* Sanal bir şirket olan Akar yazılım ın tanıtım sayfası, admin paneli, iş başvuruları ve başvuru statülerini canlandırabileceğimiz bir ortam oluşturulmuştur.
* Temel seviyede bakıldığında ilgili database diagram şekilde gözüktüğü gibidir.

metin, ekran görüntüsü, diyagram, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

* Proje temeli yukarıdadır. Kullanıcı ve roller arasında çoğa çok bir ilişki mevcuttur.
* User tablosunda belirli kolonlar var. başvurular ile user lar arasında da çoka çok bir ilişki mevcuttur.
* **Sistemde 2 tip kullanıcım olacak bunlardan birisi Admin role ü için, diğerleri ise member rolü için geçerlidir.**
* **Member role ü olan user lar oluturulan iş ilanına başvurabilecektir.**
* **Basit bir iş başvuru sistemi yapacağız.**
* Proje katmanları ile oluşturulabilir bir vaziyettedir.
* İlk etapta bizler Common katmanı (Proje içerisinde Core katmanı olarak geçer) Costume Status Resultları dönebilmemiz için bir Result sınıfı ve DataResult sınıfı oluşturduk.
* **Çalışığımız şirket adı** + **Proje adı** şeklinde bir solution ismi oluşturuyorduk. **Dll** isimleri ise **şirket adı + proje adı + katman ismi** şeklinde oluşturacağız.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

* Response yapımızı artık Common katmanında ekleyeceğiz. Biz bu response yapımızı Common (shared – core içerisinde yapılandırıyorduk )
* Bir statü enum u, message, Fluent validasyon için ilgili list<ValidatonError> larımızı property olarak ekleyeceğiz. (success, validation Error, NotFound Statüs )

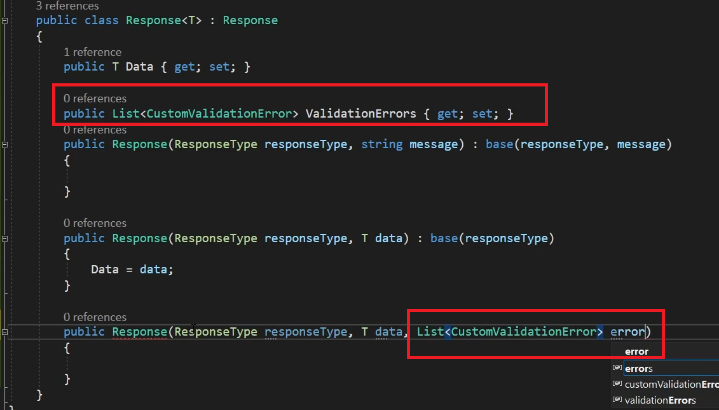
metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

* Buna uygun bir şekilde data Resultarınızı ve Result sınıflarınızı oluşturabilirsiniz. Örnekte herhangi bir response taşımayan bir yapı kurgulanmıştır.
* Data taşımayan response type ta bir validation errorları koymamış olabiliriz. Isteğe bağlı olarak opisyonel olarak konulabilir.
* Provided Services (services tablosunun amacı ) yani tanıtım sayfasında bizlere sağlanacak olan servis – hizmetler (şirket hizmetleridir. )
* Validation error için bir class ve bir Extension methoda ihtiyacımız var. Property name ve description için gereklidir.

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

Açıklama otomatik olarak oluşturuldu



Business e aktarılan data business ten validasyon a takılırsa uı a geri gönderilmesi gerekecektir.

* Entity sınıfını ekleyelim bu aşamada.
* Bizim burada tüm entitylerimiz için id propertysi olmasını isteriz. Find propertysi için. Bu noktada tabi base entity oluşturarak ilgili tüm propertyleri burada tanımlar sonrasında tüm entitylerimizi buradan kalıtım almalıyız.
* Find methodu olacak günün sonunda o yüzden tüm entitylerimizde primary key olarak id bulunmadı geri kalanları manüel indexleyebilirsiniz.
* Bu işlem için Baseentity oluşturacağız.
* BaseEntity için tüm ortak olacak propertyler eklenebilir.

metin, çizgi, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

* Tüm entityler BaseEntity den kalıtım alacaktır.
* Tüm entityler için ilgili düzenlemeleri gerçekleştir.
* AdvertiesmentsUser çoka çok tablosu için bazı düzenlemeler yapılacaktır. Burada bazı properryler üzerinde durmamız gerekecektir.
* Ilgili userlar buraya başvuru yapabilmeleri için yapılmış olan bir tablodur.
* Bu tabloya atılan her bir kayıt ilgili kullanıcı için iş başvurusu manasına gelmektedir.
* Gender tablosunda neler olabilir? erkek, kadın ve belirtmek istemiyor olabilir. Bu tarz işlemlerde sadece içerisinde parameter için data barındıran tablolara bizler lookuptable diyoruz.

metin, diyagram, sayı, numara, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

* LookUpTable da propertye özgü bir iş yapılacaksa eğer (örnek vermek gerekirse tecilli olan asker adayları için ) ilgili tecil tarihi lookup ta değil operasyon tablosunda tutulabilir.

metin, diyagram, sayı, numara, paralel içeren bir resim

Açıklama otomatik olarak oluşturuldu

* Look up tablolarını eklemeye başlayalım.

224

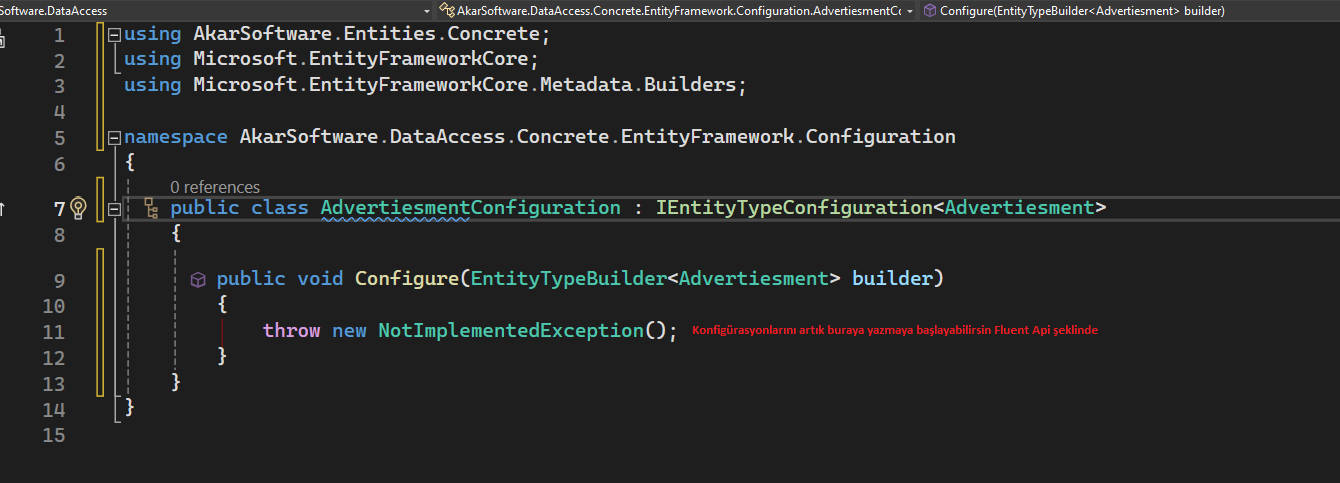
* Entitylerimizi bitirdik DAL I oluşturalım ve bu katmanımızı kodlamaya başlayalım.
* DataAccess layerimizi oluşturalım isimlendirmemize uyalım şirket + proje + katman adı.
* Entity Framework için Sql Server Providerimizi indirebiliriz Ef zaten core katmanı içerisinden otomatik olarak bizlere gelecektir. Aynı şekilde Tools ta gelecektir. (designer I uı tarafına kuracağız)
* Bu etapta bizler veritabanı işlemlerini gerçekleştirebileceğimiz bir adet **DbContext** nesnesi oluşturalım. İlgili Entity Framework kurulumlarımı gerçekleştirelim.
* UOW, DbContext, Configürtaion larımızı tutacağımız yerdir.

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

Açıklama otomatik olarak oluşturuldu

Lookuptable larda veritabanı yansımasına çoğul isim getirmeyebilirsiniz. LU ile isimlendirmenize başlayabilirsiniz.

* DbConbtext ten kalıtım laıyoruz
* Options Constructor içerisinden options alacağız ve base e göndereceğiz. Options lar consturcotr dan alınacak ama ilgili service registier ile birlikte AddDbContext dersek orada options bilgisini alıp context e göndereceğiz.
* Adddbcontext ile opitons bilgisi alınır
* Configürasyonlarınızı Configurations isminde bir klasör içerisinde oluşturabilirsiniz.
* Entitylerimizi DbSet olarak context imiz içerisinde belirttik.
* Entity isimlendirmesinde dikkat etmemiz gereken bir şey var lookuptable lar için çoğul property ismi kullanmak zorunda değilsin fakat kullanabilirsinde ama diğer entitylerde çoğul isimler kulan. Sadece Lookuptable da single isimler kullanabilirsin. (ingilizcedeki çoğul takısından bahsediyorum.)
* LU şeklinde isimlendirmelerinize de başlayabilirsiniz Lookup Table lar için
* Context oluşturulduk ve ilgili entitylerimizi property (**dbset<>)** olarak verdik.
* Configuration larımızı yazalım. **Burada Entity configürasyonlarımızı yazacağız configürasyonlarımız için Fluent Api kullanılacaktır.**
* **Configuration isimli klasörümüzü oluşturarak ilgili configürasyonlarımız geçiyoruz**.
* **IEntityTypeConfiguration<Entity>** şeklinde ilgili configürasyonlarımızı Entity Framewok ile birlikte gelen bir interface tir.
* Aşağıdaki şekilde ilgili entity niz için bir configuration file hazırlayabilirsiniz.
* Entityler



* Bir entityde default bir değer atalamak için entity e = diyerek belirleyebilirsin yada data annotation veya fluent api de ilgili method implemente **HasDefaultValueSql** bunlardan birisidir.

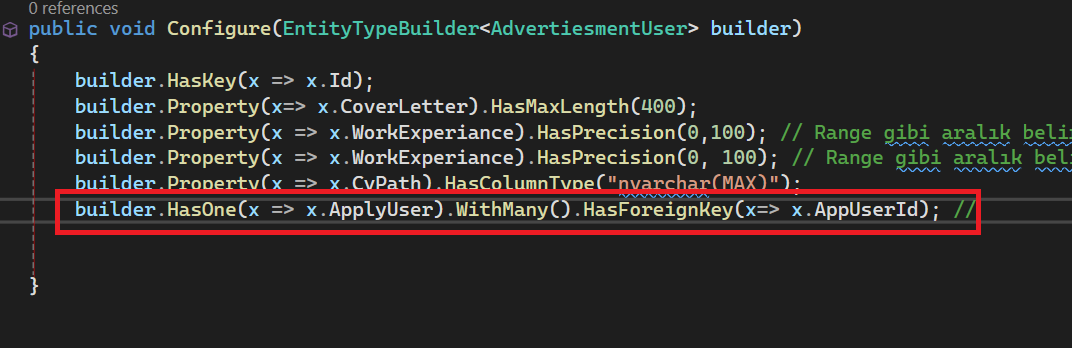
metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

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

Açıklama otomatik olarak oluşturuldu

* Tablolar arası ilişkileri belirlerken (çoka tek ilişki için) bir noktada belirtmeniz yeterlidir. birden çok alanda belirlemeniz karışıklığa yol açabilir. Çalışır ama her değişiklik iki tarafta birden gerçekleştirilmesi gerekecektir.
* Çoka tek ilişki için Tek property olarak geçilen alanlarda (entitylerde) foreign key belirtmeniz yeterlidir. Karışıklığı azaltamk için yoksa karşışıklıda belirleyebilirsiniz. (daha da açıklayıcı bir şeilde belirtmek istersek iki tablo baz alındığında tek entity nin navigation property olarak tutulduğu yerde configurasyonlarını geçmen daha kullanışlıdır. Yani sen bir product tin birden fazla kategorisi varsa kategorideki List<product> ta değil product içerisinde Category propertysinin tanımlanması sebebi ile bu ilişkiyi gidip Product sınıfı içerisinde yani product ta hizmet eden configuration nesnesi içerisinde belirtebilirisin. ) (ilişki sadece bir noktada belirtilmesi yeterli tek li propertylerin olduğu listelerin olmadığı alanlar için configürasyonlarınızı geçebilirsiniz. )
* Core katmanında ben Appuser içerisinde başvurulan işleri property olarak tutmadım. Bire çok ilişki kurabilmek için bir tarafa liste bir tarafa da tek property koyma zorunluluğum bulunmamaktadır. Fluent api içerisinde de ilgili ilişkide Withmany içerisinde ilgili alanı boş bırakabilirim. (ilişkiyi bir noktadan belirtmeniz yeterli tek li propertyler üzerinden ilişkileri belirleyebilirsiniz. )
* User id ve user nesnesi aynı yerde. Bu sebeple ilişkileri kurarken tekli taraftan kurulması daha uygundur.



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

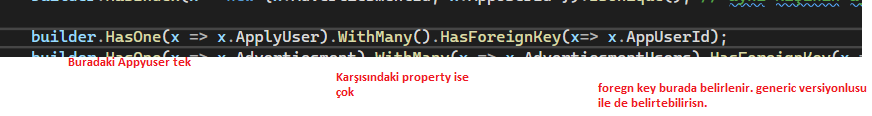
Açıklama otomatik olarak oluşturuldu

* Advertiesment alanına bakarsak ilan ve user a bakarak aynı ilana iki defa başvurunun engellenebilmesi için ikisinde unique index olarak belirlememiz gerekecektir. Unique index üzerinden iş yapacağız.1 user I 1 ilanına başvurmuş ve ikinci defa başvurmasının engellenmesini istiyoruz. Indexlemeyi bu sebeple kullanıyoruz. Fluent api tarafında builder.Hasindex(x=> new {}).IsUnique() şeklinde belirtebiliriz.

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

* Ilişkileri belirtirken bir şey daha belirtmiştik. Navigation prpertyler üzerineden ilişkiler kurulacağından bahsetmiştik. TEKLİ PROPERTYLERİ AL.



* Buradaki ilk olarak seçtiğimiz şuan belirttiğimiz class üzerinde iş yaptığımızdan ötürü bu class içerisindeki tek olark belirtimiş olan property e gidiyoruz. Buraaki apply user tek. Çok olan kim advertisement içerisindeki ise advertiesmentapppusers. Ikisi arasındaki foreign key ise appuser id olarak düşünebilirsiniz.
* Has one da tek has many de çok olan alanı belirtiyoruz.
* Geri kalan tüm nesnelerin için configürasyonlarını ve hasdata ile seed datalarını yaz.
* Ilgili entity configürasyonları tamamlanması ile birlikte bu configürasyonlarını bizlerin EntityFramework içerisinde yer alan DbContexct te **OnModelCreating() method** içerisinde çağırmamız gerekecektir. Çağırma sürecinde bir dikkat edilmesi gerekn süreç şu ya modelBuilder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly()); vasıtası ile birlikte model builder içerisinde dinamik bir şekilde dolması sağlanacak yada modelbuilder üzerinden tek tek ilgili configürasyon sınıfları tek tek modelBuilder.ApplyConfiguration(new AppUserConfiguration()); methodu gibi configurasyonları uygulamamız gerekecektir.(context içerisine configürasyonları uygulamak için OnMoodelCreating override edilecektir. )

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

Açıklama otomatik olarak oluşturuldu

* Bu başlık altında repositoryler oluşturulacaktır. Repositoryler için DataAcces içerisinde interface be Reposityory isminde sınıflar oluşturulacaktır.
* Ilk interface imizi generic bir şekilde oluşturalım. Biz generic reposityory patternini birçok projede kullanabilme ihtimaline karşılık Concrete içerisinde kurguladık.
* Repository ile birlikte bizler UOW kullanacağız. Ilgili işlemlerimizi gerçekleştirirken constructor içerisinde dbcontext alacağız. UnitOfWorkten gelecek olan Constructor ile bilrikte bizler dbcontext alacağız. Yani constructor da dependency injection ile ayağa kaldırmayacağız.
* Çeşitli methodlarımızı kullanarak ilgili methodlarımızı yazalım. Bir method asenkronik bir method ise isminde best practice açısından async takısı koyunuz. Normal şartlar altında bu bir enum ile birlikte alınabilirdi.
* Tüm verileri getirme sürecinde şu operasyonları yapabiliriz. Tüm veriyi getirebiliriz, tüm veriyi sıralayarak getirme, tüm veriyi filtreleyerek getirmek, opsiyonel olarak Tracking için asnortracking, tracking işlemleri için bir parameter ekleyebilirsin. Bulk – in operasyon yapmayacaksanız asnotracking ile işlemleri gerçekleştiriniz.

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

* Sıralama işlemleri için order by ve orderByDescanding methodları var bu methodlar için ise bizlerin order by ve order by ascending için bir func delegate atamamız gerekecektir. T ve Tkey olarak func delegate almaktadır. T entity ve Tkey im ise t ye ai tolan propertydir.

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

Açıklama otomatik olarak oluşturuldu

* Geri kalan repository kodlarını aşağıdaki şekilde görebilirsininz.

metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

* Find methodu, update delete methodları nı da projeye ekleyelim.

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

Açıklama otomatik olarak oluşturuldu

* Id ile değilde başka bir şey ile arayacağım veya asnotracking olsun istemiyorum . bu sebepten mütevellit expression geçilmiş bir versiyonun da eklenmesi gerektiği kanaatindeyim.

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

Açıklama otomatik olarak oluşturuldu

* IQueryable dönen bir methodumuz daha var onuda oluşturalım. Query şeklinde business te kullanabilmemiz için ilgili işlemi gerçekleştiriyoruz.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

* Remove da save add da update de savechanges kullanmayacağız. Çünkü bunu unit of work ile ele almaya çalışıyoruz.

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

Açıklama otomatik olarak oluşturuldu

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

Açıklama otomatik olarak oluşturuldu

Property bazında update yaptırabileceğimiz bir generic method yazabiliriz. Tüm propertyler güncellenmeden bu şekilde ilgili güncellemeleri gerçekleştirebiliriz.

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

Açıklama otomatik olarak oluşturuldu

Ilgili interface implementasyonunu gerçekleştirelim.

metin, elektronik donanım, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

* **DataAccess** katmanı için **unitOfWork** patternini implemente edelim. UOW olarak ta isimlendirilebilmektedir. Ana amaç ilgili repositoryleri tek bir context in ilgili request e gittiğinden emin olmaktır.
* Hangi context ile çalışıtorsak bizler ilgili context I ctor ile birlikte (dependency injection vasıtası ile ele alıyoruz) ilgili repositorylerin consturctur üne çağırarak basıyoruz. Dependency injection ile gelen context I yolluyorum. UnitOFWork te bir Dependency İnjection gerçekleşiyor. Bu DI ele alınan örnek ilgili context e setlendikten sonra methodlar yada propertyler üzerinden göndererek bu şekidle repsoiyroylerimiz ayağa kaldırıyoruz. Bu yüzden services kısmında bizler bağımlılıkları repository bazında kurmayacağız. Sadece unit of work için ilgili bağımlılığı eklememiz yetecektir.
* Bu aşamada artık bizler DataAccess içerisindeki düzenlemelerimizi tamamladık . Busiiness katmanını oluşturmaya başlayabiliriz. Dependencylerimiz kaydedeceğimiz bir İOC yapısı implemente etmek istiyorum. Bu aşamada Business katmanı ( bazı yerlerde manager olarak adlandırılabilir) içerisinde Abstract ve Concrete klasörlerini ekledikten sonra Concrete içerisinde DependencyResolves isminde bir klasör kurarak ilgili klasör içerisinde kullanılan teknolojiye göre (MicrosoftIOC, AutoFac …) klasörlememizi yaparak ilgili extension classımızı yazıp içerisinde ilgili bağımlılıkları geçebiliriz.
* Business I oluşturalım, Business üzerinden dependency resolves oluşturalım, ve içerisine ilgili extensions u yazalım ve database I migrate edip ayağa kaldırmaya çalışalım.
* Şirket adı + proje + katman adı eklenerek ilgili manager katmanı (business katmanı) oluşturuldu.
* Dependency resolvers içerisinde birden fazla teknoloji ve kütüphane ile birlikte İOC yapısını başka kütüphnabe ve teknoloji ile yapma durumu için klasörlenir.
* Extension method yazarken **sınıf static olur**, **methodda statik** olur, **parameter olarak ise extend edeceğimiz nesneyi this** ile belirtmemiz gerekecektir. IServiceCollection için ilgili paketin Business içerisinde kurulması gerekmektedir. **Microsoft.Extentions.DependencyInjection.Abstractions**

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

Açıklama otomatik olarak oluşturuldu

Görüldüğü gibi ilgili business katmanı artık kurulmuş bir vaziyettedir.

* **IConfiguration** interface I vasıtası ile içerisinden **GetConnectionString** method ile ilgili connection stringine erişim sağlayabilirsiniz.

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

Açıklama otomatik olarak oluşturuldu

* Bu oluşturduğumuz method artık bizler ilgili uı içerisindeki service tarafından çağırmamız gerekecektir.
* UI katmanı içerisinden migration oluşturabilmemiz için Entity Framework Desing paketinin kurulması gerekecektir.

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

Açıklama otomatik olarak oluşturuldu

* Bu aşamada package manager console seçilerek ilgili migrationlarınızı uygulayabilirsiniz.

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

Açıklama otomatik olarak oluşturuldu

* Migrationslarımız artık çalışır bir vaziyette.

(239)

* AutoMapper eklemesini geçekleştirelim.

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

Açıklama otomatik olarak oluşturuldu

Bir adet MApperConfiguration nesnesini ekliyorduk startup tarafına. Mapperconfiguration içerisie de delegateler halinde mapping profillerimizi ekkleyebiliyoruz (en üstteki kütüphane için ) Mapperconfiguration üzerinden CreateMapper diyerek bir bu bize bir IMapper çevirecektir.

metin, ekran görüntüsü, yazı tipi içeren bir resim

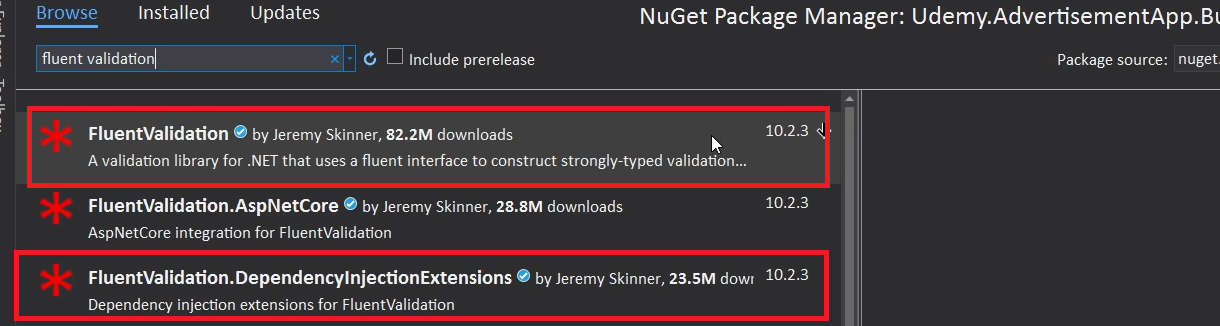
Açıklama otomatik olarak oluşturuldu

Mapper dediğimiz şey son etapta aslında bir IMapper. Bu işlemi ioc içerisinde gerçekleştirin.

Dto katmanını oluşturalım. Dto esasında bir class librarydir. Dto business tarafından görülecektir. Dto katmanı içerisinde IDto interface I oluşturuluyor. Interface I DTO sınıflarını ayırabilmemiz için kullanmaktayzız. Sen bir Dto sun demek istiyoruz. Dto aynı zamanda biz eğer generic service yazmaya kalkarsak kullanacağımız nesnelerin birer Dto olduğunu göstermek ve sınırlandırmak maksadı ile eklenmektedir.

Create DTO larda id ve CreatedDate almıyoruz sonuçta entity Framework kendisi otomatik olarka ilgili süreci gerçekleştirmektedir.

Dto lara validasyonlar geçebilmek için FluentValidaton u eklememiz gerekecektir.



Ilgili validasyonlarımızı geçebilmemiz için rule setimizi geçebileceğimiz bir adet AbstractValidator sınıfı oluşturmamız gerekecektir. Validation rule larımızı yazacağız bu sayede.