KATMANLI MİMARİ NEDİR?

KATMANLI MİMARİ GENEL TANIM

Günümüzde teknolojinin çok hızlı bir şekilde gelişmesiyle kullanıcılardan gelen isteklerin sayısı çok hızlı bir şekilde artmaktadır. Projeye yeni isteklerin implement edilmesi yazılım projesinin daha karmaşık bir yapıya dönmesine sebep olabilmektedir. Bu karmaşık yapıdan dolayı projenin okunabilirliği azalmaktadır. İşte bu karmaşayı yönetebilmek için katmanlı mimari ortaya çıkmıştır.

Katmanlı mimari, Yazılım Sektöründe büyük projeler oluştururken sıkça kullanılan bir yöntemdir. Proje için sağlam temel oluşturur. Bir veritabanı işlemleri olan her proje ve programlama dilinde rahatlıkla kullanılabilir. Çok katmanlı mimari de denir. Standartı 3 katmandan oluşur fakat ihtiyaca göre katmanlar arttırılıp azaltılabilir. Çok katmanlı mimarinin tercih edilme nedeni **proje yönetiminin kolay olması**, **kodun okunabilirliğini arttırması**, **ekip çalışmasına uygun olması**, **hata yönetiminin kolay olması** vs. gibi nedenlerden tercih edilir. Yazılımlarda veriye nasıl erişileceği, üzerinde nasıl işlemler yapılacağı ve bu işlemleri kullanıcıya nasıl gösterileceği gibi işlemleri katmanlı mimari ile çok iyi bir şekilde yönetebiliyoruz. Katmanlı mimari sayesinde bu yapıyı parçalara ayırarak bu işlemlerin daha iyi yönetilebilmesini sağlıyoruz.

Nasıl parçalara ayırıyoruz derseniz Visual Studioda add Class Library adında bir seçenek vardır. Bu seçenek tam da katmanlı mimari için biçilmiş kaftandır. Tercihe göre 2 veya 3 Class Library oluşturup bunlara DAL, BLL, SUNUM gibi isimler verilebilir. Bu katmanların her birinin kendi görevi var ve birbirinden bağımsız olacaklardır. Katmanlı Mimarinin en güzel yanlarından biridir ayrıca. Bu şekilde Temelimizi atmış oluyoruz.

Her bir Class Library bizim katmanımız oluyor. Bu Class Librarylerin her biri veritabanlı bir projeyi katmanlı mimaride yapmak için gerekli işlemleri barındıracaktır ve bu katmanlar birbirinden bağımsız olacaktır. Kodun hata ayıklaması daha kolay ve kodun esnetilebilirliği yüksek olacaktır. İhtiyaç halinde diğer katmanla iletişime geçilir ve kod karmaşıklığı ortadan kalkar.

ADO.NET NEDİR

Konumuz ADO.NET olmasa da bunun ne olduğunu ve ne işe yaradığını anlatmak isterim. Aslında Katmanlı Mimaride ve veritabanı işlerinde sıkça kullanırız. Peki nedir bu ADO.NET?

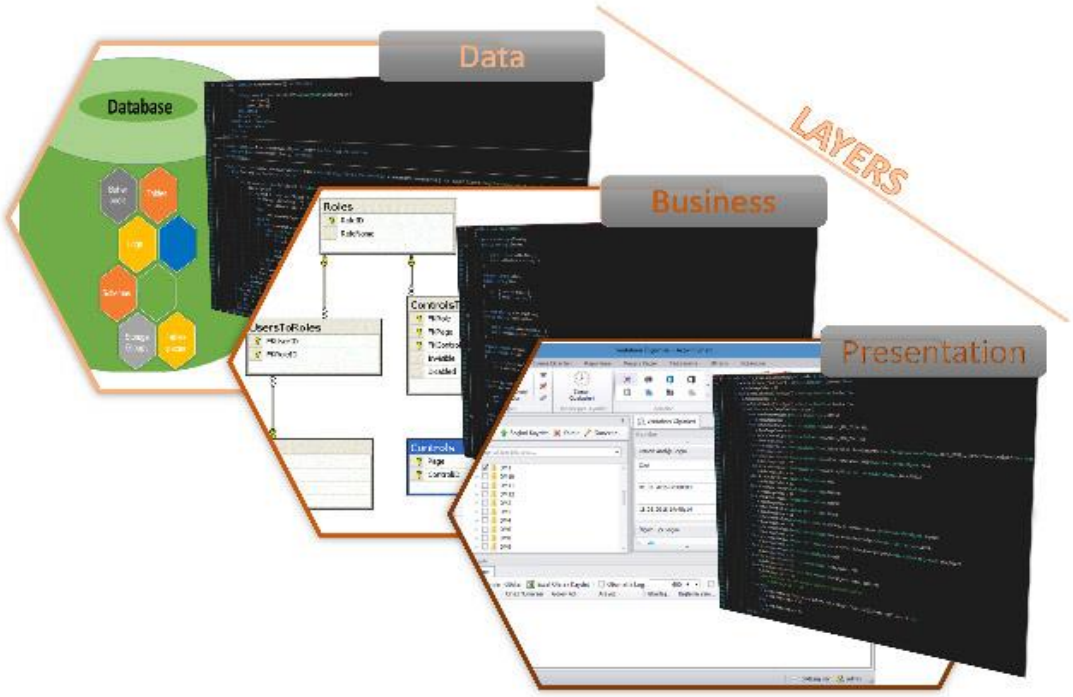
ADO.NET (ActiveX Data Objects.NET), Microsoft tarafından bize sunulan, veritabanı ile uygulamalarımız arasında köprü görevini görmektedir. Veriye Erişme teknolojisinin adıdır. Bir programlama dili değildir. ADO.NET ile uygulama tarafımızda veritabanımıza bağlanabilir, verilerimizi buradan listeleyebilir, güncelleyebilir, veri ekleyebilir veya silebiliriz. **ADO.NET** ile **SQL** sorguları, komutlarını uygulama tarafımızda kullanabiliriz. N katmanlı mimariler geliştirmek için hazırlanmış olup çevrimdışı sistemleri ve XML'e tam destek veren bir sistemdir. ADO.NET ortamını sadece SQL veri tabanı için değil, ACCESS, ORACLE gibi diğer veri tabanları içinde kullanabiliriz. Vereceğim örnekler ACCESS için olacaktır. ADO.NET'teki ACCESS araçları;

OleDbConnection: Bu sınıf ile ACCESS'te yer alan veritabanlarımıza bağlanabiliriz.

OleDbCommand : Bu sınıf SQL server üzerinde çalışacak komutlar veya StoreProcedure'ler için ilgili komut deyimlerini tutabilir.

OleDbDataReader : OleDbDataReader ile veritabanından veri döndürülmesi sağlanır. Bu sınıfta veri okunma işlemi sürekli ileri yönlüdür, geri yönlü değildir.

Bu komutların ADO.NET'e ait olduğunu öğrendiğimize göre aslında pek de konumuzla alakasız olmadığını öğrenmiş olduk. Katmanlı Mimari'de olmazsa olmazdır. Projemizde de bu araçları sıkça kullandık.



Şekil 1: Katmanlı Mimari

KATMANLI MİMARİNİN FAYDALARI

Bakım Kolaylığı: Her katman diğer katmanlardan bağımsız olduğu için, güncellemeler veya değişiklikler, uygulamayı bir bütün olarak etkilemeden gerçekleştirilebilir.

Ölçeklenebilirlik: Katmanlar katmanların konuşlandırılmasına bağlı olduğundan, bir uygulamanın ölçeklendirilmesi makul derecede kontrol edilebilir bir hal alır.

Esneklik: Her katman bağımsız olarak yönetilebilir veya ölçeklenebilir olduğundan esneklik artar.

Kullanılabilirlik: Uygulamalar, kullanılabilirliği artıran kolay ölçeklenebilir bileşenler kullanan sistemlerin etkinliğini de aynı ölçüde arttırır.

KATMANLAR

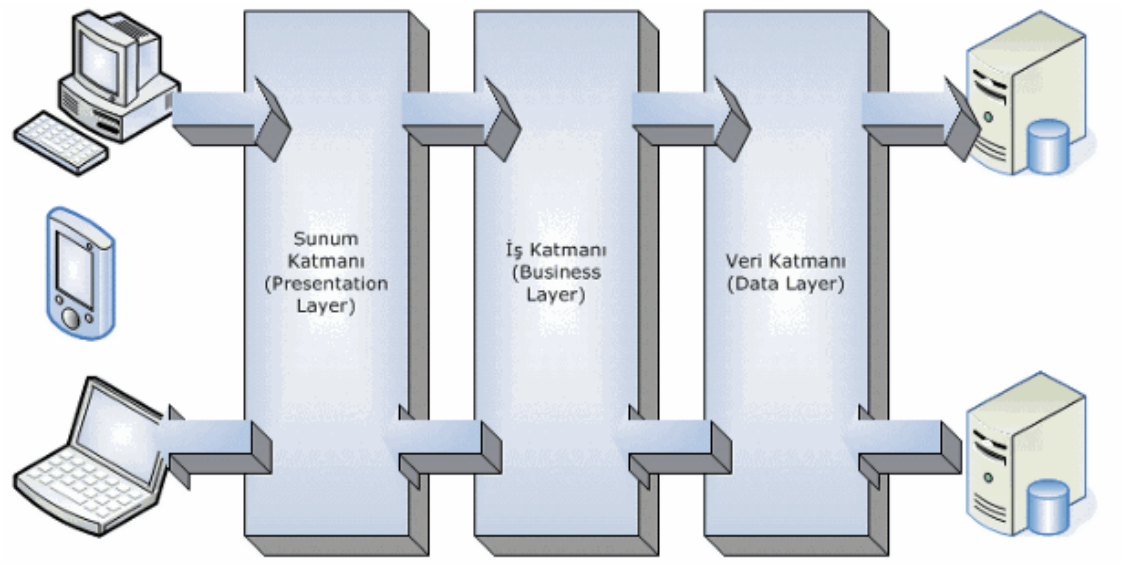
Her katman hemen hemen üstündeki ve altındaki katmanlar hariç, diğer tüm katmanlardan tamamen bağımsızdır. N inci katman yalnızca n + 1 katmanından gelen bir isteği nasıl ele alacağını, bu talebi n-l inci katmanı (varsa) nasıl yönlendirileceğini ve talebin sonuçlarını nasıl ele alacağını bilmelidir.

Standart olarak 3 katman vardır demiştik. Bu katmanlar şunlardır;

- DAL(Data Access Layer)-Veri Erişim Katmanı
- BL(Business Layer)-İş Katmanı
- PL(Presentation Layer)-Sunum Katmanı

Genel prensip, veritabanı işlemleri DAL tarafından, o veritabanının kullanılıp data manipülasyonlarının yapılması işlemi de BL tarafından yapılmalı ve sonuçlar PL ye gönderilmelidir. Sunum katmanında, kod arkasında sadece verilerin gösterilmesini sağlayan kodların yazılması makbuldür. Yani sunum katmanında veritabanı ile ilgili hiçbir işlem olmamalıdır.

3 katmanlı mimari tarzın bir örneği, güvenliğin önemli olduğu tipik bir finansal WEB uygulaması örnek verilebilir. İş katmanı, güvenlik duvarı arkasında konuşlandırılmalıdır. Sunum katmanını ise ağ üzerinde dağıtımı yapılmış web sayfalarından ibaret olabilir.



Şekil 2: Katmanlar Arası İlişki

VERİ ERİŞİM KATMANI

Üç katmanlı mimarinin en alt katmanı olan bu katman, verinin veritabanından çekilmesi ve gelen verinin veritabanına eklenmesi işleminin yapıldığı veri katmanıdır. Bu katman sadece veritabanına erişimi sağlamakla sorumludur. Veri erişim katmanı ile veritabanında hazırda bulunan veriler uygulamaya çekilir. Aynı şekilde uygulama ortamına kullanıcı tarafından girilen verilerin de hazırda bulunan veritabanına kaydedilmesi işleminde de veri erişim katmanına ihtiyaç duyulur. Bu katmanda sadece veritabanı işlemleri yapılmalıdır. Veritabanı

bağlantısı oluşturma, veritabanındaki tablolar için sınıf oluşturma ve veritabanından veri çekme işlemi dışında bu katmanda başka herhangi bir işlem yapılmamalıdır.

Veri erişim katmanı, veri depolama mekanizmalarına bağımlılıklar göstermeden veya oluşturmadan depolanan verilerin yönetim tarzlarını ortaya koyan uygulama katmanı için bir Uygulama Programlama Arabirimi (API) sağlamalıdır. Bu Katman depolama mekanizmalarına olan bağımlılıklardan kaçınmalıdır. Sonuçta, iyileştirilebilir, ölçeklenebilirlik ve sürdürülebilirlik karşılığında, veri merkezli uygulamalar için oldukça iyi sonuçlar alabileceğimiz bir mimari katmanı oluşturmuş oluruz.

```
// Bağlantı adresimizi yazıp, bağlantıyı açıyoruz
        public SqlConnection OpenConnection()
        {
            SqlConnection baglanti = new SqlConnection("Server=.;database=Northwind;trusted_connection=true");
            baglanti.Open();
            return baglanti;
        }
        // Her sorgu oluşturduğumuzda buradan yararlanacağız.
        public SqlCommand CreateConnection(string sorgu)
        {
            SqlCommand sqlCommand = new SqlCommand(sorgu, OpenConnection());
            return sqlCommand;
        }
```

Şekil 3: Veri Erişim Katmanında örnek bir SQL bağlantısı

Yukarıdaki örnekte Northwind veritabanında oluşturulmuş veritabanını uygulamaya entegre etmek için yazılmış bağlantı kodlarını görmekteyiz. İlgili Kütüphaneye ait komut olan *SqlConnection* ile bağlantı adresi yazılıp "baglanti" adında nesne oluşturulmuştur. *SqlCommand* ile yine "sqlCommand" adında oluşturulan nesneyle sorgu işlemleri yapılacaktır.

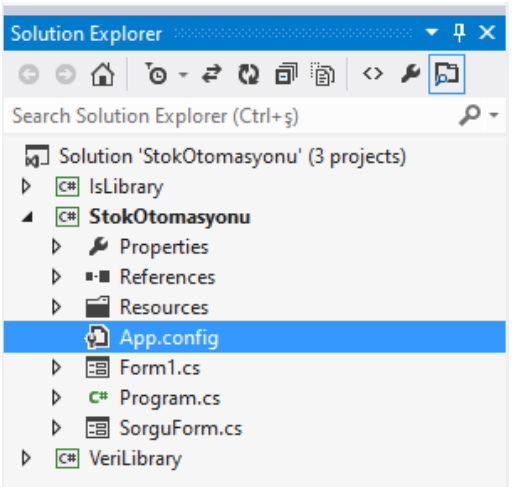
App.config İle Bağlantı

Veritabanımızı uygulamaya bağlamanın birkaç yolu vardır. Yukarıdaki örnekte bağlantı stringleri class içinde yazılmıştır. Bu bağlantı stringinde güvenlik açıklıkları oluşabilir. Çünkü örnek vermek gerekirse MS Sql Server veritabanını kullanıyorsak kullanıcı adı ve şifre değerlerinin belirtilmesi gerekir. Bunu da açık bir şekilde class içinde belirtmek pek güvenli değildir. Bu güvenlik sorununu engellemek için bir yöntem vardır ve bu yöntem daha sağlıklıdır sık kullanılır. Bağlantı stringlerimiz olsun ProviderName'imiz olsun bunları hep Proje klasörünün içinde bulunan "*app.config*" dosyasının içine yazarız. Burada daha güvenlidir ve dışarıdan erişimi zordur. Nasıl yaparıza gelince öncelikle app.config dosyasını çift tık ile açarız. Zaten içinde kendi mevcut bir kodu var. Biz buraya bağlantı ekleyeceğiz. Ben kendi projemi App.config yoluyla bağladım. Benim DAL sınıfımın adı ise *VeriLibrary*.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>

  <connectionStrings>
    <add name="baglantiStr" connectionString="Provider=Microsoft.ACE.OLEDB.12.0;Data Source=|DataDirectory|\vtStok.accdb;" providerName="System.Data.OleDb"/>
  </connectionStrings>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
    </startup>
</configuration>
```

Şekil 4: app.config dosyası



Şekil 5: App.config Dosyasının Proje Klasöründeki Yeri

Şekil 5 de "App.config" dosyasının konumunu görüyoruz. Şekil 4 de ise bu dosyanın içeriğini anlatacağım. Öncelikle bizim bağlantı için eklediğimiz şey <connectionStrings> </connectionStrings> arasında yazanlardır. <add> deyip bağlantı ekliyoruz. *name* yazan kısım bağlantımızı çağırırken kullanacağımız isim değişkenidir. *connectionString* ise Access için veritabanımızın adını ve gerekli bağlantı cümlelerini yazdığımız yerdir. Eğer MS Sql Server kullanıyor olsaydık burada ayrıca veritabanımıza bağlanacağımız kullanıcı adı ve şifre değerleri de yazacaktı. *ProviderName* ise access kullanımı için gerekli sağlayıcının ismidir.

App.configde veritabanımızı tanıttıktan sonra Veri Erişim Katmanında bağlantımızı projemize bağlayalım.

Bu katmanda öncelikte bazı kütüphaneleri eklememiz lazım

- App.config içeriği için `using System.Configuration;`

  Bu kütüphaneyi ayrıca tanımlı olduğu librarynin References kısmından referans etmemiz lazım. *References* sağ tık *Add Reference..., Assemblies>Framework System.Configuration* u bulup solundaki kutucuğa tik atmalıyız.

- Access kütüphanesi için `using System.Data.OleDb;`
- Connection State için `using System.Data;`

Bu kütüphaneleri ekledikten sonra bağlantı cümlemizi oluşturabiliriz. "baglanti" adında nesne tanımlıyoruz. OleDbConnection acceesste bağlantı komutudur. MS Sql Serverdeki karşılı SqlConnection' dur. Yukarıdaki 2 örnek resim benim projemdeki resimlerdir. Ben projemde veritabanı olarak access kullandım.

```
private static OleDbConnection baglanti = new
OleDbConnection(ConfigurationManager.ConnectionStrings["baglantiStr"].ConnectionString
);
```

Yukarıdaki bağlantı cümlemizi açıklayayım. private diğer sınıflardan erişim olmasın diye. Static diğer sınıflardan nesne oluşturmadan tanımlama yapabilmek için. OleDbConnection

bağlantı komutumuzdur. New ile yeni nesne tanımlıyoruz. ConfigurationManager app.config içeriğine girebilmek için ConnectionStrings ise app.config içinde name yazdığımız kısmı belirtmek için parametre olarak [“name”] oluşturduğumuz bağlantı adını yazıyoruz. .ConnectionString bu isimde olan bağlantıyı ConnectionString olarak kullan demektir. Yukarıdaki kod **instance**dır. Yani durum tanımlamasıdır.

```
public static OleDbConnection Baglanti
        {
            get { return baglanti;}
            set{baglanti = value;}
        }
```

Burada yapmak istediğimiz şey ise yukarıda instance olan durumun **property** kısmıdır. Yani baglanti nesnesini alıp get ve set ettiğimiz yerdir.

ExecuteNonQuery();

ExecuteNonQuery yöntemi, INSERT, DELETE veya UPDATE işlemlerinden etkilenen satır sayısını döndürür. Bu komut yalnızca ekleme, güncelleme, silme, oluştur ve set deyimleri için kullanılacaktır. *OleDbCommand* ile verdiğimiz komuttan etkilenen kayıtların adedini verir. Sonuç *integer* tipindedir. Ayrıca kendi içinde bağlantı açıp kapatır. Veri katmanıyla alakalı olduğundan bu kısımda anlattım.

İŞ KATMANI

Üç katmanlı mimarinin orta katmanıdır ve en çok iş yaptığımız, kod yoğunluğunun çok olduğu kısımdır ayrıca. Veritabanından verileri hangi sorguyla alacağımızı, verileri nasıl okuyacağımızı, verileri okuduktan sonra ne yapacağımızı bütün bu olayları bu katmanda yapıyoruz. Aslında veritabanıyla projemiz arasında bir köprü oluşturmayı sağlıyor. Ben kendi projemde iş katmanı için oluşturduğum class librarye *IsLibrary* ismini verdim. Bu kısım kendi içinde 2 ye ayrılır; ENTITY, FACADE.

ENTITY

İş Katmanının(BLL) Entity kısmındayız. Class Library içinde klasör oluşturup ismini **entity** koyuyoruz ve bu klasör içinde tablo adımızla aynı bir sınıf oluşturuyoruz. Peki Entity kısmında ne yaparız?

Getirilip gönderilen paketlerin (nesnelerin) taşındığı **class**’lardır. Örneğin, Stok tablosunu taşıyacak Stok sınıfı oluşturmak. VT’da hangi sütunlar varsa sınıf içerisinde de onu karşılayan property’ler olur.

Bu kısım Veritabanımızda bulunan her bir tabloyu eklediğimiz ve bu tablodaki her bir sütunu tanıttığımız ve bu sütunları girdi-çıktı özelliğini ayarladığımız(get, set) gerekirse özel koşul yazdığımız kısımdır. Kısaca tablolardaki kolonları değişken olarak tanımlıyoruz.

Örnek vermek gerekirse benim veritabanımda UrunAdi isminde sütunum var. Bu kısma değer ataması, değer güncellemesi, bu kısımdan değer görüntülemesi ve değer silmesi yaparken entityde oluşturduğumuz değişkeni çağıracağız. Değişkenimizi nasıl oluşturacağız?

```
public string UrunAdi { get; set, }
```

Yukarıdaki örnekteki gibi veritabanımızdaki **sütunun ismiyle birebir aynı** olacak şekilde tanımlamamızı yapıyoruz. Bu değişkenimizi gerek FACADE kısmında gerekse Sunum kısmında textboxtan değer alırken kullanacağız. Bu işlemin bir diğer adı da prop etmektir.

PROP (get; set;)

```
public int MyProperty { get; set, }
```

Property dediğimiz şey Visual Studio ortamında prop yazıp 2 kere TAB a bastığımızda çıkan kod bütünüdür. Yaptığı işlem; türü ve adını tanımladığımız değişken için kontrollü bir alış-veriş yordamı oluşturmaktır. Set ve Get metotlarını birer kontrol mekanizması olarak düşünebiliriz. Olası problemleri önlemek, işlemleri güvenilir ve kontrollü bir şekilde gerçekleştirmek için Set ve Get metotlarını kullanırız.

Örnek verecek olursak T.C. Kimlik Numarası 11 basamaklı olmalıdır. Bunun kontrolünü şu şekilde sağlayabiliriz;

```
public ulong tcno
{
get { return TCNo; }
set { if (value.ToString().Length == 11) TCNo = value;
      else MessageBox.Show("HATA! T.C. Kimlik Numarası 11 Haneli Olmalıdır.");}
}
```

Tablomuzdaki her bir sütunun propertieslerini ayarladıktan sonra yapacağımız 1 işlem daha var. Bu işlemi yapmakta zorunlu değiliz. Olmasa da olur. Yapma sebebim nesne oluşturulur oluşturulmaz propertylere değerler gönderilebilsin diye.

Yapacağımız şey parametre alan Constructor bloğudur. Öncesinde parametresiz halini de oluştururuz. Duruma göre istediğimizi kullanabiliriz.

```
public Stok() { }
```

Yukarıdaki şekilde parametresiz, sınıf adıyla aynı olmasına dikkat ederek yapıcı fonksiyonumuzu oluşturuyoruz. Eğer sınıf adıyla aynı olmasa zaten yapıcı fonksiyon olmaz.

```
public Stok(string UrunAdi, int ID) { this.UrunAdi = UrunAdi; this.Id = ID;}
```

Yukarıdaki şekilde parametreli, sınıf adıyla aynı yapıcı fonksiyonumuzu oluşturuyoruz. Propertylerle olan ilişkisine de örnektir. Kendi sınıfındaki(`this`) propertye nesne oluşup oluşmaz değer gönderir.

Bu işlemlerden sonra İş katmanının ENTITY kısmıyla işimiz bitiyor. Sırada FACADE var.

FACADE

Bu kısımda da ENTITY'de olduğu gibi Class Library içinde klasör oluşturup ismini **facade** koyuyoruz. Klasörün içine de ENTITY' de oluşturduğumuz classın çoğul ekli halini class ismi yapıp class oluşturuyoruz. Örnek: ENTITYdeki classın ismi "Stok" FACADEdaki classımın ismi "Stoklar" olacaktır. Bu bir zorunluluk değil fakat anlaşılabilir olması açısından yapmakta fayda var. Facade kısımda ne yaparız?

Getirip, gönderme, güncelleme, yazma, silme işlemlerini gerçekleştiren katmandır. Veri tabanına gider, veriyi alır, entity'e dönüştürür veya entity'i alıp veri tabanına yazar. Veritabanı ile iletişime geçen katmandır.[Getir-Götür işleri]

Bu kısımda veritabanımızla iletişime geçeceğiz. Select, Insert, Delete, Update işlemleri için SQL sorgularımızı yazacağız. Entity kısmıyla iletişim halinde verileri sorguya göre işleyeceğiz.

Öncelikle bu kısımda eklememiz gereken bazı kütüphaneler var.

- Access kütüphanesi için `using System.Data.OleDb;`
- DataTable kullanabilmek için `using System.Data;`
- Veri katmanımızı kullanabilmek için `using VeriLibrary;`
- İş katmanımızı(facade klasörü dışındakileri) kullanabilmek için `using IsLibrary;`

İlk yapacağımız işlem **Listele**dir.

Listele

```
public static DataTable Select()
        {
              OleDbDataAdapter adp = new OleDbDataAdapter("SELECT * FROM Stok",
              VeriLibrary.VeriTabani.Baglanti);
             DataTable DT = new DataTable()
             adp.Fill(DT);
             return DT;
        }
```

Metodumuzu DataTable tanımlıyoruz. Çünkü verileri veritabanından tek yönlü olarak alıp bir geçici tabloda tutacaktır. SQL sorgumuzu yazıyoruz. SELECT seç, listele demek * (yıldız) ise tüm kolonlar, sütunları getir demek. FROM ise nereden olduğunu belirtiyoruz FROM yanına tablo ismimizi yazıyoruz. Bu komutun 2. parametresine ise bağlantı propertymizi koyuyoruz. Bizim bağlantı propertymiz veri katmanındadır. Onun yolunu belirtiyoruz. *VeriLibrary.VeriTabani.Baglanti* şeklinde *"adp"* nesnesiyle bağlantımızı oluşturuyoruz. Bundan sonra *"DT"* adında DataTable nesnesi oluşturuyoruz. Gelen verileri buraya aktaracağız. *"adp"* nesnesiyle Fill metodunu kullanarak verileri DataTable'a dolduruyoruz. Geriye bu tabloyu döndürüyoruz(return).

Sorgu Listele

Bu kısım aslında Listeleye benzer. Bunu anlatmamın sebebi benim projemde kullanmış olmamdır. Listeleden biraz farkı var.

```
public static DataTable SelectMarkaSorgu(IsLibrary.Entity.Stok sorgu)
        {
```

```
        string srg = sorgu.UrunMarkasi;
        string sorgustringi = "SELECT ID, UrunMarkasi, UrunAdi, Adet FROM Stok WHERE
        UrunMarkasi Like '"+ srg +"'";
        OleDbDataAdapter adp = new OleDbDataAdapter(sorgustringi,
        VeriLibrary.VeriTabani.Baglanti);
        DataTable DT = new DataTable();
        adp.Fill(DT);
        return DT;
       }
```

Bu kısımda yapmak istediğim şey girilen UrunMarkasi değerine göre veritabanımdan o değere eşleşen verilerin Id, Marka, Ad, Adet bilgilerini almaktır. Bunun için Listele den farklı olarak DataTable fonksiyonumuz parametre alıyor. Aldığı parametre Entitydeki Stok fonksiyonudur.

Bu sefer *SELECT* den sonra *(yıldız) demedim. Çünkü tüm sütunları istemiyorum. İstediğim sütunların adını yazıp ,(virgül)'le ayırdık. *WHERE* ise alacağımız bilginin nereden olduğunu belirtiyoruz. *UrunMarkasi* sütunundan alacağız. *Like* ise arayacağımız değerin neye benzediği, ne olduğunu yazıyoruz. *Srg* isminde string oluşturdum ve bu string Entitydeki UrunMarkasi değeridir dedim. Formda da textboxa yazdığımız değer entitydeki UrunMarkasi dır diyeceğim. Böylece bilgimiz 3 katman arasında nasıl gidip geldiğini görmüş olduk. Listeleden farklı olarak SQL sorgu cümlemizi Adapter parametresi içinde değilde ayrı bir stringe tanımlayıp o stringi parametre olarak adaptere gönderdim. Aynı şey değişen bir şey yok. Kodun geri kalan kısmı zaten listeleyle aynı.

Ekle

```
public static bool Insert(IsLibrary.Entity.Stok entity)
        {
            OleDbCommand komut = new OleDbCommand("INSERT INTO
            Stok(UrunMarkasi,UrunAdi,UrunKategorisi,SonKullanmaTarihi,Adet,BirimFiyat)
            values(@m,@ad,@k,@skt,@adet,@bf)", VeriLibrary.VeriTabani.Baglanti);
            komut.Parameters.AddWithValue("@m", entity.UrunMarkasi);
            komut.Parameters.AddWithValue("@ad", entity.UrunAdi);
            komut.Parameters.AddWithValue("@k", entity.UrunKategorisi);
            komut.Parameters.AddWithValue("@skt", entity.SonKullanmaTarihi);
            komut.Parameters.AddWithValue("@adet", entity.Adet);
            komut.Parameters.AddWithValue("@bf", entity.BirimFiyat);

            return VeriLibrary.VeriTabani.ExecuteNonQuery(komut);
        }
```

Bu kısımda veritabanımıza ekleme işlemi yapacağız. SQL sorgumuzun adı *INSERT* dir. Metodumuzu bool tanımlamamın sebebi geriye oldu veya olmadı diye 2 değer döndürecektir. Bu yüzden bool kullandım. Bu sefer adapter değil komut tanımlıyoruz çünkü bu kısımda table'a veri aktarmayacağız. *INSERT* dedikten sonra *INTO* kelimesiyle hangi tabloya ekleyeceğimizi belirttiğimiz kısımdır. Tablo adını yazdıktan sonra **sırayla aralarında virgül olacak şekilde** tablodaki sütunları ardından bu değerlerin karşılıklarına yine sırasıyla *values* deyip her biri için sırasıyla değişken tanımlamalarımızı yapıyoruz. Bu değişkenleri "komut" nesnesinin "Parameters" özelliğinden "AddWithValue" metoduyla entitymizdeki değerleri kullanarak yüklüyoruz. Örnek verecek olursak;

```
komut.Parameters.AddWithValue("@ad", entity.UrunAdi);
```

`OleDbCommand` metodundan komut nesnesi oluşturmuştuk. Bunu kullanarak ekleme için yaptığımız `Parameters.AddWithValue` yoluyla SQL kısmında INSERT deyip karşılığındaki values kısmında ne yazdıysak onu belirtiriz. Virgülden sonra ise 2. parametre olarak bu sütunun entitydeki nesnesini yazarız. Bu şekilde UrunAdi kolonumuza veri eklemek için gerekenleri yapmış olduk. Metodun sonunda önceden bahsettiğim **ExequteNonQuery();** metodunu geri döndürüp işlemi bitiririz. ExecuteNonQuery 'nin ne işe yaradığını daha önce bahsetmiştim. Fakat bir konudan daha bahsedeyim. Biz bu metodu hazır olarak kullanacağımız gibi birkaç özellik ekleyip de kullanabiliriz. Benim projemde de *VeriLibrary* kısmında öyle yaptım. Try-Catch-Finally kullanarak bağlantı açık mı? Değilse aç. Kapalı mı? Değilse kapa. Özellikleri arasında ExecuteNonQuery() metodunu çağırdım ve etkilenen satır sayısına göre return ettirdim. Program içinde daha detaylı anlattım.

Sil

```
public static bool Delete(IsLibrary.Entity.Stok entity)
        {
            OleDbCommand komut = new OleDbCommand("DELETE from Stok WHERE ID=@id",
            VeriLibrary.VeriTabani.Baglanti);
            komut.Parameters.AddWithValue("@id", entity.ID);
            return VeriLibrary.VeriTabani.ExecuteNonQuery(komut);
        }
```

Silmek için SQL komutumuz *DELETE* dir. Silme işlemindeki mantık şu; Sileceğimiz satıra ait bir değeri belirtmemiz lazım. Belirteceğimiz değerin benzersiz olmasına dikkat edelim. Aksi taktirde aynı isimde başka veri olursa istemediğimiz satırı silme yapabilir. Benim referans aldığım kısım benzersiz bir değer olan ID kolonudur. Bunu *WHERE* belirteciyle <BenzersizKolonadı> = @<ulaşacağımızİsim> şeklinde yazıyoruz. Sonrasında yine Eklemede olduğu gibi komut nesnesinin Parameters özelliğiyle AddWithValue metodunu kullanarak Id belirtmemi yapıyorum. 2. Parametre alan kısım entityde tanımladığım aynı zamanda tablomdaki ID sütunudur.

Güncelle

```
public static bool Update(IsLibrary.Entity.Stok entity)
        {
            OleDbCommand komut = new OleDbCommand("UPDATE Stok set
            UrunMarkasi=@m,UrunAdi=@ad,UrunKategorisi=@k,SonKullanmaTarihi=@skt,
            Adet=@adet,BirimFiyat=@bf WHERE ID=@id", VeriLibrary.VeriTabani.Baglanti);
            komut.Parameters.AddWithValue("@m", entity.UrunMarkasi);
            komut.Parameters.AddWithValue("@ad", entity.UrunAdi);
            komut.Parameters.AddWithValue("@k", entity.UrunKategorisi);
            komut.Parameters.AddWithValue("@skt", entity.SonKullanmaTarihi);
            komut.Parameters.AddWithValue("@adet", entity.Adet);
            komut.Parameters.AddWithValue("@bf", entity.BirimFiyat);
            komut.Parameters.AddWithValue("@id", entity.ID);
            return VeriLibrary.VeriTabani.ExecuteNonQuery(komut);
        }
```

GÜNCELLE metodum EKLE ve SİL 'de olduğu gibi bool tipinde ve statiktir ve parametre olarak entitydeki tablomun olduğu kısmı alır. Böyle olmasının sebebini açıklamıştım. Güncelle Metodumuzun mantığına gelecek olursak öncelikle EKLE 'ye çok benziyor. Farkı, SQL sorgu cümleciği ve silmede olduğu gibi referans WHERE tanımlamasıdır.

Sorguyu inceleyecek olursak SQL cümlemiz *UPDATE* ardından tablo ismimiz ardından *SET* cümleciği gelir. Sonrasında aynı EKLE 'de yaptığımız gibi **değiştirmek istediğimiz** tablonun kolonlarına bir @<name> **takma ad** atarız. Bu takma adları EKLE 'deki gibi kullanırız.

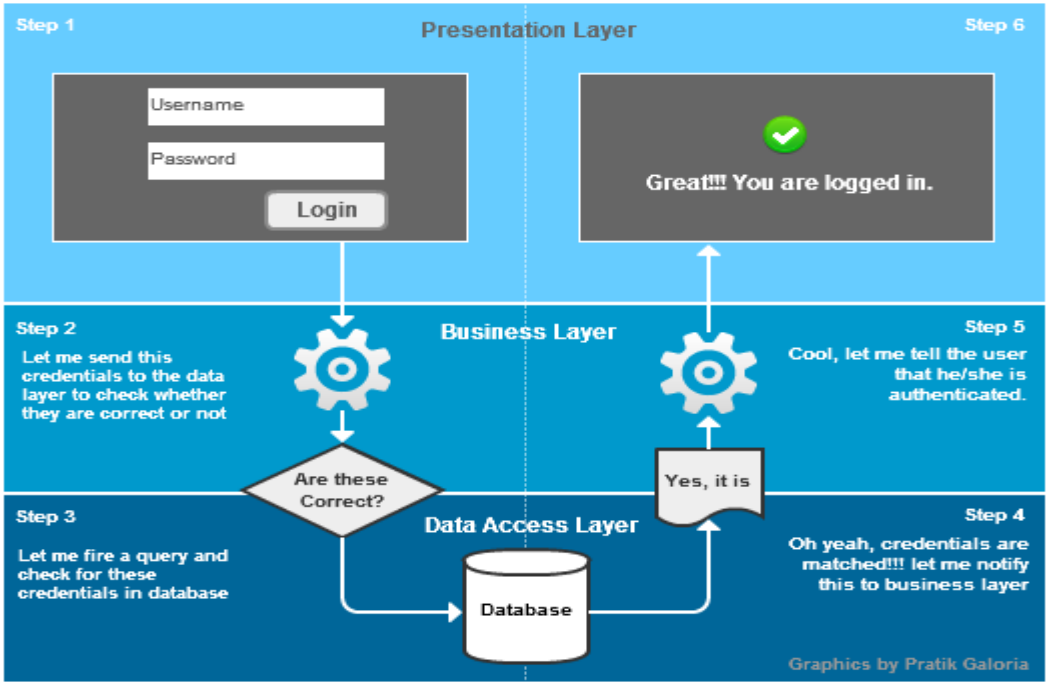
```
komut.Parameters.AddWithValue("@k", entity.UrunKategorisi);
```

GÜNCELLE 'de dikkat etmemiz gereken şey *WHERE* mantığını anlamış olmamızdır. Tabloda bir satırı değiştirmek istiyoruz tamam. Fakat bu satıra nasıl ulaşacağız? Değiştirmek istediğimiz satırı bilgisayar nerden bilecek? *WHERE* ile bunu belirtiyoruz aslında. ID 'si şu olan satırın... şu kısımlarını değiştir diyoruz aslında. Güncelle de bu şekildeydi.

Güncelleyi de anlattıktan sonra FACADE kısmını bitirmiş oluyoruz. Kısaca FACADE kısmında veritabanımızla alış-veriş yaparız. EKLE, SİL, GÜNCELLE, LİSTELE metotları bu kısımdadır. Veritabanımızla sorgular bazında iletişim halinde olmak istediğimizde İş Katmanının FACADE kısmını kullanırız.

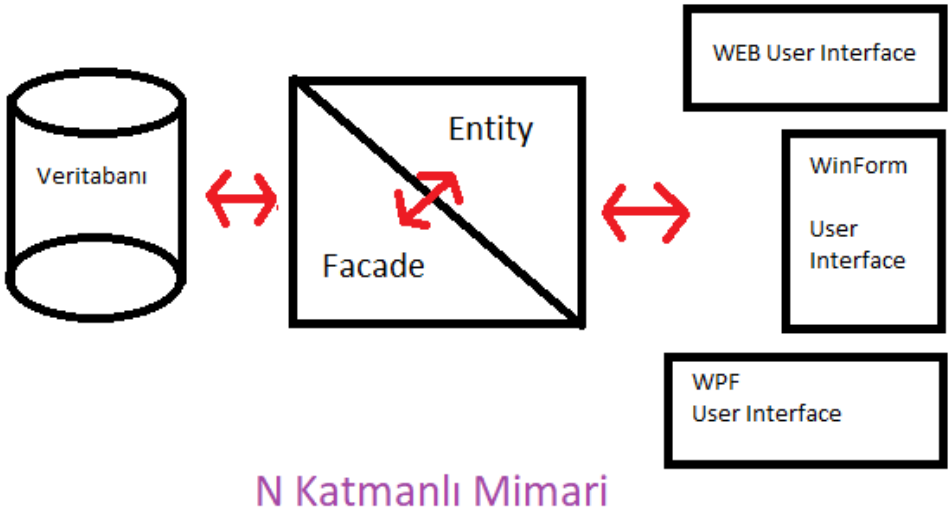
SUNUM KATMANI

Son olarak, uygulamanın kullanıcı ile etkileşimi olan arayüzün yapılandırıldığı sunum katmanı (Presentation Layer) gelmektedir. Sunum katmanı ile iş katmanında hazır hale getirdiğimiz verilerin artık kullanıcıya gidecek olan görünümü belirlenir ve artık oluşturduğumuz uygulamanın büyük bir kısmı bitmiştir. Üç katmanlı mimarinin birbirileri arasında bir etkileşim olduğu gözükmektedir. Önce veri katmanı (Data Layer) ile veriler veritabanından çekilir, iş katmanı (Business Layer) ile bu verilerin uygunluğu sağlanır ve sunum katmanı (Presentation Layer) ile bir arayüz oluşturulup kullanıcıya veriler gösterilir. Bu işlem tam tersi yönde de yapılabilir, yani kullanıcı arayüz vasıtasıyla veri girişi yapar (Sunum katmanı), girilen veri veritabanına uygun hale getirilir (İş katmanı) ve son olarak veri tabanına aktarılır (Veri katmanı).



Şekil 6: Sunum katmanına örnek bir katmanlı mimari modeli

Sunum Katmanında verilerimizin nasıl görüntüleneceği textboxlardan nasıl alınacağı ve kullanıcı için form tasarımı veyahut WEB sayfasını yaptığımız yerdir. Benim projemdeki sunum katmanımın olduğu Class Library ismi aynı zamanda projemin ismi olan *StokOtomasyonu* 'dur.

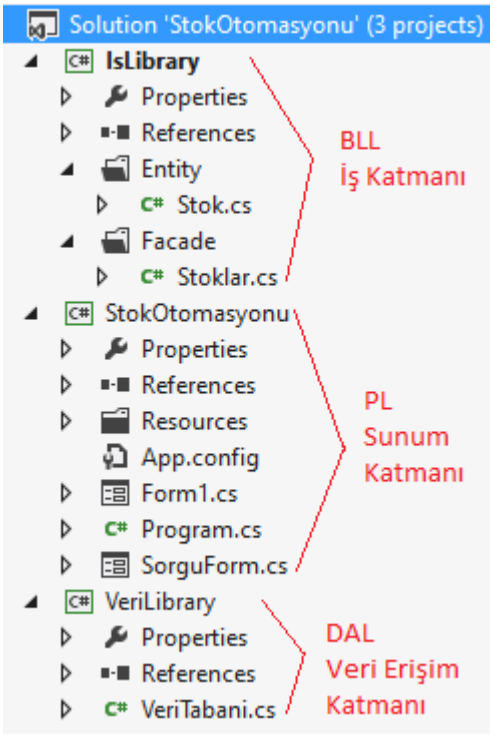


Şekil 7: Katmanlı Mimari'nin Kısımları

Şekil 7 de gördüğümüz model benim kullandığım Katmanlı Mimari modelidir. Sunum katmanı olarak Windows Form Application kullandım. Console veya WEB de kullanılabilir.

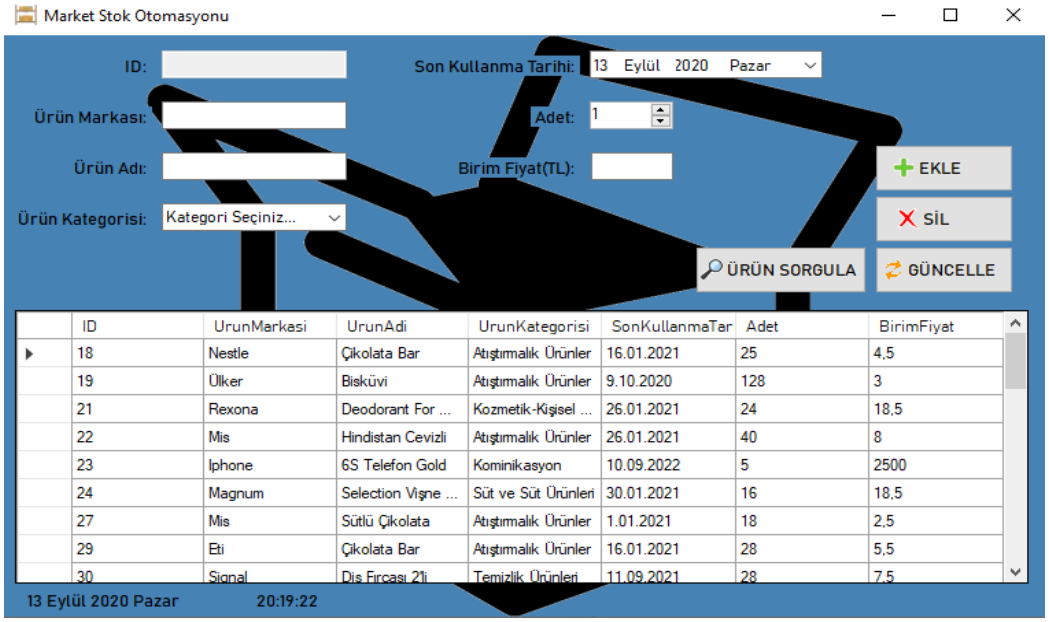
PROJEMİN SUNUM KATMANI

Bu kısımda kendi katmanlı mimarideki projemden bahsedeceğim.



Şekil 8: Projemdeki Solution Explorer kısmı

Şekil 8'de projemin Solution Explorer kısmının Katmanlı Mimarideki katmanlarını görüyoruz. Yaptığım projenin ismi Stok Otomasyonudur. Amacım Süpermarketlerde kullanılmak üzere marketin stok durumunu kontrol etmemizi kolaylaştıran ve ekleme çıkarma yapabildiğimiz pratik bir proje oluşturmaktır. Projemde 1 tablo kullandım. 2. tabloya ihtiyacım olmadı. Tek tablolu haliyle de katmanlı mimariyi anlamış ve faydalarından yararlanmış oluyoruz.



Şekil 9: Ana Form Tasarımı

Şekil 9'da Program çalışmaya başladığında karşımıza çıkan AnaForm tasarımını görüyoruz. Toolların properties kısmından **yapılabilen tüm ayarlarını propertiesten yaptım**. Form1.cs üzerinde fazla kod karmaşıklığından kaçındım.

Form1 için propertieste backgroundImage, BackgroundColor, FormBorderStyle, Icon ayarlarını yaptım. Eventste ise Click ve Load kısımlarını Form1.cs de yazdım.

Form1 üzerinde **Timer** toolunu ve DateTime metodunu kullanarak labela bir tarih saat göstergesi yaptım. Bunun sebebi tasarımsal olarak hoş gözüksün ve saate bakmak istediğimizde program üzerinden kolayca bakılabilsin diye.

```
private void saat_Tick(object sender, EventArgs e)
        {
            string tarih = DateTime.Now.ToLongDateString();
            string saat = DateTime.Now.ToLongTimeString();
            saatlbl.Text = saat;
            tarihLabel.Text = tarih;
        }
```

Timerin propertiesinden Interval değerini 1000 yani 1 saniye yaptım. Her 1 saniyede bir tetiklensin diye. Kod Kısmında da tarih ve saat adında string değerler oluşturup bunlara DateTime metoduyla saat ve tarih bilgisini aktardım. Bu bilgileri de oluşturduğum labele yazdırdım. Bu kod, Timer sayesinde her 1 saniyede tekrardan çalışacaktır. Böylece her saniyede saat ve tarih bilgisini almış oluyoruz.

Projeme ayrıca bazı araçlar için bilgilendirme amaçlı *aciklama* adında **ToolTip** ekledim. AnaForm üzerinde bulunan araçların üzerinde fare imlecimizi beklettiğimizde açıklama yazıları ekranda çıkacaktır. 4 adet butonda ve veri girilen kutucuklarda **ToolTip** mevcuttur.

Form1 üzerindeki yazıları değiştirdim 10 punto ve Bahnschrift>Regular yazı tipini kullandım. Her bir buton ve label için propertiesten bu ayarı yaptım.

Butondaki resimler için ise ImageList toolunu kullandım. Buraya internetten bulduğum yeşil tik, kırmızı çarpı, turuncu güncelle ve search ikonlarını ekledim ve buton üzerinde kullanmak içinse;

```
ekleBtn.Image = btnResimleri.Images[0];
```

kodunu form1 in Load kısmına yazdım. Images[0], 0. İndisteki resmi al demektir. Form1 yüklendiğinde resimler butonlarda olsun diye form1 load a yazdım. Ayrıca resimlerim, buton üzerindeki yazının solunda olması için butonun properties 'inden TextImageRelation değerini *ImageBeforeText* yaptım.

EKLE Butonu

```
private void ekleBtn_Click(object sender, EventArgs e)
        {
            Stok entity = new Stok();
            entity.UrunMarkasi = UrunMarkatxt.Text;
            entity.UrunAdi = UrunAdtxt.Text;
            entity.UrunKategorisi = UrunKategoricombo.SelectedItem.ToString();
            entity.SonKullanmaTarihi = SKTdatepicker.Value.ToString();
            entity.Adet = Convert.ToInt32(Adetnumeric.Value);
            entity.BirimFiyat = Convert.ToDouble(Fiyattxt.Text);
```

```
            if (!Stoklar.Insert(entity))
                MessageBox.Show("Ürün Ekleme Başarısız.\n Tüm Alanları
                Doldurduğunuzdan Emin Olun.");
            MessageBox.Show("Ürün Başarıyla
            Eklendi.","Eklendi",MessageBoxButtons.OK,MessageBoxIcon.Information);
            veriGorunum.DataSource = Stoklar.Select();
        }
```

Ekle Butonumun içeriğinden bahsedecek olursak yaptığım işlem aslında çok basit ve anlaşılır. ENTITYdeki Stok Sınıfımdan *entity* adında nesne oluşturuyorum ve bu nesnelere TextBoxlarım, ComboBoxlarım, NumericUpDownum ve DateTimePickerimdeki textleri, valueleri aktarıyorum. ve bu aktardıklarımla FACADEdaki Insert komutumu *entity* nesnesiyle çağırıyorum. Bu işlem sonucunda TextBoxlarım ve diğer InputBoxlarım üzerindeki değerler Insert metoduyla veritabanıma eklenmiş oluyor. Bunun sonucunda MessageBox kullanarak kullanıcıyı bilgilendiriyorum. Eğer işlem olmamışsa da bilgilendiriyorum.

Görüldüğü gibi Sunum katmanımda **hiçbir şekilde** veri katmanıyla ilişkim olmadı. Tamamen İş Katmanı aracılığıyla alış-verişte bulundum. Bu da Katmanlı Mimari'yi doğru yaptığımızı gösterir.

SİL Butonu

```
private void silBtn_Click(object sender, EventArgs e)
        {
            if (veriGorunum.SelectedRows.Count == 0) return;
            Stok silinecek = new Stok();
            silinecek.ID = (int)veriGorunum.CurrentRow.Cells["ID"].Value;
            if (!Stoklar.Delete(silinecek))
                MessageBox.Show("Ürün Silme Başarısız.\n Listeden Silinecek Satırı
                Seçtiğinizden Emin Olun.");
            MessageBox.Show("Ürün Silme
            Başarılı.","Silindi",MessageBoxButtons.OK,MessageBoxIcon.Information);
            veriGorunum.DataSource = Stoklar.Select();
        }
```

Sil butonumda bu sefer EKLE 'den farklı olarak TextBoxlardan herhangi bir değer almıyorum. Silme işlemini DataGridView üzerinden seçilen satır mantığında gerçekleştiriyorum. İlk if koşunu bunu sağlıyor. Eğer seçili bir satır yoksa geri dön. Ondan sonra yine ENTITY deki Stok sınıfından nesnemi oluşturuyorum. Bu nesnenin ID değerine *veriGorunum* adını verdiğim DataGridViewden seçili satırdaki ID bilgisini aktarıyorum. Ardından Delete metodumu çağırıyorum. İşlem başarılı veya başarısız diye kullanıcıyı MessageBox'larla bilgilendiriyorum.

GÜNCELLE Butonu

```
private void guncelleBtn_Click(object sender, EventArgs e)
        {
            Stok guncelle = new Stok();
            guncelle.ID = (int)IDtxt.Tag;
            guncelle.UrunMarkasi = UrunMarkatxt.Text;
            guncelle.UrunAdi = UrunAdtxt.Text;
            guncelle.UrunKategorisi = UrunKategoricombo.SelectedItem.ToString();
```

```
            guncelle.SonKullanmaTarihi = SKTdatepicker.Value.ToString();
            guncelle.Adet = Convert.ToInt32(Adetnumeric.Value);
            guncelle.BirimFiyat = Convert.ToDouble(Fiyattxt.Text);

            if (!Stoklar.Update(guncelle))
                MessageBox.Show("Ürün Güncelleme Başarısız.");
            MessageBox.Show("Ürün Başarıyla
Güncellendi.","Güncellendi",MessageBoxButtons.OK,MessageBoxIcon.Information);
            veriGorunum.DataSource = Stoklar.Select();

        }
```

Güncelle Butonunda yaptığım işlem EKLE ile %99 aynıdır. Farkları nesne tanımlamadaki ad(guncelle), çağırılan metot(Update metodu) ve ID bilgisini TextBoxın Tagını almamızdır.

veriGorunum_CellClick Eventi

```
private void veriGorunum_CellClick(object sender, DataGridViewCellEventArgs e)
        {
            DataGridViewRow satir = veriGorunum.CurrentRow;
            IDtxt.Text = satir.Cells["ID"].Value.ToString();
            IDtxt.Tag = satir.Cells["ID"].Value;
            UrunMarkatxt.Text = satir.Cells["UrunMarkasi"].Value.ToString();
            UrunAdtxt.Text = satir.Cells["UrunAdi"].Value.ToString();
            UrunKategoricombo.Text = satir.Cells["UrunKategorisi"].Value.ToString();
            SKTdatepicker.Value = (DateTime)satir.Cells["SonKullanmaTarihi"].Value;
            Adetnumeric.Value = (int)satir.Cells["Adet"].Value;
            Fiyattxt.Text = satir.Cells["BirimFiyat"].Value.ToString();
        }
```

Güncelleme yaparken kolaylık olsun diye bu kısımda yapmak istediğim şey, DataGridView ‘de tıklanan satırın bilgilerini Formdaki kutucuklara yazdırmaktır. Her seferinde veriGorunum.CurrentRow yazmamak için *satir* adında değişkenimi oluşturuyorum. Bu, Griddeki seçili satının satır sayısını bilgisini taşır. Ondan sonra yapılan işlemler aynıdır. Kutucukların bilgileri seçili satırdaki bilgilerdir diye tek tek bildirmektir.

AnaForm_Click Eventi

```
private void AnaForm_Click(object sender, EventArgs e)
        {
            veriGorunum.ClearSelection();
            IDtxt.Clear();
            UrunMarkatxt.Clear();
            UrunAdtxt.Clear();
            UrunKategoricombo.Text = "Kategori Seçiniz...";
            Adetnumeric.Value = 1;
            Fiyattxt.Clear();
        }
```

Form1 üzerinde herhangi bir yere tıklandığında doldurulan kutucuklar temizlensin metodunu yazdım. Sürekli Kutucukların dolu gözükmesi göze hoş gelmediğini düşündüğümden dolayı yaptım.

ÜRÜN SORGULA Butonu

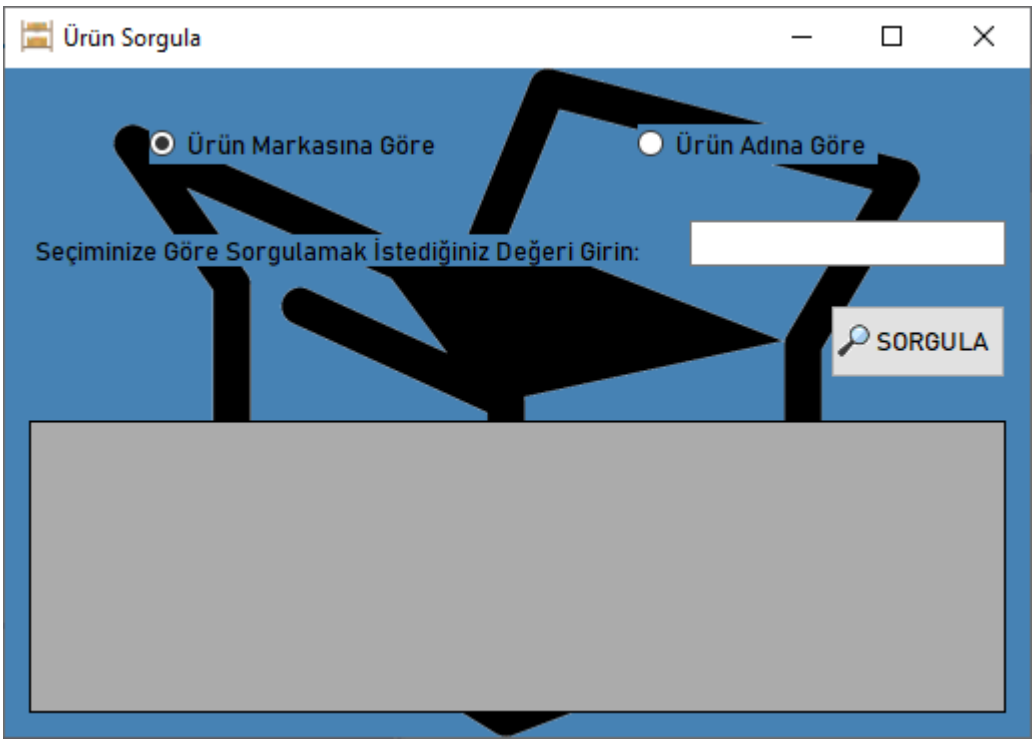
Bu butona tıklandığında karşımıza ürünün markasına göre veya adına göre sorgulama yapacağımız formumuz açılacaktır.

```
private void sorguBtn_Click(object sender, EventArgs e)
        {
            SorguForm formuac = new SorguForm();
            formuac.ShowDialog();
        }
```

Form adıyla nesne oluşturuyorum. Bu nesneyi ShowDialog(); ile ekranda açık tutuyorum.

SorguForm

Bu kısımda RadioButton'lar aracılığıyla "Ürün Markasına Göre", "Ürün Adına Göre" seçimlerine göre veritabanımızdan SELECT işlemi yapıyoruz.



Şekil 10: Ürün Sorgula Formum

Form Tasarımı AnaForm ile aynıdır. SORGULA komutumuz da bir nevi Listele komutumuzdur. Bu sefer kutucuğa girilen değere göre SELECT yapılır. O indise uyan veriler veritabanından DataGridView'e yazdırılır.

SORGULA Butonu

```
private void sorguyapBtn_Click(object sender, EventArgs e)
       {
            Stok entity = new Stok();
            if (markaRadio.Checked == true)
            {
                entity.UrunMarkasi = sorgutxt.Text;
                sorgugrid.DataSource = Stoklar.SelectMarkaSorgu(entity);
            }
            else if (adRadio.Checked == true)
            {
                entity.UrunAdi = sorgutxt.Text;
                sorgugrid.DataSource = Stoklar.SelectadSorgu(entity);
            }
            else
            {
                MessageBox.Show("Lütfen Seçim Yapınız...");
            }

            if(sorgugrid.Rows.Count == 0)
                MessageBox.Show("Girdiğiniz kelimeye ait ürün bulunamadı.\nDoğru
                yazdığınızdan emin olun.", "Sorgu Tamamlandı", MessageBoxButtons.OK,
                MessageBoxIcon.Warning);
            else
                MessageBox.Show("Sorgu Başarılı. "+sorgugrid.Rows.Count +" ürün
                bulundu.","Sorgu Tamamlandı",MessageBoxButtons.OK,MessageBoxIcon.
                Information);
        }
```

Bu butonda yaptığımız işlem kısaca hangi radioButton seçiliyse onun göre SELECT işlemi yap. Eğer markaRadio seçiliyse TextBox 'daki değer ENTITY deki UrunMarkasi adlı değerdir. DataGridView 'imizin veri kaynağı ise FACADE>Stoklar>*SelectMarkaSorgu* adındaki metottur. Bu metottan SorguListele başlığı altında bahsetmiştim. Tekrardan bahsedecek olursam yaptığı işlem, gelen veriye göre tabloda eşleşen verileri seçmektir. adRadio seçiliyse de aynı işlem yapılır. Sorgu aynı. sadece ENTITY deki değişken farklıdır. Bunun devamında eğer seçim yapılmadıysa MessageBox ile seçim yapılması gerektiğini söylüyorum eğer seçim yapıldıysa ve sonuç bulunamadıysa yine MessageBox ile sonuç bulunamadığını bildiriyorum. Sonuç Bulunmadı kontrolünü nasıl sağlıyoruz? Derseniz;

```
if(sorgugrid.Rows.Count == 0)
```

Bu kodla sorgugrid in sütun sayısı 0'a eşitse diye kontrolü sağlıyorum. Eğer satır sayısı 0 'sa veri yoktur(bulunamamıştır). Değilse yine `sorgugrid.Rows.Count` komutuyla satır sayısını yani bulunan veri sayısını MessageBox ile kullanıcıya bildiriyor ve sorgunun başarılı olduğunu söylüyorum.

KAYNAKÇA

- https://www.gencayyildiz.com/blog/c-ta-n-tier-architecturecokn-katmanli-mimari/
- https://medium.com/kodcular/katmanl%C4%B1-mimari-9fb34ef8c376
- https://sezeromer.com/c-katmanli-mimari/
- https://www.yazilimkodlama.com/c-2/cok-katmanli-yazilim-mimarisi-ornegi-c/
- https://aktif.net/tr/Aktif-Blog/Teknik-Makaleler/Yazilimda-Katmanli-Mimari
- https://medium.com/@learningzone/ntier-katmanl%C4%B1-mimari-7f515b7e600f
- https://www.youtube.com/watch?v=aGjW5FI6L9c&list=WL&index=2&t=6107s&ab_channel=AhmetYasinAk
- https://www.youtube.com/watch?v=ATo_GilxCyE&list=WL&index=3&t=0s&ab_channel=%C4%B0brahim%C3%96Z
- https://www.youtube.com/watch?v=3Ny1xv7SO4s&ab_channel=%C4%B0brahim%C3%96Z
- https://www.youtube.com/watch?v=J2ft_iLk9BM&list=WL&index=5&t=551s&ab_channel=Tasar%C4%B1mKodlama
- https://www.codeproject.com/Articles/667431/3-Tier-Architecture-in-Csharp-Web-Application