



PROJE

Haftalık Rapor – 18.12.2020

18 ARALIK 2020
KIRIKKALE ÜNİVERSİTESİ – BİLGİSAYAR MÜHENDİSLİĞİ İÖ
AHMET Mungan – 160255081

ABSTRACT VE INTERFACE'LERİN ENTEGRE EDİLMESİ

Tablolarda değişiklikler yapıldı. Entity Framework üzerindeki sorunları giderilip bazı düzenlemelerde bulunuldu. Bu düzenlemeler elbette projenin teslim edilirken yapılması gereken düzenlemeleri olmamalı. Fakat bu düzenleme kısmında raporu daha rahat ve anlaşılır bir şekilde tutabilmek için bazı söylemler ingilizce yapıp, katmanların ismi saf katman isimleri olarak değiştirildi. Ayrıca veritabanı bağlantılarında tabloların kullanılan ORM teknolojileri daha rahat algılsın diye ingilizce ve kısa söylemler olarak değiştirildi. Bu şekilde devam edilmesi hem rapor tutulmasını kolaylaştıracaktır hem de ingilizceye dönüş için rahatlık sağlayacaktır.

Burada öğrenilen en mühim şey aslında bir kural olmamasına rağmen uygulanması büyük avantajlar sağlayan bir yapı. Eğer bir class tek başına onu implemente eden herhangi bir base'i, abstract'ı veya interface'i yoksa, bu kodlama kurumsal projeler genelinde Code Smell olarak adlandırılır. Tabiri caizse kötü kokan kod denebilir. Her uygulamanın bir implementasyonu olacak diye kural yok fakat başka bir zaman diliminde bu kodlamaya güncelleme yapılmak istendiği zaman ("SOLID" prensiplerinin "O" harfi) var olan kodları değiştirmeden istenen yenilikler getirilebiliyor olması gerekir. Ufak çapta değişiklikler yapılabilmesi durumunda tüm kodlamayı baştan sonra değiştirmemek için bu tekniği uygulamanın elzem olduğu tespit edildi.

"DagitimBolgesi.dbo" isimli tablonun adı "Areas.dbo" olarak; "DagitimAboneGruplari.dbo" isimli tablonun adı ise "Subscriber.dbo" olarak değiştirildi. Dolayısıyla oluşturulan context'leri de bu yapılmış olan değişikliklere entegre edildi.

VS Code 1

```
public interface IAreaDal
{
    List<Area> GetAll();
    Area Get(int code);
    void Add(Area area);
    void Update(Area area);
    void Delete(Area area);
}
```

VS Code 1’de AreaDal için bir interface yazıldı. VS Code 1’de temel operasyonlar yazılmaya çalışıldı. Genel olarak hangi tablo olursa olsun, yapılacak tüm işlemler aynı olacağı için daha esnek bir alan yaratıldı.

VS Code 2

```
public class EfAreaDal : IAreaDal
```

VS Code 2’de öncelikle Entity Framework için, daha sonra sadece çalışıp çalışmadığını test etmek için Nhibernate için olan interface’ine implemente edildi.

VS Code 3

```
public class NhAreaDal : IAreaDal
```

VS Code 3’te interface’i ekleyip denedim. Herhangi bir problem ile karşılaşılma durumu uygulamanın çalışması konusunda. AreaManager kısmında ise Dependency Injection desenini kullanabilmek açısından bu tanımlanan interface’i kullanılması çalışıldı.

VS Code 4

```
public class AreaManager
{
    IAreaDal _areaDal;
    public AreaManager(IAreaDal areaDal)
    {
        _areaDal = areaDal;
    }
    public List<Area> GetAll()
    {
        return _areaDal.GetAll();
    }
}
```

VS Code 4’te Entity Framework veya Nhibernate ile ilgili söylem geçirmeden sunum katmanında AreaManager çağırıldığı zaman constructor olarak belirlenen ve girilmesi zorunlu olan bir veri erişim tipi belirlenmesi istenildi. Çünkü olabildiğince bağımsız bir yapı kullanmaya çalışıldı. Pek tabi burada AreaManager class’ının çıplak kaldığını(interface’inin olmadığını çıplak kaldığı ile kastediliyor) biliniyor fakat ona daha sonra başka bir teknikle yaklaşılacak. Sunum katmanında iş katmanını çağırırken constructor için bir ORM teknolojisi seçilmesini istiyor.

VS Code 5

```
private void Form1_Load(object sender, EventArgs e)
{
    AreaManager areaManager = new AreaManager(new EfAreaDal());
    dgwAreas.DataSource = areaManager.GetAll();
}
```

Doğru şekilde doldurulduğu zaman VS Code 5'teki gibi bir değişikliğe gidilip yapı güncellendi. Daha refactor süreci devam ediyor, bu yapı da elbette değişecek. Refactor için yapılması en elzem güncellemelerden bir diğeri ise AreaManager'ın daha bağımsız bir hal almasıdır. Çünkü SubscriberManager call edildiği zaman aynı operasyonlar kullanılacağı için burada bir interface daha gereklidir.

VS Code 6

```
public interface IAreaService
{
    List<Area> GetAll();
}
```

Şuan için tek bir operasyon gerçekleştirildiği için uygun kodlaması VS Code 6'da gösterilmiştir.

VS Code 7

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        _areaService = new AreaManager(new EfAreaDal());
    }
    IAreaService _areaService;
    private void Form1_Load(object sender, EventArgs e)
    {
        dgwAreas.DataSource = _areaService.GetAll();
    }
}
```

Ardından, sunum katmanında VS Code 7'deki değişiklikle biraz daha iyileştirebildiği söylenebilir. Fakat burada bir sorun olduğunu düşünerek: IAreaDal interface'inin içerisindeki operasyonları diğer tablolar için de aynısının yapılacağı için burada araştırmalar sonucu başka bir yapı kullanmak mümkündür. Bu yapı Repository adında

bir yapı ve LINQ'ler ile filtrelemeyi de daha dinamik bir hale getirebiliyor. Tüm tabloların kullanacağı bir repository yazılacağı için dosyalama konusunda üst dizinde olması kapsayıcı özelliğinden ötürü veri erişim katmanının concrete klasörünün içine değil de, yine abstract klasörünün içine yazıldı. Burada Area ya da başka bir nesne yerine daha generic bir tip kullanıldı. Ayrıca instance isimlerini de area gibi özel adlar yerine daha kapsayıcı olarak entity olarak belirlendi.

VS Code 8

```
public interface IEntityRepository<T>
{
    List<T> GetAll();
    T Get(int code);
    void Add(T entity);
    void Update(T entity);
    void Delete(T entity);
}
```

Ve şu hale getirildi, Bkz. VS Code 8.

VS Code 9

```
public interface IAreaDal : IEntityRepository<Area>
{
}
```

VS Code 9'da görüldüğü üzere, IAreaDal özelinde bu değişiklik bu şekilde kodlandı. Fakat repository kısmı daha verimli kullanabilir. Filtrelemeyi yapıp daha generic bir hale getirilmeye çalışıldı.

VS Code 10

```
public interface IEntityRepository<T>
{
    List<T> GetAll(Expression<Func<T, bool>> filter = null);
    T Get(Expression<Func<T, bool>> filter);
    void Add(T entity);
    void Update(T entity);
    void Delete(T entity);
}
```

VS Code 10'da LINQ Expression'ları kullanarak temel delegate'lerden olan ve dönüş tipi sağlayan Func hazır delegesini kullanarak boolean tipinde bir dönüş almak için bu şekilde bir düzenlemeye gidildi. Burada GetAll() methodu ile istenirse bu operasyon ile de bir filtreleme uygulayabilecek. Herhangi bir filtreleme işlemi olmaması durumunda null değerini alıp klasik GetAll() methodunu uygulanacak. Fakat

muhtemelen bir filtreleme istenen Get() methodunda ise bu filtreleme varsayılan olarak null olmayacak, illa ki bir filtreleme istenecek. Ayrıca Area gibi class'ları çıplak bırakmamak ve repository'de developer'ın elini kuvvetlendirmek amacıyla boş da olsa bir IEntity interface'i yazıldı ve Area üzerinde implemente edildi.

VS Code 11

```
public class Area : IEntity { ... }
```

VS Code 11'de bu interface'in implemantasyonu görüldüğü gibi yapıldı. Bu sayede repository interface'i generic kısıtlar koyabilir hale geldi. Tabi LINQ yapısının sunduğu hizmetlerin bu konudaki katkısı sayesinde gerçekleşti. Burada blok içindeki noktalar operasyonları belirtiyor fakat her seferinde değişmemiş olan operasyonu bu rapor özelinde yazılması gerekli bulunmadı.

VS Code 12

```
public interface IRepository<T> where T:class, IEntity, new() { ... }
```

VS Code 12'deki generic kısıt T varlığının referans edilebilen bir türden olduğunu gösteriyor. IEntity interface'inden implemente ediliyor ve son olarak da bu nesne new'lenebilir bir yapıda olması gerektiği kısıtlanmış oldu. Bu yazılmış olan repository için bir base repository yazılması yararlı olacağı yapılan araştırmalar neticesinde karara bağlanmıştır. Base'den kasıt burada inherit edilebilecek bir yapı. Bu base yapısı farklılık içereceği gibi ayrıca yine developer'ın elini kuvvetlendireceği kesin.

VS Code 13

```
public class EfRepositoryBase<TEntity, TContext> : IRepository<TEntity>  
    where TEntity:class, IEntity, new()  
    where TContext:DbContext, new() { ... }
```

VS Code 13'te Entity Framework'ün Context'ini de içeren generic bir yapı oluşturması hedeflendi. İçerisindeki operasyonlar da generic bir hal aldı. Gerisi biraz çocuk oyuncağından farksız bir hal aldı.

VS Code 14

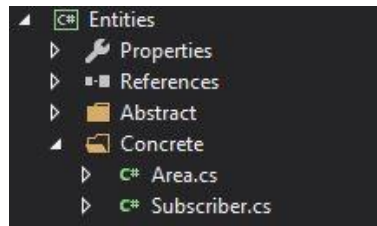
```
public class EfRepositoryBase<TEntity, TContext> :  
    IRepository<TEntity>
```

```

        where TEntity:class, IEntity, new()
        where TContext:DbContext, new()
    {
        public void Add(TEntity entity)
        {
            using (TContext context = new TContext())
            {
                var addedEntity = context.Entry(entity);
                addedEntity.State = EntityState.Added;
                context.SaveChanges();
            }
        }
        public void Delete(TEntity entity)
        {
            using (TContext context = new TContext())
            {
                var deletedEntity = context.Entry(entity);
                deletedEntity.State = EntityState.Deleted;
                context.SaveChanges();
            }
        }
        public TEntity Get(Expression<Func<TEntity, bool>> filter)
        {
            using (TContext context = new TContext())
            {
                return context.Set<TEntity>().SingleOrDefault
                    (filter);
            }
        }
        public List<TEntity> GetAll(Expression<Func<TEntity, bool>> filter =
null)
        {
            using (TContext context = new TContext())
            {
                return filter == null ?
                    context.Set<TEntity>().ToList() :
                    context.Set<TEntity>().Where(filter).ToList();
            }
        }
        public void Update(TEntity entity)
        {
            using (TContext context = new TContext())
            {
                var updatedEntity = context.Entry(entity);
                updatedEntity.State = EntityState.Modified;
                context.SaveChanges();
            }
        }
    }
}

```

VS Code 14'teki operasyonlar da base haline getirildi. Artık diğer tablolar eklenirken developer'ın işi çok daha kolay ve hızlı gerçekleşecek. Aynı zamanda bu tablo üzerindeki operasyonları yönetmek de daha kolay bir hale gelmiş olacak. Örnek olarak "Subscribers.dbo" tablosu eklenmek istediğinde:

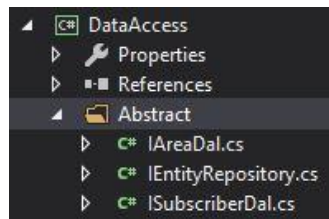


Folder 1

VS Code 15

```
public class Subscriber : IEntity
{
    public int Id { get; set; }
    public int SubscriberCode { get; set; }
    public int AreaCode { get; set; }
    public string Name { get; set; }
    public int Year { get; set; }
}
```

Folder 1'deki dizin yoluna; VS Code 15'te bir class oluşturulup, bu class veritabanı bilgileriyle match edildi.

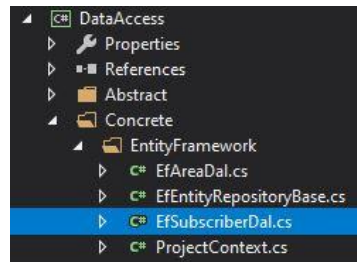


Folder 2

VS Code 16

```
public interface ISubscriberDal : IEntityRepository
<Subscriber>
{
}
```

Folder 2'deki dizin yoluna; VS Code 16'daki Dal kısmı yazıldı.

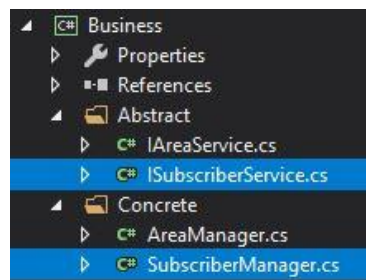


Folder 3

VS Code 17

```
public class EfSubscriberDal : EfEntityRepositoryBase
<Subscriber, ProjectContext>, ISubscriberDal
{
}
```

Folder 3'deki dizin yoluna; VS Code 17'deki kod yazılıp veri erişim katmanındaki iş bitmiş oldu.



Folder 4

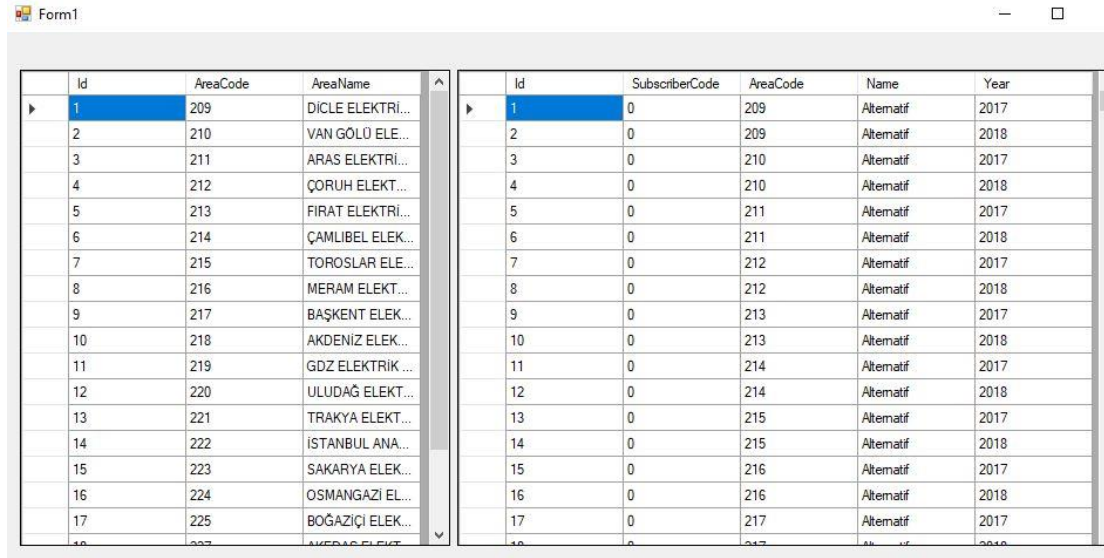
Folder 4'te iş katmanında manager class'ı ve interface'i tanımlandı.

VS Code 18

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        _areaService = new AreaManager(new EfAreaDal());
        _subscriberService = new SubscriberManager(new
EfSubscriberDal());
    }
    ISubscriberService _subscriberService;
    IAreaService _areaService;
    private void Form1_Load(object sender, EventArgs e)
    {
        dgwAreas.DataSource = _areaService.GetAll();
        dgwSubscribers.DataSource = _subscriberService.GetAll();
    }
}
```

```
}  
}
```

VS Code 18’de sunum katmanı kodlandıktan sonra



Id	AreaCode	AreaName
1	209	DICLE ELEKTRİ...
2	210	VAN GÖLÜ ELE...
3	211	ARAS ELEKTRİ...
4	212	ÇORUH ELEKT...
5	213	FIRAT ELEKTRİ...
6	214	ÇAMLIBEL ELEK...
7	215	TOROSLAR ELE...
8	216	MERAM ELEKT...
9	217	BAŞKENT ELEK...
10	218	AKDENİZ ELEK...
11	219	GDZ ELEKTRİK ...
12	220	ULUDAĞ ELEKT...
13	221	TRAKYA ELEKT...
14	222	İSTANBUL ANA...
15	223	SAKARYA ELEK...
16	224	OSMANGAZI EL...
17	225	BOĞAZIÇI ELEK...

Id	SubscriberCode	AreaCode	Name	Year
1	0	209	Alternatif	2017
2	0	209	Alternatif	2018
3	0	210	Alternatif	2017
4	0	210	Alternatif	2018
5	0	211	Alternatif	2017
6	0	211	Alternatif	2018
7	0	212	Alternatif	2017
8	0	212	Alternatif	2018
9	0	213	Alternatif	2017
10	0	213	Alternatif	2018
11	0	214	Alternatif	2017
12	0	214	Alternatif	2018
13	0	215	Alternatif	2017
14	0	215	Alternatif	2018
15	0	216	Alternatif	2017
16	0	216	Alternatif	2018
17	0	217	Alternatif	2017

Output 1

Output 1’deki gibi bir ekran çıktısı elde edildi.

ÖZET

Burada araştırılıp yapılmış olan tüm bu uygulamalar; bu projeyi yeniliklere açık ve daha esnek bir yapı haline evirmiştir. Yazılımın SOLID prensiplerinin tamamına uyulmaya çalışılmış ve projenin bu prensiplere çok yaklaştığı söylenebilir bir vaziyet almıştır. Bu yapıyı gerçekleştirmedeki asıl tekniklerin en başında repository yapısı gelmekte. Bu yapı sayesinde aslında interface'lerin bile yönetilebildiğini söylemek yanlış olmaz. Ayrıca repository yapısına interface'lerden bağımsız olarak filtrelemeler uygulamak front-end developer'lar için bir avantaj sağlamıştır. Mikro servislere kadar gidilebilir fakat burada projenin kapsamı üzerinde düşünüldüğü zaman mikro servis gibi tüm yapıların base'inin yapılması çok mantıklı olmaz. Çünkü kullanılmayacak detaylara inmek ve bu yapılara sahip olması ancak ve ancak developer için eziyet teşkil edecektir. Bu sebeple hala bilinen ve giderilmesi planlanan bazı sorunlar mevcut. Ayrıca uygulanacak birkaç teknik daha mevcut. Bu teknikler: Dependency Injection ve Fluent Validation dışında Tasarım Kalıplarının Uygunluğu, Test Driven gibi teknik süreçleri de yazılım mühendisi tarafından gerçekleştirilecektir. Ayrıca iş özelinde gerçekleştirilmesi planlanan spesifik operasyonlar gerçekleştirilecektir.

ANAHTAR KELİMELER

Koyu Mavi : Reserved Words (**public**, **void**, **int**, **interface**, vb.)

Açık Mavi : Prepared classes and special classes (**Area**, **List**, vb.)

Koyu Yeşil : Special Interfaces (**IAreaDal**, **IEntityRepository**, vb.)

X.dbo : SQL veritabanı üzerinde X tablosunun .dbo uzantılı gösterimi.

Folder X : Proje Solution'ı üzerinde klasörleme işlemlerinin X. resmi.

VS Code 19 : Microsoft ürünü Visual Studio geliştirme ortamında proje genelinde yazılmış kod enstantenelerinden X. kod bloğu.

Output X : X. ekran çıktısı

Dal : “Data Access Layer” söyleminin baş harfleriyle kısaltılması. Veri erişim katmanında isimlendirmede kullanılır.

Developer : Yazılım geliştiricisi.

EKLER

[Projenin Şuanki Hali](#) (Drive'a yüklüdür, güvenli bağlantı.) İncelenmek istenirse katmanlar ve altındaki bütün kodlar mevcut. Bu form ekranı ve genel olarak bu rapor özelinde paylaşılmış olan Çelikler Holding Enerji A.Ş. verilerinin güvenliğini riske atmamak açısından siz saygıdeğer Fahrettin Hocam'dan ricamdır: Çoğaltacak ya da başka mecralarda paylaşacak bir şekilde bu dosyayı kullanmazsanız sizlere müteşekkir olurum.