



---

# PROJE

---

Haftalık Rapor – 25.12.2020

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

## 1. SPESİFİK OPERASYONLARIN UYGULANMASI

Subscribers tablosunun verilerinin gösterilebilmesi için Areas tablosuna göre bir conditional içerecek. Areas tablosu Form1\_Load'ında gelebilir, fakat Subscribers tablosu uzun bir tablo ve Areas tablosunda seçili hücrenin ya da satırın karşılığı olan verileri Subscribers tablosundan filtreleyerek çekilmesi gerekiyor. Daha evvel kodlanmış olan Repository sayesinde bu filtrelemenin kolaylaştığı söylenebilir, fakat ek işlemler gerekir. Öncelikle dgwAreas üzerinde Cell\_Click event'ine kaydolundu.

*VS Code 1*

```
public interface ISubscriberService
{
    List<Subscriber> GetAll();
    List<Subscriber> GetSubscribersByArea(int areaCode);
}
```

VS Code 1'de ISubscriberService üzerinde bir method oluşturuldu. Tablolardaki AreaCode ortak olduğu için VS Code 1'de ilişkisel veritabanı işlemlerini de filtreler sayesinde yapabilme imkanı sunuyor. SubscriberManager yenilenen ISubscriberService için implemente edilmesi gerekir.

*VS Code 2*

```
public class SubscriberManager : ISubscriberService
{
    ISubscriberDal _subscriberDal;
    public SubscriberManager(ISubscriberDal subscriberDal)
    {
        _subscriberDal = subscriberDal;
    }
    public List<Subscriber> GetAll()
    {
        return _subscriberDal.GetAll();
    }
    public List<Subscriber> GetSubscribersByArea(int areaCode)
    {
        return _subscriberDal.GetAll(p => p.AreaCode == areaCode);
    }
}
```

SubscriberManager class'ında VS Code 2'de gelen areaCode değişkeni veritabanındaki AreaCode ile filtrelendi. Daha sonra, dgwAreas nesnesinin Cell\_Click event'inde tıklanan satırın AreaCode'u gerekli olacaktır. AreaCode ise 2. sütundadır.

### VS Code 3

```
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)
    {
        LoadAreas();
    }
    private void dgwAreas_CellClick(object sender,
DataGridViewCellEventArgs e)
    {
        var areaCode =
Convert.ToInt32(dgwAreas.CurrentRow.Cells[1].Value);
        LoadSubscribers(areaCode);
    }
    private void LoadSubscribers(int areaCode)
    {
        dgwSubscribers.DataSource =
        _subscriberService.GetSubscribersByArea(areaCode);
    }
    private void LoadAreas()
    {
        dgwAreas.DataSource = _areaService.GetAll();
    }
}
```

VS Code 3'te AreaCode değişkeni bu şekilde aktarıldı. Ayrıca VS Code 3'te extract metodlara çevirilerek daha okunur bir hale getirildi. Ayrıca iş özelinde Excel'e Aktar butonu iptal edilip SQL Procedure'leri ile daha kararı verilmemiş bir formatta export edilecek, bu nedenle Excel'e Aktar butonu yerine daha düzgün bir veritabanı doldurma kısmı yapılacaktır. Yine Entity Framework kullanılacak. Var olan katmanlar ve yapılar yeni bir uygulama için işe yarayacak çünkü generic bir yapıda kodlandığı için entegrasi kolay olacaktır. Veritabanına uygun tabloyu ekleyip sadece iş katmanında birkaç değişiklik gerekecek. Bu aşamada geriye kalan şeyler katmanların sunduğu kolaylıklar

ve daha evvel atılmış olan adımlar. Bu aşamada öncelikle çıktı ekranı düzenlenmesine gidildi.

DAĞITIM BÖLGELERİ		
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...

DAĞITIM ABONE GRUPLARI				
Id	SubscriberCode	AreaCode	Name	Year
13	0	215	Alternatif	2017
14	0	215	Alternatif	2018
48	1	215	Sanayi	2017
49	1	215	Sanayi	2018
68	2	215	Ticarethane	2017
69	2	215	Ticarethane	2018
97	3	215	Mesken	2017
98	3	215	Mesken	2018
133	4	215	Tarimsal Sulama	2017
134	4	215	Tarimsal Sulama	2018
160	5	215	Aydinlatma	2017
334	115	215	Aydinlatma - ADA...	2018
360	141	215	Aydinlatma - GAZ...	2018
364	145	215	Aydinlatma - HAT...	2018

**HESAP ALANI**

HESAPLA

TEMİZLE

AKTAR

Copyright © 2020  
ÇELİKLER HOLDİNG

Output 1

Output 1'deki ekran elde edildi. Burada Hesap Alanı kısmı üzerinde bazı kurumsal kısıtlara gidilecek. Output 1'deki Temizle butonu ile seçili satır silinebilecek, Hesapla butonu seçili satırı hesaplayacak ve Aktar butonu DataGridView nesnesi üzerindeki değişiklikleri güncelleyecek. Fakat yapılan işlemleri Log'lama gibi bir yöntem ile bu kısmın tasarımını yine DataGridView nesnesiyle gerçekleştirmek tasarım bütünlüğü açısından daha mantıklı olabilir. Öncelikle SQL veritabanım üzerinde tablo oluşturmak için script yazıldı.

#### SQL Script 1

```
CREATE TABLE [dbo].[Transfers] (  
    [Id] INT IDENTITY (1, 1) NOT NULL,  
    [T1] FLOAT (53) NULL,  
    [T2] FLOAT (53) NULL,  
    [T3] FLOAT (53) NULL,  
    [Type] INT NULL,  
    [Result] FLOAT (53) NULL,  
    [SpecialResult] FLOAT (53) NULL,  
    PRIMARY KEY CLUSTERED ([Id] ASC)  
);
```

SQL Script 1’de gereksinimlere uygun “Transfers.dbo” tablosu oluşturuldu. Bu tablo da veritabanından çekileceği için genel yapıda kodlaması yapıldı. Yeniden Entity Framework kullanılıp son aşamaya kadar (daha evvel bahsi geçtiği için tüm detaylardan ayrıca bahsedilmemiştir.) getirildi. Deneme amacıyla off-set değerleri tabloya girildi ve projenin görsel şekli belirlenmiş oldu. Bu tabloda Id clustered olduğu için ve ayrıca iş gereği Result, SpecialResult kısımları doldurulabilir olmaması için bahsi geçen column’ların R/W Mode’u ReadOnly şekilde ayarlandı. Ayrıca 20 adetten fazla log sayısı tutulmayacak iş gereği. Bu sebeple henüz doldurulmamış olan tablo hücrelerini varsayılan olarak 0,0 gibi bir numaralandırma sistemiyle doldurulmuş olacak.

#### VS Code 4

```
private void btnCalculate_Click(object sender, EventArgs e)
{
    for (int i = 0; i < 6; i++)
    {
        for (int j = 0; j < 20; j++)
        {
            ...
        }
    }
}
```

VS Code 4’teki teknik ile hesap alanındaki DataGridView nesnesinin her hücrelerini sırayla dolaşıp hesap yapmak mantıksız olacaktır. Burada pek tabii iyileştirmeler mümkündür. Döngü menziline sütun ve satır sayısı kadar seçmek gibi iyileştirmeler mümkündür. Fakat bu aşama için sütun sayısı 6 olarak kalacaktır, kesin denebilir. Fakat satır sayısı yirmi ile sınırlı kalmayabilir. Neticede log mantığı ile hareket edilirse başka bir zaman log sayısı arttırılmak istenebilir. VS Code 4’teki nested iteration’lar  $6 * 20 = 120$  gibi daha hesaplanabilir bir maliyet katsayısı sunuyorken, log’lama sayısının  $n$  olması ile birlikte  $6n$  gibi yüksek sıçrayışlara mal olabilir. Bundan ötürü bu örneklem üzerinde nested bir yapı sunmak maliyeti orantısız bir şekilde değiştirebilir. Bu sebeple hesaplama işlemi sadece seçili olan satırın hesaplaması yapılacak. Toplu bir hesap mantıklı gibi gözükse de aslında üzerinde çalışılan satırın hesabının yapılması o an için ihtiyacı yüksek oranda karşılayacaktır. Burada satırlar

yerine daha sabit ve değişmeyen bir yapıda olan sütunlar üzerinden ilerlemek hafıza yönetimini daha hesaplanabilir bir hale getirecektir.

#### VS Code 5

```
//TVariables[0] = Id;
//TVariables[1] = T1;
//TVariables[2] = T2;
//TVariables[3] = T3;
//TVariables[4] = Type;
//TVariables[5] = Result;
//TVariables[6] = SpecialResult;
double[] TVariables = new double[] { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0 };
private void dgwCalculate_CellClick(object sender,
DataGridViewCellEventArgs e)
{
    for (int i = 0; i <= 6; i++)
    {
        TVariables[i] =
Convert.ToDouble(dgwCalculate.CurrentRow.Cells[i].Value);
    }
}
```

VS Code 5'te aslında array yapısı yerine daha farklı bir yapı kullanarak bu işlem gerçekleştirilebilirdi. Fakat VS Code 5'te basit bir comment-line ile array elemanlarının hangi amaçla kullanıldığı açıklanabilirdi. Netice iş özelinde yapılmış bir işlem olduğu için bu yapının çok fazla esnetilmesi gerekli bulunmadı. Burada array yapısına ihtiyaç duyulmasının nedeni hesap alanındaki CellClick event'inden bir return değeri alıp bu değerler döndürülebilirdi. Fakat buton nesnesinin Click event'inde CellClick event'ini çağırmak daha karmaşık bir hafıza yapısı gerektirecekti. Dolayısıyla array kullanmak her zaman için daha hesaplı bir maliyet sunmuş oluyor.

#### VS Code 6

```
private void btnCalculate_Click(object sender, EventArgs e)
{
    var result =
_transferService.CalculateBySpecialResult(TVariables[1], TVariables[2],
TVariables[3]);
    dgwCalculate.Rows[Convert.ToInt32(TVariables[0]) -
1].Cells[5].Value = result;
    var tip = Convert.ToInt32(TVariables[4]);
    var specialResult =
_transferService.CalculateBySpecialResult(TVariables[tip + 1]);
    dgwCalculate.Rows[Convert.ToInt32(TVariables[0]) -
1].Cells[6].Value = specialResult;
}
```

Sunum katmanından alınması gereken tüm bilgiler VS Code 6'da service kısmına buton event'i ile gönderildi.

*VS Code 7*

```
double CalculateByResult(double T1, double T2, double T3);  
double CalculateBySpecialResult(double T1 = 0, double T2 = 0, double T3 = 0);
```

ITransferService interface'inde 2 adet method etiketi tanımlandı.

*VS Code 8*

```
public double CalculateByResult(double T1, double T2, double T3)  
{  
    return (T1 + T2 + T3) / 3;  
}  
public double CalculateBySpecialResult(double T1 = 0, double T2 = 0,  
double T3 = 0)  
{  
    return (T1 + T2 + T3) / 3;  
}
```

TransferManager class'ında VS Code 8'deki gibi 2 adet method implemente edildi. Ayrıntılar iş özelinde tanımlandı.

*VS Code 9*

```
public interface ITransferService  
{  
    ...  
    void Update(Transfer transfer);  
}
```

Aktar ve Temizle butonları için VS Code 9'daki kod ITransferService içerisine eklendi. Temizle butonuyla aslında güncelleme işlemi yapılacak. Çünkü silme işlemi aslında yapılmayacak ve seçilen satır Id dışında 0'a eşitlenecek.

*VS Code 10*

```
public class TransferManager : ITransferService  
{  
    ...  
    public void Update(Transfer transfer)  
    {  
        _transferDal.Update(transfer);  
    }  
}
```

```
}
```

VS Code 10'da implementasyonu yapıldı.

*VS Code 11*

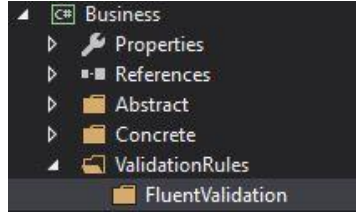
```
private void btnTransfer_Click(object sender, EventArgs e)
{
    _transferService.Update(new Transfer
    {
        Id = Convert.ToInt32(TVariables[0]),
        T1 = TVariables[1],
        T2 = TVariables[2],
        T3 = TVariables[3],
        Type = Convert.ToInt32(TVariables[4]),
        Result = TVariables[5],
        SpecialResult = TVariables[6]
    });
    LoadCalculate();
    MessageBox.Show("Seçili Satır Güncellendi.");
}
```

VS Code 11'de Aktar butonunun Click event'i yazıldı. Temizle butonu Click event'i VS Code 11'deki gibi bir yapıya benzer şekilde yazıldı. Fakat buradaki tüm güvenlik kuralları bir validation yapısı ile belirlenecektir.



## 2. VALİDASYON İMPLEMENTASYONU

Gerçekleştirilen tüm işlemlerin kurallar bütününe bağlı olması gerekir. Bu kurallar iş özelinde olduğu için iş katmanında tanımlanması ve uygulaması gerekir. Tüm bu uygulamaların genel adına validasyon denebilir. Bahsi geçen tekniği Visual Studio paket yöneticisi NuGet yardımıyla çözmek uygun bulundu.



*Folder 1*

Folder 1’de paketi kurmadan evvel iş katmanında gerekli klasörleme işlemleri yapıldı. Burada validasyon kuralları sadece şuan için Fluent Validation tekniği ile gerçekleştirileceği için ayrıca bir klasör açılması uygun bulundu. Folder 1 yapısına başka bir validasyon kuralı tekniği uygulandığı takdirde SOLID prensiplerine uygun bir biçimde yeni kurallar farklı alt klasörler ile barındırılabilir. Bu çerçevede NuGet paket yönetici yardımıyla iş katmanına FluentValidation paketi yüklendi.

*VS Code 12*

```
public class TransferValidator : AbstractValidator<Transfer>
{
}
```

FluentValidation klasörünün içerisine VS Code 12’deki gibi public bir class yazıldı. Fluent Validation’ının hazır abstract’ları inherit edildi. Transfer nesnesinin seçilmesinin nedeni Transfers tablosu üzerinde değişiklikler yapılıyor olmasıdır. Diğer tablolar üzerinde sadece gösterim yapıldığı için bir kurala ihtiyaç yok. Kuralları end-users için belirlemek gereklidir. Burada spesifik methodlar tanımlanabilir fakat en temel ve gerekli olan kurallar bu class’ın constructor’ında tanımlanması gerekir.

*VS Code 13*

```
public class TransferValidator : AbstractValidator<Transfer>
{
    public TransferValidator()
    {
        RuleFor(p => p.T1).NotEmpty();
        RuleFor(p => p.T2).NotEmpty();
        RuleFor(p => p.T3).NotEmpty();
        RuleFor(p => p.Result).NotEmpty();
        RuleFor(p => p.SpecialResult).NotEmpty();
    }
}
```

VS Code 13'te öncelikle hiçbirinin boş olmaması gerektiği kurallar olarak belirtiliyor. Burada LINQ generic yapısından faydalanılıyor. İş özelinde tanımlanmış olan kuralların bu çerçevede belirtilmesi avantaj sağlayacaktır. Neticede ne kadar az logical ve conditional işlem olursa o kadar yük hafiflemiş olur. Çünkü projenin bu aşamasında yapılan bir logical fault bütün projenin seyrini değiştirebileceği gibi, hata yönetimini de zorlaştıracaktır. Ayrıca logical hataların bulunması zordur, bu konuda IDE'ler yardımcı olmamaktadır. Logical fault'lar tespiti en zor hata türlerindendir. Bu sebeptendir ki; algoritma olarak programcının logical fault yapmaması beklenir. Bu sebeple if, switch gibi yapılardan ne kadar uzak durulursa o kadar avantaj sağlayacaktır.

#### VS Code 14

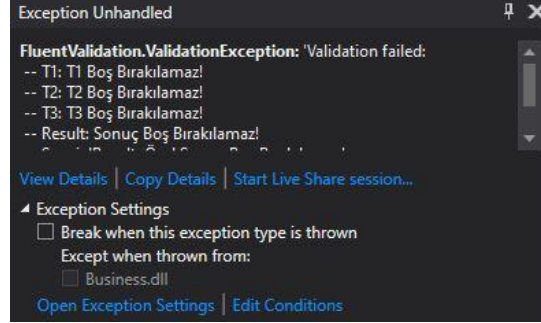
```
public TransferValidator()
{
    RuleFor(p => p.T1).NotEmpty().WithMessage("T1 Boş Bırakılamaz!");
    RuleFor(p => p.T2).NotEmpty().WithMessage("T2 Boş Bırakılamaz!");
    RuleFor(p => p.T3).NotEmpty().WithMessage("T3 Boş Bırakılamaz!");
    RuleFor(p => p.Result).NotEmpty().WithMessage("Sonuç Boş Bırakılamaz!");
    RuleFor(p => p.SpecialResult).NotEmpty().WithMessage("Özel Sonuç Boş Bırakılamaz!");
    RuleFor(p => p.T1).InclusiveBetween(0, 1).WithMessage("T1: 0 - 1 arasında bir değere sahip olmalıdır!");
    RuleFor(p => p.T2).InclusiveBetween(0, 1).WithMessage("T2: 0 - 1 arasında bir değere sahip olmalıdır!");
    RuleFor(p => p.T3).InclusiveBetween(0, 1).WithMessage("T3: 0 - 1 arasında bir değere sahip olmalıdır!");
    RuleFor(p => p.Result).NotEqual(0).WithMessage("Sonuç 0 olamaz!");
    RuleFor(p => p.SpecialResult).NotEqual(0).WithMessage("Özel Sonuç 0 olamaz!");
}
```

Aynı şekilde VS Code 14'te kurallar bu doğrultuda arttırıldı. Ayrıca her bir kuralın çığnenmesi dahilinde her biri için bir hata mesajı tanımlandı. Burada iş özelinde operasyonların gereği bir kurallandırma sistemi olduğu için detaylarının rapor kapsamında değerlendirilmesi doğru bulunmadı.

#### VS Code 15

```
public void Update(Transfer transfer)
{
    TransferValidator transferValidator = new TransferValidator();
    var validationResult = transferValidator.Validate(transfer);
    if (validationResult.Errors.Count > 0)
    {
        throw new ValidationException(validationResult.Errors);
    }
    _transferDal.Update(transfer);
}
```

Aktar butonu için TransferManager class'ında VS Code 15'teki gibi validasyon işlemleri çağırılıp burada herhangi bir hata ile karşılaşıldığı zaman programın hata fırlatması istendi. Şuan test aşamasında validasyon işlemleri doğru bir şekilde çalışıyor. Tabi görsel bir yapı test aşaması için mümkün değil, ilerleyen süreçlerde görsel bir hal almaya başlayacaktır.



*Exception 1*

Hatalı giriş yapıldığı takdirde Exception 1'deki gibi bir hata fırlatma meydana geldi. Burada daha evvel tanımlanmış olan hata mesajları da görünüyor. Bu sayede doğru çalıştığına emin olundu.

#### *VS Code 16*

```
private void btnTransfer_Click(object sender, EventArgs e)
{
    try
    {
        _transferService.Update(new Transfer
        {
            Id = Convert.ToInt32(TVariables[0]),
            T1 = TVariables[1],
            T2 = TVariables[2],
            T3 = TVariables[3],
            Type = Convert.ToInt32(TVariables[4]),
            Result = TVariables[5],
            SpecialResult = TVariables[6]
        });
        LoadCalculate();
        MessageBox.Show("Seçili Satır Güncellendi.");
    }
    catch (Exception exception)
    {
        MessageBox.Show(exception.Message);
    }
}
```

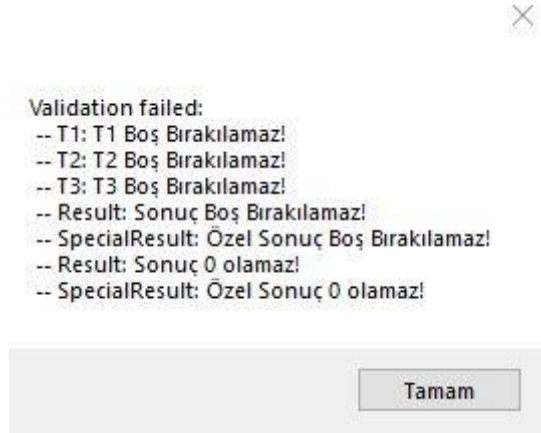
Sorun olmadığı tespit edilince görsel bir hata mesajı vermesini ve hatasız ise veritabanında güncelleme işlemi yapılması VS Code 16'da sağlandı. Ayrıca Aktar butonunun Click event'ine yazıldı ve MessageBox yardımıyla hatanın görselleştirilmesi sağlandı. Burada "Madem if, switch gibi conditional ifadeler kullanılmayacaktı, o halde neden try-catch yapısı kullanıldı?" gibi sorular oluşacaktır. Fakat bu sorulara verilebilecek en iyi cevap şudur: Kontrol işlemleri ne olursa olsun

yapılması gereken işlemlerdendir. Özellikle sisteme zarar verebileceği düşünülen tüm yapılar kontrol altına alınmalıdır. Bu kontrol elden geldiğince bağımsız ve değişkenler özelinde olmadığı takdirde daha generic bir yapı elde edilmiş olur.

*VS Code 17*

```
if (true)
{
    if (true)
    {
        if (true)
        {
            if (true)
            {
                ...
            }
        }
    }
}
```

VS Code 17'deki gibi bir örnek uygulama<sup>1</sup> karmaşıklık oluşturacağı için ve ayrıca mantıksal hataların tespitini zorlaştıracığı için if, switch yapılar burada en az seviyede kullanıldı. Bu sebeptir ki; try-catch yapısı burada hiçbir karmaşıklığa veya mantıksal hataya neden olacak bir yapı içerisinde değil. Bir ufak dipnot: VS Code 17'deki gibi bir kodlama yapılmamıştır, örnek teşkil etmesi için gösterilmiştir.



*Output 2*

VS Code 16'da yapılmış olan son değişikliklerden sonra Output 2 çıktısı elde edildi. Output 2'de normal şartlar altında MessageBox içerisinde gözüken mesajların düzenlenip "Validation failed:" gibi yazıların yazılmaması gerekir. Validasyon operasyonunun kısımları uygulama güvenliği için son kullanıcılara gösterilmemesi gerekir elbette. Fakat burada bu uygulamayı şirket dışı hiç kimsenin kullanamayacağı gerçeği var. Bir cümlelik hata mesajını silebilmek için projenin kendi hata yönetim nesnesinin yazılması uygun bulunmadı. Ayrıca son kullanıcılar için Output 2'deki

---

<sup>1</sup> Proje içeriğinde uygulanmamıştır. Söylenenleri desteklemesi ve örnek teşkil etmesi amacıyla VS Code 17 yazılmıştır.

görseli daha iyi bir hale getirmek çok da mümkün değil. Yapılacak yeni tasarım şuan ki tasarıma yakın bir görüntü sunacaktır. Yeni bir tasarımın ise maliyet olarak geri döneceği de bilindiği için Windows varsayılan MessageBox nesnesini kullanmak daha tasarruflu olacaktır. Neticede proje gereksinimleri içerisinde “maliyetin düşük olması” ibaresi bulunduğu için, buna uygun bir validasyon uygulaması gerçekleştirildi.

#### *VS Code 18*

```
public class Transfer : IEntity
{
    public int Id { get; set; }
    [Required]
    public double T1 { get; set; }
    [Required]
    public double T2 { get; set; }
    ...
}
```

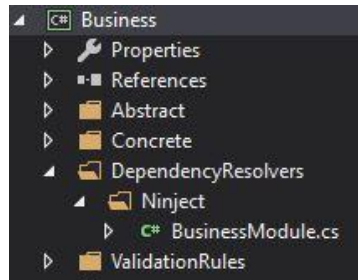
Fluent Validation’a bir alternatif olacak şekilde VS Code 18’de Data Annotations ile gerçekleştirilebilir. “Required” anahtar sözcüğü ile boş geçilemeyeceği ya da yine LINQ’ler yardımıyla başka filtrelemeler getirilebileceği göz önündedir. Fakat burada bu teknik ile tüm operasyonları kapsayacak şekilde bir biçimlendirmeye gidiliyor. Proje gereği tüm operasyonları kapsamaması istenmediğinden ötürü validasyon işlemini Data Annotations ile gerçekleştirmek mümkün değildir. Ayrıca SOLID prensiplerine de aykırı bir kullanım olur.

### 3. DEPENDENCY INJECTION VE IOC CONTAINER İMPLEMENTASYONU

VS Code 19

```
public Form1()
{
    InitializeComponent();
    _areaService = new AreaManager(new EfAreaDal());
    _subscriberService = new SubscriberManager(new
EfSubscriberDal());
    _transferService = new TransferManager(new EfTransferDal());
}
ISubscriberService _subscriberService;
IAreaService _areaService;
ITransferService _transferService;
```

Form1'in yapısı şuan için VS Code 19'daki gibi gözükmektedir. Fakat burada bir katmanın başka bir katmanı new'lememesi gerekir.<sup>2</sup> Bunun için gerekli olan çözümlerlerinin yapılması gerekir. Aslında olay burada bir noktada refactoring'e dönmektedir. Bu çerçevede kullanılan popüler IoC Container'lerden Ninject iş katmanına paket olarak yüklendi.



Folder 2

Folder 2'deki gibi bir klasörleme yapıldıktan sonra BusinessModule diye bir class oluşturuldu.

VS Code 20

```
public class BusinessModule : NinjectModule
{
    public override void Load()
    {
        throw new NotImplementedException();
    }
}
```

---

<sup>2</sup> SOLID prensiplerinin D harfi → Dependency Inversion

VS Code 20'de modül NinjectModule'den implemente edilince override bir method gelmiş oluyor. Bu override method içerisinde pkaetin genel kullanıma bağlı olarak burada Bind işlemleri gerçekleştirilir.

#### VS Code 21

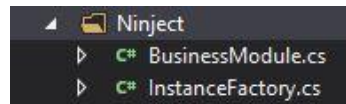
```
public override void Load()
{
    Bind<IAreaService>().To<AreaManager>();
    Bind<ISubscriberService>().To<SubscriberManager>();
    Bind<ITransferService>().To<TransferManager>();
    Bind<IAreaDal>().To<EfAreaDal>();
    Bind<ISubscriberDal>().To<EfSubscriberDal>();
    Bind<ITransferDal>().To<EfTransferDal>();
}
```

VS Code 21'de Bind işlemleri gerçekleştirilmiştir. Burada sürekli new'lemek zorunda olunan interface'leri: Ana işi yapan Manager ve Dal kısımlarına bağlanmış oldu. Burada Nhhibernate ya da servis mimarilere geçildiği zaman değişmesi gerekir bu Bind işlemlerinin. Ayrıca bu tarz durumlar için Ninject'in Load methodu yerine spesifik methodlar da yazılabilir. Bu çerçevede şuan bir değişiklik olmadığı için VS Code 21'deki yapı yeterli olacaktır. Bu yapıyı kullanabilmek için aslında Form1()

#### VS Code 22

```
public Form1()
{ ... }
```

Form1 içerisinde BusinessModule class'ında yazılmış olan Bind işlemlerinin kullanılması için Form1 in constructor ile set edilmesi gerekmektedir. ASP .NET veya .NET Core gibi uygulamalarda consturctor ile set etmek mümkünken, Windows Form uygulamalarında bu mümkün değildir. Bu sebeple instance üreten bir class yazıldı.



Folder 3

Folder 3'teki gibi InstanceFactory isimli bir class Ninject klasörünün altına yazıldı.

#### VS Code 23

```
public class InstanceFactory
{
    public static T GetInstance<T>()
    {
        var kernel = new StandardKernel(new BusinessModule());
        return kernel.Get<T>();
    }
}
```

VS Code 23'te bir çözüm sunmuş olundu. Generic bir method olan GetInstance ile verilen modüle göre<sup>3</sup> bir T döndürülmesini sağladık. Ayrıca bu method sık kullanılacağı için static bir yapıda olması yapılan araştırmalar sonucunda belirlendi.

*VS Code 24*

```
public Form1()
{
    InitializeComponent();
    _areaService = InstanceFactory.GetInstance<IAreaService>();
    _subscriberService = InstanceFactory.GetInstance<ISubscriberService>();
    _transferService = InstanceFactory.GetInstance<ITransferService>();
}
ISubscriberService _subscriberService;
IAreaService _areaService;
ITransferService _transferService;
```

VS Code 24'te ise hiçbir Manager ifadesi kullanılmadan ve en önemlisi başka bir katmanı new'lemeden kullanmış olundu. VS Code 19'daki yapı ile aynı şekilde çalışıyor fakat o yapıdaki new'leme problemi ortadan kalkmış bulunmakta.

---

<sup>3</sup> Modüller VS Code 21'de verilmiştir.



## 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 gerçekleştirmek için bu hafta uygulanan teknikler: Spesifik operasyonların gerçekleştirilmesi, validasyon implementasyonu, dependency injection ve IoC Container implementasyonu. Bu teknikler ve uygulamalar ile birlikte proje son halini almıştır.

Çelikler Holding Enerji A.Ş.

DAĞITIM BÖLGELERİ

	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...

DAĞITIM ABONE GRUPLARI

	Id	SubscriberCode	AreaCode	Name	Year
	7	0	212	Alternatif	2017
	8	0	212	Alternatif	2018
	46	1	212	Sanayi	2017
	64	2	212	Ticarethane	2017
	91	3	212	Mesken	2017
	92	3	212	Mesken	2018
	129	4	212	Tarımsal Sulama	2017
	130	4	212	Tarımsal Sulama	2018
	157	5	212	Aydınlatma	2017
	186	35	212	Sanayi-AG	2018
	209	36	212	Sanayi-OG	2018
	238	73	212	Ticarethane - AG	2018
	273	87	212	Ticarethane - OG	2018
	299	113	212	Ticarethane - OG	2018


HESAP ALANI

	Id	T1	T2	T3	Type	Result	SpecialResult
	1	0.6422342	0.554568	0.97724	1	0.724680733333...	0.184856
	2	0.8914577	0.888745	0.44613	2	0.7421109	0.14871
	3	0	0	0	0	0	0
	4	0	0	0	0	0	0
	5	0	0	0	0	0	0
	6	0	0	0	0	0	0
	7	0	0	0	0	0	0
	8	0	0	0	0	0	0
	9	0	0	0	0	0	0
	10	0	0	0	0	0	0
	11	0	0	0	0	0	0

HESAPLA

TEMİZLE

AKTAR

  
ÇELİKLER HOLDİNG  
Copyright © 2020

Output 3

Output 3 şeklini almıştır. İş yerindeki mühendislerin onayını almıştır. Çalışır ve işlevsel bir yapıdadır. Back-end kısmında yönetilebilir kolay bir katmanlama sistemine sahiptir.

## 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 X** : 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.

## **EKLER**

Projenin bitmiş hali için: [TIKLAYINIZ](#) (Drive’a yüklüdür, güvenli bağlantı). Projenin bitmiş hali burada mevcuttur. Bu raporda paylaşılmış olan tüm spesifik bilgiler Çelikler Holging Enerji A.Ş. adı altında korunmaktadır. Çoğaltılması, paylaşılması yasaktır.