipi, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Controller üzerinden CreateProduct action değerini düzenleyelim.

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

Projemizi başlatalım ve Swager üzerinden bir Ürün oluşturma işlemi yapalım.  
metin, ekran görüntüsü, yazılım, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şimdi burada sadece bu düzenlememiz yeterli olmayacak çünkü .NetCore tarafından belirlenen filtreleme yöntemini iptal edip bizim filtreleme yöntemimizi devreye sokmamız gerekmekte.

Bunun için IOC konfigürasyonlarımızda ufak bir düzenleme yapmamız gerekmekte…

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şimdi burada sadece bizim hata mesajlarımız gelsin isteyebiliriz , yani diğer hata mesajlarını görmek istemeyiz.

Bunun için kodumuzda ufak bir ekleme yapmamız gerekiyor.  
  
Amacımızı tekrar hatırlayalım; burada gelen değerleri filtrelemek yani burada gerçekleşen verilerinin doğrultusunda gelen hata mesajlarında sadece bizim hata mesajlarımızı döndürmesini istiyoruz.

Yaptığımız bu işlem bir servis işlemidir. Bundan dolayı burada yapacağımız geliştirmeleri bizim OnionArch.infrastructure içinde gerçekleştirmemiz gerekmekte.  
  
Şimdi;

öncelikle bir klasör açalım ve adını Filters koyalım daha sonra hemen altına bir class oluşturalım ve adını ValidationFilters koyalım.

metin, yazı tipi, çizgi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şimdi bunu IOC içinde tanımlamamız gerekmekte tatbikîde.



Artık sonuçlarımız backend tarafından bize ekteki gibi gelecek.

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

# Pagination

# Şimdi burada sayfalama işlemlerini yapmaya başlayacağız . Burada biz pagination yapımızı kullanırken queryString olarak geliştireceğiz. Referans olarak GetAllProducts’ı seçelim.

# metin, yazı tipi, çizgi, ekran görüntüsü içeren bir resim Açıklama otomatik olarak oluşturuldu

Şimdi burada pagination yapmızı bir şekilde karşılamamız gerekmekte ve bu gelen verileri biz bu fonksiyonun içinde işlememiz gerekmekte mantık olarak.   
Öncelikle bir tane Pagination adında bir class tanımlayalım.  
  
Bu Pagination classımız gelecek olan Request parametrelerini karşılayacak olan bir obje olacağı için bu class yapımızı bizim **Application katmanı** içinde oluşturmamız gerekmekte.  
Şimdi Application katmanı içinde birden fazla request parametresi oluşturmamız gerekebilir bunu göz önüne alarak bir adet **RequestParamaters adında bir klasör oluşturalım** ve bunun altında bu classımızı oluşturalım.  
  
  
metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu  
  
Şimdi bu çalışmadan sonra yapmamız gereken bu uyguladığımız class yapımızı kodlarımıza entegre etmek.   
metin, yazı tipi, çizgi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şimdi burada Query içinden bana pagination diye bir değer geleceğini biz biliyoruz bu değerleri default olarak değerlerini biz 0 ile 5 olarak ayarladık bunu da biliyoruz.Bura sorgulumu **Take()** kullanarak geliştirmemiz gerekiyor.

metin, ekran görüntüsü, çizgi, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şimdiye kadar basit bir şekilde uygulamamızı geliştirdik şimdi birkaç ekstra özellik ekleyerek frontend tarafında da geliştirmeler yapalım ve uygulamamızı biraz daha geliştirelim,  
metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu