Deniz Yıldırım

50160076047

Bilgisayar Mühendisliği (İÖ)

Online Çiçek Sepeti

Online Çiçek Sepeti Projesi Müşterilerin kategorilerine göre ürünleri görebileceği ve ona göre sipariş vereceği bir web sitesidir.

Proje menümde kategoriler ve ürünler olmak üzere iki link bulunmaktadır. Ürünler kısmında ürünümün listeleri (adı,fiyatı,stok sayısı ve kategori bilgileri) mevcuttur.

Kategori kısmında iste ürünlerimin hangi kategorilere ait olması gerektiğini (Creat işleminde) belirten bir View sayfası mevcuttur.

**Çiçek Sepeti web projemde kullandığım teknikler:**

**Veritabını first-code tekniği ile oluşturuldu. Bu sayede entitylerimizi ve modellerimizi yönetmek daha kolay oldu.**

Öncelikle asp.net core bizlere mimari katman yapısını zorunlu kıldığı için projemin içine oluşturuğum \*AppCore ( YAPIDAN BAĞIMSIZ HER PROJENİN BASE YAPISI )

\*Business

\*DataAccess

\*Entities

\*OnlineCicekSepeti ( UYGULAMA KATMANI )

Bulunmaktadır.

**1) AppCore Katmanı Nedir ?**

AppCore katmanı projenin yapısından bağımsız her projenin daha esnek kullanımı tek bir yerden servisleri, sorguların dönüş tiplerini, aldığımız error,exception ve success durumlarını yönetebildiğimiz katmandır.

Her servisin kullanacağı IServices adında crud işlemlerinin yer aldığı interface bir sınıf bulunmaktadır.

Normalde asp.net core ile DbContext üzerinden veritabanı işlemleri yapılırken artık entityler üzerinden RepositoryBase yardımıyla ve Dependency Injection tekniği ile her servise enjekte edilen yapı kullanılarak veritabanı işlemleri oluşturulan nesne üzerinden yapılacaktır.

RepositoryBase sınıfı bir abstract sınıf olmalıdır. Çünkü mimari yapılarda ve Solid prensipleri gereği sınıfın base soyut yapıda olması gerekmektedir. Constructor üzerinden Dependency Injection tekniği ile kullanacağımız bu sınıfın new lenmemesi gerekmektedir.

RepositoryBase sınıfını özelleştirme ihtiyaçları doğabilir mesela projemde kullandığım bu sınıf sadece generic yapıda olduğu TEntity alır ve RecordBase sınıfını implement eden sınıflar kullanabilir.

RecordBase sınıfı ise tüm entityler için ortak bir yapıdır. Tüm entity ve model sınıfları bu sınıftan kalıtım alacaklardır.

Bu katmanda son olarak bir veritabanı işleminde karşımıza çıkabilecek 3 durum olduğu için onları yöneteceğimiz bir yapı olan Bases yani dönüş değerlerimiz olan Result sınıflarım mevcut. Burada Error, Exception ve Success durumları mevcut.

**2) Business Katmanı Nedir ?**

Bir web projesinde her katman çok önemlidir. Fakat business yani iş katmanı en önemli yapıdır. Burada servislerimiz bulunmaktadır. Yani sorguların oluştuğu veritabanı işlemlerinin yapıldığı yer.

**ÖNEMLİ: Her servisin bir interface tipinde base bir yapısı bulunmaktadır. Peki neden ? Asp.net core IoC Container yapısı gereği ve AppCore projemizde her servis için tanımladığımız ve bu sınıftan implement olan yapılar birer generic tipte model almak zorundadırlar. Eğer biz base sınıf oluşturmadan direk olarak servis sınıflarına bu yapıyı implement edersek projemizde bulunan Startup dosyasında IoC Container AddScoped() yapısını set edememekteyiz. Asp.net core buna izin vermiyor. Araya bir base sınıf koyarak ve IoC Container AddScoped() yapısını kullanarak iki parametre yollamaktayız. Base yapıyı gördüğün yerde asıl bizim servis sınıflarımızdan bir nesne oluştur diyebiliriz.**

**AppCore projesinde oluşturduğumuz IServices yani bütün servislerimizin inteface base yapısı generic tipte implement edilmesi gerekmektedir. Bu ne işe yara ?**

**Bu yapı sayesinde her servis kullandığımız hangi modeli kullanacağını bilmektedir. Veritabanı işlemlerini entitylerden ayıran yapıdır.**

**Entity yapısı mevcutken neden tekrar modeller üzerinden işlemler yapıyoruz ve nasıl yapıyoruz ?**

**Bazı durumlarda entity ler ihtiyaçlarımızı karşılamıyor. Örneğin View sayfasına göndereceğimiz modellerimizi kimi zaman özelleştiremliyiz. Örneğin projemde kullandığım kategoriler listesinde hangi kategoride kaç adet ürün bulunduğu bilgisini modellerimizde özelleştirerek view yapısına model üzerinden sunduk.**

**Entity örneğin (Product) dönüştürdüğümüz (ProductModel) sınıfı ilişkili yapıları almadan aynı verileri kullanmak zorundadır. Amacımız da zaten budur.**

**Servislerimizde GetEntityQuery();**

**Yapısını kullanarak her Entity için yeni bir ProductModel oluşturarak objelerimizin atama işlemlerini burada yapıyoruz ve kullanıyoruz.**

**3) DataAccess Katmanı Nedir ?**

DataAccess katmanı servislerde kullanacağımız ve AppCore katmanında tanımladığımız veritabanı işlemlerinin bulunduğu metodları kullanacağımız nesneyi oluşturan yapıdır.

Burada aynı şekilde tekrar bir Base Repository yapı ardından entity lerimize göre normal bir Repository sınıf tanımlanmalıdır. Mantığı services yapısı yani IoC Container yapısı gibidir.

**ÖNEMLİ: Fakat bu yapı servislerden farkı generic tip olarak entity almaktadır. Zaten mantıken entity üzerinden servislerimiz de ihtiyaç gereği olarak modele çeviriyoruz. Öncelikle bize entity ve bunlar kullanacağımız bir nesne gerekmektedir. Servislerimizde kullanacağımız veritabanı işlemlerini bu sınıf yardımıyle yapılandırıcı üzerinden enjekte ederek nesnemizi oluşturarak yapıyoruz. Bu sınıfın tek bir amacı vardır. Oda AppCore da bulunan bütün veritabanı işlemlerini set edebileceğimiz yani DbContext \_db nesnesi yerine ReposityoryBase yapısını kullanmasıdır, implement etmesidir.**

Ve son olarak bu katmanda Context sınıfı mevcuttur. Entity üzerinden first-code ile veritabanı tablo ekleme işlemeleri yapılmaktadır.

**4) Entities Katmanı Nedir ?**

Entities katmanı veritabanı tablolarına karşılık gelecek şekilde hazırlanan içlerinde zorunlu olarak Id ve Guid verilerinin bulunduğu sınıflar topluluğudur. Bu sınıfların hepsi zorunlu olarak RecordBase sınıfından türemek zorundadırlar.

**5) Uygulama Katmanı Nedir ?**

Uygulama katmanında projeyle ilgili .net core teknolojisinin hazır olarak projeyi oluşturduktan sonra bizlere sunduğu yapıdır.

Burada çok önemli olan Control sınıfları View yapısı vs. bulunmaktadır.

Biraz içeriğinden bahsetmek gerekirse wwwroot klasöründe projemizde kullanacağımız bootstrap,jquery,js ,css hatta product listesi için kullandığım datatable kütüphaneleri bulunmaktadır.

Appsettings.json içerisinde veritabanı(SQL Server) adresi bulunmaktadır.

Vs.

Burada Controller sınıflarımız olmazsa olmaz yapılardır. Servislerin oluşturuğu query (sorguları) dönüşleri verileri modelleri gibi yapıların yönetildiği yerdir.

Enjeksiyon işlemi yardımıyla ihtiyaçlarımıza göre kullanacağımız servislerin;

**Önemli: Servislerin çünkü burada servislerde Repository sınıfı üzerinden işlem yaparken burada artık servislerimizi kullanıyoruz (Controllers) çünkü Repository yapılarımız generic olarak entity sınıflarımızı alıyor ve doğal olarak entity üzerinden işlemler yapıyordu bunları servisler bir query oluşturup artık controller e yollayacak ve burada listelenip modele dönüştürüldükten sonra view’a sunulacaktır. ( View a sunulan veriler model cinsinden olmalıdır. Tek sebebi ihtiyaç duyulması daha esnek kullanılabilir yapıda olması ).**

**Controller hazır olarak oluşturulup içeriğini kendimiz doldurduk.**

**View klasöründe ise webde göstereceğimiz html yapılı sayfalarımız bulunmaktadır.**

**Aynı şekilde bu yapıda hazır olarak ( Önce Controller ardından View ) oluşturulur.**

**Default olarak entity leri kullanan bu yapıya model olarak modellerimizi kullanmamız gerektiğini belirtmeliyiz.**

**BİR SORGU NASIL ÇALIŞIR VE UYGULAMA KATMANINA SUNULUR ? (ÖRNEK)**

**Mantığı ilk olarak AppCore katmanı ile başlamaktadır. Burada oluşturduğumuz soyut sınıflar yardımıyla entity, servislerimiz ardından control sınıfları ve view yapısı oluşturulur.**

**İlk olarak bir entity Add işlemi yapmak istediğimizde öncelikle entity olarak bu sınıfa aralarında ilişki de olabilir ihtiyaç duymaktayız. Bu entity lerimizi kullanacağımız veritabanı işlemlerinde verileri dolduracağımız , kullanacağımız RepositoryBase sınıfımız bulunmaktadır. Bu sınıfı kullanabilmek için Servislere ihtiyaç duyarız.**

**Bir servis sınıfı oluşturulduktan sonra IService adında sınıftan kalıtım almalıdır. İçinde tüm servislerde kullanabileceğimiz CRUD işlemleri mevcuttur.**

**Burada Add metoduda vardır.**

**Servislerimiz model kullandığı için paramtere olarak buraya eklenecek nesneyi alır.**

**(Biz View kısmında kullandığımız veriler model tipinde olduğu için ilk veri bize model olarak gelir) bu yüzden model parametresini alarak ;**

**RepositoryBase içerisindeki kodları çalıştırmak için üretilen nesneden bir sorgu yazılmalıdır.**

**Diyebiliriz ki;**

if (\_productRepository.GetEntityQuery().Any(p => p.Name.ToUpper() == model.Name.ToUpper().Trim()))

**Eğer bize modelden gelen objenin ismi ile veritabanında bulunan (entity) e ait varolan bir objenin ismi aynı ise ( küçük büyük karakter ve boşluk problem yaratmaması için ToUpper ve Trim yardımıyla ) bize burada Error dön diyebiliriz.**

**Gerekli şartlar sa ğlandıysa ;**

**var entity = new Product()**

**{**

**CategoryId = model.CategoryId,**

**Description = model.Description?.Trim(),**

**Name = model.Name.Trim(),**

**StockAmount = model.StockAmount,**

**UnitPrice = model.UnitPrice**

**};**

**\_productRepository.Add(entity);**

**return new SuccesResult();**

**}**

**Bu kodu çalıştırabiliriz. Açıklaması; entity adında bir Ürünler adında obje oluşturulur.**

**Nasıl ki servislerde kullandığımız entity model dönüşümü yaptıysak aynı işlemin tersini burada model entity dönüşümü olarak yapıyoruz.**

**Çünkü modelleri makine bilmiyor ve veritabanına ekleyemez sadece entitylerden haberdar olduğundan burada modeldeki verileri entity sınıfında bulunan ürünlere set ederek**

**ProductRepositoryBase sınıfını implement ederek oluşturduğumuz sorgu sayesinde**

**Add(); metodunu çağrarak set ediyoruz ve kaydediyoruz.**

**Dönüş tipi olarak Succes dönüyoruz.**

**Controller sınıfında ise Succes durumunu kontrol ederek , gerekse Valid mi yani modellere eklediğimiz Data annotations kontrolünden sonra view yapısına return ederek modellerimizi orada gösterebiliriz.**

**PROJE TEKNİK ÖZETİ**

**Yapıyı kısaca özetlersek müşteriye sunacağımız verileri veya alacağımız, Yetkili kişinin gerekli ürünleri ekleme, silme, güncelleme gibi işlemleri yapabileceği yapılar oluşturuldu. Mimari katmanın kullanılması ile başka bir projede de daha kullanışlı gerekli servis model işlemlerinde hatta result sonuçlarını kendimiz enum ile belirleyerek hızlı, değiştirebilir bir yapı tasarlandı.**

**Bazı özellikler düşünülerek geliştirildi ( Bir kategoriye ait ürün var ise ki bu one to many ilişki o kategoriyi kullanıcının silmesi engellendi. Aksi takdirde yapılan bir yanlışlık o kategoriye ait bütün ürünlerin silinmesine yol açabilir. Bunu Sql Server tablolar ilişkisinde No Action yardımı ile yapabildik )**

**Base sınıflar ürettik. Somut sınıfta kullanacağımız abstract ve inteface ile implement ederek kullandık.**

**IoC Container yapısı gereği araya base sınıflar eklemek zorunda kaldık çünkü RepositoryBase ve IService sınıfları generic yapıda olduğu için somut sınıflarda bunları direkt olarak tanımlayamazdık.**

**Gerekli model dönüşümlerini servislerimizde yaptıktan sonra verileri web katmanında göstermiş ve ürün ve kategori ilişkisinin bütün crud metodları sağlıkla çalışmaktadır.**