

İCİNDEKİLER

Konu	:	Sayfa No	:
ÖNSÖZ		iv	
1. BÖLÜM	: Backend (C#)		
1. DERS	: Temel Kavramlar (Python üzerinden)	1	
2. DERS	: C# ile Programlamaya Giriş (Değişkenler, şart blokları, döngüler, diziler) Kurulum: Visual Studio	2	
3. DERS	: C# => METOTLAR VE CLASSLAR Kurulum: GIT Program Kurulumu	4	
4. DERS	: C# =>Referans Tipler C# =>Liste Tanımlama	9	
5. DERS	: C# =>Inheritance C# =>Interface (POLYMORPHISM) C# =>Loglama	11	
6. DERS	: SQL (Structered Query Language) Kurulum: Nortwind Database SQL'de Veri Tabanı Oluşturma	20	
7. DERS	: Proje Katmanları Şematik Anlatım C# Projesi (MyFinalProject) Linq Joint	35	
8. DERS	: C# Çok Katmanlı Kurumsal Mimariler (Entity Framework) Proje&Entity Framework Eklienti: EntityFrameworkCore.SqlServer yükleme (Database katmanı için)	49	

<u>9. DERS</u> :	C# Çok Katmanlı Kurumsal Yazılım Mimarileri (Core Katmanı ve DTO kullanımı)	62
	Eklenti: EntityFrameworkCore.SqlServer yükleme (Core katmanı için)	
<u>10.DERS</u> :	C# Çok Katmanlı Kurumsal Yazılım Mimarileri (Business-Results&Messages)	77
<u>11. DERS</u> :	C# WebAPI katmanı	93
	Kurulum: Postman API Test Ortamı Programı	
<u>12.DERS</u> :	IOS Yapılanması (Autofac /Fluent Validation/ AOP)	105
	Eklenti: Autofac ve Autofac.Extras eklentisi kurulumu (Business ve Core Katmanı için)	
	Eklenti: FluentValidation(Jeremy Skinner) eklenti kurulumu (Business ve Core Katmanı için)	
	Eklenti: Autofac.Extensions.DependencyInjection (WebAPI ve Core Katmanı için)	
<u>13. DERS</u> :	AOP Gözden Geçirme	119
	İs kodları Yazma WorkShopları (ProductManager, CategoryManager ve AutofacBusinessModule)	
	Eklenti: Microsoft.ASPNETCORE.HTTP (Araba Projesi için Business, DataAccess ve Entities Katmanları)	
<u>14. DERS</u> :	Jason Web Token (JWT)	134
<u>15. DERS</u> :	Cache	151
	Transaction (aspect)	
	Sistem Performans (aspect)	
	Logging (aspect)	
<u>İlave Kodlar:</u>	Frontend'e yönelik veya daha önce yapmamız gereken Backend değişiklikleri	166
	Eklenti: Newtonsoft.Json (Backend için)	

2. BÖLÜM :	Frontend (Angular)	
<u>16. DERS :</u>	Angular Genel Şablonunun oluşturulması Kurulum: Visual Studio Code /Node Eklenti: Angular /Bracket Pair Colorizer ve Prettier Code Formatte	173
<u>17. DERS :</u>	NAVBAR ayarı Backend-Frontend Bağlantısı ve veri görüntüleme	185
<u>18. DERS :</u>	Backend-Frontend Bağlantı kodlarında Refactoring (standartlaştırma) Secime göre ürün listeleme Eklenti: jquery	196
<u>19. DERS :</u>	Pipe Button Angular içinde tanımlı (built-in) pipe'lar JavaScript map, filter, reduce fonksiyonları Eklenti : ngx-toastr @angular/animations	214 225 226
<u>20. DERS :</u>	Reactiv Forms Eklenti: Newtonsoft.Json (Backend için)	228
<u>21. DERS :</u>	Register ve Login İşlemleri	240
<u>İLAVE BİLGİLER:</u>	Erişim Düzenleyiciler SQL Veri Tipleri Notlar Araba Projesi	252 253 255 261

ÖNSÖZ

Sayın Engin Demiroğ ve ekibinin piyasaya nitelikli yazılımcı yetiştirmek maksadıyla hiç bir ücret almadan (aksine sitenin idamesi için ödeme yaparak) yoğun emekler harcayarak (eğitim öncesi hazırlık, eğitimin içası ve Discord desteği vb.) verdiği Yazılım Geliştirici Yetiştirme Kampı (C#-Angular) eğitim içerikleri hakkında bir çalışma notu oluşturdum. Bu notları Sayın Engin Demiroğ'un Yazılım Geliştirici Yetiştirme Kampı (C#-Angular) kurs kayıtları baz alınarak hazırladım. İlaveten Hoca'nın belirttiği ilave kaynaklar, ödevler ve ilave araştırma sonucu edinilen bilgilerle zenginlestirdim.

Tamamen kendime bilgi notu maksatlı oluşturduğum bu dokümanı paylaşmaya karar verdim. Konuya İlgilenenlerin, eğitimin içeriğine ve bu kapsamında gerekli olan bilgilere aşırı ve gereksiz gayret sarfı olmadan ulaşabileceğini değerlendirmekle beraber, eğitim esnasında anladıklarımı görsel destekli olarak kayıt altına aldığım bu dokümanda birçok hata ve eksik olmasının muhtemel olduğu göz ardı edilmemelidir

Kurs kayıtlarına, ilave kaynak bilgilerine ve kurs kapsamında verilen ödevlere kodlama.io sitesinde ilgili kursa kayıtlı giriş sonrası erişilebilmektedir. Ders kayıtlarına ayrıca Youtube üzerinden erişilebilir.

Faydalı olması temennisiyle.

Z. Mehmet Cavdar

1.BÖLÜM

1.DERS (01.2021)

Konu: Python ile Temel Kavramlar

Repl.it sitesinde Phyton üzerinden değişken tanımlama ve ne maksatla kullanıldığı anlatıldı. Değişkende yapılacak bir değişme durumunda değişkenin çağrııldığı her yeri etkiler. Yani sadece değişkeni değiştirmek yeterlidir.

2.DERS (01.2021)

Konu: C# ile Programlamaya Giriş (Değişkenler, şart blokları, döngüler, diziler)

Konu-1: C# dilinde "ternary operatörü" araştırınız.

Kullanım mantığı if-else gibidir. if - else den tek farkı kod bloglarını bir satırda yazıyoruz. Kullanım şekli : koşul ? true: false; burada bir koşul belirliyoruz. Ve bu koşul eğer sağlanıyorsa (true) soru işaretri ve iki nokta arasında yazılan kodlar çalışacaktır. Eğer koşul sağlanmıyorsa (false) iki nokta ve noktalı virgül arasındaki kod blokları çalışacaktır.

örn; int Sayı1=5; Sayı1>4 ?

```
Console.WriteLine(" Sayı1 4 'ten büyütür")
```

```
Console.WriteLine("Sayı1 4'ten küçütür");
```

Sayı1'ye atanmış değer 5 olduğundan koşul true olarak çalışacaktır. Konsol ekranına Sayı1 4 'ten büyütür çıktısı görülecektir. bir örnek daha yapalım. Şimdi hikayemiz şöyle . evli bir çift doğacak bebeği için bebek odası bakmaya gidiyor. Orada çalışan kişiye doğacak bebeklerinin kız olmadığını söylüyorlar. Çalışanın bu bilgiye göre onlara bebek odası göstermesini bekliyorlar.

```
string BebeğinCinsiyeti="Erkek"; BebeğinCinsiyeti=="Erkek" ? "
```

```
Console.WriteLine("mavi renkle süslenmiş bebek odası gösterdi");
```

```
Console.WriteLine("pembe renkle süslenmiş bebek odası gösterdi");
```

çalışan bebeğin kız olmadığı öğrendiği için mavi renkle süslenmiş bebek odası gösterdi.

Konu-2: C# dilinde "switch" yapısını araştırınız.

Switch case yapısı, çok durumlu dallanma ifadelerinde if-else blokları yerine tercihen kullanılırlar. Switch-case ile yapılabilecek tüm işlemler if-else merdiveni ile de yapılmaktedir. Fakat kod okunabilirliğini artırdığı için birçok programcı switch-case yapısını karmaşık if-else blokları yerine kullanmaktadır. Switch-case yapısının çalışma mantığı şu şekildedir. Öncelikle switch parantezi içerisindeki ifadenin değeri hesaplanır. Hesaplanan değerle eşleşen case ifadesi bulunursa, o bloktaki kodlar çalıştırılır. Eğer hiçbir case bloğuyla eşleşme sağlanmazsa default bloğundaki kodlar çalıştırılır. Break deyimi her case bloğundan sonra mutlaka kullanılmalıdır. Çünkü istenen kod bloğu çalıştırılmış olmasına rağmen, break deyimi kullanılmazsa switch dışına çıkmadan aşağıdaki case bloklarına doğru akış devam eder.

```
switch(değişken_adi)
{
    case durum_1: ..... // Kodlar
    break;
    case durum_2: ..... // Kodlar
    break;
    default: ..... // Kodlar
    break;
}
```

Konu-3: C# dilinde "while" döngüsünü araştırınız.

Bir kodu, belirlenmiş bir sayıda tekrar etmek istiyorsak for döngüsü kullanırız. Eğer kodumuzun ne kadar sefer tekrar etmesini istediğimizi bilmiyorsak while döngüsü kullanırız. While döngüsünde koşul "true" olduğu sürece kod dönmeye devam eder.

Bunu neden kullanalım dersek; bir satranç oyunu yazdığını ele alalım. Satranç oyunlarının kaç hamle boyunca devam edeceğini bilemeyeziz. Bu yüzden for döngüsü yerine.

while (true)

```
{    //oyun burada çalışır }
```

gibi bir yapı ile oyun bitmesine uygun bir zamana kadar devam eder. Oyunun bitişinde ise ile "break" komutu ile while döngüsünden çıkarız.

Konu-4: repl.it sitesini araştır

Kurulum: **Vusial studio kurulumu (dil İngilizce olsun ve yüklenecek kütüphaneler için videoya bakın)**

https://www.youtube.com/watch?v=kOnAP3fT_Vs

3.DERS (16.01.2021)

Konu : C# ile nesne yönelimli programlamaya giriş **Metotlar ve Classlar**

Kullanılan Dosya : Visual Studio => C# => KamplIntro => Metotlar

Tamamlayıcı Kaynak : Engin Demirog Youtube Sitesinde c# Dersleri No: 21-28

<https://www.youtube.com/watch?v=4r7hJwpGTL4&list=PLqG356ExoxZU5keiJwuHDpXqULLffwRYD&index=22>

Tanım: Metotlar tekrar kullanılabilirliği sağlayan kod blokları

Not: DRY: Do not repeat yourself

Not: Değişkenler büyük harfle tanımlanır

Tanım: **Snippet** hazır kodlar demektir. Çağrı adını yapıp 2xtab tuşuna basılırsa hazır kodlar gelir

Örneğin “**prop**” yapıp 2xtab tuşuna basılırsa “**public**” fonksiyonu başlangıç -bitiş temel değerleri otomatik çıkar

Console.WriteLine("-----METOTLAR-----"); //kısa yolu “cw” yapıp 2xTab

public int MyProperty { get; set; } //kısa yolu “prob” yapıp 2xTab

ENCAPSULATION : Bir Class içerisinde tanımlanan bazı özellikleri kısıtlamam (yalnızca okuma ya da yazma gibi) gerekeceği durumlarda yapılan işleme denir. **get** komutu bu metottaki değişkenlerin başka yerlerde okunabileceğini, **set** komutu ise yazdırılabilceğini belirtir.

Program içinde sadece kullanıyorsak get bloğu çalışır

Örnek : Console.WriteLine(Urun.Adi);

Program içinde eşitleme ile veya datadan okuyup eşitleme ile veri alınıyorsa **set** bloğu çalışır

Örnek : Urun.Adi="elma";

Örneğin Ayakkabı Classının “Numarası” özelliğinin dışardan değiştirilmesini istemiyorum. Yani, sadece “okunabilir” olmasını istersek **Set** komutunu iptal etmeliyiz.

Aslında **Get** ve **set** komutları yandaki komutlarda olduğu gibidir (arka planda olan). Farklı bir komut yazmak istersek görülen komutlarda değişiklik yapılır

```
namespace Classes
{
    class Customer
    {
        public int Id { get; set; }

        private string _firstName;
        public string FirstName
        {
            get { return _firstName; }
            set { _firstName = value; }
        }

        public string LastName { get; set; }
        public string City { get; set; }
    }
}
```

Not: get ve set ayarları Java'da metodlarla yapılırken C#'da properties ile yapılır

Tanım : Classlar belirli bir amaç (birden fazla değişkeni veya işlemi) kümelemek maksadıyla kullanılan yapılardır

Classlar genel olarak 2"ye ayrılır ve 2 özellik aynı Classda bir arada olmamalıdır.

- 1- İçinde özellik belirten classlar ==> Örnek Product Class
- 2- İçinde operasyon/metod olan classlar ==> Örnek Product Manager Class

Class Ekleme : Sağ tarafta “**Solution Explorer**” kısmında metodların üzerine sağ tıklayıp **add** (ekle) ile bir Class ekleyelim

Dizi ve class vb. yapıların adları değişkeni değil adresi hafızada tutar.

O yüzden tanımlama yaparken "Tip ad = new Tip []{}" şeklinde tanımlama yapılır

Class 'ları da dizi tipinde tanımlayabiliriz. Asıl işimize yarayacak olan budur. Bir veri tabanından verileri bu şekilde alabiliriz

NOT: Birden fazla programın olduğu bir projede sadece üzerinde çalıştığın programın koşturulması:

Visual Studio 'da “**Solution Explorer**” sayfasındaki (sağ taraf) koşurma istediğiniz programın üzerinde sağa tıklayıp "**Set a Startup Projekt**" seçilir

Örnek: class tanımlamada makbul olan ve makbul olmayan 2 yöntemin bir arada görülmesi

```
class SepetManager
{
    //NAMING CONVENTION--değişken isimlerinin ilk harfi (birleşik kelimelerde tüm kelimelerin ilk harfi) büyük
    public void Ekle(Urun urun)
    {
        Console.WriteLine("Tebrikler.Sepete Eklendi:" + urun.Adi);
    }
    //bu şekilde de yazabiliyoruz. Ancak istenmeyen yöntemdir.
    //Değişiklik olması durumunda ana programdaki tüm class işlemlerinde değişikliği işlemek zorunda kalırız. Veri tabanlarından alınması
    //durumunda da sadece class üzerinde yapılacak yeni tanımlama yapman yeterli olacaktır.
    public void Ekle2(string urunAdi, string aciklama, double fiyat)
    {
        Console.WriteLine("Tebrikler.Sepete 2. yöntemle Eklendi:" + urunAdi);
    }
}
```

Kullanılan Dosya : Visual Studio => C# => KamplIntro => Matematik

- Dört işlem yapılan bir Class tanımladık

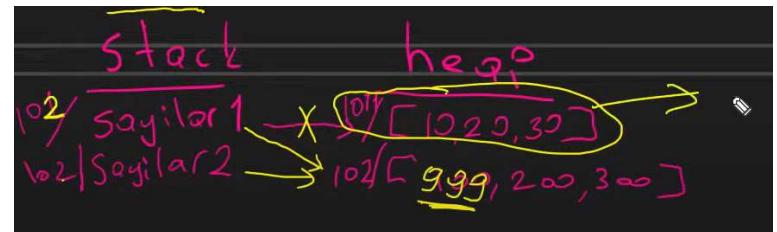
Kullanılan Dosya : Visual Studio => C# => KamplIntro => DegerlerVeReferans

Değer Tipler: int, double, bool vb. değişkenlerin birbirine eşitlenmelerinde (atanmalarında) eslenen 1. değişkende sonradan yapılacak değişim 2. değişkenin değerini etkilemez.

Bunlar hafızada sadece **Stack** kısmını kullanır

Referans Tipler: Dizi eşitlenmelerinde ise eslenen ilk dizide sonradan yapılacak değişim 2.dizinin değerinin de değişmesine neden olur. Çünkü esleme(atama) değerlerin değil adreslerin eşitlenmesidir.

Bunlar hafızada sadece **Stack** ve **Heap** kısmını kullanır



```
int sayı1 = 10;
int sayı2 = 30;
sayı1 = sayı2;
sayı2 = 65;
//sayı1=?  cevap 30
```

```
int[] sayilar1 = new int[] { 10, 20, 30 };
int[] sayilar2 = new int[] { 100, 200, 300 };
sayilar1 = sayilar2;
sayilar2[0] = 999;
//sayilar1[0]=?  cevap 999 olur .
```

Dikkat: diziler referans tip oldugu icin asagidaki örnekte sayilar2 üzerinde yapılacak her degisiklik sayilar1'in degerini de degistirir. Çünkü sayilar1 artık sayilar2 ile ayni adresi gösterir.

sayilar1 degsikeninin ilk degerlerine yani {10,20,30} degerlerine artık ulasilmaz. o dizinin adresi ve degeleri kayboldu. Garbage Collector'a gitti. O adresi baska bir dizi degiskene tutmaliydi. ciktig 999 olacak

Dikkat!!!!!!!!!!!!!!

```
String sehir1 = "Ankara";
String sehir2 = "Istanbul";
sehir1 =sehir2;
sehir2="Izmir";
System.out.println(sehir1);
```

Örnek Metinsel veri tiplerinde ise durum farklidir. String referans tip olsa da deger tip olarak calisir.

Dilleri hazırlayanlar bu seklide tanımlamış. Asagida "Istanbul" görünür.

Kurulum : GIT PROGRAM (GİTHUB HESABI AÇMA)

Adım-1: Github hesabı açılacak

Adım-2: Git sitesinden (git-scm.com) program indirilecek

Adım-3: “*Solution Center*”->“*Solution’Consoleapp3*” da sağa tıkla-> “*Create It Respiratory*” bölümünden GİTHUP hesabını bağlayıp (private kaldırmayı unutma ki public olsun) “*Create And Push*” düğmesine basılır.

Adım-4: Sağ tarafta “*GIT Changes*” bölümünden istediğiniz notu yazıp “*Push*” (yukarı ok) tuşuna basılır

Not: GITHUB'a yüklenmiş bir programda değişiklik yapıldığında

View=> “*GIT Changes*” bölümü açılır.

Comment kısmına istenen metin yazılır

“*Commit All*” tuşu ile lokale değişiklik kaydedilir.

“*Push*” (Yukarı ok) tuşu ile de GITHUB da dosya güncellenir.



Dikkat: GITHUB'dan proje alımlarında linklere, program sürümlerine ve eklentilere dikkat edilmelidir

ref ve out anahtar kelimeler

Out anahtar kelimesi argümanları metodlara referans tip olarak göndermek için kullanılır. Kod blok içerisinde değişken tanımlanırken başlangıçta herhangi bir değer atamadan bu değişkeni tanımlayabiliriz.

```
class Odev
{
    public static void main ()
    {
        int O;
        sum (out O); //Dikkat
        console.writeline("toplam : ", O);
    }
    public static void sum( out int O)
    { O= 80; O+=O; }
}
```

Ref anahtar kelimesi değişkeni referans ile göndermek için kullanılır. Tanımlanan değişkene tanımlama sırasında bir değer atanır. Değişkende yapılan herhangi bir değişikliğin, kontrol çağrıları metoda döndüğünde o değişkene yansıyacağını söyleyebiliriz.

```
class Odev2
{
    public static void main()
    {
        string ornek= "Hello";
        SetValue( ref string ornek); //Dikkat
        Console.Writeline(ornek);
    }
    public static void SetValue (ref string ornek1) //Dikkat
    {
        if( ornek1 == Hello)
            { Console.writeline("Hello World"); }

        ornek1= "Merhaba";
    }
}
```

4.DERS (20.01.2021)

Konu	: Referans Tipler
Kullanılan Dosya	: Visual Studio => C#=>KamplIntro => OOP1
Tamamlayıcı Kaynak	: Engin Demirog Youtube Sitesinde c# Dersleri
OOP icin Refeans Tipler	: https://www.youtube.com/watch?v=ruGBQ8BS_Co&feature=emb_logo
Constructor Yapısı	: https://www.youtube.com/watch?v=_gadwBmkAJ0&feature=emb_logo
Generik Yapılar	: https://www.youtube.com/watch?v=_gadwBmkAJ0&feature=emb_logo

TANIM: CRUD-Create Repeat Update

Not : Bir projede bir Class'a başka bir Class tipinde değişken gönderilirse ve o Class değişkeninde bir değer değiştirilse. O Class değişkeninin değeri her yerde kalıcı olarak değişir. CLASS ve ARRAY'lar referans tiptir

Ancak int, double, string vb. değişkenler benzer şekilde bir Class'a gönderilirse o Class içinde değeri değiştirilse bile ana yerdeki değişken değeri değişmez. Bir önceki derste benzer bir problem vardır. Sayıların değeri ile ilgili. Bunlar değer tiptir.

Benzer mantık....

Örnek Deger tip

Ana Program

```
int sayı =100;  
digerClass. Sayidegistir (sayı);  
Console.WriteLine (sayı) // cevap 100
```

Class diğer Class

```
public void Sayidegistir(int sayı)  
{  
    Sayı=99;  
}
```

Not: Bir metottan dönen bir değer yoksa **void** yazılır, Aksi takdirde dönen değer yazılır

Örnek

Ana Program

```
int toplam;  
toplam= digerClass.topla (3,6);  
Console.WriteLine (toplam) // cevap 9
```

Class digerClass

```
public int Add(int sayı1, sayı2)  
{  
    return sayı1+sayı2;  
}
```

Konu : Liste tanımlama

Kullanılan Dosya : C# => Kamplntro => Collections

Arraylar(Diziler) statik yapıda, list (listeler) dinamik yapıdadır. Arraylara sonradan anı bir değişken eklenemez. Ancak listelere ekleme yapılabilir

Using komut satırı eklenerek Class/metotu tanıtma :

Çalışılan Class 'ta bir kütüphane Metotunun veya başka katmandaki class'ın kullanılabilmesi/çağrılabilmesi için o Metot/Class 'in bulunduğu katmanın ismi using komut satırı ile çalışılan class'in en üstüne tanıtılmalıdır.

İlgili Class 'ta kodlara ekli/tanımlı olmayan bir kütüphane Metotunun veya başka katmandaki class'ın ismi yazılarak çağrılmak istenmesi durumunda, yazılan Metod/Class ismi tanınmadığı için altı kırmızı çizili olarak görünecek ve altında/solunda ampul/tornavida işaretini çıkacaktır. Ampul/tornavida üzerine tıklanıp açılan sekmeden ilgili Using komut satırı tıklanır ve o satır en üstte eklenerek ilgili Class' ta tanımlı hale gelir

Örneğin; programda List yapıları kullanılabacaksa **list** yazısı yazılıncaya göre **ampul** 'den "**using System.Collections.Generic**" komutu tıklanarak eklenir/çözülür. Bu komut satırının eklenmesiyle List yapısı artık tanımlanmıştır ve çağrılabılır.

```
string[] isimler = new string[] {"Engin", "Murat", "Kerem", "Halil" };  
Console.WriteLine(isimler[0]);  
Console.WriteLine(isimler[1]);  
Console.WriteLine(isimler[2]);  
Console.WriteLine(isimler[3]);
```

```

List <string> isimler2 = new List<string> {"Engin", "Murat", "Kerem", "Halil"};
Console.WriteLine(isimler2[0]);
Console.WriteLine(isimler2[1]);
Console.WriteLine(isimler2[2]);
Console.WriteLine(isimler2[3]);
isimler2.Add("Ilker");
Console.WriteLine(isimler2[4]);

```

Not: Komutları toplu Comment(Yorum satırı) yapma veya geri alma

Ctrl k + ctrl c ===== seçilen komutlar Comment haline gelir. Seçili satırların başına “//” işaretleri eklenir

Ctrl k + ctrl u ===== seçilen Commentlar komut haline gelir. Seçilen satırların başındaki “//” işaretleri kalkar

Kullanılan Dosya : C# => KamplIntro => GenericsIntro

- **Normal Class tanımlama**

```
class ClassAdı { }
```

- **Liste olarak Class tanımlama ve tipini program çağrıırken belirleme**
(Class şeklinde dinamik bir liste oluşturmuş oluruz ve yeni eleman eklemesi yapabiliriz)

```

class Mylist <T> // T belirlenen bir Tip olacak. Bu herhangi bir değişken
                  olabilir. Bu Class çağrırlarken belirlenir
{
    T[] items;

    // "Constructor" yapısı (ctor yazıp 2xtab a bas)
    public Mylist()
    {
        items = new T[0];
        // bu komutla Class hemen çalışmaya baslar ve 0 elemanlı bir dizi oluşur
    }
    // değişkenin tipi T şeklinde tanımlandı.
}

```

```

public void Add (T item)
{
    // gecici dizinin referansını itemin referansına atadım ki kaybolmasın
    T[] tempArray = items;

    //items listesinin genişliğini +1 yapıyorum (1 artırıyorum)
    //burada yeni bir referans numarasıyla eleman sayısı 1 fazla liste oluştur
    items = new T[items.Length +1];
    // eskiden var olan elemanları eleman sayısı 1 artmış olan yeni diziye alıyoruz
    for(int i = 0; i < tempArray.Length; i++)
    {
        items[i] = tempArray[i];
    }
    // şimdi listenin en sonuna istenen bilgiyi ekliyoruz
    items[items.Length - 1] = item;
}

```

Tanım: Constructor bir Class new'lenliğinde çalışan metottur

Constructor, kesinlikle bir metottur. Ama herhangi bir değer döndüren metot değildir. Ya da void metot olarak da düşünülemez. Constructor, yalnızca üyesi bulunduğu class'dan nesne üretimi sırasında çalışacak olan metoddur.

Constructor, classdan instance alınırken çalışır ve amacı, class üyelerinizin değerlerini ayarlayarak nesne referansına geçirir.

Enum= bir değişkenin alabileceği değerleri bu şekilde sınırlayabilir, tanıtabiliriz.

Arastirma Konusu: C# dilinde dictionary yapısı

Dictionary sınıfı içerisinde belirleyeceğimiz bir key' e uygun bir value saklayabilecek listeler oluşturmamızda kolaylık sağlar. Dictionary sınıfında key(anahtar) türümüzü belirtmek zorundayız (int, string, vb.). Kısaca " arama ve aranılana kolayca ulaşma " denirse de dictionary özetlenmiş olur. Tipik bir Dictionary tanımlaması aşağıdaki gibi olmaktadır :

```
Dictionarykelimeler= newDictionary();
```

Örnek olarak şehirler ve plaka kodlarıyla ekleme yapısını verilirse :

```
class Program
{
    public static void Main()
    {
        Dictionary ilPlaka=new Dictionary();
        ilPlaka.Add("Malatya",44);           // Yeni eleman için Add sözcüğü kullanıldı.
        ilPlaka.Add("Sinop",57); ilPlaka.Add("Edirne",22);
        ilPlaka.Add("Elazığ",23); ilPlaka.Add("Samsun",55);
        ilPlaka.Add("Ankara",06); ilPlaka.Add("İzmir",35);

        foreach( var element in ilPlaka)      //Burada foreach ile ilPlaka dolaşılıyor ve ekran yazdırılıyor.
        {
            Console.WriteLine(element);
        }
    }
}
```

Cıktı olarak ekranımıza :

Malatya, 44

Sinop 57

Edirne 22

Elazığ 23

Samsun 55

Ankara 06

İzmir 35 şeklinde gelecektir.

5.DERS (23.01.2021)

Konu ADI : INHERITANCE (Abstract Siniflar)

Kullanılan Dosya : C# =>KamplIntro =>OOP2

Tamamlayıcı Kaynak : Engin Demirog Youtube Sitesinde c# Dersleri No: 38-48

<https://www.youtube.com/watch?v=aomLseXFZeg&list=PLqG356ExoxZU5keiJwuHDpXqULLffwRYD&index=38>

https://www.youtube.com/watch?v=YdK6w8Swofc&feature=emb_logo

https://www.youtube.com/watch?v=6VYDltTF2b4&feature=emb_logo

INHERITANCE Nesneleri kategorize etmek maksadıyla kullanılır

Örneğin **müşteri** Classı yerine "**tüzelmüsteri**" ve "**gercekmüsteri**" şeklinde ayrı ayrı tanımlayabiliriz.

Her iki Class içinde aynı işlemi yapacaksak veya aynı değişkeni kullanacaksak ayrı ayrı değişken tanımlamak veya ayrı ayrı operation Classı yazmak yerine Base Class'ta tanımlanan ortak değişkenlerle ve burada tanımlanan **Operation Class** 'la işlem yapabiliriz. Bunun için bir 3.Class (base Class) oluştururuz ve diğer iki Class'in o tipte olduğunu belirtebiliriz. Ortak özellikleri de burada belirtebiliriz. Operasyon Class'ına da ebeveyn Class şeklinde gönderilebilir

Ebeveyn Class diğer 2 Class 'ında referans numarasını tutabilir. Böylece Ortak operasyon referansları tanımlanabilir ve oraya ebeveyn Class şeklinde gönderilebilir

Soyutlama önemlidir. **Abstract, Inheritance**

Not: Bir işlem yapılmayacak, sadece kayıt tutulacak değişkenler sadece rakam bile içerde **string** olarak tanımlamak daha faydalıdır

Not: Eğer ki bir nesnede bir değeri kullanmak zorunda değilse ama tanımlamışsan hata yapıyorsun demektir

Base(Temel)/ Ebeveyn Class

```
class Musteri // Ebeveyn Class
{
    public int Id { get; set; } //prop 2xtab
    public string MusteriNo { get; set; }
}
```

Derived (Türetilmiş) Class

```
class GercekMusteri : Musteri // (Inheritance) Tüzel müşteri bir müsteridir anlamındadır  
DIKKAT  
{  
    public string TcNo { get; set; }  
    public string Adi { get; set; }  
    public string Soyadi { get; set; }  
}
```

Derived (Türetilmiş) diğer Class

```
class TuzelMusteri : Musteri // (Inheritance Tanimlama) DIKKAT  
{  
    public string SirketAdi { get; set; }  
    public string VergiNo { get; set; }  
}
```

Ana program

```
Musteri musteri3 = new GercekMusteri(); // dikkat!!!! sol taraf Ebeveyn Class seklinde  
Musteri musteri4 = new TuzelMusteri(); // dikkat!!!! sol taraf Ebeveyn Class seklinde
```

```
MusteriManager.Ekle (musteri3); // operasyon Classina GercekMusteri sınıfından bir Class  
gönderildi
```

```
MusteriManager.Ekle (musteri4); // operasyon Classina TuzelMusteri sınıfından bir Class  
gönderildi
```

Ebeveyn Class diğer 2 Classin 'da referans numarasını tutabilir.

Böylece Ortak operasyon Classları tanımlanabilir ve oraya ebeveyn Class seklinde gönderilebilir

Örnek

Base Class adı: Yemek

Base Class Özellikleri: İçecek, Yemek; Tatlı

Türetilmiş Class 1: Sabah

İçecek: Cay, Kahve; meyve Suyu

Yemek: Omlet

Türetilmiş Class 1: Öğle

İçecek; Cay ,Kahve; meyve Suyu ve Alkollü İçecek

Yemek: Et Kavurma

Türetilmiş Class 1: Aksam

İçecek; Cay, Kahve; meyve Suyu ve Alkollü İçecek

Yemek: Et Kavurma

Konu : INTERFACE (POLYMORPHISM) Çok biçimlilik

Kullanılan Dosya : Visual Studio => C# => Kamplntro =>OOP2

Tanım: Ortak Class'ta yer alan bir özellik diğer tüm Class'larda kullanılacaksa ancak özelliğin kullanımında farklılıklar varsa kullanılır.

INTERFACE BIR REFERANS TUTUCUDUR

Interfaceleri birbirinin alternatifleri olan ancak kod içerikleri farklı olan durumlar için kullanırız. Bu örnekte kredi türlerinin hepsinden farklı şekilde de olsa hesaplamalar vardır. Bu hesaplama yöntemi değiştiği için C# program hesaplamasında Interface kullanılır. Interface Proje katmanları (DataAccess, Business vb.) arasında bağlantıda kullanmak için idealdir

Ebeveyn Class'ta Class tanımlamada **Class** kelimesi yerine **Interface** yazılır ve ortak işlem yapılacakta public yapısında süslü parantezler kaldırılır ve noktalı virgülle bitirilir. Interface class adı da "I" harfi ile başlar ki Interface olduğunu anlaşılır

Inheritance yalnız Interface ile yaparız. IKredimanager üzerine lamba işaretini çıkışınca implement tıklanınca Interface yapılan Class'taki ortak public yapısı çıkıyor. Buraya her birine özel hesaplama yöntemi yazılabilir.

Polymorphism → Bir başka Class'ta da yine diğer Classlar ve özellikleri bu şekilde kullanılabilir. Belirsiz sayıda (birden fazla) değişken kullanılacaksa **Liste yapısı** kullanmamız gerektiğini unutmayalım. Ve bu listeyi **foreach** komutu ile gezeriz

Interface/Ebeveyn CLASS

```
Interface IKrediManager // DİKKAT: Class tipini Interface olarak değiştirdik, Adı da "I" harfi ile başları  
{  
    public void Hesapla(); // süslü parantezi sildik ve noktalı virgül kullandık, Class 'ların içinde ayrı ayrı tanımlayacağız  
    public void BiseyYap(); // süslü parantezi sildik ve noktalı virgül kullandık, Class 'ların içinde ayrı ayrı tanımlayacağız  
}
```

Bir Operasyon Classı

```
class KonutKrediManager : IKrediManager // Inheritance yalnız Interface ile
//İkredimanager üzeine gelince lambda işaretü cikinca implement tıklanınca Interface yapılan Classtaki ortak public yapısı çıkıyor
{
    public void Hesapla()
    { Console.WriteLine("Konut kredisı ödeme Planı Hesaplandı"); } // Mesela burada farklı faiz oranıyla hesaplama yapılacak
}
```

Bir diğer Operasyon Classı

```
class İhtiyaçKrediManager : IKrediManager // Inheritance yalnız Interface ile
//İkredimanager üzeine gelince lambda işaretü çıkışınca implement tıklanınca sağlı Interface yapılan Classtaki ortaktaki public yapısı çıkıyor
{
    public void Hesapla()
    { Console.WriteLine("İhtiyaç kredisı ödeme Planı Hesaplandı"); } // Mesela burada farklı faiz oranıyla hesaplama yapılacak
}
```

Bir diğer Operasyon Classı (birden fazla diğer Classı kullanıyor)

```
class BasvuruManager
{
    public void BasvuruYap(IKrediManager krediManager)
        // burada değişkeni bu şekilde tanımlarsak tüm krediler için kullanabileceğimiz bir metod olur
    {
        krediManager.Hesapla(); // buraya hangi tip kredi gelirse onun hesapla metodu çalışır
    }

    public void KrediOnBilgilendirmesiYap(List<IKrediManager> krediler) //belirsiz sayıda kredi talebi olabilir
    {
        foreach (var kredi in krediler)
        { kredi.Hesapla (); }
    }
}
```

Ana Program

```
IKrediManager ihtiyacKrediManager = new IhtiyacKrediManager();  
ihtiyacKrediManager.Hesapla();
```

```
IKrediManager tasitKrediManager = new TasitKrediManager();  
tasitKrediManager.Hesapla();
```

```
IKrediManager konutKrediManager = new KonutKrediManager();  
konutKrediManager.Hesapla();
```

Konu : LOGLAMA

Kullanılan Dosya : Visual Studio => C# =>KamplIntro =>OOP2

TANIM: Loglama: kim ne zaman hangi operasyon kullandığının dökümüdür.

Logları bir dosyada veya bir veri tabanında tutabiliyoruz. Logları SMS olarak atabiliyoruz

Solution Explorer=>ADD=>CLASS=>INTERFACE secimi yap işlem yapmayan şablon bir metod oluştur

Solution Explorer=>ADD=>CLASS=> ile Interface class'i refere ederek şablon metodu çağırarak işlem tanımla oluştur

NOT: Bir Class aynı anda iki Interface Classe Interface edilebilir.

Tanım:Abstract kelimesinin Türkçesi soyut demektir. Abstract classlarda aslında interfacelerin bir değişigi gibidir abstract classlarda genelde projemizde base olarak kullanırız. interfaceler gibi buradan da nesne üretmemeyiz. içi Boş methodlar barındıran yani abstract methodlar barındıran classdır. Bir sınıfı abstract yapmak için abstract keywordunu kullanmamız gereklidir. İçerisinde abstract olmayan methodlarda barındırılabilir. Constructors bulunabilir.

<https://medium.com/software-development-turkey/abstract-class-ve-interface aras%C4%B1ndaki-farklar-nelerdir-3c0a4f956eba>

Abstract Class ve Interface Arasındaki Farklar

ABSTRACT CLASS	INTERFACE
Constructor içerebilir.	Constructor içermemelidir.
Static üyeler içerebilir.	Static üyeler içermemelidir.
Farklı tiplerde access modifier (erişim belirleyicisi) içerebilir. public, private, protected gibi.	Farklı tiplerde access modifier içermemelidir. Interface'te tanımlanan her metod default olarak public kabul edilir.
Sınıfın ait olduğu kimliği belirtmek için kullanılan. (is-a ilişkisi)	Sınıfın yapabileceği kabiliyetleri belirtmek için kullanılan. (can-do ilişkisi)
Bir sınıf sadece bir tane abstract sınıfı inherit edebilir.	Bir sınıf birden fazla interface'i inherit edebilir.
Eğer birçok sınıf aynı türden ve ortak davranışları sergiliyorsa abstract sınıfı base class olarak kullanmak doğru olacaktır.	Eğer birçok sınıf yalnızca ortak metodları kullanıyor ise interface'ten türetilmeleri doğru olacaktır.
Abstract sınıf metod, fields, contants vb. üyeleri içerebilir.	Interface yalnızca metod imzalarını içerebilir.
Türetilen sınıflar abstract sınıfı tamamen veya kısmi implemente edebilir.	Türetilen sınıflar interface'i tamamen implement etmek zorundadır.
Metod imzaları veya implementasyonları içerebilir.	Yalnızca metod imzalarını içerebilir.

5-gün dersine ait 5. Ödevi

Cevap Engin Demirog GITHub'ta

Bir oyun yazmak istiyorsunuz. Bu yazılım için backend kodlarını C# ile geliştirmeyi planlıyoruz. Yeni üye, satış ve kampanya yönetimi yapılması isteniyor. Nesnelere ait özellikleri istediğiniz gibi verebilirisiniz. Burada amaç yazdığınız kodun kalitesidir. **Ödevde gereksinimleri tam anlamadığınız durum benim için önemli değil, kendinize göre mantık geliştirebilirisiniz.**

Gereksinimler

1. Oyuncuların sisteme kaydolabileceği, bilgilerini güncelleyebileceği, kayıtlarını silebileceği bir ortamı simule ediniz. Müşteri bilgilerinin doğruluğunu e-devlet sistemlerini kullanarak doğrulama yapmak istiyoruz. (E-devlet sistemlerinde doğrulama TcNo, Ad, Soyad, DoğumYılı bilgileriyle yapılır. Bunu yapacak servisi simule etmeniz yeterlidir.)
2. Oyun satışı yapılabilecek satış ortamını simule ediniz. (Yapılan satışlar oyuncu ile ilişkilendirilmelidir. Oyuncunun parametre olarak metotta olmasını kastediyorum.)
3. Sisteme yeni kampanya girişi, kampanyanın silinmesi ve güncellenmesi imkanlarını simule ediniz.
4. Satışlarda kampanya entegrasyonunu simule ediniz.

6.DERS (27.01.2021)

Konu : **SQL (Structered Query Language)**

Tamamlayıcı Kaynak : <https://www.w3schools.com/sql/default.asp>

Tanım: Veri tabanı işlemleri için (depolama, erişme, işleme, değerlendirme) kullanılan bir alt dil olarak da bilinir

Veri işlemlerinde SQL ve EXCEL PHYTON'dan daha çok kullanılır ve tercih edilir.

SQL sistemi ile sorgulama yapan yazılımlar

2 tür Sistem vardır. Yerine göre aynı anda kullanılabilir. Birbirini alternatifî diye tanımlamaktan ziyade tamamlayıcı demek daha doğru olur

1. RDBMS (Relational Database Management System): Oracle, SqlServer, MySql, Postgresql, Access

PISql: Oracle kullanır

T-Sql: SqlServer

Sql standartları : ANSI/ISO

(ANSI) American National Standards Institute in 1986,

(ISO) International Organization for Standardization in 1987

2. NoSql : Not Only Sql

Yazılımlar MongoDb, Firebase

Normalisation: data modelleme tekniği

Customers

	A	B	BC	D	E	F	G
1	Id (Pk)	CityId (Fk)	FirstName	LastName			
2	1	6	Engin	Demirog			
3	2	34	Kerem	Demiralay			
4	3	34	Merve	Cetindag			
5							

Not: Bir veri dosyasında ilk sütun her zaman Id olur. Sonrasında ise başka veri tabanından tipleri alınan (alabileceği değer) değişkenler yer almalıdır

Her veri tabanında 1 kolon **PK (Primary Key)** olur ve bu değer sadece biri alır. Başka değerlerde aynı şekilde ise onları **unique** yaparsın ve ayrıca bir PK olarak tanımlarsın

FK: Foreign Key: Değişkenlerin değeri bir başka veri dosyasından geliyorsa (**Burada Category ID**)

Cities (Customers bilgileri için City seçiminin belirlenmesi, hatanın önlenmesi için)

	A	B	C	D	E	F	G
1	Id	CityId	Name				
2	1	1	Adana				
3	6	6	Ankara				
4	34	34	Istanbul				
5							

Product

	A	B	C	D	E	F	G
1	Id	Name	UnitPrice				
2	1	Elma	10				
3	2	Mandalina	7				
4	3						
5							

Orders

	A	B	C	D	E	F	G
1	Id	CustomerId	Date				
2	1	1	27.01.2021				
3	2	3	27.01.2021				
4	3						
5							

OrderDetails bir siparişte birden fazla ürün sipariş edilmiş olabilir (orderId aynı olanlar aynı siparişte yer almaktadır)

	A	B	C	D	E	F	G
1	Id	OrderId	ProductId	Price			
2	1	1	1	6			
3	2	1	1	7			
4	3						
5							

Not: Udemy'de Engin Demirog'un ücretsiz veri tabanı eğitimi vardır

Not: iyi bir Class içerisinde sadece 1 metot olmalıdır

Şehir adlarında yazım hatalarının önüne geçmek için ayrı bir veri dosyasında tuttuk. Peki aynı ad soyada iki insan olması engellenmelidir. Aynı müşteri değilse sorun yoktur zaten. Bunu yapmak imkânsız olabilir. Risk yönetimi yapıp bunu ihmali edebiliriz. Ya da **TCKimlikNo** ile **unique** olması sağlanabilir

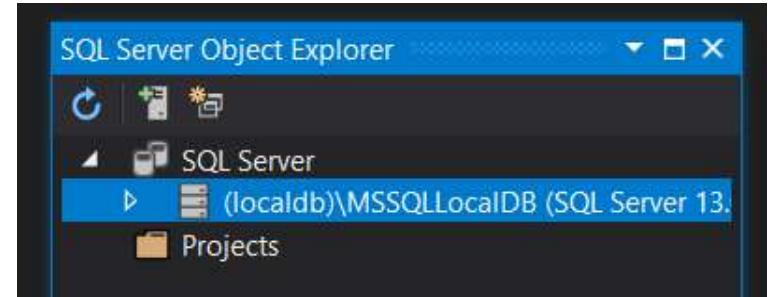
Kurulum: Bir SQL kodu netten alıp Visual Studio programında açacağız ve koşturup veri tabanının olmasını sağlayacağız. Projemizde o veri tabanının üzerinde çalışacağız (Northwind sample database)

Step-1: <https://github.com/microsoft/sql-server-samples/blob/master/samples/databases/northwind-pubs/instnwnd.sql>

Linkde “view raw” kısmına basılıncı görülen komutları kopyalayıp Visual Studio programında açalım

Step-2: Visual studio programı açıldığında “**Kodsuz devam et**” seçeneği tıklanarak işleme ekranın gelmesi sağlanır.

Step-3: “view” bölümünde SQL Server Objekt Explorer açılır



Step-4: Sağa tıklayıp dosya açalım (New Query) ve kopyaladığımız kodları buraya yapıştırıp koşuralım. (**SQL üst sekmesinden koşturma yapılması gereklidir**)

Commands completed successfully yazması lazım

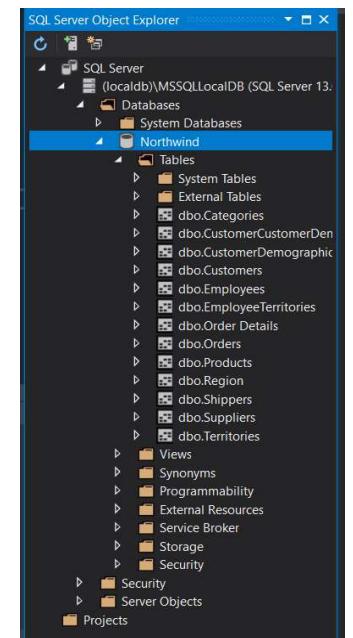
Step-5: **SQL Server Objekt Explorer** dan **databases** bölümü **refresh** yapıldığında **northwind** veri dosyaları ekrana gelir

Step-6: **SQL Server Objekt Explorer** da **Nortwind** yazısında sağa tıklayıp “**new Query**” tuşuna basarsak kod yazabileceğimiz bölüm açılır

Dikkat: Kodları yazıp run (üçgen) tuşuna basarsak komutlar çalışır (dikkat SQL query başlığının altındaki run tuşuna basmalıyız)

SQL 'de Yorum satırı 2 adet çizgi yazıldıktan sonra olur **--yorum satırı**

Komutlar ((ANSI Standartları ,Tüm SQL programlarında çalışır))



Some of The Most Important SQL Commands

- **SELECT** - extracts data from a database
- **UPDATE** - updates data in a database
- **DELETE** - deletes data from a database
- **INSERT INTO** - inserts new data into a database
- **CREATE DATABASE** - creates a new database
- **ALTER DATABASE** - modifies a database
- **CREATE TABLE** - creates a new table
- **ALTER TABLE** - modifies a table
- **DROP TABLE** - deletes a table
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index

Komut: Veri dosyasından işlem yapmak için data seçmek (**select**)

`select * from Customers` -- bu şekilde tüm değerleri gösterir

`select ContactName, CompanyName, City from Customers` -- bu şekilde sadece seçilenler gösterilir

`select ContactName Adi, CompanyName Sirketadi, City sehir from Customers` -- bu şekilde sadece seçilenler bizim belirttiğimiz isimlerle (ALIAS) gösterilir

Komut: Filtreleme (**Select....where** komutları) - Özellige göre (sütun adına göre)

`select * from Customers where City = 'London'` -- bu şekilde sadece Customer veri dosyasından Şehir adı 'London' olanlar gösterilir

`select * from Products where CategoryId =1 or CategoryId =3` --bu şekilde sadece Product veri dosyasından CategoryId 1 ve 3 olanlar gösterilir

`select * from Products where CategoryId =1 and UnitPrice>10`

Komut: **Sıralı Filtreleme (Selectorder by komutları)**

`select * from Products order by UnitPrice desc` -- descending (azalan) ve ascending (artan) default asc dir.

`select * from Products where CategoryId =1 order by UnitPrice desc` -- seçili ve sıralı filtreleme

Komut : **Adet sorgulama (select....count) – istenen nitelikteki veri sayısı sorgulanabilir**

`select count (*) from Products` -- ürün sayısını gösterir

`select count (*) UrunAdeti from Products where CategoryId= 1` -- ürün sayısını gösterir. Sütun adı **ürün adeti** olarak gözükmür

`select CategoryId from Products group by CategoryId` --- listedeki kategorinin her bir değerini göster demek (tekrarsız)

Komut Gruplama yaparak filtreleme (group by)

`select CategoryId, count(*) from Products group by CategoryId` --- listedeki kategorinin her bir değerini göster demek (tekrarsız)

`select CategoryId, count(*) from Products group by CategoryId`--- Hangi kategoride kaç farklı ürün var

Komut: **having (belirli değişkene belirlenen değerlere sahip olanların filtrelenmesi)**

`select CategoryId, count(*) from Products group by CategoryId having count (*)<10`
-- <10 olan ürünleri kategori gruplarına göre kaç farklı ürün var

`select CategoryId, count(*) from Products where UnitPrice>20 group by CategoryId having count (*)<10` ----

Komut: Select Distinct (sadece farklı olanların listelenmesi)

The SELECT DISTINCT statement is used to return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

```
SELECT DISTINCT Country FROM Customers;
```

--Bu şekilde Customers veri dosyasında sadece farklı ülke kayıtları listelenir (her ülkeden 1 kayıt)

```
select Count (Distinct Country) from Customers --Bu şekilde Customers veri dosyasında kaç adet farklı ülke kayıtlı olduğu öğrenilir
```

Komut Komut: Insert Into (Dosyaya yeni giriş)

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
```

Komut : NULL Operator

The IS NULL operator is used to test for empty values (NULL values).

The following SQL lists all customers with a NULL value in the "Address" field:

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NULL;
```

Komut: UPDATE Table

The following SQL statement updates the first customer (CustomerID = 1) with a new contact person *and* a new city.

```
UPDATE Customers  
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'  
WHERE CustomerID = 1;
```

Komut :Delete

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

Komut: TOP, LIMIT and ROWNUM Examples

İstenen dosyanın istenen sayıda verisi görüntülenir. İlk 3 komut türü de aynı işlevi görür

The following SQL statement selects the first three records from the "Customers" table (for SQL Server/MS Access):

```
SELECT TOP 3 *FROM Customers;
```

The following SQL statement shows the equivalent example using the LIMIT clause (for MySQL):

```
SELECT * FROM Customers  
LIMIT 3;
```

The following SQL statement shows the equivalent example using ROWNUM (for Oracle):

```
SELECT * FROM Customers  
WHERE ROWNUM <= 3;
```

Komut: istenen dosyanın istenen yüzdede verisi görüntülenir

```
SELECT TOP 50 PERCENT * FROM Customers;
```

Komut: Aşağıdaki 3 komutta aynı işlevi görür: istenen dosyada **istenen özelliğe** **haiz** istenen sayıda veri gözükmür.

The following SQL statement selects the first three records from the "Customers" table, where the country is "Germany" (for SQL Server/MS Access):

```
SELECT TOP 3 * FROM Customers  
WHERE Country='Germany';
```

```
SELECT * FROM Customers  
WHERE Country='Germany'  
LIMIT 3;
```

```
SELECT * FROM Customers  
WHERE Country='Germany' AND ROWNUM <= 3;
```

Not: SQL ve İlişkisel Veri tabanı tasarımları öğrenilmelidir*****

Konu: Joint (tabloları birleştirme)

Belirlenen değeri uyusan tabloları birleştirme

Select (Sergilenmesi istenen değişkenler)

from (birleştirilecek tablo_1) **inner join** birleştirilecek tablo_2

on iki tabloda yer alan ve karşılaşılacak değişken

where koşul

Select *

from Products inner join Categories

on Products.CategoryId = Categories.CategoryId

NOT: birden fazla kelimelek ve ayrı yazılmak istenen başlıklar köşeli parantez içinde gösterilir

select*

from Products p **inner join** [Order Details] od

on p.ProductId = od.ProductId

Not: **inner Join** sadece iki tabloda da yer alıp eşleşen değerleri gösterir

Solda olup sağda olmayanları da göster anlamında **left join** komutu vardır. Tam tersi olanlar için **right join**

select*

from Products p **left join** [Order Details] od

on p.ProductID = od.ProductID

-- Hiç sipariş vermemiş olanları listeledik

```
select * from Customers c left join Orders o  
on c.CustomerID = o.CustomerId  
where o.CustomerId is null
```

DTO-Data Transformation Objekt

İkiden fazla tabloyu birleştirmek

```
select*  
from Products p inner join [Order Details] od  
on p.ProductId = od.ProductId  
inner join Orders o  
on o.OrderId = od.OrderId
```

diğer komutlar

in

distinct

union

DIKKAT: SQLServer ≠ SQL

DİKKAT: 5-gün dersine ait 5. Ödevi hoca yaptı ve GITHub'a ekledi

Ödev-2: Her bir üründen toplamda ne kadar para kazandığımızı bulunuz.

İpucu : Group by kullanılacak

İpucu : Products, Orders, Order Details tabloları join edilecek.

İpucu : Sum kullanılacak.

Sonuç aşağıdaki formatta olmalıdır.

Ürün Adı, Kazanılan Toplam Miktar

Cevap (kodlama.io yorumlardan bakabilirim)

```
SELECT Products.ProductName AS 'Ürün Adı',
SUM([Order Details].UnitPrice*[Order Details].Quantity) AS 'Kazanılan Toplam Miktar'
```

FROM Products

INNER JOIN [Order Details]

ON Products.ProductID = [Order Details].ProductID

INNER JOIN Orders

ON Orders.OrderID = [Order Details].OrderID

GROUP BY Products.ProductName

Sinan Fen 4 hours ago

```
select p.ProductName as 'Ürün Adı', sum(od.quantity * od.unitprice) as 'Kazanılan Toplam Miktar'
from Products p inner join [Order Details] od
on p.ProductID = od.ProductID
inner join Orders o
on od.OrderID = o.OrderID
group by p.ProductName
```

Konu:

SQL'de veri tabanı oluşturmak

Visual Studio 'da **View=>Sql Server Object Explorer** açılır

Databases üzerinde **sağ** tıkla **Add new Database** seçilir ve oluşturulmak istenen Database'e İsim verilir.

1.Yöntem: **Tables** üzerinde **sağa** tıklayıp “**Add new Table**” yeni bir **OperationClaims** adlı Tablo oluşturacağız

ID üzerinde sağ tıklayıp açılan pencerede “**Id Specification**” altında yer alan “**Is Identity**” özelliğini **True** yapmalıyız (Bu özellik Id'nin sırayla birer artmasını sağlar)

OperationClaims adını sol alta iki köşeli parantez arasına yazarız

Sol üstteki **Update** tuşıyla oluştururuz

2.Yöntem: Oluşan **Database** üzerine **sağ** tıklanır “**new Query**” seçilir

Açılan SQL penceresine eklemek için gerekli kodlar yazılır ve **SQL** altındaki **Execute** Tuşuna basılır

```
Create table Brands(
    BrandId  int Primary key IDENTITY(1,1),
    BrandName varchar(50)
)

Create table Colors(
    ColorId  int Primary key IDENTITY(1,1),
    ColorName varchar(20)
)

Create table Cars(
    CarId int PRIMARY KEY IDENTITY(1,1),
    BrandId int,
    ColorId int,
    ModelYear int,
    DailyPrice decimal,
    Descriptions varchar(200),
```

```
        Foreign key(ColorId) References Colors(ColorId),  
        Foreign key(BrandId) References Brands(BrandId)  
)
```

```
CREATE TABLE Customers(  
  
    Id int PRIMARY KEY IDENTITY(1,1),  
    UserId int,  
    CompanyName nvarchar(50),  
    FOREIGN KEY (UserId) REFERENCES Users(Id)  
)
```

```
CREATE TABLE Rentals(  
  
    Id int PRIMARY KEY IDENTITY(1,1),  
    CarId int,  
    CustomerId int,  
    RentDate datetime,  
    ReturnDate datetime,  
    FOREIGN KEY (CarId) REFERENCES Cars(CarId),  
    FOREIGN KEY (CustomerId) REFERENCES Customers(Id)  
)
```

//DİKKAT!!! **nvarcahr** yerine **varchar** kullanmak daha performanslı. Çünkü **varchar** kullanmadığımız karakter alanlarını boş bıraktığı halde, **nvarchar** onu boşluklarla doldurarak ekstradan bellek kaplamasına sebep oluyor

```
Insert into Brands(BrandName) values ('Audi'),('BMW'),(3,'Hyundai'),(4,'Mitsubishi'),(5,'Nissan'),(6,'Mazda'),(7,'Porsche');
```

```
Insert into Colors(ColorName) values ('Black'),('White'),('Silver'),('Blue'),('Red'),('Brown'),('Green');
```

```
-- Esas tabloda Primary key değişkenine değer atayamazsin
Insert into Cars(BrandId,ColorId,ModelYear,DailyPrice,Descriptions) values
(1,5,2020,450,'AUDI Q8 - Red'),
(2,2,2018,370,'BMW 2 Gran Coupé - White'),
(3,4,2015,250,'HYUNDAI i10 - Blue'),
(4,7,2016,290,'Mitsubishi Outlander - Green'),
(5,6,2017,350,'NISSAN QASHQAI - Brown'),
(6,3,2019,630,'MAZDA CX-5 - Silver'),
(7,1,2021,720,'PORSCHE P911 Turbo S - Black');

Insert into Users(FirstName, LastName,Email,Password) values
('Ahmet','Zimba', 'ahmet@gmail.com', '12345678 '),
('Mehmet ','Caglayan', 'mehmet@gmail.com','87654321'),
('Cevdet','Tasdelen', 'cevdet@gmail.com', 'sffgrrr');

Insert into Customers(UserId, CompanyName) values(1,'Coskunlar'),(2,'Caglayan'),(3,'Tasdelen');

//calisolmali tarih formatina?????????????????????ß
Insert into Rentals (CarId, CustomerId,RentDate) values (1,1,01.02.2021),(2,2,01.02.2021),(7,3,05.02.2021);
```

7.DERS (30.01.2021)

Konu : Proje Katmanları (Backend)

Ana Katman	Maksadı	Açıklama
Data Access	Veri erişim yöntemleri	Veri erişim yöntemlerini de birden fazla kısmı halinde tasarlamalıyız. Böylece, Değişiklik yapacağımız zaman sadece Data Access kısmında istenen kişim değiştirirsek veri etkilenmeden ya da diğer erişim yöntemleri etkilenmeden program çalışır. İşlem kolaylığı olur. Ana Katmanlardan sadece Business ile etkileşimde bulunabilir. API Katmanı ile etkileşimde bulunmamalıdır.
Business	Kuralları yazdığımız yerdir	Data Access ile Business kısmına yapılacak kısımları iyi ayırmalıyız. Hem DataAccess hem de API ile etkileşimde bulunabilir
API	Kullanılan ara servis katmanı	IOS/ Android ara yüzlerini yapsak bile Business ile direk görüşüremeyiz. Bir standart kullanmalıyız. Service katmanı kullanmalıyız. Ana Katmanlardan sadece Business ile etkileşimde bulunabilir. DataAccess Katmanı ile etkileşimde bulunmamalıdır.

Yardımcı Katmanlar	Maksadı	Açıklama
Entities	Verilerin karşılığı özelliğ tutan Class'lar	Veri tabanındaki tabloların varlıkların karşılığı, özellik tutan klasları bu katmanda tutarız. Tüm katmanlarla etkileşim halindedir. Örneğin bir ürünü DataAccess veri tabanından alır, ekler, Business ürünün uygunluğunu kontrol eder. API kullanıcıya sunar veya kullanıcılardan alır
Core	Aynı isi gören işlerin genel haliyle yazılıarak tutulduğu katman	Aynı isi gören kodları sürekli yazmak yerine evrenselleştirmek gereklidir. Bu kodlar her yerde kullanmaya imkân sağlamak üzere burada tutarız. Bu kodlar diğer projelerde de kullanılabilen yapıda olmalıdır. Büyük şirketlerde sadece Core katmanın hazırlayan yazılımcı ekip mevcuttur. Bu şekilde Core kodları farklı bir proje altında kayıt altında tutulur.

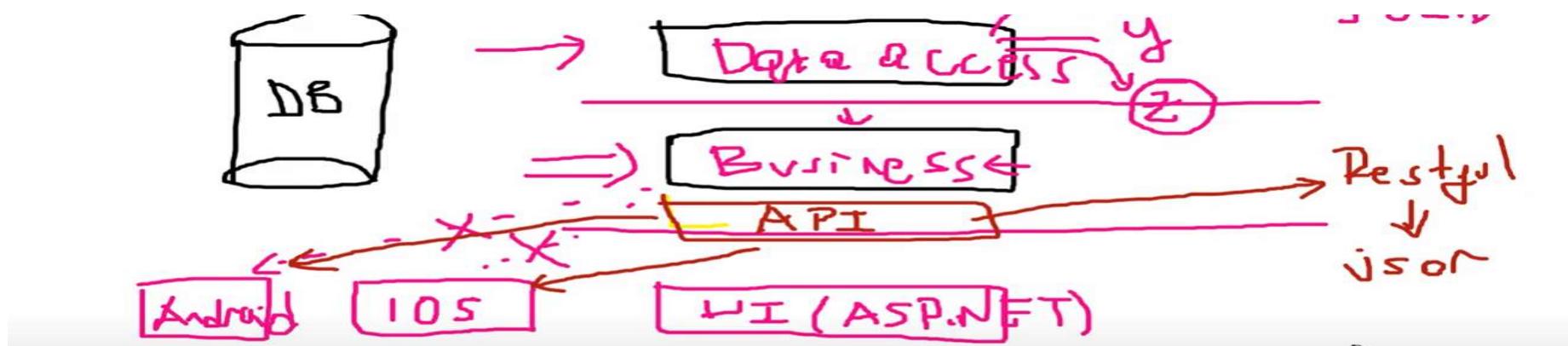
Alt Katmanlar	Maksadı	Açıklama
Concrete	iş yapan asıl Class'lar yer alır	
Abstract	Katmanların birbirine etkileşimi sağlayan Interfaceler yer alır	Katmanlar interface'le üzerinden Dependency İnjektion ile birbirleriyle etkileşimde bulunurlar. Constructor'da ona göre bir bağlantı kuruldu. Böylece sıkı bir bag yerine gevsek bir bag kurulur ve yöntem/kural degistiginde sadece bağlantıyi değiştirmek yeterli olur

Frontend kismı : Örnegin Android, IOS, Angular/React/Web Sayfasi, Swing

Katmanlar ne işe yarar. Doğru ve standart klasörleme ile kolay tasarım, daha anlaşılır ve bilgiler daha erişilir olur. Örneğin Foto klasöründe 2013 yılında çekilen bir fotoğrafı başka yila ait klasörün altına kaydedersen ona erişmek daha zor olur.

Katmanların işleyişi hakkında bir anlatım: Kullanıcı API ile Business'e ürün eklemek için istekte bulunur. API ile ürün ekle sayfasına ürününe ait bilgileri girdi ve ekle butonuna bastı. **Business** ürünün belirlenmiş kriterde uygunluğunu test etti (**validation**). Uygun değilse API'ye geri bildirim yaptı. Uygun ise DataAccess emir kulu olarak ürün bilgilerini veri tabanına ekledi ve Business'e ürünün eklendiği bilgisini verdi. Bir projede API ile DataAccess iletişimde bulunmamalıdır. Bulunuyorsa o proje yanlış tasarlanmış demektir.

Örnegin DataAccess hibernate, JDBC vb. veya baska bir teknik kullanırsa business katmani da ayni tekniği kullanır .



Backend Proje Oluşturma

Kullanılan Dosya : Visual Studio => Project => MyFinalProject

Step-1: Visula Studio'da yeni proje (katman) oluşturacağız.

Burada "Blank Solution" ile yeni kod blokları açacağız. (daha önce Console.app den açtıyorduk)

Adı *MyFinalProject* olsun

Step-2: Açıtığımız Projeye **DataAccess** adlı yeni proje (katman) (Class.Library olarak) ekleyeceğiz. İçine 2 klasör oluşturacağız.

Sağ tarafta **Solution** üzerinde sağ tıklayıp **add=> new project** seçilir. Sonra **class library.net standart** seçilmelidir.

Adı **DataAccess** olsun

DataAccess kısmında Sağ tarafta oluşan Class1 adlı kısmı (uyduruk) silelim.

Sağ tıklayıp yeni klasör ekle'den **Abstract** ve **Concrete** adlı 2 klasör oluşturalım

Step-3: Açıtığımız Projeye **Business** adlı yeni proje (katman) (Class.Library olarak) ekleyeceğiz. İçine 2 klasör oluşturacağız.

Yeni bir **proje** oluşturalım. Aynı şekilde oluşturalım ve adına **Business** verelim

Business kısmında Sağ tıklayıp oluşan Class1 adlı kısmı (uyduruk) silelim.

Sağ tıklayıp yeni klasör Ekle'den **Abstract** ve **Concrete** adlı 2 klasör oluşturalım

Step-4: Açıtığımız Projeye **Entities** adlı yeni proje (katman) (Class.Library olarak) ekleyeceğiz. İçine 2 klasör oluşturacağız.

Yeni bir **proje** oluşturalım. Aynı şekilde oluşturalım ve adına **Entities** verelim

Entities kısmında Sağ tarafta oluşan Class1 adlı kısmı (uyduruk) silelim.

Sağ tıklayıp yeni klasör Ekle'den **Abstract** ve **Concrete** adlı 2 klasör oluşturalım

Step-5: Açığımız Projeye **ConsoleUI** adlı yeni proje (katman) (**Console.App** olarak) ekleyeceğiz.

Yeni bir proje oluşturalım Aynı şekilde değil. Bu sefer **Console** seçelim ve adına **ConsoleUI** verelim

ConsoleUI “yi Set a startup projekt seçelim

Step-6: **Entities**.projesi **Concrete** klasörü içine **Product ve Category** adlı 2 adet yeni Class (sağa tıklayıp add) ekleyelim.

Product Class’ı **public** yapalım. Bu Projedeki tüm Class’lar public yapılacak.

Entities tüm Domaine hizmet edecek değişkenlerdir

```
public class Product //public yapalım. default internaldir.  
{  
    public int ProductId { get; set; }  
  
    public int CategoryId { get; set; }  
    public string ProductName { get; set; }  
  
    public short UnitsInStock { get; set; }  
  
    public decimal UnitPrice { get; set; }  
}  
  
public class Category //public yapalım. default internaldir.  
{  
    public int CategoryId { get; set; }  
  
    public string CategoryName { get; set; }  
}
```

Step-7: **Entities**, **Abstract** kısmına sağ tıklayıp **IEntity** adlı yeni Class (**Interface olarak**) ekleyelim **Interface Public** yapalım.

```
public Interface IEntity
```

Step-8: `Category` ve `Product` klaslarını da `IEntity` ile inherit edelim. Class'in en üstüne aşağıda komutu ekleyelim. Ampul gelirse onla seçelim

```
using Entities.Abstract;

public class Category:IEntity //public ve Interface.

public class Product:IEntity //public ve Interface.
```

Step-9: `DataAccess` projesine `Abstract` kısmında `IProductDAL` adlı Interface'ni oluşturalım. Bu Interface'e Ekle, güncelle ve Sil metotları ekleyelim

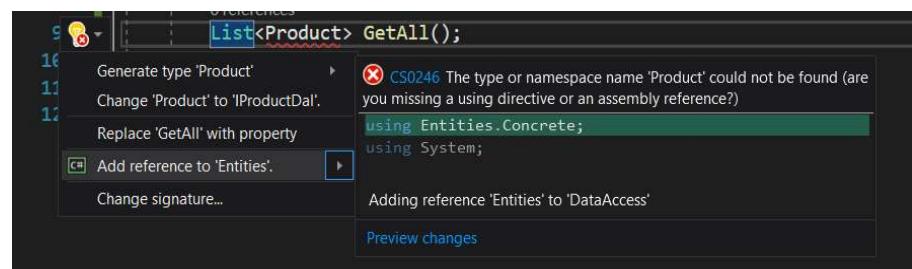
Adı : `IProductDAL (DAL =>Data Access Layer)`

Burası; Product ile ilgili veri tabanında yapacağım işlemlerin Interface'i

Liste Tanımlamasında çıkan Ampülden Entities'e referans eklemeli ya da (aynı işlem). `DataAccess` sağ tıkla add `ProjektReference` kısmında **Entities** seçmeliyiz Ardından ampulden **UsingReference** seçmeliyiz (en üsté komut ekler). Gereken metotları çağır

```
namespace DataAccess.Abstract
{
    public Interface IProductDal // public yapalım. Default internaldir. Interface'in operasyonları ise default publictir
    {
        List<Product> GetAll();
        void Add(Product product);
        void Update(Product product);
        void Delete(Product product);

        List<Product> GetAllByCategory(int categoryId); //ürünleri CategoryId ye göre liste
    }
}
```



Step-10: **DataAccess** 'te **Concrete** klasörüne. **InMemory** ve **EntityFramework** adlı klasörleri (sağ tıklayıp) oluşturalım.

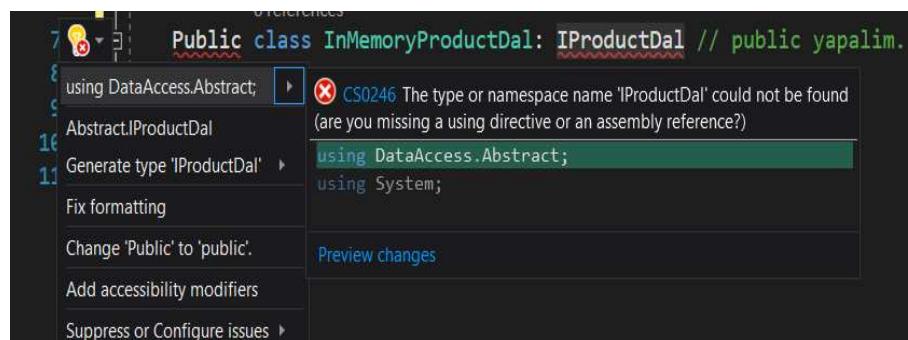
EntityFramework adlı klasöründe daha sonra çalışacağımız

InMemory klasöründe **InMemoryProductDal** adlı bir public Class ekleyelim.

IProductDal Interface ile ilişkilendirelim:

Çıkan Ampülden “Using” komut satırının en üstte eklenmesini sağlayalım

Çıkan Ampülden “Implement” ile ilgili metodların eklenmesini sağlayalım



```
namespace DataAccess.Concrete.InMemory
{
    Public class InMemoryProductDal: IProductDal // public yapalim. Default internaldir. IProductDal Interface ile ilişkilendirelim
    {
    }
}
```

Step-11: **InMemoryProductDal** adlı Class içinde metodların dışında product listesi oluşturacağız

Burada Product ekle, sil, güncelle vb. metodları buraya özgü tanımlayacağız.

ÖNEMLİ: Artık Foreach ile bir liste elemanı bulup işlem yapmak yerine **LISTQ** yapısını kullanacağız

Step-12: **Business Abstract** klasöründe **IProductService** adlı Interface ekleyelim.

Business de **Add** kısmından **ProjectReference** kısmında **DataAccess** ve **Entities** seçeceğiz

Böylece Bu Interface'da yazmak istediğimiz Liste için ampulden **using** komutu ekleyebiliriz. Ya da kendimiz en yukarı yazabiliriz

```
using Entities.Concrete;
```

Step-13: **Business Concrete** klasöründe **ProductManager** adlı **Class** ekle. Bu Classı **IProductService** Interface ile inherit et:

Ampulden **using** komut satırını ekleyeceğiz

Implement edeceğiz (**Interface** da tanımlı metodlar yazılması için bu **Classta** ortaya çıkacak)

Not: bir is sınıfı başka sınıfları new'lemez. Onun yerine kendisi alt çizgi ile yeni bir Class tanımlar.

DependencyInjection yöntemi= Temel olarak bağımlılıkların kontrolü ve yönetimi için kullanılmaktadır. Dependency Injection tekniğinde bağımlılık oluşturacak parçalarının ayrılip, bunların sisteme dışarıdan verilmesi (enjekte edilmesi) ile meydana gelir.

Bir katman diger bir katmandaki ilgili/gerekli yapının sadece referans tutucu olan abstract kısmındaki yapıya, yani Interface'ine baglanmalıdır

Sonra ampulden using satırını ekler ve **generate constructor** yaparak burada çalışacağın bir Class üreten metot oluşturur.

Step-14: Ana programdan çalışma (**ConsoleUI**) // Set a StartUp Project Yapmıştık

ConsoleUI de Add kısmından **Project Reference** kısmında hepsini (3 ünү de) seçeceğiz.

Kullanacağımız Veri kaynağını tanımlayacağız.

İstenen işlemi yapacağız (şimdilik ürün adları yazdırıldı)

```
class Program
{
    static void Main(string[] args)
    {
        //newleme yapacağız. Daha sonra gerek kalmayacak
        ProductManager productManager = new ProductManager(new InMemoryProductDal());
        foreach (var product in productManager.GetAll())
        {
            Console.WriteLine(product.ProductName);
        }
    }
}
```


No	Yeni Proje Oluşturma
1	Visula Studio'da MyFinalProject adlı yeni proje oluştur. Burada "Blank Solution" template ile yeni proje aç. (Daha önce Console.app den açıyordu)
2	Projeye DataAccess adlı yeni proje (Class.Library template) ekle. İçine Abstract ve Concrete adlı 2 klasör oluştur.
3	Projeye Business adlı yeni proje (Class.Library template) ekle. İçine Abstract ve Concrete adlı 2 klasör oluştı.
4	Projeye Entities adlı yeni proje (Class.Library template) ekle. İçine Abstract ve Concrete adlı 2 klasör oluştur.
5	Projeye ConsoleUI adlı yeni proje (Console.App template) ekle. ConsoleUI "yi Set a startup projekt seç
6	Entities ,=> Concrete klasörü içine Product ve Category adlı 2 adet yeni Class ekle.
7	Entities => Abstract klasörüne sağ tıklayıp IEntity adlı yeni public Interface ekle.
8	Category ve Product klaslarını da IEntity ile Interface ile inherit et:
9	DataAccess => Abstract kısmında ProductDAL adlı public Interface ekleyelim. Ekle, güncelle ve SİL metotları ekleyelim DataAccess add kısmında ProjektReference kısmında Entities seç
10	DataAccess => Concrete klasörüne. InMemory ve EntityFramework adlı klasörleri ekle. EntityFramework adlı klasöründe daha sonra çalışacağız. InMemory klasörüne InMemoryProductDal adlı bir public Class ekleyelim. IProductDal Interface ile ilişkilendir:
11	InMemoryProductDal adlı Class içinde metodların dışında product listesi oluşturacağız Burada Product ekle, sil, güncelle vb metodları buraya özgü tanımla. ÖNEMLİ: Artık Foreach ile bir liste elemanı bulup işlem yapmak yerine LISTQ yapısını kullan
12	Business=> Abstract klasöründe IProductService adlı Interface ekle. Business de Add kısmından ProjectReference kısmında DataAccess ve Entities seç
13	Business=> Concrete klasöründe ProductManager adlı Class ekleyelim Bu Classı IProductService Interface ile inherit et Burada istenen is kodlarını metota yaz. Not: bir iş sınıfı başka sınıfları new'lemez. Onun yerine kendisi alt çizgi ile yeni bir Class tanımlar
14	Ana programdan çalışma (ConsoleUI) ConsoleUI de Add kısmından Project Reference kısmında hepsini (ucunu de) seç. Kullanacağımız veri kaynağını tanımla. İstenen işlemi yap (şimdilik ürün adları yazdırıldık)

7.Ders Sonu Projenin Durum Tablosu

No	Projekt/Template	Projekt Reference	Klasör	Class/Interface	İşlem
1	MyFinalProject (Blank Solution)	---			
2	DataAccess (Class.Library)	Entities	Abstract	IProductDAL	Ekle, güncelle ve Sil metotları çağır.
			Concrete		
			Concrete- InMemory	InMemoryProductDal	IProductDal ile inherit et ekle, sil, güncelle vb metotları tanımla
			Concrete- EntityFramework		
3	Business (Class.Library).	DataAccess Entities	Abstract	IProductService	
			Concrete	ProductManager	IProductService ile inherit et istenen is kodları metodu tanımla.
4	Entities (Class.Library) .	---	Abstract	IEntity	
			Concrete	Product ve Category	IEntity ile inherit et.
5	ConsoleUI (Console.App)	Business DataAccess Entities	Set a startup project seç Kullanacağımız veri kaynağını tanımla. İstenen işlemi yap. Metotları çağır (örnek:ürün adları yazdır)		

Entities tüm Domaine hizmet edecek değişkenlerdir. Entities elemanları kullanacağımız katmanlarda tanımlarız.

Konu : Linq

Kullanılan Dosya : C# => CSharpIntro => LinqProject

Eğitim Video Linki: https://www.youtube.com/watch?v=uGcxmmwYLO8&feature=emb_logo

(Tanım) Linq: Hem Veri tabanı üzerinde ve hem kendi verilerimize erişmeye ve çalışmaya imkân veren kütüphanedir. EntityFramework ile beraber çalışır

Step-0: CSharpIntro adlı bir proje oluşturdum. Konu çalışmalarını bu projenin altında yapacağım

Step-1: LinqProject adlı yeni bir **console.Net** proje oluşturalım.

Program kısmına **Product** ve **Category** adlı iki class oluşturalım

```
class Product
{
    public int ProductId { get; set; }
    public string ProductName { get; set; }
    public string QuantityPerUnit { get; set; }
    public decimal UnitPrice { get; set; }
    public int UnitsInStock { get; set; }
}
class Category
{
    public int CategoryId { get; set; }
    public string CategoryName { get; set; }
}
```

Step-2: LinqProject Program kısmına **Product** ve **Category** listeleri oluşturalım.

Dikkat: Liste tanımlamada sonra noktalı virgül var. Sonra **Filtreleme** ile istenen özellikte ürünlere erişelim

Örneğin: fiyatı 5000'den fazla ve stokta 3'den fazla olan ürünlerin isimlerini ekran'a yazdıralım.

Bunun için **Algoritmik** ve **Linq** sistemlerini ayrı ayrı kodlayalım.

İşler karmaşıklaşıkça **Linq** 'in etkinliği belirginleşiyor

Linq komutlarını kullanabilmek için “**using System.Linq;**” komut satırı en üsté eklenmelidir

```

static void Main(string[] args)
{
    List<Category> categories = new List<Category> //*****kategori tanımlama*****
    {
        new Category{CategoryId =1, CategoryName = "Bilgisayar"},
        new Category{CategoryId =2, CategoryName = "Telefon"},
        new Category{CategoryId =1, CategoryName = "Bilgisayar"},
    };

    List<Product> products = new List<Product> //*****Ürün tanımlama*****
    {
        new Product{ProductId =1, CategoryId = 1, ProductName ="Acer Laptop",
                    QuantityPerUnit = "32 Gb Ram", UnitPrice = 10000, UnitsInStock=5},
        new Product{ProductId =2, CategoryId = 1, ProductName ="Asus Laptop",
                    QuantityPerUnit = "16 Gb Ram", UnitPrice = 8000, UnitsInStock=3},
        new Product{ProductId =3, CategoryId = 1, ProductName ="Hp Laptop",
                    QuantityPerUnit = "8 Gb Ram", UnitPrice = 6000, UnitsInStock=2},
        new Product{ProductId =4, CategoryId = 2, ProductName ="Samsung Telefon",
                    QuantityPerUnit = "4 Gb Ram", UnitPrice = 5000, UnitsInStock=15},
        new Product{ProductId =5, CategoryId = 2, ProductName ="Apple Telefon",
                    QuantityPerUnit = "4 Gb Ram", UnitPrice = 8000, UnitsInStock=0}
    };
    Console.WriteLine("Algoritmik yazım formати-----");

    foreach (var product in products && product.UnitsInStock>3) //*****istenen ürünlere erişme*****
    {
        if (product.UnitPrice > 5000)
            {Console.WriteLine(product.ProductName); }
    }

    Console.WriteLine("Linq ile -----");

    var result = products.Where(p=>p.UnitPrice>5000 && p.UnitsInStock>3);
    foreach (var product in result)
    {
        Console.WriteLine(product.ProductName);
    }
}

```

Metot yazımında Algoritmik ve Linq komutları kıyaslama

```
// Algoritmik
static List<Product> GetProducts(List<Product> products)
{
    List<Product> filteredProducts = new List<Product>();
    foreach (var product in products)
    {
        if (product.UnitPrice > 5000 && product.UnitsInStock > 3)
        {
            filteredProducts.Add(product);
        }
    }
    return filteredProducts;
}

//Linq ile
static List<Product> GetProductsLinq(List<Product> products)
{
    return products.Where(p => p.UnitPrice > 5000 && p.UnitsInStock > 3).ToList(); //dikkat. ToList() yazmazsa hata verir
}
```

Hazır Liste Özellikleri

Tanımlanan liste adı komu satırına yazıldığında ve sonunda nokta eklendiğinde liste alt değişkenleri ve bazı hazır özellikler görünür
Bu özellikler döngü olmaksızın veriyi inceler ve istenen bilgiye ulaşır

Any: Liste içinde istenen özellikte eleman olup olmadığını belirleme(true-false)

```
var result = products.Any(p => p.ProductName == "Acer Laptop") //liste içinde eleman arama
Console.WriteLine(result); //result True veya False olur
```

Not: “=>” ise anlamındaki bu işaretre Lambda denir

Find : Liste içinde istenen özellikte elemanı bulma (tüm detaylarına erişme)

```
var result = products.Find (p=>p.ProductId==3);  
Console.WriteLine(result.ProductName); //result product cinsi olur
```

FindAll /Where: Liste içinde istenen özellikte eleman bütün elemanları bulma (tüm detaylarına erişme)

```
var result = products.FindAll (p=>p.ProductName.Contains("top")).OrderByDescending(p=>p.UnitsPrice).ThenBy(p => p.ProductName);  
//fiyata göre azalan sıralama, eşit fiyatlarda alfabetik sıralama  
Console.WriteLine(result); //result product cinsi ve liste olur. bulamazsa null olur.  
foreach (var product in result)  
{ Console.WriteLine(product.ProductName);}  
  
var result = products.Where(p => p.ProductName.Contains("Laptop")).OrderBy(p=>p.UnitsPrice).ThenByDescending(p => p.ProductName);  
//artan sıralama eşit fiyatlarda alfabetik tersi sıralama  
Console.WriteLine(result); //result product cinsi ve liste olur. bulamazsa null olur.  
foreach (var product in result)  
{ Console.WriteLine(product.ProductName);}
```

Orderby: Liste ile veri tabanından alınan bilgiler istenen özelliğe göre artan şekilde sıralanıp sergilenebilir. (default Ascending)

OrderByDescending : Azalan şekilde sıralanır

Örneğin Fiyata göre sıralama şekilleri yukarıdadır.

Fiyatı eşit olanları ise veri tabanı sırasına göre sıralar.

ThenBy: orderBy ile sıralamada aynı olanları neye göre sıralayacağını belirleme. (default Ascending)

ThenByDescending

Örneğin Fiyata göre sıralanana ürünlerden aynı fiyata sahip olanı stok durumuna göre veya satış oranına göre sıralar

From...in.....: istenen verideki ürünlere erişme

```
var result = from p in products
              where p.UnitPrice >6000 // filtreleme
              orderby p.UnitPrice descending, p.ProductName ascending // fiyat azalan isim artan sıralama
              select p;
```

Konu : Joint (DTO- Data Transfer Object)

Kullanılan Dosya : C# => CSharpIntro => LinqProject

(Tanım) **Joint:** Farklı veri tablolarından istenen verileri alıp birleştirme

Veri tablosundan sadece istenen veriyi çekebilirim

Örneğin 2 farklı Ürün Class'ından kullanacağım özellikleri hava yeni tek bir Class oluşturabilirim

```
class ProductDto //Dto:Data
{
    public int ProductId { get; set; }
    public string ProductName { get; set; }
    public decimal UnitPrice { get; set; }
    public string CategoryName { get; set; }
}

//Ana Program
var result = from p in products
             join c in categories
             on p.CategoryId equals c.CategoryId
             where p.UnitPrice >=10000
             orderby p.UnitPrice
             select new ProductDto {ProductId = p.ProductId, CategoryName = c.CategoryName, ProductName = p.ProductName,
UnitPrice=p.UnitPrice };

foreach (var productDto in result)
{
    Console.WriteLine(productDto.ProductName + " " + productDto.CategoryName);
    //Aynı işlevi görür. Bellekte daha az yer tutar
    //Console.WriteLine("{0}..{1}", productDto.ProductName, productDto.CategoryName);
}
```

8.DERS (03.02.2021)

Konu	: C# EntityFramework
Kullanılan Dosya	: Visual Studio => Project => MyFinalProject DEVAM
Ek kaynak	: İlgideki 16.Bölümü Proje&Entity bölümünü çalış https://www.udemy.com/course/c-sharp-programlama-kursu/

Dev Architect Framework: Veri tabanına bağlanmak ve işlem yapmak istediğimizde veya WEB sayfası tasarım işlemlerinde Programlama dilimizle (Burada C#) uzun işlemler yapmak yerine geliştirilmiş Framework'lar (**Açık kaynak proje yönetim modülü**) kullanabiliriz. Uygun girdiler (veri ve istenilenler) sonrası bir proje otomatik oluşturan bir sistemdir. (<https://www.devarchitecture.net/>)

Bu tarz Framework'ların kullanımı yaygınlaşmıştır. **Java** içinde Spring adlı bir takımın geliştirdiği **Spring** adlı Framework'un kullanım oranı %100'e yakındır.

NEYI NICIN YAPIYORUZ: **EntityFrameWork** yapısı ile “Northwind” veri tabanını projemize bağlayacağız ve veri tabanı üzerinde işlem yapma (ekle, sil ve güncelle) ve veriyi görüntüleme kodlarını güncelleyeceğiz

Step-15: **DataAccess** projesinde **Abstract** Klasöründe **ICategoryDal** Interface oluştur

IProductDal Interface'de olduğu gibi benzer işlemleri yapacağız.

Benzer işlemleri yapacağımıza göre her iki **Interface** kullanabileceğimiz ortak bir jenerik yapı oluşturabiliriz.

Adı Jenerik Repository Dizayn patern

Step-16: **Abstract** Klasöründe **IEntityRepository** adlı Interface oluşturalım

IEntityRepository Interface içinde **IProduktDal** Interface de daha önce yazdığımız kodları kesip buraya alacağız.

Bunları **IProduktDal** ve **ICategoyDal** da ortak kullanacağız.

Burada işlem sonrası istenen değerleri geri döndürmeliyiz. Ödevi de buna göre değiştirebiliriz

```
public interface IEntityRepository<T> where T : class, IEntity, new()
{
    List<T> GetAll(Expression<Func<T, bool>> filter = null); //Çıkan Ampulden Using komutunu eklemeyi unutmamalı
    T Get(Expression<Func<T, bool>> filter); //İstenen değerleri geri döndüren bir Get operasyonu yazmalıyız
    void Add(T entity);
    void Update(T entity);
    void Delete(T entity);
}
```

Yeni IProductDal

```
public Interface IProductDal: IEntityRepository<Product>
{
}
```

Yeni ICategoryDal

```
public Interface ICategoryDal:IEntityRepository<Category>
{
}
```

Not:Generic Constraint : // Bir değişkene gönderilecek tipleri Sınırlamak istediğimiz durumda Generic Constraint kullanılır

Örnek: `public Interface IEntityRepository <T> where T : class, IEntity`
// T değişkeni sadece class ipi referans tip olacak ve IEntity veya ondan türetilerek bir nesne olabilir

Step-17: **Entities** Projesi **Concrete** Klasöründe **Customer** Class oluşturuyoruz

DataAccess Projesi **Abstract** Klasöründe **ICustomerDal** Interface oluştur

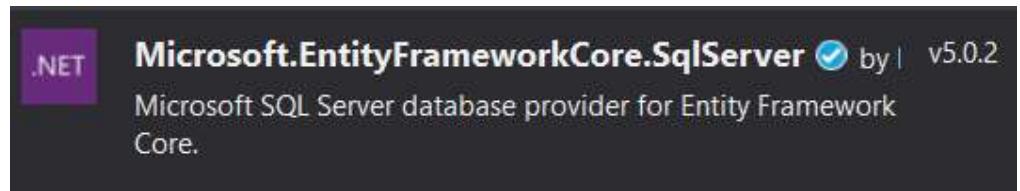
```
public Interface ICustomerDal:IEntityRepository<Customer>
{}
```

- Step-18:** **DataAccess** projesinde **Concrete** klasöründe **EntityFramework** adlı klasör oluştur
EntiyFramework klasörü içinde **EfCategoryDal** adlı Class oluştur // public yapmayı unutma
EfCategoryDal Classini **ICategoryDal** ile inherit et. Ampulden **Using** komut satirini eklemeyi ve sonra **Implement Interface** etmeyi unutma
Implement edince Interface in tüm komutları buraya gelir
EntiyFramework klasörü içinde **EfProduktDal** adlı Class oluştur // public yapmayı unutma
EfProduktDal Classini **IProductDal** Interface et. Ampulden **Using** komut satirini eklemeyi ve sonra **Implement Interface** etmeyi unutma
Implement edince Interface'in tüm komutları buraya gelir

Not: Daha sonra **InMemoryProductDal** Interface **EntityFrameWork'a** bağlayacağız

- Step-19:** DataAccess projesine sağ tıklayıp **Manage Nuget Packages** seçilir. Browser kısmına **entityframeworkcore.sql** yazacağımız
Entity Framework : Link Destekli Çalışır. Veri Tabanındaki tabloyu Class gibi ilişkilendirip SQL'leri Linq ile yaptığımız bir Ortam
Şimdiye kadar C# dat.net içindeki Implemetation'ları kullandık. Bundan sonra başkalarının da hazırladıkları yapıları kullanabiliyoruz
Adı **NuGet**

EKLENTİ: Yalnız bizim Version 3.1.11 olduğu için bu versiyon yüklenmelidir



- Step-20:** **DataAccess** projesinde **Concrete** Klasörü-**EntityFrameWork** Klasörüne **NorthwindContext** adlı Class ekle.
Class'in tipini **DbContext** seçelim. Projeye veri tabanını ilişkilendireceğiz

```

public class NorthwindContext:DbContext
{
    // Projeye ver tabanını ilişkilendireceğimiz kısım
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(@"Server=localdb\mssqllocaldb;Database=Nortwind;Trusted_Connection=true");
        // optionsBuilder.UseSqlServer(@"Server=175.45.2.12"); Normalde Sql serverin bulunduğu yerin IP adresi
        // Trusted_Connection=true password olmadan bağlanmak için yazılmalıdır
        // @ ters slashları komut olarak algılamaması için yazılır
    }
    // veri tabanındaki hangi nesneye hangiye karşılık gelecek belirtmeliyiz
    public DbSet<Product> Products { get; set; }
    public DbSet<Category> Categories { get; set; }
    public DbSet<Customer> Customers { get; set; }
}

```

Step-21: **EfProductDal** üzerinde çalışacağız

Artık **EntityFramework** kullanarak kodlarını yazabiliriz

NOT: "Us" yazıp 2x tab yapınca **using** metodu gelir. Bu metot program çalışınca işlem görür sonra silinir (using komut satırı ile karıştırma)

Çıkan metot içinde data Classini tanımlayıp işleme yapacağız (ekle, sil, güncelle)

Burada yaptığımız aynı işlemler Category içinde geçerli. O yüzden jenerik yapabiliriz (sonra ??????)

```

public class EfProductDal : IProductDal // public ve inherit etme
{
    public void Add(Product entity)
    {
        // disposable pattern implemenatation of c#
        using (NorthwindContext context = new NorthwindContext())
        {
            var addedEntity = context.Entry(entity);
            addedEntity.State = EntityState.Added;
            context.SaveChanges();
        }
    }
}

```



Hazır Komutlar

```

public void Delete(Product entity)
{
    using (NorthwindContext context = new NorthwindContext())
    {
        var deletedEntity = context.Entry(entity);
        deletedEntity.State = EntityState.Deleted;
        context.SaveChanges();
    }
}
public Product Get(Expression<Func<Product, bool>> filter) //dikkat liste tipi değil Product tipi
{
    using (NorthwindContext context = new NorthwindContext())
    {
        return context.Set<Product>().SingleOrDefault(filter);
    }
}
public List<Product> GetAll(Expression<Func<Product, bool>> filter = null)
{
    using (NorthwindContext context = new NorthwindContext())
    {
        return filter == null
            ? context.Set<Product>().ToList()
            : context.Set<Product>().Where(filter).ToList();
    }
}
public void Update(Product entity)
{
    using (NorthwindContext context = new NorthwindContext())
    {
        var UpdatedEntity = context.Entry(entity);
        UpdatedEntity.State = EntityState.Modified;
        context.SaveChanges();
    }
}

```

The diagram consists of two red-bordered boxes, each containing the text "Hazır Komutlar". One arrow points from the top box to the line of code "var deletedEntity = context.Entry(entity);". Another arrow points from the bottom box to the line of code "var UpdatedEntity = context.Entry(entity);". These arrows indicate that the highlighted code snippets are examples of ready-made commands.

Step-22: Console projesi program kısmını açtık

Not: SOLID kodlama prensibinin bas harfleri burada O harfini yaptık: **Open Closed Principle**. Yeni bir özellik ekliyorsak mevcut koduna dokunmayın prensibi

Burada 7.dersten farklı olarak üretilen Classini tipini **Entityframework** tipinde yaptık

*****Artık program veri tabanıyla ilişkilendi. *****

Programı Run (koşturduğumuzda) yaptığımda kendi uydurduğumuz ürünler değil, Northwind veri tabanındaki ürünler listelenenecek

HATA: Programı koşturduğumuzda “**SQL client hatası**” alırsak **DataAccess =>Concrete =>EntityFramework** klasöründe **NorthwindContext Class’ında** yazdığımız kısa yol hatalı demektir

```
class Program
{
    static void Main(string[] args)
    {
        ProductManager productManager = new ProductManager(new EfProductDal()); //newleme yapacağız. Daha sonra gerek kalmayacak
        foreach (var product in productManager.GetAll())
        {
            Console.WriteLine(product.ProductName);
        }
    }
}
```

Step-23: Business Abstract klasörü **IProductService** Interface açalım

EntityFrameWork nimetlerinden faydalananmaya başlayalım

Yeni listeler ekledik.

1. liste: GetAll tümünü getiriyordu. Şimdi CategoryId sine göre getirecek **GetAllByCategory** listesi ekledik. Örnek E ticaret sitesinde Kategori seçildiğinde açılacak kısımlar için komut yazıyor

2. liste: Belirlenen fiyat aralıklarındaki ürünler listelenenecek/sergilenecek

```
public Interface IProductService // public yap
{
    List<Product> GetAll();
    List<Product> GetAllByCategoryId(int Id); //ekledik 8.ders 03.02.2021
    List<Product> GetByUnitPrice(decimal min, decimal max); //ekledik 8.ders 03.02.2021
}
```

Step-24: *Business Concrete* klasörü *ProductManager* artık sorun yaratacak. Implement et (Ampulden).

GetAllByCategory ve *GetByUnitPrice* için hazır metot gelecek.

Sorun kalkacak

Burada değişiklik yapacağız. Metotlara işlem komutunu yazacağız.

1.liste: Seçilen kategoride ürünleri listeleyeceğiz

```
public List<Product> GetAllByCategoryId(int id) //dikkat
{
    return _productDal.GetAll(p => p.CategoryId == id);
}
```

2.liste: Belirlenen fiyat aralıklarındaki ürünler listelenecek/sergilenecek

```
public List<Product> GetByUnitPrice(decimal min, decimal max)
{
    // sadece seçilen fiyat aralığındaki ürünler listelenecek
    return _productDal.GetAll(p => p.UnitPrice >= min && p.UnitPrice <= max); // notasyona dikkat
}
```

Step-25: **Console** projesi program kısmını açtık

Örneğin **Foreach** komutunu **CategoryId** değeri “2” olanları göstermek üzere aşağıdaki şekilde değiştirelim.

ProductManager’dan **GetAllByCategoryId(2)** metodunu çağıralım

Koşturma: Programı **Run** (koşturduğumuzda) yaptığımızda Northwind veri tabanında **CategoryId=2** olan ürünler listelenecik

```
static void Main(string[] args)
{
    ProductManager productManager = new ProductManager(new EfProductDal());
    foreach (var product in productManager.GetAllByCategoryId(2)) //*****değiştirdik*****
    {
        Console.WriteLine(product.ProductName);
    }
}
```

Veya **Foreach** komutunu birim fiyatı 50 ile 100 arasında olanları göstermek üzere aşağıdaki şekilde değiştirelim.

Programı **Run** (koşturduğumuzda) yaptığımızda “**Northwind**” veri tabanında fiyatı 50 ile 100 arası olan ürünler listelenecik

```
static void Main(string[] args)
{
    ProductManager productManager = new ProductManager(new EfProductDal());
    foreach (var product in productManager.GetByUnitPrice(50,100)) //*****değiştirdik*****
    {
        Console.WriteLine(product.ProductName);
    }
}
```

Hatırlatma: Projeyi kaydedip **Get** ile yoruma bugünün tarihini yazıp **“Commit All”** yapıp Push yapmayı unutmayalım.

8.Ders Sonu Projenin Durum Tablosu

No	Projekt/Template	Projekt Reference	Klasör	Class/Interface	İşlem
1	MyFinalProject (Blank Solution)	---			Manage Nuget Packages kısmından DataAccess için entityframeworkcore.sql (v 3.1.11) yükleyeceğiz
2	DataAccess (Class.Library)	Entities	Abstract	IProductDal	IEntityRepository<Product> ile inherit et 7.derste oluşturduğumuz metotları silip IEntityRespository 'e yazacağız
				ICategoryDal	IEntityRepository<Category> ile inherit et
				ICustomerDal	IEntityRepository<Customer> ile inherit et
				IEntityRepository	Diğer 3 interface için ortak ekle, sil, güncelle vb metotları çağır (EntityFramework kodları)
			Concrete	---	----
			Concrete-EntityFramework	InMemoryProductDal	IProductDal ile inherit et ekle, sil, güncelle vb metotları tanımla
				EfCategoryDal	ICategoryDal ile inherit et. ekle, sil, güncelle vb metotları tanımla
				EfProductDal	IProductDal ile inherit et ekle, sil, güncelle vb metotları tanımla
				NorthwindContext	DbContext ile inherit edilecek Ilgili Class'ları (Entities) veri Tabanındaki karşılıkları ile eşleştir Product / Category / Customer
3	Business (Class.Library).	DataAccess Entities	Abstract	IProductService	GetAll, GetAllByCategory metodunu çağır
			Concrete	ProductManager	IProductService ile inherit et GetAll, GetAllByCategory vb metotları tanımla
4	Entities (Class.Library) .	---	Abstract	IEntity	
			Concrete	Product Category Customer	IEntity ile inherit et.
5	ConsoleUI (Console.App)	Business DataAccess Entities			Set a startup project seç Kullanacağımız veri kaynağını tanımla. Veri kaynağından işlem yapabilmek için EfProductdal tipinde class new'le İsteneden işlemi yap (örnek:ürün adları yazdır)

Ek Kaynak : Proje&Entity Framework <https://www.udemy.com/course/c-sharp-programlama-kursu/>

Kullanılan Dosya : Visual Studio => Project =>CSharpIntro

Veri tabanları birden fazla programla etkileşim halinde olabilir

Northwind Database kurulumunu daha önce yaptık (View-SQL Server Object Explorer 'dan Databases altında Northwind veri tabanını görmeliyiz)

ADO.Net= .Net içerisinde veri tabanına soru yazmamızı sağlayan Kütüphane

Günümüzde nesneye yönelik program ile ilişkisiyi kolaylaştmak adına **ORM (Object Relational Mapping)** adlı yapılar kullanılır

Entity Framework Microsoft tarafından geliştirilmiş başarılı bir **ORM**'dir

Burada Veri tabanındakilere karşılık nesneler oluşturacağız

Step-EF-1: *Solution Explorer* üzerinde **Add=>New Project** ile **Console App.Net** ile **EntityFrameworkDemo** adlı katmanı(projesi) oluşturalım

Step-EF-2: **EntityFrameworkDemo Product** adlı **class** ekle

```
public class Product //public // Veri Tabanında products adlı çoklu tablo var (burada tekil)
{
    public int ProductId { get; set; } //Primary Key
    public int CategoryId { get; set; } // Foreign Key
    public string ProductName{ get; set; }
    public string QuantityPerUnit { get; set; }
    public decimal UnitPrice { get; set; }
    public short UnitsInStock { get; set; } // veri tabanında bu şekildedir (short int'in küçüğü)
}
```

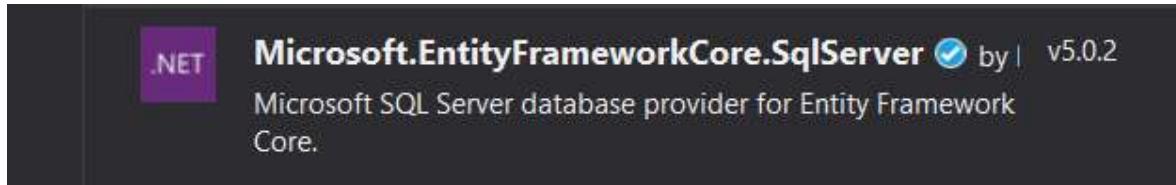
*******EntityFrameWork ile Tablo arasındaki ilişkisiyi sağlayacağınız*******

Step-EF-3: **EntityFrameworkDemo** sağ tıkla **Manage Nuget Packages and Solutions** seçerek

MyFinalProject Step-19"de olduğu gibi hazır paket kodları (EF) yükleyeceğiz

Browser kısmına **entityframeworkcore.sql** yazacağız

Entity Framework : Link Destekli Çalışır. Veri Tabanındaki tabloyu Class gibi ilişkilendirip SQL'leri Linq ile yaptığımız bir Ortam



Seçilecek . Yalnız bizim Versiyon 3.1.11 olduğu için bu versiyon yüklenmelidir

Step-EF-4: *EntityFrameworkDemo* *NorthwindContext* adlı **class** ekle

DbContext adlı hazır Interface (EF) Inherit et. Ampulden **Using** komut satırını ekle

Linki sağlayarak veri tabanıyla projemizi ilişkilendirelim (MyFinalProject Step-20 gibi)

Veri Tabanı ile Program işletilen bilgisayar ayrı da olabilir

```
class NorthwindContext : DbContext // EntityFramework Class'ına Interface edilir
{
    // Projeye veri tabanını ilişkilendireceğimiz kısım
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=Northwind;Trusted_Connection=true");
        // optionsBuilder.UseSqlServer(@"Server=175.45.2.12"); Normalde Sql serverin bulunduğu yerin Ip adresi
        // Trusted_Connection=true password olmadan bağlanmak için yazılmalıdır
        //@ ters slashları komut olarak algılamaması için yazılır
    }

    //veri tabanındaki hangi nesneye karşılık gelecek belirtmeliyiz
    public DbSet<Product> Products { get; set; }
}
```

override üzerine Yazmak anlamındadır. Inherit Edilen kısımda değişiklik yapabilme yetkisi tanımlanmış oluyor

Step-EF-5: *EntityFrameworkDemo Program* kısmına komutlar yazarak *EntityFramework* özelliklerini kullanalım
EntityFrameworkDemo sağ tıklayıp “Set a StartUp Project” yapmayı unutmayalım.

```
class Program
{
    static void Main(string[] args)
    {
        GetAll();
        GetProductsByCategory(1);
    }

    private static void GetAll()
    {
        NorthwindContext northwindContext = new NorthwindContext(); // Class newledik
        foreach (var product in northwindContext.Products)
        {
            Console.WriteLine(product.ProductName);
        }
    }

    private static void GetProductsByCategory(int categoryId)
    {
        NorthwindContext northwindContext = new NorthwindContext(); // Class newledik
        var result = northwindContext.Products.Where(p => p.CategoryId == categoryId);
        foreach (var product in result)
        {
            Console.WriteLine(product.ProductName + " Kategorisi:" + product.CategoryId);
        }
    }
}
```

Konu: Custom Mapping Veri tabanıyla programda isimlendirmeler arasında fark olabilir.

Context üzerinden kendi adlandırmamızı ver tabanında karşılığı ile ilişkilendirebiliriz.

```
class NorthwindContext : DbContext
{
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=Northwind;Trusted_Connection=true");
    }

    public DbSet<Personel> Personels { get; set; }

    //***** Custom Mapping*****
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Personel>().ToTable ("Employees", "dbo"); // dbo default temadır ancak bazı durumlarda yazılmalıdır
        modelBuilder.HasDefaultSchema ("admin"); // sadece adminlerin tablosu gibi tanımlamalar yapılabilir,
        modelBuilder.Entity<Personel>().Property(p=>p.Id).HasColumnName ("EmployeeId"); // Alt tanımlamaları eşleştiriyoruz
        modelBuilder.Entity<Personel>().Property(p=>p.Name).HasColumnName ("FirstName");
        modelBuilder.Entity<Personel>().Property(p=>p.Surname).HasColumnName ("LastName");
    }
}

EntityFramework program kısmına yazdırma komutlarını yazalım
class Program
{
    static void Main(string[] args)
    {
        GetAllPersonel();
    }
    private static void GetAllPersonel()
    {
        Console.WriteLine("Custom Mapping" );
        NorthwindContext northwindContext = new NorthwindContext(); // Class newledik
        foreach (var personel in northwindContext.Personels)
        {
            Console.WriteLine("Personel Adı:" + personel.Name);
        }
    }
}
```

9.DERS (06.02.2021)

Konu : C# Çok Katmanlı Kurumsal Yazılım Mimarileri (Core Katmanı ve DTO kullanımı)

Kullanılan Dosya : Visual Studio => Project => MyFinalProject (DEVAM)

Ek Kaynak :

NEYI NICIN YAPIYORUZ: Gecen derste kurumsal mimarının altyapısını kurduk.

EntityFrameWork yapısı ile bir veri tabanını projemize bağladık.

Veri tabanı üzerinde işlem yapma (ekle, sil ve güncelle) ve veriyi görüntüleme kodları yazdık

Bu derste belli başlı kodları **Core** bir katman (proje) oluşturup oraya ekleyeceğiz.

Yazılan kodlar bu proje haricinde kodlarda da geçerli olacak

API yazmaya başlayacağız

Açıklama: Neden **Core** katman gereklidir?

Metotlarda hep bir çalıştığımız bir **Entity** (burada Produkt, Category, Order vb.) var. **NorthwindContext** context üzerinde işlem yapıyoruz

Bunu başka bir yerde yapı oluşturarak standartlaştırırız. Jenerik yapı kurarız. Veri tabanında bağımsız olacak bu yapıyı başka projelerde de kullanabiliriz

Not Car ödevinde **IEntityRepository** benzeri kodlarımız var

Step-26: **DataAccess=>Concrete=>EntityFramework EfProductDal** adlı Class açalım

Solution=>Add =>New Project class library.dat.net seçelim **Core** adlı bir proje oluşturalım. (Uyduruk Class nesneyi silelim)

İlgileneceğimiz Katmana atfen klasörler oluşturacağız

- **Core Katmanı içine DataAccess** klasörü oluşturalım. Metin içinde bahsederken **Core (DataAccess)** diyelim ki karışmasın
DataAccess – Abstract klasörü içindeki **IEntityRepository** Interface Classini buraya kesip alalım

- **Core** katmanına **Entities** adlı yeni bir klasör oluşturalım.
- Entities-Abstract** klasörü içindeki **IEntity** Classini buraya (**Core(Entities)**) kesip al
- (**Core(Entities)**) **namespace** satırını düzenleyelim **Core.Entities** yapalım. “**namespace**” kısmı klasör konumunu belirtir.

Core (DataAccess) IEntityRepository interface namespace kısmını **Core.DataAccess** yapalım.

Hata veren “Using Entities.Abstract” satırını silelim.

(using komut satırı , özellikleri kullanılmak istenen namespace ile aynı olmalıdır)

Burada ampulden **Using Core.Entities** komut satırını ekleyeceğiz

Not: Core katmanı diğer katmanları referans almaz. Bağımsız olmalıdır

```
using Core.Entities; //Eklandı
using System;
using System.Collections.Generic;
using System.Linq.Expressions;
using System.Text;

namespace Core.DataAccess // burayı değiştirelim
{
    // Generic Constraint
    // T değişkeni sadece class tipi referans tip olacak ve IEntity veya ondan türetilerek bir nesne olabilir
    // new() newlenebilir olmalı
    public Interface IEntityRepository <T> where T : class, IEntity, new ()
    {
        List<T> GetAll(Expression<Func<T,bool>> filter=null); //Çıkan Ampulden Using komutunu eklemeyi unutmamalı
        T Get(Expression<Func<T, bool>> filter); //İstenen değerleri geri döndüren bir Get operasyonu yazmalıyız
        void Add(T entity);
        void Update(T entity);
        void Delete(T entity);
    }
}
```

```
namespace Core.Entities // ( bu Classin dosya uzantısı şeklinde olan adını düzelttik)
{
    public interface IEntity
    {
    }
}
```

Step-27: **DataAccess** klasöründe sağ tıklayıp “add Projekt Reference” ile **Core (Entity** zaten seçiliydi) da seçilir

Bu seferde **DataAccess** klasörü içindeki **IproductDal** Classı Interface edildiği **IEntityRepository** Classını bulamadığı için hata verir.

Ampulden “Add reference to Core” seçilir. (using Core.DataAccess komut satır eklenir)

DataAccess-Concrete_EntityFramework klasöründe yer alan **ICategoryDal** Interface Classına gel

Ampulden “Add reference to Core” seçilir. (using Core.DataAccess komut satır eklenir)

DataAccess-Concrete_EntityFramework klasöründe yer alan **ICustomerDal** Classına gel

Ampulden “Add reference to Core” seçilir. (using Core.DataAccess komut satır eklenir)

Not: Kısacası **DataAccess-Concrete_EntityFramework** klasöründe yer alan Dal (Data Access Layer) olan tüm interface Classlar için Ampulden “Add reference to Core” seçilir.

Step-28: **Entities** klasöründe sağ tıklayıp Add **Project Reference** ile **Core** seçilir
Entities-Concrete klasörü altındaki **Produkt** Classına gecelim
Bu seferde **Entities** klasörü içindeki **Product** Class Interface edildiği **IEntity** Classını bulamadığı için hata verir.
Ampulden “**Add Reference to Core**” seçilir. (**using Core.Entities** komut satır eklenir)
using Entities.Abstract; satırı hepsinde artık gereksiz, silelim

Entities-Concrete klasörü altındaki **Category** Classına gecelim. Aynı işlemi yapalım. Ampulden “**Add Reference to Core**” seçilir.
Entities-Concrete klasörü altındaki **Customer** Classına gecelim. Aynı işlemi yapalım. Ampulden “**Add Reference to Core**” seçilir.

Not: Kısacası **Entities-Concrete** klasöründe yer alan tüm Classlar için Ampulden “**Add reference to Core**” seçilir.

Not: **Build** menüsünden **Built Solution** yapalım. Hata olup olmadığını kontrol edelim.
Build Succeeded görürsek işler yolunda gidiyor demektir.

Step-29: **Core** katmanına gelelim **Core(DataAccess)** klasörüne **EntityFramework** klasörü ekleyelim
EntityFramework klasörüne **EfEntityRepositoryBase** adlı Class oluşturalım (sağ tıkla **Add ile**). **Public** yapmayı unutma
Entity tipi ve **Context** tipi ile veriye ulaşabileceğimiz bir yapıyı tanıtmalıyız
EntityFramework kodu yazacağız. **EntityFramework** eklemeliyiz

```
public class EfEntityRepositoryBase <TEntity, TContext> // Entity tipi ve Context tipi ile veriye ulaşabiliriz
{ }
```

Step-30: **EKLENTİ:** *Solution* sağ tıkla **ManageNugetPackages und Solutions** seçilir. **Installed** kısmından Aşağıdaki seçilir.

(step 19'da Browser'dan zaten yüklemiştik)

Core seçilir versiyon 3.1.11 olduğu kontrol edilir ve **Install** yapılır (**DataAccess** 'in yanında yüklü olan 3.1.11 versiyon numarası görünüyor)



Step-31: Simdi oluşturacağımız yapıyı her yerde kullanabiliriz

Core(DataAccess) *EntityFramework* klasörüne **EfEntityRepositoryBase** adlı Class 'a gelelim

Buradaki değişkenleri tanımlayacağız,

IEntity ve **DbContext** 'lerden ampulden **using** komut satırını ekleyelim

DataAccess **Concrete-EntityFrameWork** klasörü içindeki **Category**, **Product** ve **Customer** için **Entityframework** işlem komutları yazmıştık.

Şimdi de burada hepsini simule edeceğiz. (All in one)

```
public class EfEntityRepositoryBase <TEntity, TContext> : IEntityRepository<TEntity> // entity tipi ve Context tipi ile veriye ulaşabiliriz
    where TEntity : class, IEntity, new()
    where TContext : DbContext, new ()
{
}
```

Ampulden **Implement** etmemeliyiz veya komutları yazmamızı (bu durumda kaba komut satırlarının içini doldurmalıyız)

Onun yerine **DataAccess-Concrete-EntityFramework** klasörü **EfProductDal** klasından tüm komutları kesip buraya alalım

Product yerine **TEntity** yazalım

NorthwindContext yerine **TContext** yazalım

Expression ve **SingleOrDefault** komutlarını ampulden using komut satırlarını ekleyelim

```
using System.Linq;
using System.Linq.Expressions;

public class EfEntityRepositoryBase <TEntity,TContext> : IEntityRepository<TEntity> // entity tipi ve Context tipi ile veriye ulaşabiliriz
{
    where TEntity : class, IEntity, new()
    where TContext : DbContext, new()

    {
        public void Add(TEntity entity)
        {
            //disposable pattern implementation of c#
            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<, bool>> filter)
{
    using (TContext context = new TContext())
    {
        return context.Set<().SingleOrDefault(filter);
    }
}

public List< GetAll(Expression<Func<, bool>> filter = null)
{
    // Filtre verebilsin filter null ise bir ilsem değilse diğer işlem çalışır
    //burası bizim için sql kodu olan select * products yapıyor
    using (TContext context = new TContext())
    {
        return filter == null
            ? context.Set<().ToList()
            : context.Set<().Where(filter).ToList();
    }
}

public void Update(TEntity entity)
{
    using (TContext context = new TContext())
    {
        var UpdatedEntity = context.Entry(entity);
        UpdatedEntity.State = EntityState.Modified;
        context.SaveChanges();
    }
}

}

```

Step-32: **DataAccess Concrete EntityFrameWork** klasörü içindeki **EfProductDal** a gelelim

Eskiden sadece **IProductDal** a inherit idi. Simdi **EfEntityRepositoryBase** 'ede inherit edeceğiz

Ampulden **Using** komut satirini da ekleyelim

```
public class EfProductDal : EfEntityRepositoryBase<Product, NorthwindContext>, IProductDal{}
```

Aynı hususu **EfCategoryDal** içinde yapacağız

Eskiden sadece **ICategoryDal** a inherit idi.

Eskiden içinde kodlar vardı. Artık gerek yok. silelim

Ortak Komutları yazdığımız **EfEntityRepositoryBase** 'ede inherit edeceğiz

Ampulden **Using** komut satirini da ekleyelim.

Not: Başka bir veri tabanını, başka bir program kullanan bir sisteme entegre olabilmek için bu şekilde Her katmanı değiştirebiliriz

```
public class EfCategoryDal : EfEntityRepositoryBase<Product, NorthwindContext>, ICategoryDal { }
```

Step-33: Yeni bir nesne oluşturalım. SQL serverdan bir bakalım. **Order** nesnesi oluşturalım

Entities Concrete Klasörü içine **Order** adlı **Class** oluşturalım

IEntity inherit edelim. Ampulden **Implement** edelim

```
public class Order:IEntity
{
    public int OrderId { get; set; }

    public string CustomerId { get; set; }
    public int EmployeeId { get; set; }

    public DateTime OrderDate { get; set; }
    public string ShipCity { get; set; }
}
```

Step-34: **DataAccess Abstract** klasörüne **IOrderDal Interface** ekleyelim. Public yapalım

IEntityRepository<Order> inherit edelim

```
public Interface IOrderDal:IEntityRepository<Order>
{
}
```

Step-35: **DataAccess Concrete EntityFramework** klasörüne **EfOrderDal Class** ekleyelim. Public yapalım

Inherit edelim (Core EntityFramework ve Abstract Interface 'den)

```
public class EfOrderDal :EfEntityRepositoryBase<Order, NorthwindContext>, IOrderDal
{
}
```

Step-36: **NorthwindContext** Classına Order 'in karşılığını da tanıtmalıyız

```
public class NorthwindContext:DbContext
{
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=Northwind;Trusted_Connection=true");
        // Trusted_Connection=true paswrd olmadan bağlanmak için yazılmıştır
        // @ ters slashları komut olarak alılamaması için yazılır

    }
    public DbSet<Product> Products { get; set; }
    public DbSet<Category> Categories { get; set; }
    public DbSet<Customer> Customers { get; set; }
    public DbSet<Order> Orders { get; set; } ****yeni ekledik*****
}
```

Step-37: **Business Abstract** klasörüne **ICategoryService** adlı Interface oluşturalım. Public yap

```
public Interface ICategoryService
{
    List<Category> GetAll();
    Category GetById(int categoryId);
}
```

Step-38: **Business Concrete** klasörüne **CategoryManager** adlı Class oluşturalım. Public yap

ICategoryService Interface et

Using Komut satırını ekle

Implement Et. Hazır metodlar gelsin

```
public class CategoryManager : ICategoryService
{
    ICategoryDal _categoryDal;

    public CategoryManager(ICategoryDal categoryDal)
    {
        _categoryDal = categoryDal;
    }

    public List<Category> GetAll()
    {
        //İs kodları
        return _categoryDal.GetAll();
    }

    public Category GetById(int categoryId)
    {
        return _categoryDal.Get(c => c.CategoryId == categoryId);
    }
}
```

Step-39: **Console Program** Classına gelelim

Kosturma: **Run** yapalım ve kontrol edelim

Bazı satırları seçip sağ tıklayınca **Implement** ederek fonksiyon haline getirebilirsin. Böylece sadece fonksiyon komut satırını iptal ederek engelleyebilirsin

```
private static void CategoryTest()
{
    CategoryManager categoryManager = new CategoryManager(new EfCategoryDal());
    foreach (var category in categoryManager.GetAll())
    {
        Console.WriteLine(category.CategoryName);
    }
}
```

Not: Metot ismini "**Get**" yerine "**GET**" yazdım için 20 dk. Uğraştım. Program hata verdi

Step-40:

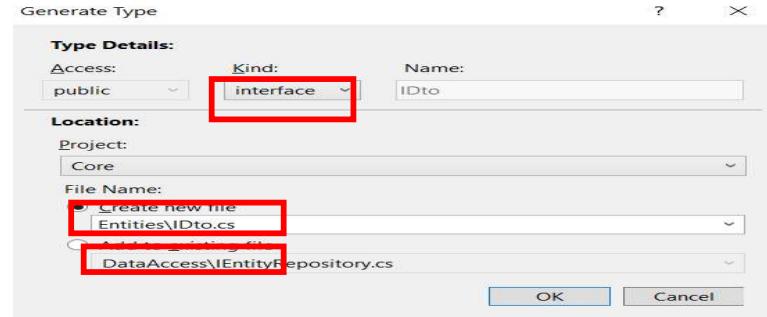
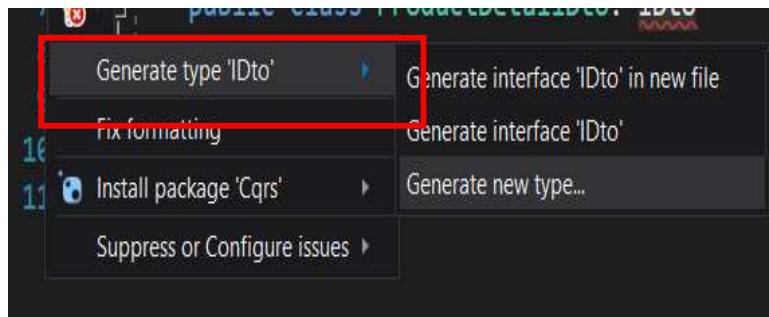
Core projesi **Entities** Klasörüne gidip **IDto** adlı Interface tanımla

Entities projesi içinde **DTOs** adlı yen bir klasör oluşturalım

Tanım: **DTO:** Data Transformation Objekt (Taşıyacağım objeler)

Farklı dosyalarından ortak özelliklerini kullanarak **joint** yaparak yeni bir veri listesi oluşturmak için kullanılır

DTOs içine **ProductDetailDto** adlı Class ekle **public** yap ve **IDto** ile **inherit** et.



Dikkat: Using Komut satirini kendisi otomatik buraya da ekler (kontrol edelim. Entities alt uzantılı using satiri olmalı)

Dikkat: Oluşan DTO Interface **namespace** Core.Entities olmalı (Entities Alt klasör çıkmamış olabilir)

Not: Ampul yoluyla alt Klasörde oluşturulmak istenen Class/Interface 'lerde using ve namespace satırları hatalı oluyor

STEP41: **DataAccess Abstract** klasörü **IProduktDal** Interface gel. Liste oluşturacağız

Built ile kontrol edince **IProductDal'a** Interface edilmiş olan 2 Classe hata olmuşmuş olmalı.

Birincisi **InMemoryProductDal** Classı. Hatalı olduğu için altı çizili olan **IProductDal** ampulden **using** komut satırını Implement et

İkincisi **EfProductDal** Classı. Hatalı olduğu için altı çizili olan **IProductDal** ampulden **using** komut satırını Implement et

Burada bir komut daha yazacagiz

```
public class EfProductDal : EfEntityRepositoryBase<Product, NorthwindContext>, IProductDal
{
    public List<ProductDetailDto> GetProductDetails()
    {
        using (NorthwindContext context = new NorthwindContext())
        {
            // Contexte ulaşıp ürünlerle kategorileri join ediyoruz
            var result = from p in context.Products
                        join c in context.Categories
                        on p.CategoryId equals c.CategoryId
                        select new ProductDetailDto
                        {
                            ProductId = p.ProductId,
                            ProductName = p.ProductName,
                            CategoryName = c.CategoryName,
                            UnitsInStock = p.UnitsInStock
                        };
            return result.ToList();
        }
    }
}
```

Step 42: **Business Abstract** klasörü **IProductService** Interface e getiyoruz

ProductDetailsDto Listesi ekleyeceğiz. Ampulden **Using** komut satırı ekleyeceğiz

```
public Interface IProductService // public yap
{
    List<Product> GetAll();
    List<Product> GetAllByCategoryId(int Id); //ekledik 8.ders 03.02.2021
    List<Product> GetByUnitPrice(decimal min, decimal max); //ekledik 8.ders 03.02.2021
    List<ProductDetailDto> GetProductDetails(); // 06,02.2021 ****
}
```

Step 43 : **Business Concrete** klasörü **ProductManager** Classına getiyoruz

IproductService Interface inde değişiklik yaptığımız için hata verdi. Ampulden **Implement** yapalım

```
public class ProductManager : IProductService
{
```

Step 44: **Console** klasöründe **Program** Classına gelelim

Kosturma: **Run** ettiğimizde **Product** ve **Category** tablosundan bilgileri **joint** ederek görüntülenmesini sağladık (DTO ile)

```
ProductManager productManager = new ProductManager(new EfProductDal());
foreach (var product in productManager.GetProductDetails())
{
    Console.WriteLine(product.ProductName + "/" + product.CategoryName);
}
```

Hatırlatma: Projeyi kaydedip **Get** ile yoruma bugünün tarihini yazıp **“Commit All”** yapıp **Push** yapmayı unutmayalım.

9.Ders Sonu Projenin Durumu

No	Projekt/Template	Projekt Reference	Klasör	Class/Interface	İşlem
1	MyFinalProject (Blank Solution)	---	Manage Nuget Packages	kısmından DataAccess a ilaveten Core içinde entityframeworkcore.sql (v 3.1.11) yükleyeceğiz	
2	DataAccess (Class.Library)	Entities Core	Abstract	<i>IProductDal</i>	<i>IEntityRepository<Product></i> ile inherit et
				<i>ICategoryDal</i>	<i>IEntityRepository<Category></i> ile inherit et
				<i>ICustomerDal</i>	<i>IEntityRepository<Customer></i> ile inherit et
				Önceden burada olan <i>IEntityRepository</i> tamamen <i>Core.DataAccess</i> altına alındı (Kes-Yapıştır)	
				<i>IOrderDal</i>	<i>IEntityRepository<Order></i> ile inherit et
			Concrete	--	----
			Concrete-InMemory	<i>InMemoryProductDal</i>	<i>IProductDal</i> ile inherit et
			Concrete- EntityFramework	<i>EfCategoryDal</i>	<i>ICategoryDal</i> ile inherit et. <i>EfEntityRepositoryBase<Product,NorthwindContext></i> ile inherit et 8.derste oluşturduğumuz metodları sil
				<i>EfProductDal</i>	<i>IProductDal</i> ile inherit et <i>EfEntityRepositoryBase<Product,NorthwindContext></i> ile inherit et <i>ProductDetailDto</i> tipinde Liste tanımla 8.derste oluşturduğumuz metodları silip <i>IEntityRespository</i> 'e yazacağız
				<i>NorthwindContext</i>	<i>DbContext</i> ile inherit edilecek <i>İlgili Class'ları (Entities) veri Tabanındaki karşılıkları ile eşleştir</i> <i>Product / Category / Customer / Order</i>
				<i>EfOrderDal</i>	<i>IOrderDal</i> ile inherit et <i>EfEntityRepositoryBase<Product,NorthwindContext></i> ile inherit et
3	Business (Class.Library).	DataAccess Entities	Abstract	<i>IProductService</i>	<i> GetAll, GetAllByCategory</i> vb. metodları çağır ve listelerini oluştur <i>ProductDetailsDto listesi olustur</i>
				<i>ICategoryService</i>	metodu çağır
			Concrete	<i>ProductManager</i>	<i>IProductService</i> ile inherit et <i> GetAll, GetById</i> vb <i> GetAll, GetAllByCategory</i> vb metodları tanımla
				<i>CategoryManager</i>	<i>ICategoryService</i> ile inherit et <i> GetAll, GetById</i> vb metodları tanımla

No	Projekt/Template	Projekt Reference	Klasör	Class/Interface	İşlem
4	Entities (Class.Library) .	Core	Abstract	----	Önceden burada olan IEntity Interface Core -Entity altına alındı) (Kes-Yapıştır)
			Concrete	Product Category Customer	IEntity ile inherit et.
				Order	IEntity ile inherit et.
			DTOs	ProductDetailDto	IDto ile inherit et.
5	Core (Class.Library)		DataAcces	IEntityRepository	Diğer 3 interface için ortak ekle, sil, güncelle vb metotları çağır (EF kodları)
			DataAccess- EntityFramework	EfEntityRepositoryBase	Burada hem IEntity hem de DbContext Class'tan yeni bir Class newleyeceğiz EfProductDal ve EfCategoryDal 'da 8.derste tanımladığımız metotları kesip buraya alıp yeniden tanımla
			Entities	IEntity IDto	IEntity Class Entities-Abstract klasöründen buraya taşınır
6	ConsoleUI (Console.App)	Business DataAccess Entities	Set a startup project seç Kullanacağımız veri kaynağını tanımla. Veri kaynağından işlem yapabilmek için EfProductdal tipinde class new'le İstenen işlemi yap (örnek: ürün adları yazdır) //Listeleri Joint yap		

Not: Core katmanındaki her yapı diğer projelerde de kullanılabilir

10.DERS (10.02.2021)

Konu : C# Çok Katmanlı Kurumsal Yazılım Mimarileri (Business-Results&Messages)

Kullanılan Dosya : Visual Studio => Project => MyFinalProject (DEVAM)

Ek Kaynak :

NEYI NICIN YAPIYORUZ: Bugün Business kısmıyla ilgileneceğiz

Business ile işletim sistemlerinin birbirile etkileşime geçmesi için **Web API** standart kullanılmalıdır

Bu etkileşime (Etki-Reaksiyon) request-response denir

Business içinde **result'ları** ve resulta göre yazılacak **mesajları** toparlayalım

Step 45: **Business-Abstract** klasörü **IProduktService** interface gel. Ürün ekleme metodunu çağırıralım. Artık veri tabanından veri almanın yanı sıra veri tabanında işlem yapabilmek için hazırladığımız metotları **Business** katmanı için düzeltiyoruz

```
public interface IProductService // public
{
    List<Product> GetAll();
    List<Product> GetAllByCategoryId(int Id);
    List<Product> GetByUnitPrice(decimal min, decimal max);
    List<ProductDetailDto> GetProductDetails();

    void Add(Product product);                                // Eklendi 10.02.2021
    Product GetById (int productId);                          // Eklendi 10.02.2021
}
```

Step 46: **Business-Concrete** klasörü **ProductManager** Class ta **IProduktService** deki değişiklik nedeniyle Implement etmeliyiz.

Metot tanımlamak için hazır olarak oluşur

```
public Product GetById(int productId)
{
    return _productDal.Get(p => p.ProductId == productId);      }
```

Step 47: **Core** katmanına **Utilities** adlı klasör oluştur.

Utilities klasörü altına da **Results** adlı klasör oluştur.

Results klasörü altına da **IResult** adlı **Interface** oluştur.

İşlem sonucunu bildiren komutlar yazalım. (sadece okunabilir olduğu için sadece get kullandık)

```
public interface IResult //public
{
    bool Success { get; }
    string Message { get; }
}
```

Step 48: **Results** klasörü altına da **Result** adlı **Class** oluştur ve **IResult** ile **inherit** et. Ampulden **Implement** et. Metotlar yazılmak üzere görünüşün

Not: Result altına Abstract ve Concrete adlı klasörler de oluşturabilirdik ancak çok fazla olur diye gerek görmedik

Not: Classlar çiplak kalmayacak demek mutlaka bir Interface ile veya başka bir class ile inherit edilecek anlamadadır

Step-49: **Business-Abstract** klasörü **IProduktService** interface geri gel. Daha önce void tanımladığımız **Add** metodunu **IResult** tipinde tanımlayalım ve ampulden using komut satirini ekleyelim

Business Abstract klasörü **ProductManager** Class ta **IProduktService** deki değişiklik nedeniyle Implement etmeliyiz. Daha önce etmiştim. Daha önce çıkışmış olan metotta void tipini **IResult** yapalım ve ampulden **using** komut satirini ekleyelim ve **Result** döndürelim

```
public IResult Add (Product product); // eklendi 10.02
{
    _productDal.Add(product);
    return new Result(true,"ürün eklendi");
}
```

Result içine değer döndüğünde başarılı olduğunda eklendiğini belirten cümle yazdık.

Doğal olarak Hata belirdi. Constructor

Result seçelim. Ampulden **generate Constructor (with field)**

Result class içinde beliren **constructor** yapısını değiştireceğiz



```
public class Result : IResult
{
    public Result(bool success, string message):this(success) //bu durumda aşağıdaki aynı isimdeki tek parametreli constructor da çalışır
    {
        Message = message; // set edilemez demistik get'ler constructor içinde set edilebilir
        //Success = success; // buradan silebiliriz- çünkü aşağıda zaten çalışıyor. gerek yok
    }

    public Result(bool success) // bu yapı hem tek başına hem de yukarıdaki yapı çalıştığında çalışır
    {
        Success = success;
    }

    public bool Success { get; } // DİKKAT

    public string Message { get; }

}

//şimdi biraz daha değiştirelim
```

Step-49 Core-Utilities-Results içine **SuccessResult** adlı **Class** oluşturalım. **Result** Class' e **inherit** edelim. 2 farklı türde değer alabilir

```
public class SuccessResult : Result
{
    public SuccessResult(string message) : base(true, message)
    {
    }

    public SuccessResult() : base(true)
    {
    }
}
```

Step-50 Business-Concrete klasöründe **ProductManager** içindeki **Add** metodunu değiştirelim

```
public IResult Add(Product product)
{
    _productDal.Add(product);

    if (product.ProductName.Length < 2)
    {
        return ErrorResult("Ürün ismi minimum 2 karakter olmalıdır"); // iyi bir ifade değil. Sonra iyileştireceğiz
    }
    _productDal.Add(product);

    return new SuccessResult(); // step 49

    // return new Result(true,"ürün eklendi"); // step 47 de yaptık. iptal ettik. Geliştirmeye devam
}
```

Step-51: **Core-Utilities-Result** içine **ErrorResult** adlı class oluşturalım **Result** class 'a inherit edelim. Kodlar **SuccessError** class benzeri ancak **false** değerli.

```
public class ErrorResult : Result
{
    public ErrorResult(string message) : base(false, message)
    {
    }
    public ErrorResult() : base(false)
    {
    }
}
```

Step-52: **Business** katmanı altına **Constants** (sabit) adlı klasör oluştur. **Northwind** contexte ait özel proje sabitlerini buraya koyacağız. Metinler vb.

Step-53: **Business** katmanı **Constants** (sabit) adlı klasör altına **Messages** adlı Class oluştur. Public static yap (static yapınca newlemeye gerek yok)

Public static yapabiliriz

```
public static class Messages
{
    public static string ProductAdded = "Ürün eklendi";
        // değişken olmasına rağmen public olduğu için büyük harfle başladık
    public static string ProductNameInvalid = "Ürün ismi geçersiz";
}
```

Step-54 **Business-Concrete** klasöründe **ProductManager** içindeki **Add** metodunu değiştirelim

```
public IResult Add(Product product)
{
    _productDal.Add(product);

    if (product.ProductName.Length < 2)
    {
        return new ErrorResult(Messages.ProductNameInvalid);
    }
    return new SuccessResult(Messages.ProductAdded); // step 49
    // return new Result(true, "ürün eklendi"); // step 47 de yaptık. iptal ettik
}
```

Step-55 **Core-Utilities-Results** altına **IDataResult** adlı interface Class oluştur. Mesajlardan faydalananmak için **IResult** ile inherit edelim

```
public interface IDataResult<T> : IResult // public, T herhangi tipte veri dönecek ve Inherit (mesajlardan faydalananacak)
{
    T Data { get; }
}
```

Step-56: **Business-Abstract** klasöründe **IProduktService** i artık **IDataResult** ile yazabilirim

```
public interface IProductService // public yap
{
    IDataResult<List<Product>> GetAll(); //10.02.2021 değiştirdik
    IDataResult<List<Product>> GetAllByCategoryId(int Id); //ekledik 8.ders 03.02.2021
    IDataResult<List<Product>> GetByUnitPrice(decimal min, decimal max); //ekledik 8.ders 03.02.2021
    IDataResult<List<ProductDetailDto>> GetProductDetails(); // 06.02.2021

    IDataResult<Product> GetById(int productId); // eklendi 10.02 dikkat burada liste değil

    IResult Add(Product product); // eklendi 10.02

}
```

Not: **Business-Concrete** klasöründe **ProductManager** sorun verir. **IProductService** te yapılan değişikliklere göre burayı uyarlamalıyız.

Öncelikle Data sonuçları verecek class oluşturmalıyız

Step-57: **Core-Utilites-Results** klasörü altında **DataResult Class** oluştur. Public Yap **Result** Class 'ına ve **IDataResult** Interface inherit edelim. Ampülden Implemente edelim
Ctor 2*tab yapalim. Constructor yapısı otomatik olusun. İçini dolduralım

```
public class DataResult<T> : Result, IDataResult<T> // public tip duruma göre değişecek
{
    public DataResult(T data, bool success, string message): base(success, message) // resulttan tek farkı datasının olması
    {
        Data = data;
    }

    public DataResult(T data, bool success):base (success) //mesajsız formatı
    {
        Data = data;
    }
    public T Data { get; }
}
```

Step-58: **Business-Concrete** klasöründe **ProductManager** Classına gel

```
public IDataResult<List<Product>> GetAll()
{
    // is kodları

    if (DateTime.Now.Hour == 22) // saat kismi 22 ise hata döndür (denemk icin)
    {
        return new ErrorDataResult();
    }

    // yetkisi var mi?
    // tüm ürünleri listeleyecek metot
    return new SuccessDataResult<List<Product>>(_productDal.GetAll(),true,"ürünler listelendi"); //10.02.2021
}
```

Step-59: **Core-Utilites-Results** klasörü altında **SuccessDataResult** Class oluştur. Public Yap. **DataResult** Classına inherit et

```
public class SuccessDataResult<T>:DataResult<T> // public ve inherit
{
    public SuccessDataResult(T data, string message):base(data,true,message)
    {
    }
    public SuccessDataResult(T data):base(data,true) // mesajsız formatı
    {
    }
    public SuccessDataResult(string message):base(default,true,message)
    {
    }
    public SuccessDataResult():base (default,true)
    {
    }
}
```

Step-60: **Core-Utilites-Results** klasörü altında **ErrorDataResult** Class oluştur. Public Yap. **DataResult** Classına inherit et
SuccessDataResult metni **Error** için de yazacağımız. Buradan kopyalarız. Adını değiştir ve true değerlerini **false** yapalım

Step-61: **Business-Concrete** klasörü içindeki **ProductManager** Classına gel

```
public IDataResult<List<Product>> GetAll()
{
    // is kodları

    if (DateTime.Now.Hour == 22) // saat 22 (2200-2259) ise hata döndür (denemek için)
    {
        return new ErrorDataResult<List<Product>>(Messages.MaintenanceTime);
    }

    // yetkisi var mı?
    // tüm ürünleri listeleyecek metod
    return new SuccessDataResult<List<Product>>(_productDal.GetAll(),Messages.ProductsListed); //10.02.2021
```

}

Messages.MaintenanceTime üzerine gel Ampulden **Generate field** seçelim

Messages.ProductsListed üzerine gel Ampulden **Generate field** seçelim

Not: Ampulden Generate field ile messages Classında oluşan bu tanım satırları kontrol edilmeli ve düzeltilmelidir

Tüm **ProductManager** içindeki metodları **IDataResult** şeklinde bilgi girişi ve **SuccesDataResult** veya **ErrorDataresult** şeklinde dönüş olacak şekilde düzeltelim

```
public class ProductManager : IProductService
{
    IProductDal _productDal;

    public ProductManager(IProductDal productDal)
    {
        _productDal = productDal;
    }

    public IResult Add(Product product)
    {
        _productDal.Add(product);

        if (product.ProductName.Length < 2)
        {
            return new ErrorResult(Messages.ProductNameInvalid);
        }

        return new SuccessResult(Messages.ProductAdded); // step 49
    }
}
```

```

    // return new Result(true,"ürün eklendi"); // step 47 de yaptık. iptal ettik
}

public IDataResult<List<Product>> GetAll()
{
    // is kodları

    if (DateTime.Now.Hour == 22) // saat kismi 22 ise hata döndür (denemk icin)
    {
        return new ErrorDataResult<List<Product>>(Messages.MaintenanceTime);
    }

    // yetkisi var mi?
    // tüm ürünleri listeleyecek metot
    return new SuccessDataResult<List<Product>>(_productDal.GetAll(),Messages.ProductsListed); //10.02.2021
}

public IDataResult<List<Product>> GetAllByCategoryId(int id) //dikkat 02.03.2021
{
    // sadece secilen kategoride ürünler listelenecek
    return new SuccessDataResult<List<Product>>(_productDal.GetAll(p => p.CategoryId == id));
}

public IDataResult<Product> GetById(int productId) // Tanimladik 10.02..2021
{
    return new SuccessDataResult<Product>(_productDal.Get(p => p.ProductId == productId));
}

public IDataResult<List<Product>> GetByUnitPrice(decimal min, decimal max)
{
    // sadece secilen fiyat araligindaki ürünler listelenecek
    return new SuccessDataResult<List<Product>>(_productDal.GetAll(p => p.UnitPrice >= min && p.UnitPrice <= max));
}

public IDataResult<List<ProductDetailDto>> GetProductDetails()
{
    return new SuccessDataResult<List<ProductDetailDto>>(_productDal.GetProductDetails());
}
}

```

Step-62: **Business-Constants** klasörü içindeki **Messages** Classini aşağıdaki şekilde düzelt. **Maintenannce** ve **Productlisted** satırlarını düzelt

```
public static class Messages
{
    public static string ProductAdded = "Ürün eklendi";
    // degisen olmasına ragmen public oldugu icin büyük harfle başladık

    public static string ProductNameInvalid = "Ürün ismi geçersiz";
    public static string MaintenanceTime= "Sistem bakımda";
    public static string ProductsListed= "Ürünler listelendi";
    // Ampulden Generate field ile oluşan bu son 2 satır bu şekilde düzeltilmeli
}
```

Step-63: **Console Program** kısmını açalım

```
private static void ProductTest()
{
    ProductManager productManager = new ProductManager(new EfProductDal());
    var result = productManager.GetProductDetails();

    if (result.Success == true)
    {
        foreach (var product in result.Data)
        {
            Console.WriteLine(product.ProductName + "/" + product.CategoryName);
        }
    }
    else
    {
        Console.WriteLine(result.Message);
    }
}
```

Hatırlatma: Projeyi kaydedip **Get** ile yorumu bugünün tarihini yazıp “**Commit All**” yapıp **Push** yapmayı unutmayalım.

10.Ders Sonu Projenin Durumu

No	Projekt/Template	Projekt Reference	Klasör	Class/Interface	İşlem
1	MyFinalProject (Blank Solution)	---	Manage Nuget Packages	kısmından DataAccess a ilaveten Core içinde entityframeworkcore.sql (v 3.1.11) yükleyeceğiz	
2	DataAccess (Class.Library)	Entities Core	Abstract	<i>IProductDal</i>	<i>IEntityRepository<Product></i> ile inherit et
				<i>ICategoryDal</i>	<i>IEntityRepository<Category></i> ile inherit et
				<i>ICustomerDal</i>	<i>IEntityRepository<Customer></i> ile inherit et
				Önceden burda olan <i>IEntityRepository</i> tamamen Core.DataAccess altına alındı (Kes-Yapıştır)	
			<i>IOrderDal</i>	<i>IEntityRepository<Order></i> ile inherit et	
			<i>Concrete</i>	--	---
			<i>Concrete-InMemory</i>	<i>InMemoryProductDal</i>	<i>IProductDal</i> ile inherit et
			Concrete- EntityFramework	<i>EfCategoryDal</i>	<i>ICategoryDal</i> ile inherit et. <i>EfEntityRepositoryBase<Product,NorthwindContext></i> ile inherit et 8.derste oluşturduğumuz metotları sil
				<i>EfProductDal</i>	<i>IProductDal</i> ile inherit et <i>EfEntityRepositoryBase<Product,NorthwindContext></i> ile inherit et <i>ProductDetailDto</i> tipinde Liste tanımla 8.derste oluşturduğumuz metotları silip <i>IEntityRespository</i> 'e yazacağız
				<i>NorthwindContext</i>	<i>DbContext</i> ile inherit edilecek İlgili Class'ları (Entities) veri Tabanındaki karşılıkları ile eşleştir Product / Category / Customer / Order
				<i>EfOrderDal</i>	<i>IOrderDal</i> ile inherit et <i>EfEntityRepositoryBase<Product,NorthwindContext></i> ile inherit et
3	Business (Class.Library).	DataAccess Entities	Abstract	<i>IProductService</i>	<i>GetAll</i> , <i> GetAllByCategory</i> vb. metotları çağır ve listelerini oluştur <i>ProductDetailsDto</i> listesi oluştur <i>IResult</i> ile inherit et
				<i>ICategoryService</i>	metodunu çağır
			Concrete	<i>ProductManager</i>	<i>IProductService</i> ile inherit et <i>GetAll</i> , <i> GetById</i> , <i> GetAllByCategory</i> , <i>Add</i> vb metotları tanımla Ekle, sil, güncelle vb metotları <i>IResult</i> tipinde tanımla <i>GetAll</i> , <i> GetById</i> , <i> GetAllByCategory</i> vb İşlem Metotlarını <i>IDataResult</i> tipinde tanımla
				<i>CategoryManager</i>	<i>ICategoryService</i> ile inherit et <i>GetAll</i> , <i> GetById</i> vb metotları tanımla

No	Projekt/Template	Projekt Reference	Klasör	Class/Interface	İşlem
4	Entities (Class.Library) .	Core	Constants(sabit)	Messages	İşlemlere göre sonucu bildirecek metinleri serbest metin olarak değişkenlere tanımla
			Abstract	----	Önceden burada olan IEntity Interface Core -Entity altına alındı) (Kes-Yapıştır)
			Concrete	Product Category Customer	IEntity ile inherit et.
				Order	IEntity ile inherit et.
			DTOs	ProductDetailDto	IDto ile inherit et.
5	Core (Class.Library)		DataAcces	IEntityRepository	Diğer 3 interface için ortak ekle, sil, güncelle vb metotları çağır (EF kodları)
			DataAccess- EntityFramework	EfEntityRepositoryBase	Burada hem IEntity hem de DbContext klasından yeni bir klass newleyeceğiz EfProductDal ve EfCategoryDal 'da 8.derste tanımladığımız metotları kesip buraya alıp yeniden Tanımla
			Entities	IEntity	IEntity Class Entities-Abstract klasöründen buraya taşınır
				IDto	
			Utilities-Results	IResult	Başarı durumu ve mesaj değişkeni içersin
				Result	IResult ile inherit et. Farklı şekilde sonuç döndürecek metotlar yaz (basari +mesaj, sadece basari vb.)
				SuccessResult	Result Class' e inherit edelim. Farklı şekilde sonuç döndürecek metotlar yaz
				ErrorResult	Result Class' e inherit edelim. Farklı şekilde sonuç döndürecek metotlar yaz
				IDataResult	IResult Interface e inherit edelim. Farklı şekilde sonuç döndürecek metotlar yaz
				DataResult	IResult ve IDataResult ile inherit et. Farklı şekilde sonuç döndürecek metotlar yaz
				SuccessDataResult	DataResult Class' e inherit edelim Farklı şekilde sonuç döndürecek metotlar yaz
				ErrorDataResult	DataResult Class' e inherit edelim Farklı şekilde sonuç döndürecek metotlar yaz
6	ConsoleUI (Console.App)	Business DataAccess Entities	Set a startup project seç Kullanacağımız veri kaynağını tanımla. Veri kaynağından işlem yapabilmek için EfProductdal tipinde class new'le İstenen işlemi yap (ProductManager) (örnek: ürün adları yazdır) //Listeleri Joint yap Sonuçlara(results) göre bildirim (Messages) yap		

Not: Core katmanındaki her yapı diğer projelerde de kullanılabilir

Ek Kaynak: Clean Code Temiz Kod Yazma Sanatı (Bağlantı: https://www.youtube.com/watch?v=1MJpaMUdodU&feature=emb_logo)

Kendimi Nasıl geliştirmir? Robert C.Martin Clean Code Adlı Kitabı

- Udemy Eğitimlerim - Kupon Kodu : SUPER24
- C# Kursu: En Baştan En Sona, Pro Tekniklerle Programcılık | 4.7 puan
- Profesyonel C# Tasarım Kalıpları (Design Patterns) | 4.7 puan
- C# Kurumsal Backend 1- SOLID,Entity Framework,NHibernate,ORM | 4.7 puan
- C# Kurumsal Backend 2- AOP,IoC,Validation,Cache,Transaction | 5.0 puan
- C# Kurumsal Backend 3- ASP.NET,Log4Net,Performans | 4.7 puan
- C# Kurumsal Backend 4-ASP.NET, Rol Bazlı Kullanıcı Yönetimi | 4.7 puan
- C# Kurumsal Backend 5- Web Api, WCF,AutoMapper, AutoMapper | 4.7 puan
- C# Kurumsal Backend 6| Proje | Microservice İmplementasyon | 4.7 puan

Sorulması gereken sorular?

Projeniz ilerledikçe, değişen talepler, yeni özellikler, projeye olan aidiyetinizi nasıl etkiliyor?

Başkasının kodlarını okurken nasıl hissettiniz?

Başkası sizin kodunuzu okursa diye endişe ettiniz mi?

Gecen yıl yazdığınız koda bakınca ne hissediyorsunuz?

Yazılım işi bittikten sonra **Refactoring** yapmak imkânsızı yakındır?

Her müşteri, fazla ve uygulaması zor isteklerde bulunur. Daha sonra fikir değiştirebilir veya yeni talepte bulunabilir. Clean Code yazarsan sisteme müdahale edebilmek mümkündür.

Interface yazılımda değişim odaklı bir sistem olduğunun farkında değiliz demektir

Öğrendiklerimizi “**gerçek Hayatta nasıl kullanılır?**” şeklinde düşünmeliyiz.

Clean Code Kurumsal Hafıza için de çok önemlidir. (Kodu sadece siz yazıyor ve kullanıyor da olsanız)

“**SonarCube**” adlı araç kodunuzu **clean code** prensibine göre inceleyerek ve tavsiyede bulunur

Review Code çok önemli

“Çalışıyorsa dokunmam” mantığı burada geçerli olmayabilir. Çünkü Kodlar, taleplere ve etkileşimde olduğu diğer projelerdeki değişime istinaden sürekli değişim içerisinde olmalıdır

1. İsimlendirme Kuralları

C# Naming Conventions şeklinde C# adlandırma kurallarını araştırabilirsin

Adlandırmalar **anlamlı** olmalı (Mümkünse İngilizce)

Static değişkenlerin adlandırması küçük harfle başlamalı, (Kısaltma olmasın, yeni bilgisayarlarda Complexity bu isimlendirmelerden artık çok etkilenmez)

Dinamik değişkenlerin (dizi, liste, class vb) adlandırması (birleşik kelimelerde her kelime bas harfi) büyük harfle

2. Code Refactoring (C# Nesnel bir dildir)

1 Fonksiyon sadece 1 işi yapacak şekilde oluşturulmalıdır. Fonksiyon içinde 1'den fazla fonksiyonu çağrıbilirsin. Adlandırmalarda yapılacak işi anlatır. Ayrıca alt fonksiyonlardan biri başka yerlerde lazım olduğunda bunu orada da çağrıbilirsin

Denge de önemlidir. **Overdesign** olmasın ancak işlemler mümkün mertebe en küçük birim halinde tanıtılsın

3. Yorumlar

Fonksiyon başlıklarının üstüne genel yorum yazılabılır ancak fonksiyon içerisinde Kurumsal Hafıza için yapılan bu işlem yanlış yerde yapılıyor demektir

4..Soyutlama:

Interface önemli. Çıplak Class kalmayacak. Her Class başka bir Classa veya Interface'e inherit edilecek

5.SOLID :

Single Responsibility

Open Closed

Liskov's Substitution

Interface Segregation

Dependency Inversion

6. Aspect Oriented Programming (AOP)

11.DERS (13.02.2021)

Konu : İleri seviye Web API kodlama

Kullanılan Dosya : Visual Studio => Project =>**MyFinalProject (DEVAM)** ve Postman API Test programı

Ek Kaynak : <https://www.btkakademi.gov.tr/portal/course/c--7008#!/about>
(25,26,27,28. Bölümleri) Attribute ile başlıyor, Events ile bitiyor.

KURULUM :Postman (<https://www.postman.com/downloads/>) **WebApi için Test Ortamı sağlayan bir Program**

STEP-64: **MyFinalProject Solution** üzerinde sağ tıklayıp **ASP.NET.Core** türünde **WebAPI** adlı bir proje ekleyeceğiz.

Create tuşu sonrası önceki proje türlerinden farklı olarak veri transferine müsaade eden **API** tipinde olması için seçim yapacağız

Versiyon seçimi dikkat edelim (3.1)

Oluşturduğumuz **WebAPI** projesini “**Set a StartUp Project**” olarak seçelim

Bugüne kadar testleri **Console** katmanı aracılığı ile yapıyorduk. Artık **WebAPI** ile yapacağız

Business Abstract içinde bulunan Service Interface ‘ların içindeki fonksiyonların her biri bir işlem yapmaktadır.

Bunların her birini API aracılığıyla tetikleyen ve sergileyen bir ara yüz hazırlayacağız

STEP-65: **WebAPI** üzerinde sağ tıklayıp **Add Project Reference** kısmında “**Console haric**” hepsini seçelim (etkileşimde olacaklar)

STEP-66: WebAPI => Controllers klasöründe Sağ tıklayıp ProductsController adlı controller oluşturalım

(Not: Kendi oluşmaz ise WebAPI katmanında sağ tıklayarak Controllers (cogul) adlı klasör oluşturalım)

ADD =>controller=>API=>API Controller-Empty

Burada kendimiz ürün tanıtalım ve API'de sergiletem

```
namespace WebAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ProductsController : ControllerBase
    {
        [HttpGet]
        public List<Product> Get()
        {
            return new List<Product>
            {
                new Product{ProductId=1, ProductName = "Elma"},
                new Product{ProductId=2, ProductName = "Armut"},
            };
        }
    }
}
```

STEP-67: Programı **kósturduğunda** açılan **browsera** aşağıdaki linki kopyalarsak çalışması lazım. <https://localhost:44356/api/products>

Güvenlik nedeniyle çalışmayabilir

Postman 'da deneyelim: <https://localhost:44356/api/products> bağlantı linkini **GET** kısmına yazıp **Sent** tuşuna basınca çalışması gerekiyor

[HttpGet] için **GET**
[HttpPost] için **POST**

STEP-68: Aşağıdaki kodu deneyelim. Burada veri tabanında yer alan bilgileri API'de sergiletelim

```
namespace WebAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ProductsController : ControllerBase
    {
        [HttpGet]
        public List<Product> Get()
        {
            IProductService productService = new ProductManager(new EfProductDal());
            var result = productService.GetAll();
            return result.Data;
        }
    }
}
```

STEP-69: **ProductsController** üzerinde aşağıdaki tanımlamayı yapalım (fonksiyon ampulden **Generate** yapınca otomatik oluşturuldu)

Burada Class ' hemen new'lemek yerine local class ataması yapılır ve startup kısmına yapılacak ilaca kodla sadece lazım olduğunda newleme yapılır

```
namespace WebAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ProductsController : ControllerBase
    {
        //Loosely coupled
        //naming convention
        //IoC Container -- Inversion of Control
        IProductService _productService;           //yaptığımız tanımlama

        public ProductsController(IProductService productService)
        {
            _productService = productService;
        }
}
```

```

[HttpGet]

public List<Product> Get()
{
    //Dependency chain --
    var result = _productService.GetAll();
    return result.Data;

}
}

```

STEP-70: WebAPI içindeki **Startup Classına** gel. Aşağıda zaten mevut olan fonksiyon içini doldur

Burasına yazacağımız kod başlangıçla beraber tanıttığımız referans tipleri gerektiğinde new'ler

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers(); // başlangıçta yanlışlıkla boş api seçtiğim için bu kişim bende otomatik çıkmamıştı
    services.AddSingleton<IProductService, ProductManager>(); // bu newleme gerektiğinde new'ler
    services.AddSingleton<IProductDal, EfProductDal>(); // bu newleme gerektiğinde new'ler
}

```

STEP-71: Postman 'da deneyelim:

<https://localhost:44356/api/products> bağlantı linkini **GET** kısmına yazip

Sent tuşuna basınca çalışması gerekiyor

Postman Status "200 Ok" yazması her şeyin yolunda olduğunu gösterir

400 kodu olsaydı yetkisiz giriş anlamına gelirdi

The screenshot shows the Postman application window. In the center, there's a request card for a GET request to 'https://localhost:44356/api/products'. Below the card, the status bar displays 'status: 200 Ok'. A red circle is drawn around this status code to emphasize it.

STEP-72: **WebAPI içindeki Controller =>ProductsController klasörüne gel.**

public IActionResult Get() kısmını sonuca göre **status** veren ve işlem yapan bir yapı kuralım ve koşturalım.
ProductManager'dan saat bakım durumundaki saatı mevcut zaman göre ayarlayıp çalışıralım. Sistem bakımında yazsın

```
public class ProductsController : ControllerBase
{
    //Loosely coupled
    //naming convention
    //IoC Container -- Inversion of Control
    IProductService _productService;

    public ProductsController(IProductService productService)
    {
        _productService = productService;
    }

    [HttpGet]
    public IActionResult Get()                                //Dikkat birazdan başka yapıda kullanacağız
    {                                                       // Dikkat burası da duruma göre değişecek
        //Dependency chain --
        var result = _productService.GetAll();
        if (result.Success)
        {
            return Ok(result);
        }
        return BadRequest(result);
    }
}
```

STEP-73: **WebAPI içindeki** Controller ProductsController klasörüne bu fonksiyonu ekle

Fonksiyon Adının olduğu satırı BREAKPOINT KOY (Visual Studio 'da o satırın en soluna tıklayınca kırmızı nokta çıkar)

Postman 'da deneyelim: <https://localhost:44356/api/products> bağlantı linkini POST **DİKKAT** kısmına yazıp **Send** tuşuna basınca çalışması gerekiyor

Program hata verir. Sebebi ürün ekle diyoruz ama bir ürün bilgisi göndermiyoruz

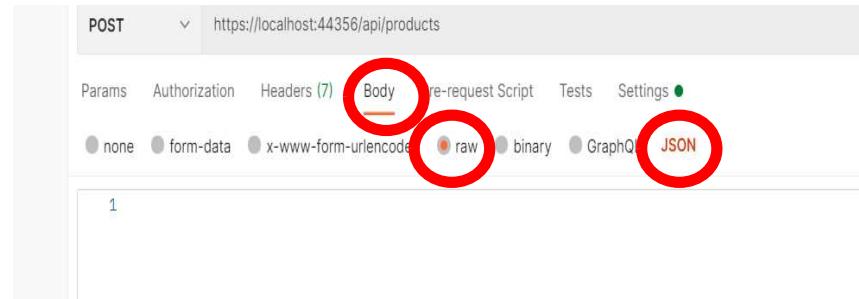
```
[HttpPost] //DİKKAT Silme ve güncelleme içinde [HttpPost] tercih edilir

public IActionResult Post(Product product) //DİKKAT
{
    var result = _productService.Add(product);
    if (result.Success)
    {
        return Ok(result);
    }
    return BadRequest(result);
}
```

STEP-74: Bit Get ile **Postman** 'i çalıştır bir ürün bilgisini alalım.

Postman 'in yandaki şeklinde açılan kısmına yapıştırıralım yalnız **ProductID** satırını silelim. Çünkü veri tabanı PK(Primary Key olan bu değeri) otomatik veriyor. Diğer bilgileri rasgele girelim ve **send** tuşuna basalım

Yandaki işlem aslında web sayfasında yaptığımımız işlemler



POST https://localhost:44356/api/products

Params Authorization Headers (8) Body (highlighted) Pre-request Script Tests Settings Cookies

Body (JSON)

```
1
2   "categoryId": 1,
3   "productName": "Bardak",
4   "unitsInStock": 15,
5   "unitPrice": 25
6
```

Send

Programımızda **Breakpoint** 'in üstündeki produkt yazısına tıklayınca bizim girdiğimiz bilgiler görünmelidir. Bilgi girişi olduğunu teyit ettik

Devam edelim (F5 basalım.)



https://localhost:44356/api/products

POST https://localhost:44356/api/products

Params Authorization Headers (8) Body (highlighted) Pre-request Script Tests Settings Cookies

Body (Raw)

```
{"success": true, "message": "Ürün eklenmiştir."}
```

Raw

Status: 200 OK Time: 2 m 45.96 s Size: 225 B Save Response

STEP-75: Postmanda GET ile Send yapalım ve listemizi görelim. En alta girilen yeni ürün ve sonuç raporu görülmüyor. İşlem başarılı

The screenshot shows the Postman application interface. At the top, there's a search bar labeled "Search Postman" and various navigation icons. Below the header, the URL "https://localhost:44356/api/products" is entered into the address bar. The main area displays a GET request configuration with "GET" selected and the URL again. To the right of the URL is a large blue "Send" button, which is also circled in red. The response section shows a JSON object with a list of products. A large red oval highlights the entire JSON structure, including the array of products and the success message at the bottom.

```
529     "productId": 76,
530     "categoryId": 1,
531     "productName": "Lakkalikööri",
532     "unitsInStock": 57,
533     "unitPrice": 18.0000
534   },
535   {
536     "productId": 77,
537     "categoryId": 2,
538     "productName": "Original Frankfurter grüne Soße",
539     "unitsInStock": 32,
540     "unitPrice": 13.0000
541   },
542   {
543     "productId": 78,
544     "categoryId": 1,
545     "productName": "Bardak",
546     "unitsInStock": 15,
547     "unitPrice": 25.0000
548   }
549 ],
550 "success": true,
551 "message": "Ürünler listelendi"
552 }
```

STEP-76: **WebAPI-Controllers-ProductController** içine aşağıdaki fonksiyonu yazalım.

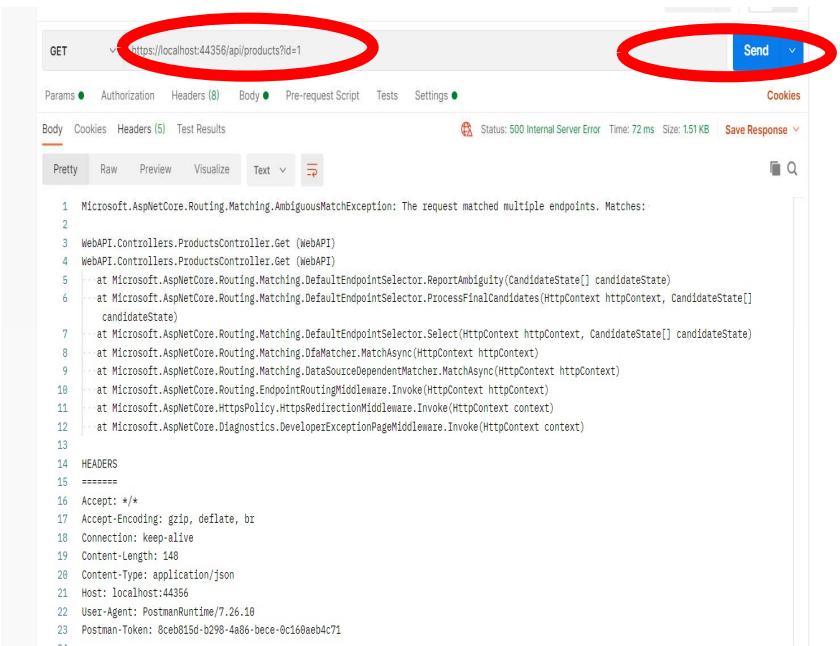
Yazdığımız fonksiyonun **if** satırına **Breakpoint** koyalım.

Çalıştırılm. Hata verir.

Postman 'a GET olarak <https://localhost:44356/api/products?id=1> yazıp

Çalıştırınca da hata olduğu görülür

```
[HttpGet]
public IActionResult Get(int id)
{
    var result = _productService.GetById(id);
    if (result.Success)
    {
        return Ok(result);
    }
    return BadRequest(result);
}
```



STEP-77: **[HttpGet]** ve **[HttpPost]** komutlarını ilgili fonksiyonlara göre aşağıdaki gibi isimlendirelim ve Fonksiyon adlarını da değiştirelim

```
[HttpGet ("getall")]
public IActionResult GetAll()
{
    //Dependency chain --
    var result = _productService.GetAll();
    if (result.Success)
    {
        return Ok(result);
    }
    return BadRequest(result);
}
```

*//fonksiyon Adına göre değişti
// adı Get idi GetAll oldu*

```

[HttpGet("getbyid")]
//fonksiyon adına göre değişti
public IActionResult GetById(int id)
{
    var result = _productService.GetById(id); // adı Get idi GetById oldu
    if (result.Success)
    {
        return Ok(result);
    }
    return BadRequest(result);
}

[HttpPost("add")]
//fonksiyon adına göre değişti
public IActionResult Add(Product product)
// adı post idi Add oldu
{
    var result = _productService.Add(product);
    if (result.Success)
    {
        return Ok(result);
    }
    return BadRequest(result);
}

```

Yine Çalıştırıralım. Status 404 hatası verdi (kaynak hatası)

The screenshot shows the Postman interface with a failed API request. The URL 'https://localhost:44356/api/products' is highlighted with a red circle. Below it, the 'Send' button is also circled in red. The status bar at the bottom displays 'Status: 404 Not Found'. The interface includes tabs for Body, Cookies, Headers (4), and Test Results.

Postman Get kısmına artık özel isimlendirdiğimiz [**HttpGet**] adlarına göre çağrırsak çalışır [**HttpPost("Getall")**] ve [**HttpPost("Getbyid")**]

Veya **Post** kısmına isimlendirdiğimiz [**HttpPost**] adlarına göre çağrırsak çalışır isimlendirdiğimiz [**HttpPost("Add")**])

The screenshot shows the Postman interface with a red circle highlighting the URL field containing "https://localhost:44356/api/products/getall" and the "Send" button at the top right.

Request details:

- Method: GET
- URL: https://localhost:44356/api/products/getall
- Headers: (8)
- Body: (Green dot)
- Pre-request Script: (Green dot)
- Tests: (Green dot)
- Settings: (Green dot)

Response status:

- Status: 200 OK
- Time: 4.63 s
- Size: 8.1 KB

Body (Pretty):

```
1
2   "data": [
3     {
4       "productId": 1,
5       "categoryId": 1,
6       "productName": "Chai",
7       "unitsInStock": 39,
8       "unitPrice": 18.0000
9     }
]
```

The screenshot shows the Postman interface with a red circle highlighting the URL field containing "https://localhost:44356/api/products/getbyid?id=1" and the "Send" button at the top right.

Request details:

- Method: GET
- URL: https://localhost:44356/api/products/getbyid?id=1
- Params: (Green dot)
- Authorization: (Grey)
- Headers: (8)
- Body: (Green dot)
- Pre-request Script: (Green dot)
- Tests: (Green dot)
- Settings: (Green dot)

Response status:

- Status: 200 OK
- Time: 358 ms
- Size: 310 B

Body (Pretty):

```
1
2   "data": {
3     "productId": 1,
4     "categoryId": 1,
5     "productName": "Chai",
6     "unitsInStock": 39,
7     "unitPrice": 18.0000
8   },
9   "success": true,
10  "message": null
11 }
```

Sık Karşılaşılan HTTP Statü Kodları

200: Ok (İstek Başarılı)
201: Created (Oluşturuldu)
301: Moved Permanently (Kalıcı taşındı)
400: Bad Request (Kötü İstek)
401: Unauthorized (Yetkisiz)
403: Forbidden (Yasaklandı)
404: Not Found (Sayfa Bulunamadı)
405 :Method not Allowed
500: Internal Server Error (Dahili Sunucu Hatası)
502: Invalid Gateway(Geçersiz Ağ Geçidi)

Postman Metotları

GET methods retrieve data from an API.
POST sends new data to an API.
PATCH and PUT methods update existing data.
DELETE removes existing data.

12.DERS (17.02.2021)

Konu : **Autofac (her API için ortak)**

Fluent Validation (her katmanda kullanılabilecek hususlar)

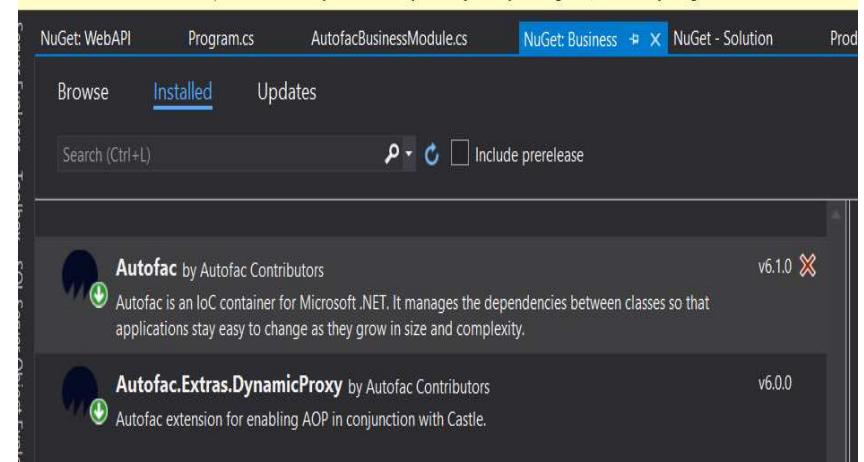
AOP

Autofac :IOS yapılandırmasını Autofac”e çevireceğiz.

NEYI NICIN YAPIYORUZ: WebAPI projesi Startup içinde **sevices.AddControllers()**; kısmında yapabiliriz ancak yeni bir API eklediğimizde bu işlemi bir daha yapmak zorunda kalırız. Bu yapılandırmayı daha geriye taşıyabiliyoruz. Onun yerine **AUTOFAC** yapılandırması yapabiliriz

Step-78 EKLENTİ: Business sağ tıkla “Manage nuget Packages” kısmını açalım

Browse kısmında **Autofac** yazarak arayalım Son versiyonu (V 6.1..0) kuralım. Sıkıntı olursa 3.11.1i kurabiliriz



Step-79 EKLENTİ: Browse kısmında **autofac.EXTRAS** yazarak arayalım

Autofac.EXTRAS.DynamicProxy adlı eklentiyi kuralım (Son versiyon)

Step-80

BUSINESS içinde **DependencyResolvers** adlı klasör oluşturalım.

Altına **Autofac** adlı bir klasör oluşturalım Burada **IPRODUCTDAL'in** karşılığını tanımlayacağız

WebAPI startup içinde içinde yaptığımz ayarları **Businessa** taşıyacağız

Step-81

Autofac içinde **AutofacBusinessModule** adlı class ekle (aynisi kendi core katmanımızda da yapacağız)

Public yapalım **Module** 'e inherit yapalım. ampulden “**using Autofac**” seçelim

Override yazıp **space** tuşuna basarsak seçenekler çıkar

“**Load**” adlı metodu seçelim

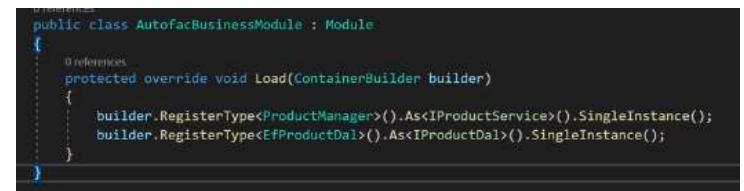
WebAPI startup da **services.addsingleton<...>** olarak yazdığımız kodun karşılığını yazacağız

ProductManager ve **IProductService** ampulde **using** komut satırını ekle.

Aynisini **EFProductDal** ve **IProductDal** içinde yap.

(örneğin bir web sayfasında 100 kişi için aynı eylemi istediğiinde ayrı ayrı çalışırmak yerine tek bunu kullanıyor)

Bunu diğerleri içinde yapalım

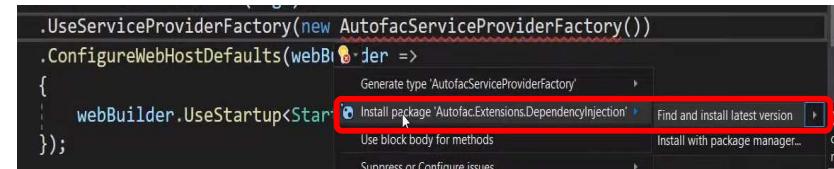


```
0 references
public class AutofacBusinessModule : Module
{
    protected override void Load(ContainerBuilder builder)
    {
        builder.RegisterType<ProductManager>().As<IProductService>().SingleInstance();
        builder.RegisterType<EFProductDal>().As<IProductDal>().SingleInstance();
    }
}
```

STEP-82 :**WebAPI** startup kısmındaki `services.AddSingleton.....` diye başlayan satırları iptal edelim. Çalışmayacaktır
Burada kendi IOS yapılandırması yerine kendi tanımladığımız yapılandırmayı kullanmasını sağlayacağız

STEP-83: **WebAPI** 'nin Program kısmını açalım

Service Provider olarak “**AutofacServiceProviderFactory**” tanıtabileceğiz ve ampulden “**Install Packages AutofacExtensionsDependencyInjection**” şeklinde 2. Seçenekten “**Find and Install Latest Version**” seçeceğiz.



EKLENTİ: Yukarıda aslında kurulum yaptıktı

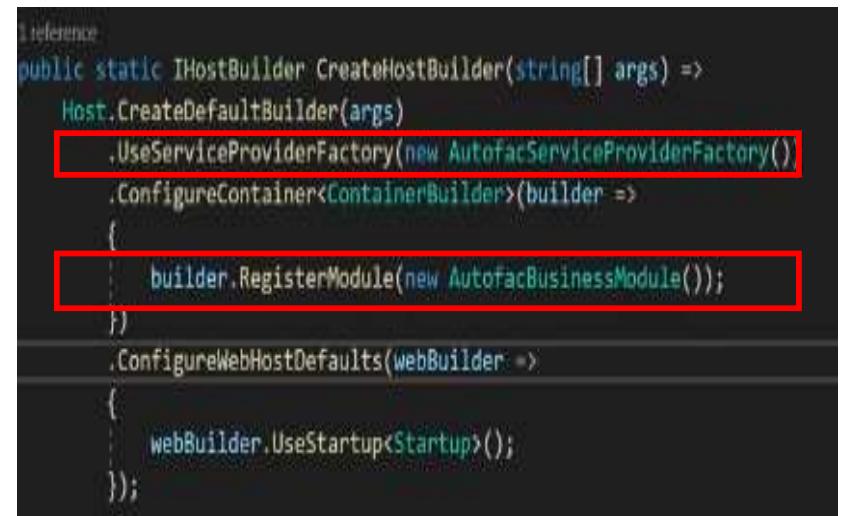
Referansı tutan bir yapı oluşturdu

Configure Container kısmını yapacağız. Yazdığımız **AutofacBusinessModule** yapılandırımda kullanılacağını belirteceğiz

ContainerBuilder için Ampulden **Using** komutu satırını çözelim

AutofacBusinessModule için de **Using** komut satırını çözelim

Şimdi **WebAPI** programı tekrar çalışıralım ve **Postman**’de çalıştığını görelim.
Çalıştı



FluentValidation : FluentValidation AOP ile birleştireceğiz.

NEYI NICIN YAPIYORUZ:

Business projesi **Concrete** klasörü **ProductManager** Add metoduna gelelim

Business kodu ile Validation kodunu karıştırmayalım. Hepsi yerinde olmalı.

Add metodunda ismin en az 2 karakter olması bir validation kodudur

Bunları merkezi bir noktada yapabiliyoruz. **FluentValidation** ile yapacağız

Bunları **Attributes** ile de yapabiliyoruz. Ama mesela eskiden TC kimlik No ile girişe göre ayarladığınız bir kodu yabancılarında girmesine müsaade etmek için uyarlamaya kalkarsanız çok uğraşırsınız.

STEP-84: **Business** projesine sağ tıkla “**Manage nuget packages**” tıkla.
Browse **FluentValidation** yazılım ve **Jeremy Skinner** tarafından yazılmış olan paketi yükleyelim

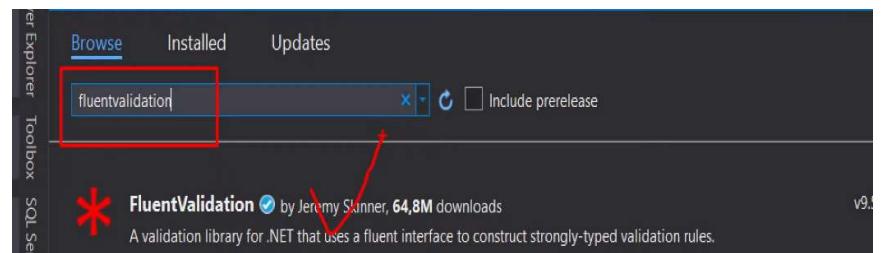
STEP-85: **Business** projesine sağ tıkla **ValidationRules** adlı klasör oluşturalım

Onun altına da “**FluentValidation**” adlı bir klasör oluşturalım

```
public IResult Add(Product product)
{
    //business codes
    //validation

    if (product.UnitPrice <= 0)
    {
        return new ErrorResult(Messages.UnitPriceInvalid);
    }

    if (product.ProductName.Length<2)
```



STEP-86: **FluentValidation** üzerinde sağ tıklayıp **ProductValidator** adlı bir Class ekleyelim

Public yapalım, ile inherit edelim.

Hangi nesne için oluşturuyorsak onu yazacağımız.

Burada **AbstractValidator<Product>** ile inherit edecegiz

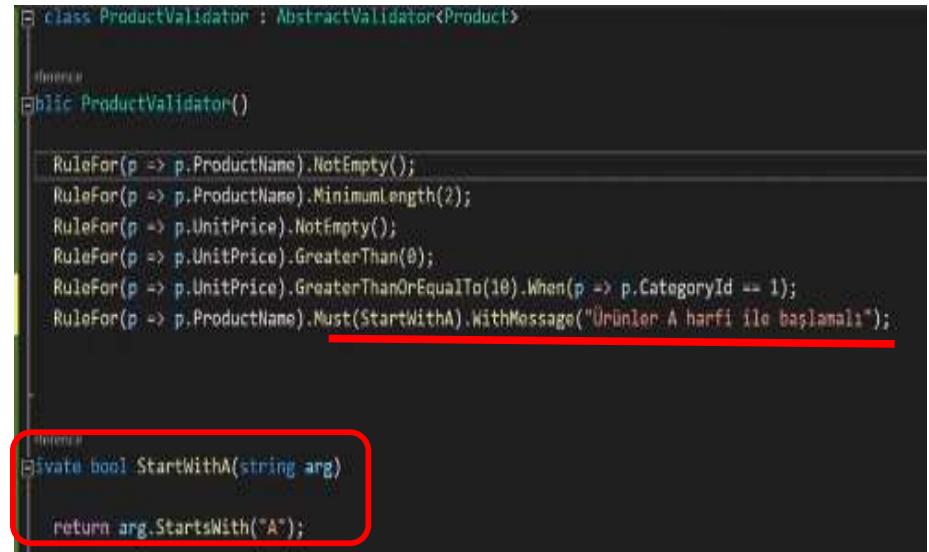
Ampulden **AbstractValidator** ve **Product** için **Using** komut satırını ekleyelim

Ampulden **AbstractValidator** çözelim. Metot olusun

Kuralları hazır kütüphaneden kullanalım veya biz oluşturalı. Hata mesajlarını da aynı şekilde yapabiliriz.

Fluentvalidator içinde 19 dilde Otomatik hata mesajları tanımlıdır

ancak **withmessages** kodu ile kendi özel mesajını verebiliriz



```
class ProductValidator : AbstractValidator<Product>
{
    public ProductValidator()
    {
        RuleFor(p => p.ProductName).NotEmpty();
        RuleFor(p => p.ProductName).MinimumLength(2);
        RuleFor(p => p.UnitPrice).NotEmpty();
        RuleFor(p => p.UnitPrice).GreaterThan(0);
        RuleFor(p => p.UnitPrice).GreaterThanOrEqualTo(10).When(p => p.CategoryId == 1);
        RuleFor(p => p.ProductName).Must(StartWithA).WithMessage("Ürünler A harfi ile başlamalı");
    }

    private bool StartWithA(string arg)
    {
        return arg.StartsWith("A");
    }
}
```

Ctrl k+Ctrl k : boşluksuz veya fazla boşluklu yazılan kodları otomatik düzenler

Otomatik bazı validation kuralları metodu mevcuttur. Ancak biz de metot oluşturabiliriz. Örneğin ürünlerin A harfi ile başlaması kuralını koyabiliriz

Kurallar içine **StartWithA** adlı bir metod adı ile bir kural koyalım ve o Metot içine kural kodlarını yazalım. Yukarıda metot isminin üzerinden ampulle generate Metot yapabiliriz

STEP-87: **Business Concrete** klasörü **ProductManager** classına gel

ProductManager içersinde **Add** metodu içinde olan kendi yazdigimiz validation kodlarini silelim. Yerine yeni kodlarimizi yazalim

ValidationContext newleyip ProductValidator newleyecegiz

Bunları yazınca ampulden using komut satrini ekleyelim/cözelim

Kurallara uygunlugunu kontrol ettirecegiz

Kodu birazdan iyileştirecegiz (bu en spaghetti yöntemi)

Kosturma(Run): Bunu bir çalıştırıalım.

Postmandan bir ürün ekleyelim kurallara uygun ve uygun olmayan ürün ekleyelim

Untuma! **Post** kısmını çalıştıracağiz. Body ,Raw ve Json şeklinde iken bilgiyi girecegiz

Basari ve Hata kodlarını, dolayısıyla eklientini calistagini bir görelim.Sonra kendi yapimiza göre iyileştirelim

(A ile baslayan ve başlamayan ürün adı ile ürün adeti 10 dan az ve fazla olarak gir)

Simdi bunu bütün projelerde kullanabilmek için **Core** katmanına koyabiliriz

STEP-88 :**Core** Katmanına **CrossCuttingConcerns** adlı yeni bir klasör ekle

Altına da **Validation** adlı yeni bir klasör ekle

Onun altına da **ValidationTool** adlı class ekle. **Public static** yap.

ProductManagement icine Validation icin yazdigim kodu taşıyacağız

Yanda **ProductManagement** yeni hali gözükmekte

```
2 references
public IResult Add(Product product)
{
    var context = new ValidationContext<Product>(product);
    ProductValidator productValidator = new ProductValidator();
    var result = productValidator.Validate(context);

    if (!result.IsValid)
    {
        throw new ValidationException(result.Errors);
    }

    _productDal.Add(product);
    return new SuccessResult(Messages.ProductAdded);
}
```

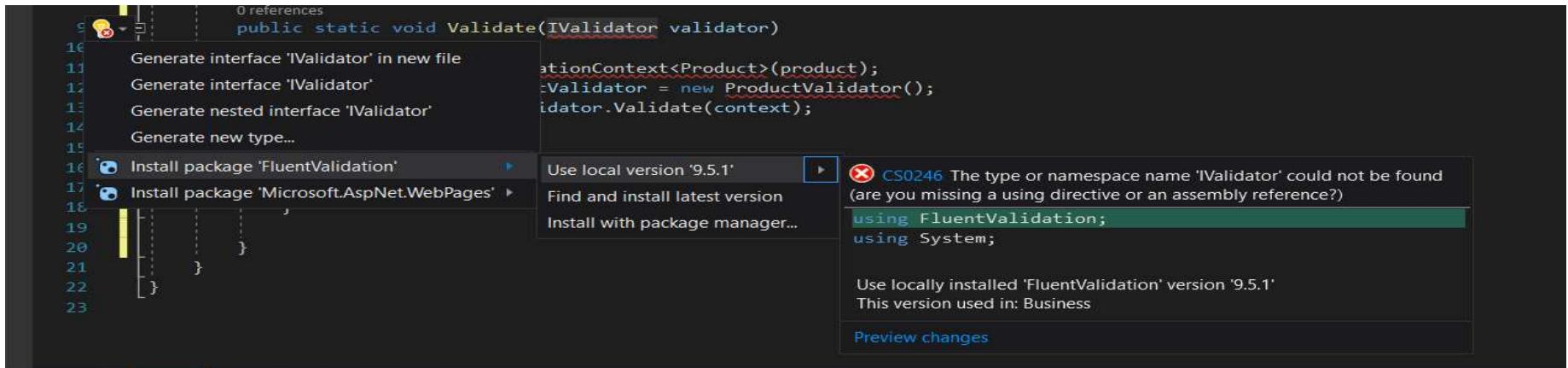
```
2 references
public IResult Add(Product product)
{
    // business codes

    _productDal.Add(product);

    return new SuccessResult(Messages.ProductAdded);
}
```

Validate adlı Metota **IValidator** tipinde bir nesne gönderecegiz. Ivalidate yazınca Ampulden “Install Package “FluentValidation” => “Use Local version 9.5.1.” sececegiz.

Ekleni: Aslinda bu sekilde projede yüklü olan **FluentValidation** ekini **Core** icin de yüklemis/kurmus olduk



Artık Metota sadece **Produkt classtan** gelecek **produkt nesnesine** göre degil, herhangi bir **entity'e** göre calisacak sekilde uyarladik.
Yukarda **Produkt classindan** newledigimiz nesne satirini sildik. Metota nesen geliyor zaten.

Produkt class isimi yerine **object** yazdik
produkt nesnesi yerine de **Entity** yazdik

Not: bir class vb. yapinin üzerine gelip F12 'ye basarsan o yere gider

Aciklama

Cross Cutting Corcerns: **Loglama, Cache, Trasaction , Authoruzation, Validation** vb yapılar projenin tüm katmanlarında kullanilan yapılardır (Database, Business ve UI (User Interface))

STEP-89: **Business** Katmanı **Concrete** Klasörü **ProductManager** Classına **Add** metoduna sadece bu yazdığımız metodu çağıracak bir kod yazacağız

ValidationTool Classını ampulden **using** komut satırını ekleyeceğiz/çözeceğiz

Koşturma (Run): Çalışıp çalışmadığını kontrol edeceğiz

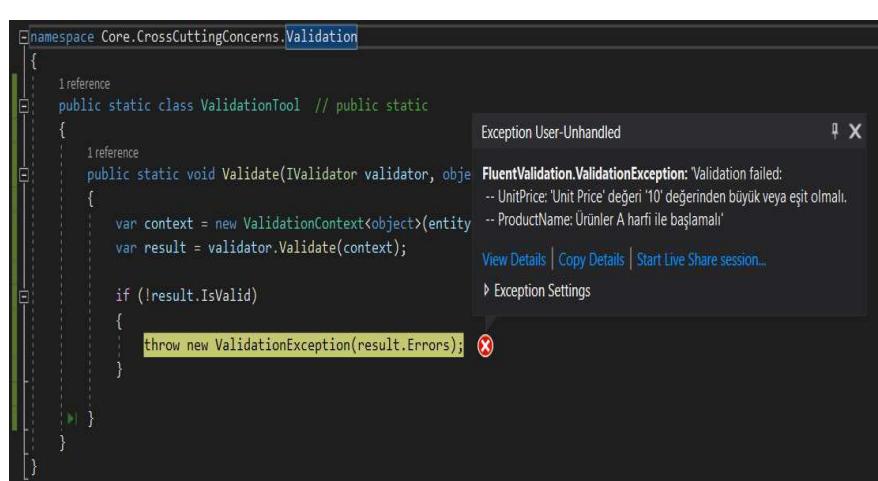
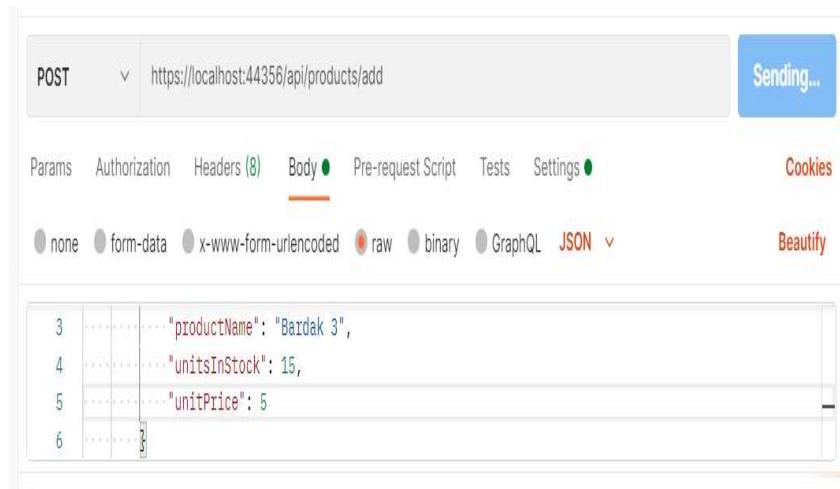
```
2 references
public IResult Add(Product product)
{
    ValidationTool.Validate(new ProductValidator(), product);

    _productDal.Add(product);

    return new SuccessResult(Messages.ProductAdded);
}
```

A ile başlamayan ve fiyatı 10'dan küçük olan bir ürün eklemeye çalışınca Program Validation'a uymayan ürün bilgisi girildiği uyarısı verdi.

Demek ki program doğru çalışıyor



AOP Destegi:

NEYİ NİCİN YAPIYORUZ : Add metodu içine **Business** kodu yazmalıyız. Ancak **Validation**, **Loglama**, **Performance**, **transaction**, **yetkilendirme** kodları da yazmalıyız.

Asıl işlevi dışında birçok komut satırı olacak. Onun yerine Add, GetAll vb metodu üstüne yazacağımız kod ile kuracağımız yapı çalışarak **Validation** yapacak, Add metodunda sadece business kod olacak (AOP yapısı) **Validation**, **Loglama** vb işlemleri işlem görecek metotun içine komut yazmak yerine üzerine (dişında üstünde) **Attributes** olarak ekleyeceğiz. **Attribute** bir metod çalışmak için tetiklendiğinde önce çalışmasını istediği işlemleri tanıtmaya yarar Metotlara anlama yükler. **Attributes** metodlarına tip gönderdiğimizi de unutmamalım. Metodu çağırırken nesnenin tipini göndereceğiz

Örnek : [AttributeUsage(AttributeTargets.Class|AttributeTargets.Method,AllowMultiple = true, Inherited = true)]

Anlamı : Classlara veya metotlara ekleyebilirsin, birden fazla ekleyebilirsin ve inherit edilen bir noktada bu Attribute cal ıssın

Hem veri tabanına hem bir dosyaya loglama yapmak isteyebiliriz

Tanım: **AOP:** Metotları loglamak istediğimizde çalışmasını istediği zamanlarda (örneğin başında, sonunda ve hata verdiğinde) çalışmasını **INSERCEPTION** yöntemiyle sağlayabiliyoruz

Step 90: **Solution Explorer** ‘dan **Solution** üzerinde **sağa** tıklayıp “**Manage NuGet Packages und Solutions**” açalım
Installed kısmını açalım

Eklenti:

1. **Autofac by Autofac Contributors** (**Business** katmanı için Installed yaptığımız olanları) Core için de yükleyeceğiz
2. **Autofac.Extras.DynamicProxy** (**Business** katmanı için Installed yaptığımız olanları) Core için de yükleyeceğiz
3. **WebAPI** katmanı için Installed yaptığımız **Autofac.Extensions.DependencyInjection** Core içinde yükleyeceğiz

Step 91: **Core** katmanı=> **Utilities** klasörü altına **Interceptors** adlı bir Klasör ekleyelim

Onun içine de adını bile vermeyeceğimiz bir **Class ekle.**

Aşağıda belirtilen **Repository** 'deki projenin içindeki 3 **Class** İçindeki kodları bu eklediğimiz Class içine **kopyalayalım (karışmasın)**

Engin Demirog'un GITHUP Repository'dan hazır kod alacağız
“**NetCoreBackend**” adlı Proje **Core** katmanı=> **Utilities** klasöründen

1. **MethodInterceptionBaseAttribute** adlı Class 'i açalım. (kodlar burada)
Ekleyince; **IInterceptor** interface ampulden **using Castle.DynamicProxy** satırını ekle/cöz

2. “**NetCoreBackend**” adlı Proje **Core** katmanı=> **Utilities** klasöründen **MethodInterception** adlı Class 'i açalım. (kodlar burada)
Attribute' lerin ne zaman çalışacağına belirten kodlar

3. “**NetCoreBackend**” adlı Proje **Core** katmanı=> **Utilities** klasöründen **AspectInterceptorSelector** adlı Class 'i açalım. (kodlar burada)

Metodun Attribute 'ları okuma, Priority değerine göre sıralama kodları

Aşağıdakiler için ampulden **using** komut satırlarını ekle/çöz
(tanımama uyarıları (altı kırmızı çizili) gider)

```
MethodInfo icin      „using System.Reflection”
Tolist icin          using System.Linq
```

Loglamaya ilgili bu satırı sileceğiz (şimdilik isimiz yok)

```
classAttributes.Add(new ExceptionLogAspect(typeof(FileLogger)));
// Otomatik tüm metodları loga dahil eden komut satırı
```

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method, AllowMultiple = true, Inherited = true)]
3 references
public abstract class MethodInterceptionBaseAttribute : Attribute, IInterceptor
{
    1 reference
    public int Priority { get; set; } // atributeslar arası calisma öncelik belirtme

    1 reference
    public virtual void Intercept(IInvocation invocation)
    {
    }
}
```

```
0 references
public abstract class MethodInterception : MethodInterceptionBaseAttribute
{
    1 reference
    protected virtual void OnBefore(IInvocation invocation) { }

    1 reference
    protected virtual void OnAfter(IInvocation invocation) { }

    1 reference
    protected virtual void OnException(IInvocation invocation, System.Exception e) { }

    1 reference
    protected virtual void OnSuccess(IInvocation invocation) { }

    1 reference
    public override void Intercept(IInvocation invocation)
    {
        var isSuccess = true;
        OnBefore(invocation);
        try
        {
            invocation.Proceed();
        }
        catch (Exception e)
        {
            isSuccess = false;
            OnException(invocation, e);
            throw;
        }
        finally
        {
            if (isSuccess)
            {
                OnSuccess(invocation);
            }
        }
        OnAfter(invocation);
    }
}
```

```
0 references
public class AspectInterceptorSelector : IInterceptorSelector
{
    0 references
    public IInterceptor[] SelectInterceptors(Type type, MethodInfo method, IInterceptor[] interceptors)
    {
        var classAttributes = type.GetCustomAttributes<MethodInterceptionBaseAttribute>(
            true).ToList();
        var methodAttributes = type.GetMethod(method.Name)
            .GetCustomAttributes<MethodInterceptionBaseAttribute>(true);
        classAttributes.AddRange(methodAttributes);
        //////////////////classAttributes.Add(new ExceptionLogAspect(typeof(FileLogger))); // Otomatik tüm metodları loga dahil eder
        return classAttributes.OrderBy(x => x.Priority).ToArray();
    }
}
```

Bunları birazdan refactor edeceğiz

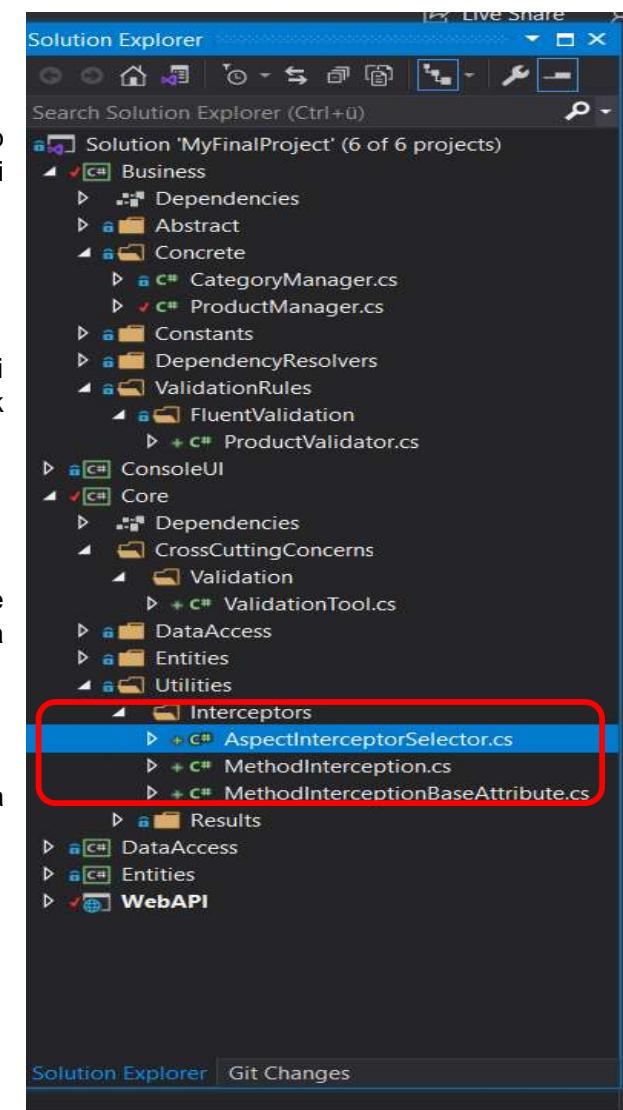
Respository içinden 3 classtan alıp eklediğimiz Class üzerinde ilk 2'sinin Metotların isminin üzerine gelip açılan tornavida işaretinden ayrı dosyalar haline getireceğiz. Kalan 3. Burada bırakıp classin ismini değiştireceğiz

1. Class içinde **MethodInterceptionBaseAttribute** adlı **class** yazısının üzerine gel. Bas taraftaki Tornavida sekline gelince açılan “**Move Type to MethodInterceptionBaseAttribute.cs**” yazan ilk sıradaki komutu tıkla. Metodun üzerindeki Attribute da buna bağlı olduğu için otomatik tasındı

2. Class içinde **MethodInterception** adlı **class** yazısının üzerine gel. Bas taraftaki Tornavida sekline gelince açılan “**Move Type to MethodInterceptionBaseAttribute.cs**” yazan ilk sıradaki komutu tıkla

3. Class içinde **AspectInterceptionSelector** adlı **class** yazısının üzerine gel. Bas taraftaki Tornavida sekline açılan “**Rename file to AspectInterceptionSelector**” yazan ilk sıradaki komutu tıkla

Not: Tüm bu kodlar 1 kere yazılır. Tüm meslek süresince kullanılır-



Step-92: Core katmanına alt alta 3 klasör oluştur (adları sırasıyla **Aspects**, altına da **Autofac**, altına da **Validation** adını ver) **Validation** içine de adını bile vermeyeceğimiz bir **Class ekle**.

Bunların hepsini **Autofac** kullanarak **Aspect** yazacağız,. yani **AOP** ile yazacağız

Engin Demirog'un GITHUP Repository'dan "NetCoreBackend" adlı Proje **Core => Aspects/Autofac => Validation** klasöründen **ValidationAspect** adlı class' dan komutları kopyala (class adı ve başlangıç bitiş süslü parantezleri hariç)

Validation içindeki isimsiz **Class** içine kodları yapıştır

...	Add project files.	10 months ago
Caching	Add project files.	10 months ago
Exception	Add project files.	10 months ago
Logging	Add project files.	10 months ago
Performance	Add project files.	10 months ago
Transaction	Add project files.	10 months ago
Validation	Add project files.	10 months ago

MethodInterception adlı interface **ampulden using.Core.Utilities.Interceptors;** komut satırını ekle/çöz

Kodlarda if komutu içinde yazan **IValidator** tipini ampulden **using FluentValidation;** komut satırını ekle/çöz

Kodun içini aşağıdaki şekilde düzelt. Aşağıdaki kod tip uygunluk kontrolü yapan metotdur

```
private Type _validatorType;
public ValidationAspect(Type validatorType)
{
    if (!typeof(IValidator).IsAssignableFrom(validatorType))
    {
        throw new System.Exception("Bu bir doğrulama sınıfı değil");
    }
    _validatorType = validatorType;
}
```

İkinci kısmı ise **MethodInterception** ile tanımlanan ve içi boş olan **onbefore** metodunu buraya özgü çalıştıracak komuttur

IInvocation ampulden **using Castle.DynamicProxy;** komut satırını ekle/cöz

Where için ampulden **using System.Linq;** komut satırını ekle/cöz

ValidationTool için ampulden **using Core.CrossCuttingConcerns.Validation;** komut satırını ekle/cöz

```
protected override void OnBefore(IInvocation invocation)
{
    var validator = (IValidator)Activator.CreateInstance(_validatorType); //reflektion: calisma aninda birseyleri calistirir
    var entityType = _validatorType.BaseType.GetGenericArguments()[0]; // base tipini ilkini bul ve parameterlerini bul
    var entities = invocation.Arguments.Where(t => t.GetType() == entityType); //invocation metod demektir
    foreach (var entity in entities) // bulunan parametreleri bu ttolu kullanarak valide et
    {
        ValidationTool.Validate(validator, entity);
    }
}
```

Step-93: **Business=>Concrete=>ProductManager** içindeki Add metoduna gel

Attributes metodlarına tip gönderdiğimizi de unutmamalı. Metodu çağrıırken nesnenin tipini göndereceğiz

Add metodu üzerine yazdığımız alttaki kodun anlamı **ProductValidator** kullanarak **Add** metodunu doğrula...

`[ValidationAspect(typeof(ProductValidator))]`

ValidationAspect için **using Core.Aspects.Autofac.Validation;** komut satırını ekle // manuel ekledim

ProductManager Add metodu son hali

```
[ValidationAspect(typeof(ProductValidator))]  
public IResult Add(Product product)  
{  
    _productDal.Add(product);  
  
    return new SuccessResult(Messages.ProductAdded);  
}
```

STEP-94: kurduk ama interseptlerin deveye gireceğini söylemeliyiz

Engin Demirog'un GITHUP Repository'dan "NetCoreBackend" adlı Proje **Business =>DependencyResolvers/Autofac** adlı klasörden **AutofacBusinessModule** adlı Classı ac

Bu kodlar kaydettiğimiz sınıflar var. **Autoc** kullanma sebebimiz intercept de veriyor olmasıdır.

Çalışan uygulama içinde implemente edilmiş interface 'leri bul, onlar için **AspectInterceptorSelector()** çağrırmaya imkan veriyor

Yani; bütün sınıflar için önce **AspectInterceptorSelector** calistiriyor. Bu kısmı kopyalacagız (var ile baslayan)

Business => DependencyResolvers => Autofac =>AutofacBusinessModule adlı Classe ilk metodun içine kodları yapıştıracağız

Kod içindeki **EnableInterfaceInterceptors** için ampulden **using Autofac.Extras.DynamicProxy**; komut satırını ekle/cöz

ProxyGenerationOptions için ampulden **using Castle.DynamicProxy**; komut satırını ekle/cöz

AspectInterceptorSelector için ampulden **using Core.Utilities.Interceptors**; komut satırını ekle/cöz

Kosturma(Run): ürün kriterlere uymadığı için işlem yapamadı. Validation hatası uyarısı verdi. Program başarılı bir şekilde calisti

13.DERS (20.02.2021)

Konu : AOP Gözden Geçirme

Iskodlari Yazma WorkShop (ProductManager)

AOP neydi? Is kurallarını yazdığımız Business katmanında Manager class değişkeninde standart işlemleri yazıp şışirmek yerine ve her mana ger için tekrar tekrar yazmak yerine Attribute kullanarak ve ortak klasörler tanımlayarak projemizi yazmak için kullanıyoruz.

Bu is kodları ara yüze yazılmamalıdır. Bu durunda her ara yüz için tekrar tekrar yazmak zorunda kalırız

GÖZDEN GEÇİRME

Business üzerine Demo yapalım. Sonra sileceğiz

GG-1 : Business içinde **CCS** adlı klasör oluşturalım

CCS içine **ILogger** adlı Interface oluşturalım public yapalım

CCS içine **FileLogger** adlı Class oluşturalım public yapalım, ILogger ‘a inherit edelim

CCS içine **DatabaseLogger** adlı Class oluşturalım public yapalım, ILogger ‘a inherit edelim

(Biz File Loger içine aynı Classı kopyaladık. Daha sonra çıkan Tornavida Generate dierek classı olmasını sağladık)

GG-2: Business=>Concrete klasörü içinde **ProductManager** içinde Add için yazılmış Attribute satırını iptal edelim

Add metodu içinde Try 2*tab yapalım çıkan metod içine Add içindeki işlem kodlarını alalım

```
ILogger _logger; tanımlamasını burada yapalım

public class ProductManager : IProductService
{
    IProductDal _productDal;
    ILogger _logger; //EKLENDİ GG

    public ProductManager(IProductDal productDal, ILogger logger) //EKLENDİ GG
    {
        _productDal = productDal;
        _logger = logger; //EKLENDİ GG
    }
    .....

    //#[ValidationAspect(typeof(ProductValidator))]
    public IResult Add(Product product)
    {

        _logger.Log();
        try
        {

            //business Codes

            _productDal.Add(product); // kodları buraya aldık

            return new SuccessResult(Messages.ProductAdded); // kodları buraya aldık
        }
        catch (Exception exception)
        {

            _logger.Log();
        }

        return new ErrorResult();
    }
}
```

GG-3: AutoFAc içinde gerekiğinde newlenmesi için ILogger satirini ekleyelim

```
protected override void Load(ContainerBuilder builder)
{
    builder.RegisterType<ProductManager>().As<IProductService>().SingleInstance();
    builder.RegisterType<EfProductDal>().As<IProductDal>().SingleInstance();
    builder.RegisterType<FileLogger>().As<ILogger>().SingleInstance(); //EKLENDI GG
```

Koşturma (RUN): Çalıştırırsak ürün eklendiğini görebiliriz. Yalnız validation Attributes satirini sildik. O yüzden validation yapmadan ekledin

AOP ile süreci nasıl ele alırız. Şimdi onu görelim GG-4'e kadar sadece gözden geçirelim

Core katmanında **Utilities** klasöründe **Interceptors** **methodinterception** **klassina** **gelelim**

Virtual Classina gelelim. Ezilmeyi bekleyen metodlar

Aspect demek **MetotInterceotion** i temel alana çalışmasını istediğiniz yeri belirleyeceksin (başında, sonunda, hata verdiğinde çalışmasını istediğiniz kodlar)

CrossCuttingConcern Klasörü Validation Klasörü içinde **ValidationTools** **Classina** Gelelim

Aspect içinde ValidationAspect Class MethodInterception interface inheritdir

Ezmeni istediğiniz metodu ez. (Validation olduğu için onbefore metodunu ezeceğiz)

Burada tipi kontrol ediyoruz. Tip doğruysa işlem yapıyoruz.(burada onbefore)

Burada lazımlı olunca çalışmasını için newleme yapacak kodu yazıyoruz (on before içinde reflection)

Generic argumanlarından 0 olanı yakala diyoruz (0(sıfır) ilk elemanı ifade eder. Zaten 1 adet eleman var)

Tipi uygunsa valide kurallarına kontrol et diyoruz (validation tool)

*****ProductManageri eski haline getirelim*****

İs kodları yazacağımız Workshop yapacağız

WorkShop-1 : Yönetim bir ürün eklemek istersen eklemek istediği ürünün kategorisinde maksimum 10 adet ürün olabilir şeklinde bir isteği olursa ilişkin productManager içinde bir is kuralı yazalım

Yanlış Cevap: bir değişkenle girilmek istenen ürünün Category bilgisi alınacak. Tüm listede aynı Kategoride olan ürün sayısını kontrol edeceğiz. Ürün sayısı 9 veya daha az ise **Add** işlemini yapacağız. Aksi takdirde işlemi yapmayıp ikaz edeceğiz. Mesaj Class 'na ilgili ikazı ekleyeceğiz

Bu yanlış bir işledmdir. Bu kural ever bir is kuralıdır ancak UPDATE metodu içinde geçerlidir Bu Merkezi bir yerde tanımlanırsa ve sadece orada değişme yaparsak programa çalışır. Ancak Add ve Update için ayrı ayrı yazarsak sıkıntı olur (DO NOT REPAT YOURSELF)

```
[ValidationAspect(typeof(ProductValidator))]
public IResult Add(Product product)
{
    //business Codes

    var result = _productDal.GetAll(p => p.CategoryId == product.CategoryId).Count;
    if (result >= 10)
    {
        return new ErrorResult(Messages.ProductCountCategoryError);
    }

    _productDal.Add(product);

    return new SuccessResult(Messages.ProductAdded);
}

public static class Messages
{
    ....
    public static string ProductCountCategoryError = "bir kategoride en fazla 10 ürün olabilir";
}
```

DOGRU CEVAP:

STEP-95: (WorkShop-1)

Şimdiye Kadar hic aktive etmemiştik. IProductService içinde Update Metodunu yazalım

ProductManager içinde Update Metodunun Update Benzeri tanımlayalım

Biraz önce yazdığımız is kodlarını seçip ampulden “Quick Actions and Refactoring” secmeliyiz. Olması gerek. Olmadıysa manuel yapacağız

ProductManager içinde bu is kodunu **Private** bir Metod içinde tanımlayacağız (public olması gerekseydi IproductService Inteface da tanımlayacağımız bir metot olurdu)

Sonra Add ve Update metodu içinde çağırabiliriz

```
private IResult CheckIfProductCountOfCategoryCorrect (int categoryId)
{
    var result = _productDal.GetAll(p => p.CategoryId == categoryId).Count;
    if (result >= 10) // Bu kural değiştiğinde sadece buraya işlem yaparız
    {
        return new ErrorResult(Messages.ProductCountCategoryError);
    }
    return new SuccessResult();
}
```

```
[ValidationAspect(typeof(ProductValidator))]
public IResult Add(Product product)
{
    //business Codes
    if (CheckIfProductCountOfCategoryCorrect(product.CategoryId).Success)    // metodu kullanık
    {

        _productDal.Add(product);

        return new SuccessResult(Messages.ProductAdded);
    }
}

public IResult Update(Product product)
{
    if (CheckIfProductCountOfCategoryCorrect(product.CategoryId).Success)
    {

        _productDal.Update(product);

        return new SuccessResult(Messages.ProductAdded);
    }
    return new ErrorResult();
}
```

WorkShop-2 : Ayni isimde ürün eklenemez

Cevap: başka bir metod yazacağız. Veri tabanında Ayni ada sahip ürün olup olmadığını kontrol ederiz. Yoksa metod başarılı döner Başarılı dönerse **Add** ve **Update** metodları başarıyla çalışır

STEP-96: WorkShop-2 : Ayni isimde ürün eklenemez

```
private IResult CheckIfProductNameExist(string productName)
{
    var result = _productDal.GetAll(p => p.ProductName == productName).Any(); // varmi yokmu nun kontrolu icin Any yeterli
    if (result) // Bu kural degistiginde sadece buraya islem yapariz
    {

        return new ErrorResult(Messages.ProductNameAlreadyExist); // MESAJ EKLEMELIYIZ
    }
    return new SuccessResult();
}

[ValidationAspect(typeof(ProductValidator))]
public IResult Add(Product product)
{
    //business Codes
    if (CheckIfProductCountOfCategoryCorrect(product.CategoryId).Success)
    {
        if (CheckIfProductNameExist(product.ProductName).Success) // IF KOSULARINI ICI ICE YAZMAK DAHA IYIDIR
        {
            _productDal.Add(product);

            return new SuccessResult(Messages.ProductAdded);
        }
    }
    return new ErrorResult();
}
```

```

public IResult Update(Product product)
{
    if (CheckIfProductCountOfCategoryCorrect(product.CategoryId).Success)
    {
        if (CheckIfProductNameExist(product.ProductName).Success)          // IF KOSULARINI ICI ICE YAZMAK DAHA İYİDIR
        {
            _productDal.Update(product);

            return new SuccessResult(Messages.ProductAdded);
        }
    }
    return new ErrorResult();
}

public static class Messages
{
    ....
    public static string ProductNameAlreadyExis= "bir kategoride en fazla 10 ürün olabilir"; // MESAJ EKLEDİK
}

// if satırlarını ayrı ayrı yazmak daha avantajlı ancak aşağıdaki gibi de yazabilirdik
if ((CheckIfProductCountOfCategoryCorrect(product.CategoryId).Success) &&(CheckIfProductNameExist(product.ProductName).Success))

```

NEYİ NICİN YAPIYORUZ

İs kuralları artarsa dosya karışabilir. O zaman başka dosyalara taşıyabiliriz

Tüm is Metodları Iresult döndürüyor. Bunu bir motora göndersek ve bizim için hepsini çalıştırılan bir class oluşturacağız

Polymorphism Kodu yazacağız

STEP-97: **Core-Utilities**- icine **Business** adlı yeni Klasör ekle

Icine **BusinessRules** adlı **Class** ekle. **Public** yap

(bu Class Utilities olduğu için çiplak kalabilir. Normale bir Interface oluşturup bunu ona inherit edebilirdik ancak **overdesign** olurdu)

Params istediği kadar parametre gönderebilmemize imkan tanır. C# bu metoda gönderdiğimiz her şeyi **IResult** döndürecek bir dizi olarak atar.

İstesek Liste şeklinde **IResult** da döndürebiliriz

```
public class BusinessRules
{
    public static IResult Run(params IResult[] logics) // Params
    {
        foreach (var logic in logics)
        {
            if (!logic.Success)
            {
                return logic; // hatalı olanı geri döndürüyoruz
            }
        }
        return null;
    }
}
```

ProductManager içinde is kodlarını oluşturduğumuz Classa göndereceğiz ve kontrol ettireceğiz

Yeni Add ve Update Metotları

```
[ValidationAspect(typeof(ProductValidator))]
public IResult Add(Product product)
{
    IResult result = BusinessRules.Run(CheckIfProductNameExist(product.ProductName),
                                         CheckIfProductCountOfCategoryCorrect(product.CategoryId));
    if (result != null)
    {
        return result;
    }

    _productDal.Add(product);
    return new SuccessResult(Messages.ProductAdded);
}

public IResult Update(Product product)
{
    IResult result = BusinessRules.Run(CheckIfProductNameExist(product.ProductName),
                                         CheckIfProductCountOfCategoryCorrect(product.CategoryId));
    if (result != null)
    {
        return result;
    }
    _productDal.Update(product);
    return new SuccessResult(Messages.ProductUpdated);
}
```

WorkShop-3: Eger mevcut kategori sayısı 15'ii geçtiyse sisteme yeni ürün eklenemez

(uydurma bir kural ancak başka bir tablodan işlem yapman gereklili bir örnek olması için)

Microservis mimarilere bakış açısını geliştirecek bir çalışma

Category tablosundan sayıyı alırız. 16 kategori varsa ürün eklemeyi durdur

Cevap: ICategoryDal içinde bu bilgi var. Injection sözkonusu

*****Bir entity Manager kendisi hariç başka entity için oluşturulmuşdal şeklinde classı enjekte edemez, ancak başka bir Service interface enjekte edebiliriz *****

YANLIS CÖZÜM

O yüzden aşağıdaki çözüm yanlışdır (İlk metotda product nesnesi newliyor idik. Simdi Category de newleyecegiz)

```
public class ProductManager : IProductService
{
    IProductDal _productDal;
    ICategoryDal _categoryDal;

    public ProductManager(IProductDal productDal, ICategoryDal categoryDal)
    {
        _productDal = productDal;
        _categoryDal = categoryDal;
    }
}
```

DOGRU CÖZÜM: .:Dal enjekte edemeyiz ancak başka bir Service interface enjekte edebiliriz

STEP-98:

```
public class ProductManager : IProductService
{
    IProductDal _productDal;
    ICategoryService _categoryService;           //DIKKAT

    public ProductManager(IProductDal productDal, ICategoryService categoryService)
    {
        _productDal = productDal;
        _categoryService = categoryService;

    }
    ....
```

CategoryService ve **CategoryMananger**'ı zaten düzeltmemiz gerekiyordu. Onları düzeltelim. Sonuçlar **IDataResult** olarak dönecek. **IDataResult** ise tüm managerler için ortak olan Utilities içindeki Results içinden kullanılacak

STEP-99:

```
namespace Business.Abstract
{
    public interface ICategoryService
    {

        IDataResult<List<Category>> GetAll();    //utilities içinde genellemiştiğinden Using.Core.Utilities.Results komut satırı eklenmeli
        IDataResult<Category> GetById(int categoryId);
    }
}
```

STEP-100: **ProductManager** içine bir metod yazıp **Add** ve **Update** metotlarını güncelleyelim. Gerekli mesajı **Messages** Classına ekleyelim

```
private IResult CheckIfCategoryLimitExceeded()
{
    var result = _categoryService.GetAll();
    if (result.Data.Count > 15)
    {
        return new ErrorResult(Messages.CategoryLimitExceeded); // BU MESAJİ MESSAGES CLASSINA EKLEYELİM
    }
    return new SuccessResult();
}

public static class Messages
{
    ...
    public static string CategoryLimitExceeded = "kategor limiti astı";
}

[ValidationAspect(typeof(ProductValidator))]
public IResult Add(Product product)
{
    IResult result = BusinessRules.Run(CheckIfProductNameExist(product.ProductName),
                                        CheckIfProductCountOfCategoryCorrect(product.CategoryId),
                                        CheckIfCategoryLimitExceeded()); //YENİ YAZDIĞIMIZ METODU EKLEDİK

    if (result != null)
    {
        return result;
    }
    _productDal.Add(product);

    return new SuccessResult(Messages.ProductAdded);
}
```

```

public IResult Update(Product product)
{
    IResult result = BusinessRules.Run(CheckIfProductNameExist(product.ProductName),
        CheckIfProductCountOfCategoryCorrect(product.CategoryId),
        CheckIfCategoryLimitExceeded());                                         //YENİ YAZDIĞIMIZ METODU EKLEDİK
    if (result != null)
    {
        return result;
    }
    _productDal.Update(product);
    return new SuccessResult(Messages.ProductUpdated);
}

```

STEP-101: AutoFacBusinessModule da CategoryManager Clasinin Newlemeliyiz

```

public class AutofacBusinessModule : Module
{
    protected override void Load(ContainerBuilder builder)
    {
        builder.RegisterType<ProductManager>().As<IPrintService>().SingleInstance();
        builder.RegisterType<EfProductDal>().As<IPrintDal>().SingleInstance();

        builder.RegisterType<CategoryManager>().As<ICategoryService>().SingleInstance();      // eklenmeli
        builder.RegisterType<EfCategoryDal>().As<ICategoryDal>().SingleInstance();           // eklenmeli
    }
}

```

//şimdilik Console program Classı bu değişikliklerden dolayı hata verdi. Commentleyelim isimiz yok

Koşturma (Run): programı çalıştırıldım.

Başarılı işlem: İlk olarak Ürünleri Get ile listeledim oldu

Hatalı İşlem: Post ile Category 1,2 ve 3 'e ürün eklemek istedim. "Bir kategoride en fazla 10 ürün olabilir" hata mesajı verdi.

Başarılı İşlem: Kategori 5 'e "productName": "A harfi 24.02 son kısmı için" ürün ekledim

Hatalı İşlem: Post ile Kategori 4 için aynı isimli ürünler eklemek istedim. "Aynı isimle birden fazla ürün olamaz" hata mesajı verdi.

Başarılı İşlem: Kategori 5' e "productName": "A harfi 24.02 son kısım için 2. işlem" ürün ekledim

The screenshot shows a Postman request to `https://localhost:44356/api/products/add`. The 'Body' tab is selected, containing JSON data:

```
1 {"productName": "A harfi 24.02 son kisim icin",  
2 "unitsInStock": 40,  
3 }  
4
```

The 'Test Results' section shows a 400 Bad Request response with the following body:

```
1 {"success": false,  
2 "message": "Ayni isimle birden fazla Ürün olamaz"  
3 }  
4
```

A red circle highlights the error message in the test results.

POST https://localhost:44356/api/products/add

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON

1 "categoryId": 1,
2 "productName": "A harfi 24.02 son kisim icin 2.deneme",
3 "unitsInStock": 40,
4

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

1 "success": false,
2 "message": "bil ketoride en fazla 10 GiOn olabilir"
3

The screenshot shows a POST request to `https://localhost:44356/api/products/add`. The 'Body' tab is selected, showing JSON input:

```
1 {  
2   "categoryId": 5,  
3   "productName": "A harfi 24.02 son kisim icin 2.deneme",  
4   "unitsInStock": 48,  
5 }
```

The response tab shows a successful response with status 200 OK, duration 40 ms, and size 225 B. The response body is:

```
1 {  
2   "success": true,  
3   "message": "Ürün eklendi"  
4 }
```

Konu: Güvenlik

NEYİ NİCİN YAPIYORUZ?

Doğru kişileri yetkilendirme önemlidir.

Business katmanında **Aspectlerle** yöneteceğimiz **Jason Web Token (JWT)** oluşturacağız

Yetkilendirmeyi **API** kısmına taşımak hatalıdır. Yeni bir **API** geldiğinde buraya bastan tanıtmak zorunda kalırız

Business katmanında **Aspect** yaptı. Yetkilendirmede **CCC (Cross Cutting Console)** olarak her katmanda kullanılabilir

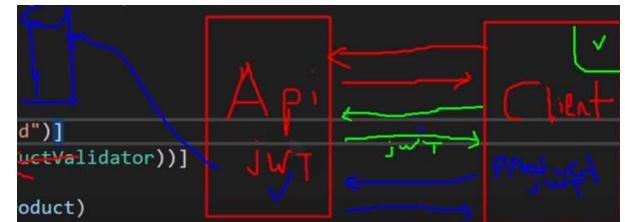
Örneğin **ProductManager Add** metodu için **Admin** yetkisine sahip birisi olması gereklidir. O yüzden **Add** metodu üzerine Aspekt çağrılabılır

Aspektte belirteceğimiz Claim 'leri (iddiaları) sağlayan kişi **add** metodunu kullanabilir. Örnegin `[SecuredOperation("product.add,admin")]`

Client: Her tarayıcı Cilenttir ve API'yi kullanmak isteyebilir. JWT ile tanımladığımız kriteri sağlayan kişi bunu kullanabilir (Örneğin kullanma yetkisi)

Bu kriter daha önceden kaydolmak olabilir. Sistemde Kayıtlı olan kayıt bilgisi ile sisteme erişmek isteyen kişiye müsaade edilir.

Yada erişmek isteyen kişi **JWT** ile birlikte gelir ve erişmek ister



Encryption, Hashing açıktan okunmasını engellemek için veri tabanında gizli tutmak için kullanılır

Hashing için MD5 ve SHA1 şifreleme algoritmalarıdır. Geri döndürülemez şekilde değiştirir ve şifre o şekilde saklanır. Kullanıcı her sefer kendi şifresini girer ve aynı algoritma çevirir ve şifreyi kayıtlı sekilde kontrol eder

Salting: Girilen şifreyi ekleme yaparak güçlendirme yoluyla **hash 'leme** yapmaktadır

Encryption: Geri dönüşü olan veridir. Algoritma bilinmelidir ki geri döndürülebilir olsun (evin anahtarı olduğu sürece girip çıkabilirsiniz)

STEP-102:

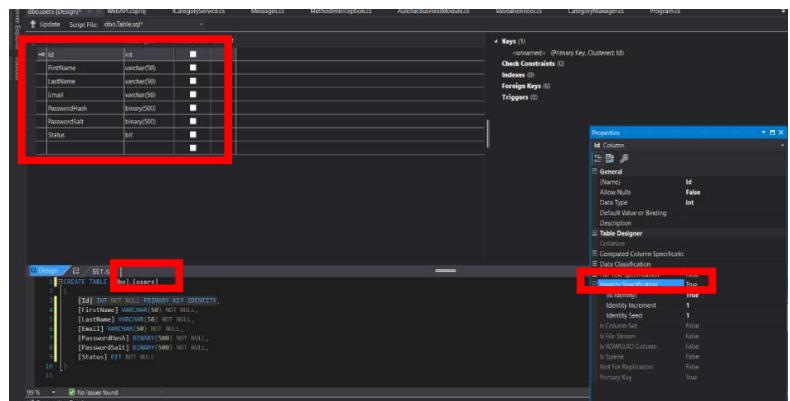
Northwind veri tabanını açalım

Tables üzerinde sağ tıklayıp “Add new Table” ile Users adlı yeni bir Tablo oluşturacağız

ID üzerinde sağ tıklayıp açılan pencerede “Id Specification” altında yer alan “Is Identity” özelliğini True yapmalıyız (Bu özellik Id'nin sırayla birer artmasını sağlar)

Users adını sol altta iki köşeli parantez arasına yazarız

Sol üstteki Update tuşıyla oluştururuz

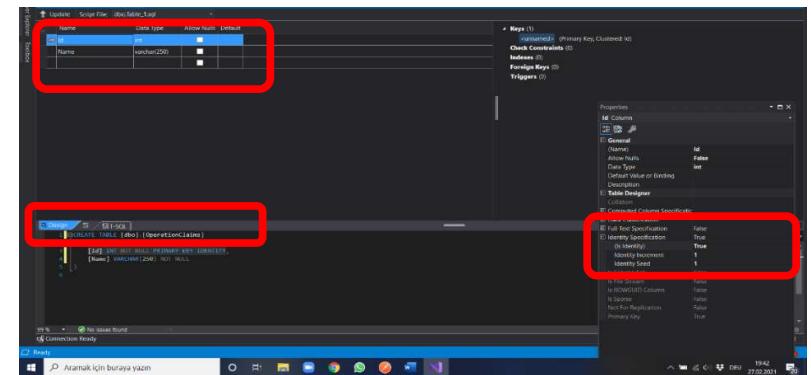


Tables üzerinde sağ tıklayıp “Add new Table” yeni bir OperationClaims adlı Tablo oluşturacağız

ID üzerinde sağ tıklayıp açılan pencerede “Id Specification” altında yer alan “Is Identity” özelliğini True yapmalıyız (Bu özellik Id'nin sırayla birer artmasını sağlar)

OperationClaims adını sol altta iki köşeli parantez arasına yazarız

Sol üstteki Update tuşıyla oluştururuz

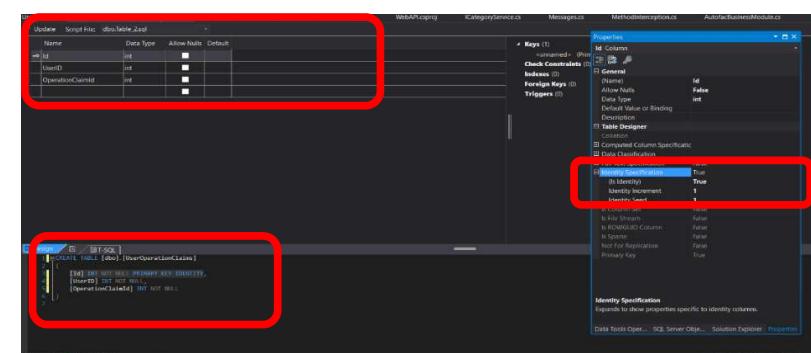


Tables üzerinde sağ tıklayıp “Add new Table” yeni bir UserOperationClaims adlı Tablo oluşturacağız

ID üzerinde sağ tıklayıp açılan pencerede “Id Specification” altında yer alan “Is Identity” özelliğini True yapmalıyız (Bu özellik Id'nin sırayla birer artmasını sağlar)

UserOperationClaims adını sol altta iki köşeli parantez arasına yazarız

Sol üstteki Update tuşıyla oluştururuz



STEP-103:

Bu tablolarının Entitylerini oluşturacağız. Ancak bu varlıklar bütün Projelerde kullanılabileceğim için Core Katmanına tanımlayalım

Core Entities içine **Concrete** adlı klasör oluştur Tabloların karşılığı olan 3 adet Class tanımlayalım . **Public** yapalım ve **IEntity** ile inherit edelim

1. User

```
public class User:IEntity
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Email { get; set; }
    public byte[] PasswordHash { get; set; }
    public byte[] PasswordSalt { get; set; }
    public bool Status { get; set; }
}
```

2. OperationClaim

```
public class OperationClaim : IEntity
{
    public int Id { get; set; }

    public string Name { get; set; }
}
```

3. UserOperationClaim

```
public class UserOperationClaim : IEntity
{
    public int Id { get; set; }
    public int UserId { get; set; }
    public int OperationClaimId { get; set; }
}
```

STEP-104: WebAPI katmanında AppSettings.json adlı Jason dosyasını açalım

(zaten var)

İçine **Token Options** adlı bir tanımlama yapacağız

Bu değerler şifreleme için önemli

AccessTokenExpiration şifrenin geçerlilik süresi

SecurityKey ise şifrelerken kullanacağı anahtar

```
{
    "TokenOptions": {
        "Audience": "engin@engin.com", // web sayfası
        "Issuer": "engin@engin.com", // kullanıcılar
        "AccessTokenExpiration": 10, // 10 dk, gecerli
        "SecurityKey": "mysupersecretkeymysupersecretkey"
    },
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft": "Warning",
            "Microsoft.Hosting.Lifetime": "Information"
        }
    },
    "AllowedHosts": "*"
}
```

STEP-105: Core Utilities altına Security adlı klasör oluştur. Altına da Hashing, Encryption ve JWT adlı 3 tane klasör oluştur

Hashing altına HashingHelper adlı Class oluştur (Çiplak kalabilir)

Bu class ta ilk kısım kullanıcı tarafından girilen şifreye göre HMAC 512 ile hash ve salt password üretecek Bu classin 2. kısmında kayıt sonrası girilen şifrenin doğrulanmasını yapacak (Aslında kullanıcının gireceği şifre hash 'lenip saltlanacak, esleniyorsa doğru girilmiş demektir)

```
namespace Core.Utilities.Security.Hashing
{
    public class HashingHelper //ciplak kalabilir
    {
        public static void CreatePasswordHash(string password, out byte[] passwordHash, out byte[] passwordSalt)
        {
            using (var hmac = new System.Security.Cryptography.HMACSHA512()) // kullanılacak algoritma
            {
                passwordSalt = hmac.Key;
                passwordHash = hmac.ComputeHash(Encoding.UTF8.GetBytes(password)); // string gelen passwordun byte değerini buraya vermeliyiz
            }
        }

        public static bool VerifyPasswordHash(string password, byte[] passwordHash, byte[] passwordSalt) // out iptal
        {
            using (var hmac = new System.Security.Cryptography.HMACSHA512(passwordSalt)) // burada Önceki salt'i göndermeliyiz
            {
                var computedHash = hmac.ComputeHash(Encoding.UTF8.GetBytes(password)); // dizilerin değerlerini karşılaştıracağız

                for (int i = 0; i < computedHash.Length; i++)
                {
                    if (computedHash[i] != passwordHash[i])
                    {
                        return false;
                    }
                }
            }
            return true;
        }
    }
}
```

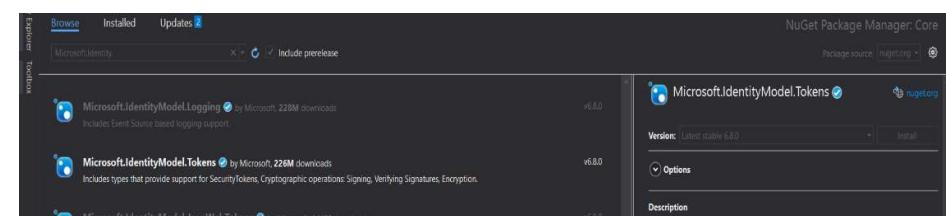
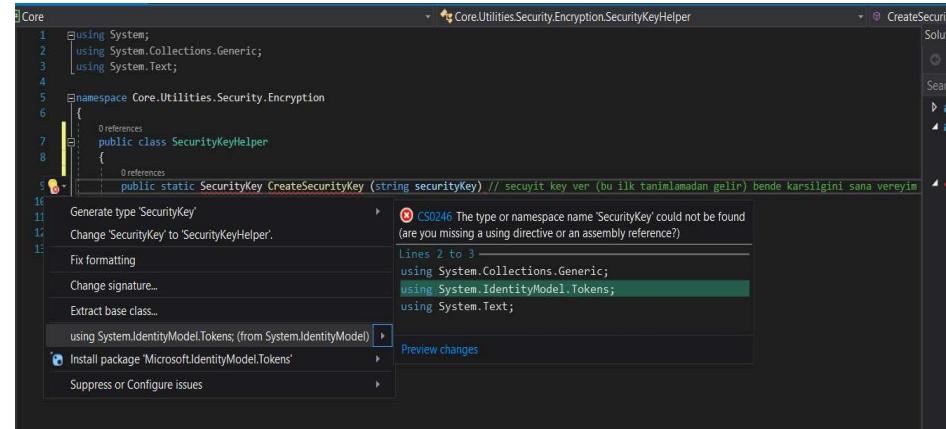
Kurulum: Ampulden ilgili yardımcı ekleniyi ekleyebilir. Olmazsa klasik yöntemi kullanırız. Core katmanı sağ tıkla **Manage NuGet Packages** kısmından **Microsoft.IdentityModel.Tokens** adlı ekleni

STEP-106: **Security =>Encryption** altına **SecurityKeyHelper** adlı bir Class oluştur. **Public yap**

SecurityKey için ampulden **using Microsoft.IdentityModel.Tokens;** satırını çöz

```
namespace Core.Utilities.Security.Encryption
{
    public class SecurityKeyHelper
    {
        public static SecurityKey CreateSecurityKey(string securityKey)
// secuyit key ver (bu ilk tanimlamadan gelir) bende karsilagini sana vereyim der

        {
            return new SymmetricSecurityKey(Encoding.UTF8.GetBytes(securityKey));
        }
    }
}
```



STEP-107: **Security =>Encryption** altına **SignedCredentialsHelper** adlı bir Class oluştur. **Public yap**
Yöneteceğin Json için kullanacağın anahtar ve şifreleme metodu tanımlanır
SignedCredentials için **using Microsoft.IdentityModel.Tokens** komut satırını çöz

```
namespace Core.Utilities.Security.Encryption
{
    public class SignedCredentialsHelper
    {

        public static SignedCredentials CreateSignedCredentials(SecurityKey securityKey)
        {
            return new SignedCredentials(securityKey, SecurityAlgorithms.HmacSha512Signature ); //anahtar olarak bu key'i kullan, şifreleme
olarak ta güvenlik algortmalarından hmac 512 kullan
        }
    }
}
```

STEP-108: **Security =>JWT** altına **AccessToken** adlı bir **Class** oluştur **Public yap**

```
namespace Core.Utilities.Security.JWT
{
    public class AccessToken // anlamsız gözüken anahtar değeri ve bitiş süresi tanımlanır
    {
        public string Token { get; set; }

        public DateTime Expiration { get; set; }

    }
}
```

STEP-10: **Security =>JWT** altına **ITokenHelper** adlı bir **Interface** oluştur. Public yap

User ampulden using komut satırı ekleyelim

Girilen şifre doğruya burası çalışacak. Veri tabanında Claimlerinin bulacak , **JasonWebToken** üretecek ve buraya verecek

```
namespace Core.Utilities.Security.JWT
{
    public interface ITokenHelper
    {
        AccessToken CreateToken(User user, List<OperationClaim> operationClaims);
    }
}
```

STEP-11: Engin Demirog Github 'a gidelim NetCoreBackHand =>Core=> Utilities=> Security =>JWT **JWTHelper** adlı classın hepsini kopyalacağız (sadece class tamimi içindeki kodları)

JWT klasörü içinde **JwtHelper** adlı Class oluştur ve kopyaladığın kodları buraya yapıştır

User ve **OperationClaim** adlı nesneler için ampulden using komut satırlarını çözelim

STEP-11: Kopyalanan kodların anlatımı ve yapılması gereken işlem

i) Kopyaladığımız ilk komut satırı **public IConfiguration Configuration { get; }** için ampulden 2. sıradaki **using Microsoft.Extensions.Configuration;** komut satırını çöz

- ii) 2.satır(`private TokenOptions _tokenOptions;`) için aynı yerden (Engin GITHUB) Token Options adlı Classı kopyala
JWT altına **TokenOptions** adlı Class oluştur ve buraya kodları yapıştır

```
namespace Core.Utilities.Security.JWT
{
    public class TokenOptions
    {
        public string Audience { get; set; }
        public string Issuer { get; set; }
        public int AccessTokenExpiration { get; set; }
        public string SecurityKey { get; set; }
    }
}
```

- iii) CreateToken Metodu=> User ve Claim bilgisine göre Token oluşturur

```
*****
public AccessToken CreateToken(User user, List<OperationClaim> operationClaims)
{
    _accessTokenExpiration = DateTime.Now.AddMinutes(_tokenOptions.AccessTokenExpiration);
    // verilen gecerlilik süresi. Suandan itibaren bizim tanımladığımız süre (WebAPI konfigürasyonunda tanımladık)

    var securityKey = SecurityKeyHelper.CreateSecurityKey(_tokenOptions.SecurityKey);
    // WebAPI konfigürasyonunda tanımladığımız securitykey istenilen formatta sisteme verecek kişiim

    var signingCredentials = SigningCredentialsHelper.CreateSigningCredentials(securityKey);
    // kullanılacak algoritmayı belirtiyoruz

    var jwt = CreateJwtSecurityToken(_tokenOptions, user, signingCredentials, operationClaims);
    // JWT üretimi için gerekenler tanımladığımız CreateJwtSecurityToken metodu çalışacak

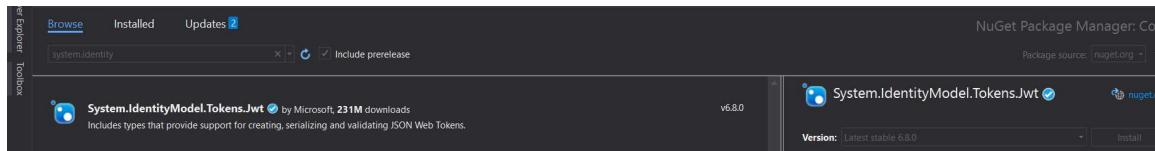
    var jwtSecurityTokenHandler = new JwtSecurityTokenHandler();
    var token = jwtSecurityTokenHandler.WriteToken(jwt);
    return new AccessToken
    {
        Token = token,
        Expiration = _accessTokenExpiration
    };
}
```

iv) Tanımladığımız CreateJwtSecurityToken metodu

```
public JwtSecurityToken CreateJwtSecurityToken(TokenOptions tokenOptions, User user,
    SigningCredentials signingCredentials, List<OperationClaim> operationClaims)
{
    var jwt = new JwtSecurityToken(
        issuer: tokenOptions.Issuer,
        audience: tokenOptions.Audience,
        expires: _accessTokenExpiration,
        notBefore: DateTime.Now,
        claims: SetClaims(user, operationClaims),
        signingCredentials: signingCredentials
    );
    return jwt;
}
```

// ekle
//ampulden using komut satirini çöz

Eklenti: Core katmanı için Manage NuGet Packages kısmından System.IdentityModel.Tokens.jwt ekantisini ekle



v) Claim için metod: Yetki ve kullanıcı bilgileri (Gerekenler için Using kod satirini çöz)

```
private IEnumerable<Claim> SetClaims(User user, List<OperationClaim> operationClaims)
{
    var claims = new List<Claim>();
    claims.AddNameIdentifier(user.Id.ToString());
    claims.AddEmail(user.Email);
    claims.AddName($"{user.FirstName} {user.LastName}");
    claims.AddRoles(operationClaims.Select(c => c.Name).ToArray()); // array üzerinde rol ekleme
    return claims;
}
```

Github da Core=>Extensions içinde **Claimextensions** adlı class 'i aç. Kopyala

Core altına **Extensions** adlı klasör oluştur altına **ClaimExtensions** adlı Class oluştur ve kodları yapıştır. Aşağıdaki using komut satırlarını çöz

Claim için **using System.Security.Claims;**

JwtRegisteredClaimNames için **using System.IdentityModel.Tokens.Jwt;**

Tolist için **using System.Linq;**

Tanım (Extension) varolan hazır class/metotlara değiştirme/özelleştirmeye **Extension** adı verilir

```
namespace Core.Extensions
{
    public static class ClaimExtensions
    {
        public static void AddEmail(this ICollection<Claim> claims, string email)
        {
            claims.Add(new Claim(JwtRegisteredClaimNames.Email, email));
        }

        public static void AddName(this ICollection<Claim> claims, string name)
        {
            claims.Add(new Claim(ClaimTypes.Name, name));
        }

        public static void AddNameIdentifier(this ICollection<Claim> claims, string nameIdentifier)
        {
            claims.Add(new Claim(ClaimTypes.NameIdentifier, nameIdentifier));
        }

        public static void AddRoles(this ICollection<Claim> claims, string[] roles) // BANA GÖNDERİLEN ROLLERİ LİSTEYE CEVİR ROLÜ CLAİME EKLE
        {
            roles.ToList().ForEach(role => claims.Add(new Claim(ClaimTypes.Role, role)));
        }
    }
}
```

Github da **Core=>Extensions** klasörü isinde **ClaimsPrincipalExtensions** adlı Class 'i açalım

JWT gelen Claimlerin bilgilerine okumayı kolaylaştıran bir **Extensions** yazılmış

Core altına **Extensions** adlı klasör oluştur altına **ClaimsPrincipalExtensions** adlı Class oluştur ve kodları yapıştır.

ClaimsPrincipal için `using System.Security.Claims;` Select için `using System.Linq;` komut satırlarını çöz

Github içinde **NetCoreBackend** adlı projeden **Business=>BusinessAspects=>Autofac** içindeki **SecuredOperation** adlı Classin içindeki kodları kopyala

Bizim **Business =>BusinessAspect** adlı klasör altına da **Autofac** adlı klasör ekle altına **SecuredOperation** adlı Class olusur ve kopyaldagımız class yapıştır.

Kurulum: Business sağ tıkla “Manage Nuget Packages” kısmından **Microsoft.AspNetCore** eklentisini yükle (**IHttpContextAccessor** için)

SeviceTool adlı metot olmadığı için uyarı veriyor (hazır bir metot da degil). ServiceTool bizim injection altyapısını okuyabileceğimiz bir araç olacak. WEbAPI de oluşturduğumuz injectionları tetikleyecek bir kod lazım

STEP-112: GITHUB içinden **CORE=>Utilities=IOC=>Servicetool** Class ina gel

CORE=>Utilities klasörü içinde **IOC** klasörü oluştur. Altına da **IOC** Klasörü oluştur. Onun altına **ServiceTool** adlı bir Class oluştur ve kodlar buraya yapıştır

`IServiceCollection` için `using Microsoft.Extensions.DependencyInjection;` komut satirini çöz

Bir önceki Stepte (**Business.BusinessAspects.Autofac Secured Operations CClass**) Service Tool uyarı veriyordu. Artık Ampulden using komut satirini çözebiliriz

Eklenti: Solution sağ tıkla “Manage Nuget Packages” kısmına gel **Autofac.Extensions.DependencyInjection** adlı eklenti Business içinde kuralım

Ampul çözülmese de elle komut satırlarını ekleyelim

```
using Castle.DynamicProxy;
using Microsoft.Extensions.DependencyInjection;

throw new Exception(Messages.AuthorizationDenied); // bu satır için Yetkiniz olmadığına dahil mesajı Class 'ina ekleyelim

public static string AuthorizationDenied = "Yetkiniz Yok";
```

Github DataAccess Abstract içinde IUserDal adlı Interface 'i al ve aynı yere Interface oluştur ve kodları yapıştır

Ampulde Using Kodları çöz

Github **DataAccess Concrete.EntityFramework** içinde EfUserDal adlı Classı al ve aynı yere **EfUserDal** Class oluştur ve kodları yapıştır.

Ampulden Using Kodları çöz

Github **DataAccess=>Concrete=>EntityFramework** içinde **NorthwindContext** adlı Classı içinde User, OperationClaims ve UserOperationsClaims için olan kod satırlarını al ve aynı yerdeki **NorthwindContext** Class içine kodları ekle.

Ampulden Using Kodları çöz

Eksik olan **using System.Linq;** // elle ekledik

Github **Business=>Abstract** içinde **IUserService** ve **IAuthService** adlı Interface kopyala ve aynı yerde aynı isimli Interfaceler oluştur ve kodları yapıştır
Ampulden Using Kodları çöz. Ancak **UserForRegisterDto** ve **UserForLoginDto** adlı nesneler eksik olduğu için altı hala çizili

GitHub=>Entities=>DTOs içindeki **UserForRegisterDto** ve **UserForLoginDto** adlı Classları kopyala ve aynı yerde aynı isimli Class'lar oluştur ve kodları yapıştır

GitHub=>Business=>Concrete=>UserManager ve **AuthManager** adlı Class'ları kopyala ve aynı yerde aynı isimli Classlar oluştur ve kodları yapıştır

Github=>**WebAPI=> Controller** **AuthController** adlı Contoller içindeki kodları kopyala ve aynı yerde oluşturduğum aynı adlı Controller içine yapıştır

Github=>**Business=>DependencyResolvers=>Autofac** içindeki **AutofacBusinessModules** adlı Class in içinden **User** ve **Auth** için gerekli 4 satırı kopyalayalım ve aynı klasörde aynı yere yapışralım. Ampülden using satirini çözelim

```
builder.RegisterType<UserManager>().As< IUserService>();  
builder.RegisterType<EfUserDal>().As< IUserDal>();  
  
builder.RegisterType<AuthManager>().As< IAuthService>();  
builder.RegisterType<JwtHelper>().As< ITokenHelper>();
```

Github=> **WebAPI** içindeki **Startup** Classından bir metot kopyalayacağız. **WebAPI** içindeki **Startup** Classına **ConfigureServices** metodu içine yapışracagız
TokenOptions icin **using Core.Utilities.Security.JWT;** komut satirini çöz
TokenValidationParameters **icin using Microsoft.IdentityModel.Tokens;** komut satırını çöz.
SecurityKeyHelper **icin using Core.Utilities.Security.Encryption;** komut satırını çöz.

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers(); // vardi zaten

    var tokenOptions = Configuration.GetSection("TokenOptions").Get<TokenOptions>();

    services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
        .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidIssuer = tokenOptions.Issuer,
            ValidAudience = tokenOptions.Audience,
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = SecurityKeyHelper.CreateSecurityKey(tokenOptions.SecurityKey)
        };
    });
}

```

Eklenti: **WebAPI** Sağ tıkla **Manage NUGet Packages** kısmında **Microsoft.AspNetCore.Authentication.JwtBearer (version 3.1.11)** yükleyelim

Ampulden using satırlarını çözelim. JWT Token tanıtılmış oldu

Startup içinde mevcut 2.metot içine aşağıdaki komutu ekledik

```

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    ...
    app.UseAuthentication(); // ekledik 14. ders
    ...
}

```

Koşturma: Ürün eklemek istedik. Program koşmadı (Hocada Versiyon Farkı nedeniyle çalışmadı)

Bir kullanıcı kaydı yaptı. Çalıştığını gördük.

Kullanıcı kaydımız sonrası admin şifresi ile ürün ekledim. Sonuç başarılı

Yalnız Burada Veri girişinin yanında header kısmında Authorizarion seçerek ““Bearer” kodu +bosluk+ Token değeri” (login olduğumuzda çıkan Token kodunu (sadece tırnak içini) kopyaladık. Yetkimiz olduğunu bildirmiş olduk

Login kayıttı ise parola doğru olduğu halde hatalı uyarısı verdi. Tavsiye üzerine Tabloda PasswordHash ve PasswordSalt tipini Binary(500) ‘ den Binary (128)’e düşürünce sorun kalktı

The image shows two side-by-side screenshots of development tools. On the left is a screenshot of the Postman API testing tool. It displays a POST request to `https://localhost:44356/api/auth/register`. The 'Body' tab is selected, showing a JSON payload with fields: `"FirstName": "Engin", "LastName": "Demir", "email": "Engin@engin", "password": "2345"`. The response status is 200 OK with a token returned in the JSON body. On the right is a screenshot of SQL Server Object Explorer showing the `dbo.users` table. It contains one row with Id=1, FirstName='Engin', LastName='Demir', Email='Engin@engin', PasswordHash='0x7A4451EA26...', PasswordSalt='0x5D7405E2930...', and Status=True. Below this is another Postman screenshot showing a POST request to `https://localhost:44356/api/products/add`. The 'Body' tab is selected, showing a JSON payload with fields: `"categoryId": 5, "productName": "A 14.ders", "unitsInStock": 39, "unitPrice": 18.0000`. The response status is 200 OK with a success message returned in the JSON body.

POST https://localhost:44356/api/products/add

Headers (9)

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJodHRwOi8vd3Lncz...	
Key	Value	Description

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```

1
2   "success": true,
3   "message": "Ürün eklendi"
4

```

```
{
    "categoryId": 7,
    "productName": "A 14.ders",
    "unitsInStock": 39,
    "unitPrice": 18.0000
}
```

POST https://localhost:44356/api/auth/login

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings (1)

none form-data x-www-form-urlencoded raw binary GraphQL **JSON**

```

1
2   "FirstName": "Engin",
3   "LastName": "DEDE",
4   "email": "ddsdssdsd@gmail.com",
5   "password": "123456788"
6

```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```

1
2   "token":
      "eyJhbGciOiJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy8yMDAxLzA0L3htbGRzaWctbW9yZSNobWFjLXNoYTUxMiIisInR5cCI6IkpxVCJ9.
      eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy8yMDAxLzA1L21kZW50aXR5L2NsYWhlcy9uYW1laWR1bnRpZml1ciI6IjIiLCJlbWFpbCI6ImNhdmlAY2F2aSISimh0dHA6Ly9zY2hlbWFzLnhtbHNvYXAb53nL3dzLzIwMDUv
      IjM1Lj1bwPpbCI6ImRkc2RzZMKqG0tWlslmNbSiisInh0dHA6Ly9zY2hlbWFzLnhtbHNvYXAb53nL3dzLzIwMDUv
      awR1bnRpDHkvY2xhaW1zL25hbWUiOiJNZWhtZXQgQ2F2aSISim5iZiI6MTYxNDQ2NzI4NiwiZXhwIjoxNjE0NDY3ODg2LCJpc3MiOiJlbmdpbkB1bmdp
      bi5jb20iLCJhdWQiOiJlbmdpbkB1bmdpbi5jb20ifQ.RIkMR3orsKjpmHWKzDQnBq1UxqFB1CxNRh0LIOAmVhEIWLZVHphA8rX8QtERHKVZYSkcEW-2W_TIwu39MlgTw
      mB2z9He1buXa_EUXQhehsPZhXU47AX2i8yzsdM395bh89R5qPoqN600pM7_dgSvKYCuS20nCwMbNjk4Rgg",
      "expiration": "2021-02-28T01:11:09.0937348+01:00"
3
4

```

```
{
    "FirstName": "Engin",
    "LastName": "DEDE",
    "email": "ddsdssdsd@gmail.com",
    "password": "123456788"
}
```

Token Değeri (her login de farklı değer alır)

eyJhbGciOiJodHRwOi8vd3LnczLm9yZy8yMDAxLzA0L3htbGRzaWctbW9yZSNobWFjLXNoYTUxMiIisInR5cCI6IkpxVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy8yMDAxLzA1L21kZW50aXR5L2NsYWhlcy9uYW1laWR1bnRpZml1ciI6IjIiLCJlbWFpbCI6ImNhdmlAY2F2aSISimh0dHA6Ly9zY2hlbWFzLnhtbHNvYXAb53nL3dzLzIwMDUvMDUvaWR1bnRpDHkvY2xhaW1zL25hbWUiOiJNZWhtZXQgQ2F2aSISim5iZiI6MTYxNDQ2NzI4NiwiZXhwIjoxNjE0NDY3ODg2LCJpc3MiOiJlbmdpbkB1bmdpbi5jb20iLCJhdWQiOiJlbmdpbkB1bmdpbi5jb20ifQ.RIkMR3orsKjpmHWKzDQnBq1UxqFB1CxNRh0LIOAmVhEIWLZVHphA8rX8QtERHKVZYSkcEW-2W_TIwu39MlgTw

Tamamlayıcı Kaynak: Engin Demirog UDEMY Kursu :Net Core & C# Backend Frontend Geliştirici Kurs

<https://www.udemy.com/course/net-core-c-sharp-kursu/>

Proje Standart Kurumsal Mimari Katmanlar	
Business (Class Library.Net Standart)	İş Kuralları Abstract (IService) Concrete(Manager) BusinessAspects =>Autofac Constants (Messages) DependencyResolvers (Autofac) Cross Cutting Control Öğeleri (Loglanma, Cashlama, Validation) // Her katmanda kullanılabilir
DataAccess (Class Library .Net Standart)	Veri Tabanı Yönetimi Veriye Erişim Kodları burada yazılır Abstract (Dal) Concrete =>(EntityFramework(Dal ve Context), InMemory)
Entity (Class Library.Net Standart)	Proje için gerekli varlıklar/Nesneler veri tabanının projedeki karşılıkları veya birden fazla veri tabanından alınan değerleri joint ederek biz bir nesne oluşturabiliriz Concrete DTO
Core (Class Library.Net Standart)	Çekirdek Kodlar // .NetCore ile karıştırmayın Veri tabanından bağımsız her projede kullanılabilecek unsurlar JWT , Web Token icin gerekli nesneler Dependencies Aspects=>Autofac=>Validation Cross Cutting Concerns=>Validation DataAccess =>EntityFramework Entities =>(Concrete, IDTO, IEntity) Extensions Utilities =>(Business, Interceptors, IOC, Results, Security(Hashing, Encryption ve JWT))
WebAPI (Asp.NET Core Web Application)	API Ara yüzü Controllers Startup
Console (Asp.NET Core Web Application)	Kodları test yapmak için

15.DERS (03.03.2021)

Konu : Cache , Transactions, Performance, Logging
Kullanılan Dosya : Visual Studio => Project => MyFinalProject (DEVAM)

STEP 113: **İyileştirme yapıyoruz** Lazım olduğunda çalışan Class 'ları buraya alacağız. Yükleme isini burası yapacak

Core=>Utilities=> IOC içine ICoreModule adlı interface oluştur. Public yap

IServiceCollection için using Microsoft.Extensions.DependencyInjection; komut satırını ampulden çöz

```
public interface ICoreModule
{
    void Load(IServiceCollection serviceCollection);
}
```

Eskinden Business içindeki DependencyResolvers altındaki Autofac içindeki yapıları Core içine alacağız

STEP 114: Core içine **DependencyResolvers** adlı klasör oluştur. Altına **CoreModule** adlı Class ekle ICoreModule inherit et public yap.

Ekleni: Ampulden Using Komut satırını çöz. IHttpContextAccessor için ampulden Microsoft.AspNetCore.Http; eklenisini yükley veya core için sağ tıkla **manage Nuget packages** kullanarak

```
public class CoreModule : ICoreModule
{
    public void Load(IServiceCollection serviceCollection)
    {
        serviceCollection.AddSingleton < IHttpContextAccessor, HttpContextAccessor > ();
    }
}
```

STEP 115: Extensions yazılım. Polymorphism yapalım

Core =>Extensions içine ServiceCollectionsExtensions adlı Class ekle. Public Static yapmayı unutma.

Core katmanı dahil olmak üzere ekleyeceğimiz bütün injectionları bir araya toplayacağımız bir yapı oluşturacağız

Ampülden aşağıdaki using komut satırlarını çöz

IServiceCollection için `using Microsoft.Extensions.DependencyInjection;`

ICoreModule için `using Core.Utilities.IOC;`

```
public static class ServiceCollectionsExtensions
{
    public static IServiceCollection AddDependencyResolvers
        (this IServiceCollection serviceCollection, ICoreModule[] modules)
    {
        foreach (var module in modules)
        {
            module.Load(serviceCollection);
        }
        return ServiceTool.Create(serviceCollection);
    }
}
```

STEP 116: WebAPI Startup içine ConfigureServices metodu içine aşağıdaki yapıyı ekleyelim

```
services.AddDependencyResolvers(new Core.Utilities.IOC.ICoreModule[] {
    new CoreModule()
});
```

AddDependencyResolvers için `using Core.Extensions` komutunu çöz

CoreModule için using komut satırını çöz

NEYI NICIN YAPIYORUZ

Cache yazacağız

Bir işlemi biz veya başka bir kullanıcı yaptığında tekrar kullanmak istediğimizde Cash 'ten kullanabiliriz. Bunun süresini belirleyebiliriz

Birkaç yöntemle yapabiliyoruz. Microsoft'un kendi içindeki Inmemory 'e Cash mekanizmasını kullanacağız. Kendimize uyarlayacağız. İlerde başka bir yapıya geçtiğimde sorun yaşamamak için. Dinamik bir yapı kuruyoruz

Veri tabanında bir değişim olması durumunda (ekleme, silme, güncelleme) durumunda da önceden Cache belleğe alınan söz konusu veri tabanı değerleri degistigi için kullanılamayacağı için ilgili cache'in uçurulmasını kodlayabiliriz (çünkü kullanmayacağız)

Cash 'ları bellekte Key Value Pair ile tutuyoruz. Parametreli olanları da tutabiliyoruz. Her parametreye göre değişen bir cache yapısı kuracağız

Manipulasyon yapan metodlarınızı cache aspekt ile yönetiriz. Bu aspect kullanımlarında Veri tabanına manuel düzenleme sıkıntısı aratabilir

STEP 117: **Core =>Cross Cutting Controls** klasörü içine **Caching** adlı klasör oluşturalım

İçine **ICacheManager** adlı interface oluşturalım public yapalım.

Bir metot çağrıldığında cache de varsa oradan getiririz. Yoksa çağrııp sonrası için cache içine alırız

Güncellenen ve silinenleri e uçurmak istiyoruz. Çünkü eski bilgi kullanılamaz. Bu kapsamda lazım olabilecek metodları tanımlayalım

```
public interface ICacheManager
{
    T Get<T>(string key);                                // generic Metodlar
    object Get(string key);                               // generic olmayan versiyonu
    void add(string key, object value, int duration);   // her tipi kapsayacak şekilde yazılır, cache süresi de belirtilir

    bool IsAdd(string key);
    void Remove(string key);
    void RemoveByPattern(string pattern);
}
```

STEP 118: Core=>DependencyResolvers adlı klasöründe CoreModule adlı Class açalım. Bir sonraki metottaki Cache için Class çağrıracagız

ICacheManger icin `using Core.CrossCuttingConcerns.Caching;`

```
public class CoreModule : ICoreModule
{
    public void Load(IServiceCollection serviceCollection)
    {
        serviceCollection.AddMemoryCache(); // Cach kapsamında ekledik
        serviceCollection.AddSingleton < IHttpContextAccessor, HttpContextAccessor > ();
        serviceCollection.AddSingleton<ICacheManager, MemoryCacheManager>(); // Cach kapsamında ekledik
    }
}
```

STEP 119: Caching klasörü içine Microsoft adlı bir klasör oluşturalım. Altına da MemoryCacheManager Class oluştur.

Public yap ve ICacheManager inherit et

Interface Implement edelim (metotlar tanımlanmak üzere görünsün) metotlarını dolduralım.

Daha sonra Microsoft harici başka bir cache yöntemi istersek başa bir klasörde aynı yapıyı kurabiliriz

Kullanacağımız nesneyi tanımlayalım ve using komut satirini çözelim (`using Microsoft.Extensions.Caching.Memory`)

```
IMemoryCache _memoryCache;
```

```

using Core.Utilities.IOC;
using Microsoft.Extensions.Caching.Memory;
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.Extensions.DependencyInjection; // ekledik, otomatik çalışmadı

namespace Core.CrossCuttingConcerns.Caching.Microsoft
{
    class MemoryCacheManager : ICacheManager
    {
        IMemoryCache _memoryCache;
        public MemoryCacheManager()
        {
            _memoryCache = ServiceTool.ServiceProvider.GetService<IMemoryCache>(); // service tool için using komut satırını ekle. ayrıca
yukarıya elle yazdım
        }
        public void add(string key, object value, int duration)
        {
            _memoryCache.Set(key, value, TimeSpan.FromMinutes(duration));
        }
        public T Get<T>(string key)
        {
            return _memoryCache.Get<T>(key);
        }
        public object Get(string key)
        {
            return _memoryCache.Get(key);
        }
        public bool IsAdd(string key)
        {
            return _memoryCache.TryGetValue(key, out _); // cache de olup olmadığı ve varsa değer döndüren bir metod. biz sadece kontrol
istiyoruz out _ kullandık
        }
    }
    public void Remove(string key)
    {
        _memoryCache.Remove(key); //
    }
}

```

STEP-120: Core=>Caching=>Microsoft klasörü **MemoryCacheManager** Class 'in **RemoveByPattern** metotunu GITHUB'daki bir Projeden alacagiz

Engin Demirog'un Github 'indan NetCoreBackend projesinden Core.CrossCuttingConcerns.Caching.Microsoft MemoryCacheManager Classindan **RemoveByPattern** metodunu kopyaliyoruz

```
public void RemoveByPattern(string pattern)
{
    var cacheEntriesCollectionDefinition = typeof(MemoryCache).GetProperty("EntriesCollection",
System.Reflection.BindingFlags.NonPublic | System.Reflection.BindingFlags.Instance);
    var cacheEntriesCollection = cacheEntriesCollectionDefinition.GetValue(_memoryCache) as dynamic;
    List<ICacheEntry> cacheCollectionValues = new List<ICacheEntry>();

    foreach (var cacheItem in cacheEntriesCollection)
    {
        ICacheEntry cacheItemValue = cacheItem.GetType().GetProperty("Value").GetValue(cacheItem, null);
        cacheCollectionValues.Add(cacheItemValue);
    }

    var regex = new Regex(pattern, RegexOptions.Singleline | RegexOptions.Compiled | RegexOptions.IgnoreCase);
    var keysToRemove = cacheCollectionValues.Where(d => regex.IsMatch(d.Key.ToString())).Select(d => d.Key).ToList();

    foreach (var key in keysToRemove)
    {
        _memoryCache.Remove(key);
    }
}
```

Regex için **using** System.Text.RegularExpressions;

Where icin **using** System.Linq; çözelim

Bu metot Bellekteki Collection'a gider. Her birini gezer. Belirttiğimiz kurala uyanları atılacak listeye kaydedecek ve sonra cache den silecek

STEP-121: biz bu cache kullanımı her metot içine **CacheManagerer** yazabilriz/kullanabiliyoruz.ancak bir kere yazacağımız aspect ile yapmak daha kolay

Business Concrete klasörü ProductManager Classi içinde **GetAll** ve **GetById** metodlarının hemen üzerine [CacheAspect] komutunu ekle. Şimdi bu adda aspekt yazacağız

Business Concrete klasörü ProductManager Classi içinde **Add**, **Update** ve **Delete** metodlarının hemen üzerine [CacheRemoveAspect] komutunu ekle. Şimdi bu adda aspekt yayacağız

STEP-122: Engin Demiroğ'un Github 'ından NetCoreBackend projesinden **Core=>Aspects=>Autofac =>Caching** klasörünü açalım

CacheAspect Classından kopyalıyoruz

Core=>Aspects=>Autofac =>Caching klasörünü oluşturalım ve **CacheAspect** adlı Class oluşturalım ve buraya kodları yapıştırıralım.

Using komut satırlarını çözelim. Elle yazmamız gereken bir using komut satırı var

Burada yapılanlar: (bu metotun başında çalışacak bir aspect olduğunu unutma)

Metota ulaşıyoruz. Class 'in adını alıyoruz (Reflected Type) kullanılcak metotun adını alıyoruz ve cach içinde işlem yapabilmek için key oluşturuyoruz.

Kullanılacak parametre varsa onu da alır

Belleke gider ve bellekte key varsa metotu (bu aspect hangi metotun başında yazıldıysa) çalıştırmadan çıkar. Aksi takdirde metot çalışır. Bundan sonra bir daha çalışmasın diye metot işlemi cach içine alınır

```
using Castle.DynamicProxy;
using Core.CrossCuttingConcerns.Caching;
using Core.Utilities.Interceptors;
using Core.Utilities.IOC;
using System;
using System.Collections.Generic;
```

```

using System.Text;
using Microsoft.Extensions.DependencyInjection; // elle ekledik
using System.Linq;

namespace Core.Aspects.Autofac.Caching
{
    public class CacheAspect : MethodInterception
    {
        private int _duration;
        private ICacheManager _cacheManager;

        public CacheAspect(int duration = 60)
        {
            _duration = duration;
            _cacheManager = ServiceTool.ServiceProvider.GetService<ICacheManager>(); // burası için using komut satırı elle yazıldı
        }

        public override void Intercept(IInvocation invocation)
        {
            var methodName = string.Format($"{invocation.Method.ReflectedType.FullName}.{invocation.Method.Name}");
            var arguments = invocation.Arguments.ToList();
            var key = $"{methodName}({string.Join(", ", arguments.Select(x => x?.ToString() ?? "<Null>"))})"; /
                / ?? varsa bu yoksa diğerini yap anlamındadır

            if (_cacheManager.IsAdd(key))
            {
                invocation.ReturnValue = _cacheManager.Get(key);
                return;
            }
            invocation.Proceed();
            _cacheManager.add(key, invocation.ReturnValue, _duration);
        }
    }
}

```

STEP-123: Engin Demiroğ'un Github 'ından NetCoreBackend projesinden **Core=>Aspects=>Autofac =>Caching** klasörünü açalım

CacheRemoveAspect Class indan kopyalıyoruz. İsimize yaramayacak (veri tabanında değişmiş, silinmiş, eklenmiş veriler olduğu için veri tabanından önceden alınan cache kullanılamaz) verileri cache'den uçuracağız

Core=>Aspects=>Autofac =>Caching klasörünü oluşturalım ve **CacheRemoveAspect** adlı Class oluşturalım ve buraya kodları yapıştırıyalım.

Using komut satırını çözelim. Elle yazmamız gereken bir using komut satırı var

```
using Castle.DynamicProxy;
using Core.CrossCuttingConcerns.Caching;
using Core.Utilities.Interceptors;
using Core.Utilities.IOC;
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.Extensions.DependencyInjection; // elle ekledik

namespace Core.Aspects.Autofac.Caching
{
    public class CacheRemoveAspect : MethodInterception
    {
        private string _pattern;
        private ICacheManager _cacheManager;
        public CacheRemoveAspect(string pattern)
        {
            _pattern = pattern;
            _cacheManager = ServiceTool.ServiceProvider.GetService<ICacheManager>();
        }
        protected override void OnSuccess(IInvocation invocation) // Add, Delete ve Update metotları başarılı olursa bu cache teki
                                                               // Get olan yani veri tabanından veri okuyan cacheler kullanılamaz/Uçuracağız
        {
            _cacheManager.RemoveByPattern(_pattern);
        }
    }
}
```

STEP-124: Business=> Concrete Klasörü **ProductManager** Class içinde Update ve Add metodlarının üzerine **CacheRemoveAspect** adlı aspect yazacağız

```
[ValidationAspect(typeof(ProductValidator))]
[CacheRemoveAspect("IProductService.Get")]
public IResult Update(Product product)
{
-----
}

[SecuredOperation("product.add,admin")]
[ValidationAspect(typeof(ProductValidator))]
[CacheRemoveAspect("IProductService.Get")]
public IResult Add(Product product)
{
-----}
```

Koşturma(Run): CacheAspect içinde `if (_cacheManager.IsAdd(key))` satırına bir break point koyup çalıştıralım

Login olalım

Post dan GetById?id=1 ile 1. ürünü sorgulayalım. İlk kez çalıştığında Breakpoint 'in olduğu Cache komutları çalışmaz, çünkü cache bos, veri direkt veri tabanından gelir. Ancak 2. Kez ve sonrası GetById sorgusunda bu komutlar çalışır ve değerler veri tabanından değil, Cache 'den görüntülenir

Yeni bir ürün ekleyim:

GetByld ile 1.ürünün sorgusu yapalım. Bu sefer ise ilgili veri tabanında değişme olduğundan bu komut satırları çalışmaz ve değerler veri tabanından alınarak görüntülenir. Cache içindeki bu veri tablosuna ait değerler uçurulur (silinir)

Neyi niçin yapıyoruz

Transaction yönetimi uygulamalarda tutarlılığı korumak için yapılan bir yöntemdir. Örneğin benim hesabımda olan 100 tl den 10 tl 'sini keremin hesabına aktarmak istiyorum. Bunun sonunda herkesin malumu benim hesabım 90 tl'ye düşecek ve keremin hesabı 10 tl artması gerekir. Ancak keremin hesabına bir şekilde para yatırılamaz ve hata verirse işlemi geri alınmalıdır. Yani benim hesabım tekrar 100 tl olmalı

STEP-Deneme: Business=> Concrete Klasörü IProductService içine IResult AddTransactionalTest(Product product); adlı uydurmasyon bir metot ekleyelim

STEP-Deneme: Business=> Concrete Klasörü ProductManager Class içinde bu metodu tanımlayalım. Uydurma bir gerekçe yazdık. Gerekce (if komutu) gerçekleşmezse yapılan işlemi geri alınması gerekir (burada add metodu)

[TransactionScopeAspect] adlı bir aspect ile bu durumu aspect olara yazalım ki bu kodlar buraları kirletmesin

```
//[TransactionScopeAspect]
public IResult AddTransactionalTest(Product product)
{
    Add(product);
    if (product.UnitPrice < 10)
    {
        throw new Exception(""); // senaryo gereği Exception fırlatıyoruz
    }

    Add(product);

    return null;
}
```

STEP-125: Github 'da NetCoreBackend=> Core=>Aspects/Autofac=>Transaction Klasörü **TransactionScopeAspect** adlı Class içinde kodları alalım

Kendi Projemizde **Core=>Aspects/Autofac=>Transaction** adlı Klasörü oluşturalım ve içine **TransactionScopeAspect** adlı Class tanımlayalım. Github 'tan aldığımız kodu ekleyelim ve gerekli olan metodlar için using Komut satırlarını çözelim.

```
public class TransactionScopeAspect : MethodInterception
{
    public override void Intercept(IInvocation invocation) // invaocation buraya gönderilen metot demektir. O metot yerine bu metot çalışır
    {
        using (TransactionScope transactionScope = new TransactionScope())
        {
            try
            {
                invocation.Proceed();
                transactionScope.Complete(); // Sorun çıkmazsa ilerler
            }
            catch (System.Exception e)
            {
                transactionScope.Dispose(); // sorun çıkarsa iptal olur
                throw;
            }
        }
    }
}
```

Not: Bunu ödevde herkes derste örnek olarak verildiği sekliye tanımlamış. Bence İstenen metotun üzerine veya her metota çalışacak sekilde yazılmalı

Sistem Performans Sorgulama

Programdaki her bir metotun (işlemin) hızını yani performansını ölçen, sorgulayan ve zafiyet varsa bildiren bir yapı kurabiliriz

STEP-126: Github 'da NetCoreBackend=> Core=>Aspects/Autofac=>Performance Klasörü **PerformanceAspect** adlı Class içinde kodları alalım. Bunu Attribute şeklinde tanımlayabiliriz

Kendi Projemizde Core=>Aspects/Autofac=>Performance adlı Klasörü olusturalım ve icine **PerformanceAspect** adlı Class tanımlayalım. Githubtan aldığımız kodu ekleyelim ve gerekli olan metodlar için using Komut satırlarını çözelim.

GetService metodu için manuel **using** kodu ekleyelim Visual otomatik getirmede

```
using Microsoft.Extensions.DependencyInjection; // GetService metodu için ekledik. Visual otomatik getirmede

public class PerformanceAspect : MethodInterception
{
    private int _interval;
    private Stopwatch _stopwatch;    // timer (sayac) metod basında calisir , bittiğinde gecen süreyi ölçer
    public PerformanceAspect(int interval)
    {
        _interval = interval;                      //
        _stopwatch = ServiceTool.ServiceProvider.GetService<Stopwatch>();      // instance şeklinde
    }
    protected override void OnBefore(IInvocation invocation)
    {
        _stopwatch.Start();
    }
    protected override void OnAfter(IInvocation invocation)
    {
        if (_stopwatch.Elapsed.TotalSeconds > _interval)
        {
            Debug.WriteLine($"Performance : {invocation.Method.DeclaringType.FullName}.{invocation.Method.Name}-->{_stopwatch.Elapsed.TotalSeconds}");           // uyarı yazısı
        }
        _stopwatch.Reset();
    }
}
```

STEP-127: Burada kullanılan **Stopwatch** metodu için **Core=>DependencyResolvers** klasöründe **CoreModule** Class içine aşağıdaki kodu ekleriz ve ampulden using komut satırını çözeriz

```
using System.Diagnostics;

public void Load(IServiceCollection serviceCollection)
{
    serviceCollection.AddMemoryCache();
    serviceCollection.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();
    serviceCollection.AddSingleton<ICacheManager, MemoryCacheManager>(); // Cache kapsamında ekledik
    serviceCollection.AddSingleton<Stopwatch>(); // sistem performansının ölçülen ve sıkıntısı varsa bildiren bir aspect için ekledik
}
```

STEP-128: **Business=>Concrete** klasöründe **ProductManager** içinde örneğin **GetById** metodu önüne metot 5 sn'yi geçerse uyarı verecek bir aspect yazalım

```
[CacheAspect]
[PerformanceAspect(5)] // bu metodun çalışması 5 saniyeyi geçerse uyarı verir
public IDataResult<Product> GetById(int productId) // Tanımladık 10.02..2021
{
    return new SuccessDataResult<Product>(_productDal.Get(p => p.ProductId == productId));
}
```

Not: Bir aspecti her şeye çalışmasın istersek **Core =>Utilities** Klasörü içindeki **Interceptors** Class içine koyarız

Aynı şekilde Performans ölçümünü de öyle yapabiliriz

STEP-Deneme: Core=>Utilities=>Interceptors Klasörü **AspectInterceptorSelector** adlı Class açalım

Burası çalıştırılmak istenen bir metodun üstüne bakıyordu. Oradaki Interceptorları (aspectleri) buluyor ve çalıştırıyordu

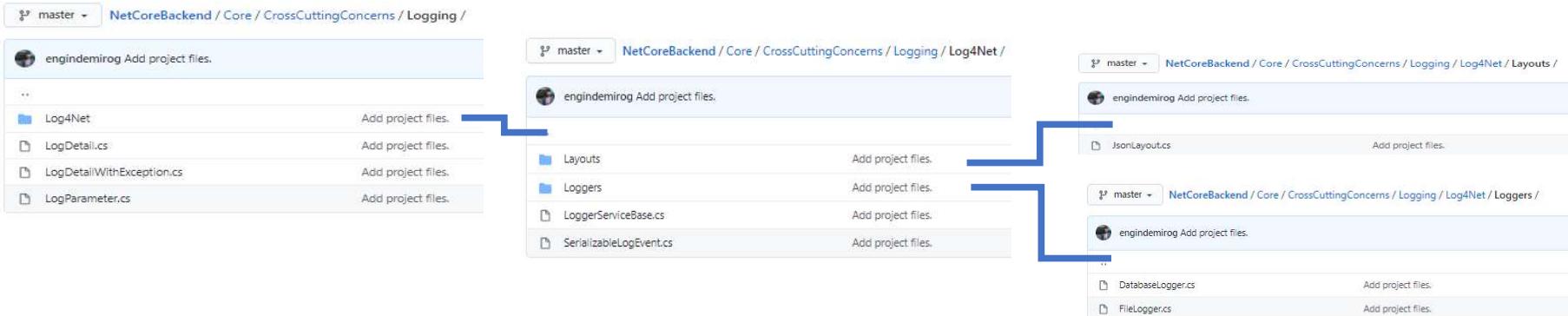
Buraya eklenen aspectler (Log, Perfomaance) ise tüm metotlarda çalışır

Örnegin Log Kaydi: `classAttributes.Add(new ExceptionLogAspect(typeof(FileLogger))); // Otomatik tüm metotları loga dahil eder`

Loglama

Loglama için aşağıdaki gibi birçok işlem gerekiyor (Engin Demirog NetCoreBackend içinde var)

Core=>Aspects=>Autofac =>logging =>**LogAspect.cs** ve aşağıdaki yapı kurulmalı ve tüm metotlarda çalışacak şekilde yukarıdaki kod yazılmalı



İlave Backend Kodları: Frontend'e yönelik veya daha önce yapmamız gereken Backend değişiklikleri

İşlem-1: Startup'a açalım Frontend erişimi için Konfigürasyon yapmalıyız

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to the container.
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddControllers();
        services.AddCors(); //Frontend'ten erişimi için Configurasyon yapmalıyız 17.Hafta
        -----
        -----
    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        ----

        app.UseCors(builder=>builder.WithOrigins ("http://localhost:54694").AllowAnyHeader()); //Frontend'ten erişimi için Configurasyon yapmalıyız 17.Hafta. Buradaki konumu önemli
        -----
    }
}
```

İşlem-2: Backend icinde WebAPI=>Controller içine CategoriesController adlı bir controller ekleyelim

Injecton tanımlayacağız ve ampulden Using Business.Abstract komutunu ekleyelim

_categoryService henz tanımlanmadı. Onun için metot yazacağız veya ampulden “Intilialize field from Constructor” seçilirse otomatik oluşur.

Bende otomatik çıkmadı çünkü bir öğrenci email kaydıyla ücretiz kurulabilen Rescharper adlı eklenti gerekiyor

ProductsController'dan GetAll Metotunu kopyalayalım ve uyarlayalım

```
public class CategoriesController : ControllerBase
{
    ICategoryService _categoryService;
    public CategoriesController(ICategoryService categoryService)
    {
        _categoryService = categoryService;
    }

    [HttpGet("getall")]
    public IActionResult GetAll()
    {
        var result = _categoryService.GetAll();
        if (result.Success)
        {
            return Ok(result);
        }
        return BadRequest(result);
    }
}
```

Backend icinde Business=>DependencyResolvers=> Autofac=> AutofacBusinessModule adlı Classı açalım

CategoriesControler icinde injection yaptığı CategoryService karşılığı olup olmadığını kontrol etmeliyiz. Daha önce tanımlamışız

İşlem-3: Veri tabanına Frontend üzerinden Add (Ekleme) için

Backend → Concrete => ProductManager içinde “[SecuredOperation("product.add,admin")]” satırını şimdilik iptal edelim Çünkü Frontent altyapısı henüz hazır değil

```
//[SecuredOperation("product.add,admin")]
```

İşlem-4: Validation Hatalarını da listeleyecek bir yapı kuralım. Frontent'e validation hata listesi gerekiyor

Adım 4-1: Backend Core → Extentions içine ExceptionMiddleware adlı bir Class oluştur ve Engin Demirog GITHUB'ından NetCoreBackend projesinden aynı yerden (Core-Extensions-ExceptionMiddleware) kodları kopyala. Gerekli Using komut satır çözümlemeleri yap. Sarı renkli düzenlemeleri yap

```
using System;
using System.Collections.Generic;
using System.Net;
using System.Text;
using System.Threading.Tasks;
using FluentValidation;
using FluentValidation.Results;
using Microsoft.AspNetCore.Http;

namespace Core.Extensions
{
    public class ExceptionMiddleware
    {
        private RequestDelegate _next;

        public ExceptionMiddleware(RequestDelegate next)
        {
            _next = next;
        }
    }
}
```

```

public async Task InvokeAsync(HttpContext httpContext)
{
    try
    {
        await _next(httpContext);
    }
    catch (Exception e)
    {
        await HandleExceptionAsync(httpContext, e);
    }
}
private Task HandleExceptionAsync(HttpContext httpContext, Exception e)
{
    httpContext.Response.ContentType = "application/json";
    httpContext.Response.StatusCode = (int) HttpStatusCode.InternalServerError; // burada hata sistem hatası 500 olarak belirtilmiş

    string message = "Internal Server Error";
    IEnumerable<ValidationFailure> errors; // hata listesi oluşturacağız
    if (e.GetType() == typeof(ValidationException))
    {
        message = e.Message;
        errors = ((ValidationException)e).Errors; // hataların hepsini yakaladık ve listeye aldık
        httpContext.Response.StatusCode = 400; // hata statü kodunu da kötü sonuç olacak şekilde düzeltelim (500'den 400'e)

        // validation'a uygun bir hata listesi oluşturduk ve ekledik
        return httpContext.Response.WriteAsync(new ValidationErrorDetails
        {
            StatusCode = 400,
            Message = message,
            Errors = errors
        }.ToString()); // Json'a çevirmek için
    }
    return httpContext.Response.WriteAsync(new ErrorDetails // burası sistemsel hatayı belirtir. Validation Hata istesi içermez
    {
        StatusCode = httpContext.Response.StatusCode,
        Message = message
    }.ToString());
}
}

```

Adım 4-2: Backend → Core → Extentions içine **ErrorDetails** adlı bir Class oluştur ve Engin Demirog GITHUB'ından NetCoreBackend projesinden aynı yerden (Core-Extensions-**ErrorDetails**) kodları kopyala.

JsonConvert için ampülden Install Package Newtonsoft.Json ile eklenti kuralım ve using komut satırını ekleyelim

```
using System;
using System.Collections.Generic;
using System.Text;
using FluentValidation.Results;
using Newtonsoft.Json;

namespace Core.Extensions
{
    public class ErrorDetails
    {
        public string Message { get; set; }
        public int StatusCode { get; set; }
        public override string ToString()
        {
            return JsonConvert.SerializeObject(this);
        }
    }
    // hata kodu oluşturan metodу tanımlayalıım. Yukarıdakilere ilaveten Validation hata listesi var
    public class ValidationErrorDetails : ErrorDetails
    {
        public IEnumerable<ValidationFailure> Errors { get; set; }
    }
}
```

Adım 4-3: Backend → Core → Extentions içine **ExceptionMiddlewareExtensions** adlı bir Class oluştur ve Engin Demirog GITHUB'ından NetCoreBackend projesinden aynı yerden (Core-Extensions-**ExceptionMiddlewareExtensions**) kodları kopyala. IApplicationBuilder için Ampülden Using komut satırını çöz

Kendi Midleware'imizi ekleyeceğiz

```

using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.AspNetCore.Builder;

namespace Core.Extensions
{
    public static class ExceptionMiddlewareExtensions
    {
        public static void ConfigureCustomExceptionMiddleware(this IApplicationBuilder app) //StartUp'ta zaten kullanılan bir yaşam döngüsü
        {
            app.UseMiddleware<ExceptionMiddleware>(); //Startup'ta olan yaşam döngüsüne hata olup olmadığını gözetmek için eklemek yapıyoruz
        }
    }
}

```

Adım 4-4: Backend → WebAPI Startup içindeki mevcut metoda ExceptionMiddleware çalışması için komut ekliyoruz

```

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    app.ConfigureCustomExceptionMiddleware(); // 20. derste ekledik. Frontend veri tabanı işleminin barası/hata durumu için
    app.UseCors(builder=>builder.WithOrigins ("http://localhost:4200").AllowAnyHeader()); //Frontend'ten erişimi için Configurasyon
yapmalıyız 17. Hafta. Buradaki konumu önemli
    app.UseHttpsRedirection();
    app.UseRouting();
    app.UseAuthentication(); // ekledik 14. ders
    app.UseAuthorization();
    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllers();
    });
}

```

İşlem-5: Yetkili personelin kontrolü için değerlerin uygunluğunun sağlanması ve kontrol Aspectinin yeniden aktivasyonu

Adım-114: **Backend**→**WebAPI**→**authController** içinde bir satırı düzelttik

```
[HttpPost("login")]
public ActionResult Login(UserForLoginDto userForLoginDto)
{
    var userToLogin = _authService.Login(userForLoginDto);
    if (!userToLogin.Success)
    {
        return BadRequest(userToLogin.Message);
    }
    var result = _authService.CreateAccessToken(userToLogin.Data);
    if (result.Success)
    {
        return Ok(result); /////////////////////////////////////////////////////////////////// 21.derste düzelttik
    }
    return BadRequest(result.Message);
}
```

Adım-115: **Backend**→**Business**→**Concrete**→**ProductManager** içinde add metodu üzerinde ekleme yetkisi kontrolp yapan aspect satrını yeniden aktive edelim . Bir önceki derste Add metodunu yapabilmek için iptal etmişik

```
[SecuredOperation("product.add,admin")] // Frontent kapsamında 21.derste tekrar aktiv ettik
```

2. BÖLÜM : Frontend (Angular)

16.DERS (06.03.2021)

Konu : Genel Sablonun olusturulması

Kullanılan Program : Visual Studio Code

İlave Kaynak: HTML: https://www.youtube.com/watch?v=pkYSPtvDqc&list=PLqG356ExoxZUQtzluVqxRkqU_rqanJAdp

Tanım: Angular Microsoft, React ise Facebook tarafından web uygulamaları (E-ticaret sitesi) geliştirmek için hazır şablonlar sunan bir framework/platform. Javascript, HTML, typescript konusunda temel bilgileri olan bir kişi başka hiçbir şeye ihtiyaç duymadan platform üzerinden e-ticaret sitesi geliştirebilir.

En popüler frontend Web ve mobil'dir

Kurulum: Visual Studio Code indirmeliyiz (<https://visualstudio.microsoft.com/tr/>) Frontend için daha iyidir . Visual Studio Code açık kalsın

Kurulum: node (<https://nodejs.org/en/download/>) 14.16.0 LTS (Kararlı Sürüm-test edilmiş ve onaylanmış versiyon)
diğer 15.11.0 test edilmiş ancak henüz onaylanmamıştır (Halihazırda kararlı sürümler degillerdir)

Adım-1:

Eklenti: <https://angular.io/cli> : bileşenleri hızlıca Angular uygulamalarını canlıya almak için Angular ekibi tarafından geliştirilen paketi indireceğiz

```
npm install -g @angular/cli
```

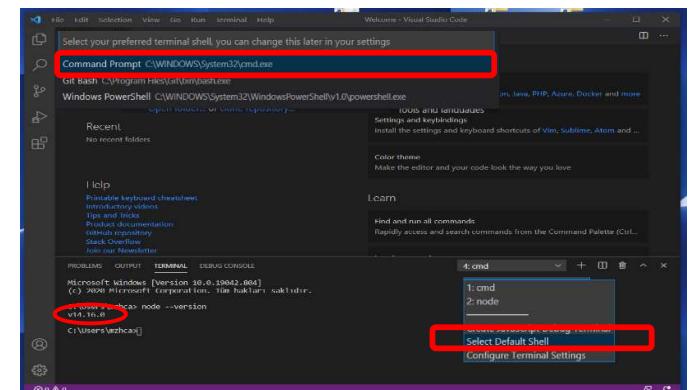
 satırını kopyala

“Visual Studio Code” yukarıda “terminal” sekmesinden “new terminal” sağ altta seçeneklerden “select default Shell” seçelim. Yukarıda çıkan “command prompt” seçelim.

Altta kod yazabileceğimiz bölüm çıkar

“node --version” → yazınca yüklü olan versiyon gözükmür (bende v14.16.0)

Sorun çıkarsa Visual Studio Code kapatıp tekrar açınız



Alt ekranda sağa tıklayınca kopyaladığımız yükleme komutu ekrana gelir

(`npm install -g @angular/cli`) ve entere basınca yükleme yapar. “-g” tüm işletim sisteminde global olarak kurulduğu anlamına gelir

“Y enter” ➔ Yükleme sonunda bilgi paylaşımı için Y/N (Yes oder No) secince kurulum tamamlanıyor

Bilgisayarın C:\ dizinine **kampfrontend** adlı klasör oluşturalım

cd.. ➔ bir üst klasöre çıkar (**2 defa yapalım**)

cd kampfrontend ➔ klasöre gelebiliriz

“Y” yazıp **enter** (Y/N ekranından) yapalım(2 defa)

ng new NORTHWIND komutu girelim (ilgili klasöre bu adda dosyayı acar) // <https://angular.io/cli> sayfasında bu ve birkaç kod yazıyor

CSS ➔ Yükleme yaparken hangi template istendiği soruluyor. Default CSS'dir. **Entere** basınca dosyamıza Angular ile ilgili yükleme başlıyor

cd Northwind ➔ dosyaya eristik

code . (**code boşluk nokta**) ➔ ilgili klasörü Visual Studio Code da aç demektir

Açılan sayfadan üstten **Terminal** → **new Terminal** diyelim

ng serve --open → bir tarayıcı da aç anlama geliyor

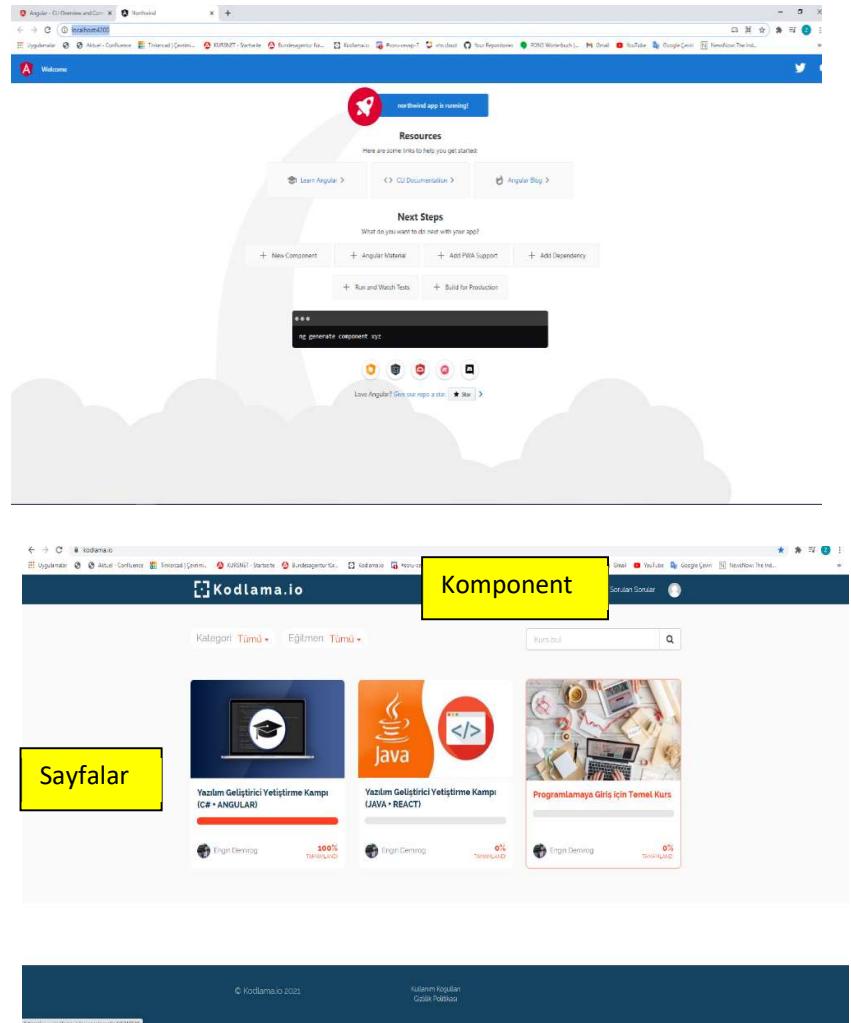
Y enter yapalım (Y/N ekranı çıkar)

\$200 porta sağdaki web sayfası şablonu görünür

ng serve -open -port 4201 → *eğer 4200 no'lu port sıkıntı çıkarırsa*

Normalde kodlama.io vb. web sayfalarında her bir kurs ayrı bir sayfadır

Angular ise bir SPA Single Page Applicationdur. Yani tek sayfalık uygulamalardır. Bu geçişler için Component kullanır örneğin kodlama.io da kategori ve eğitim seçimi bir komponenttir



“Visual Studio Code” → sol taraf menülerinin bazılarını inceleyelim

Not: Github ‘dan başkasına ait Angular uygulamasını indirirsen açtıktan sonra yapılması gereken ilk is “**Visual Studio Code**” kod kısmına “**Npm install**” yazmaktadır (gerekli tüm paketleri yükler)

Node. Modules → Angular kapsamında kullanılan tüm paket modüller burada yer alır

Package.json → Uygulayıcıya ve kullanıcıya gerekli paketler (dependency) ayrı ayrıdır. Biz gerekeni yükleyeceğiz

Src → Kaynak Kodları

Src → **App** → Uygulama kodları

Src → **App** → **Index.html** → Kullanıcı sayfası (buraya dokunmayacağız) burası html kodlarıdır Sadece <app-root> komponenti Angular için kullanılmış

Adım-2: **Src** → **App** sağ tıklayıp “**new Folder**” ile **component** adlı klasör oluştur (uygulamanın Componentlerini buraya koyacağız)

app.component ise ana sayfaya ilgili dosyalardır. Aynı adda farklı uzantıda 4 dosya vardır.

app.component.css → tasarım

app.component.html → ekran **kodları(görüntü)**. // hayatı

app.component.spec.ts → unit test

app.component ts → asıl component (kodları) . Datayı buradan yönetiriz // hayatı

Adım-3: app.component.html açalım ve son satır hariç (<router-outlet></router-outlet>) tüm kodları silelim

“File” kısmından “**save all**” yapalım. 4200 HTML sayfamız bembeyaz bir sayfa oldu. Bundan sonra her sayfada yapacağımız değişiklik sonrası Ctrl+S yapalım

→ app.component.html kısmına “**Merhaba**” yazalım. **4200 HTML** sayfasında da merhaba yazısı göründü

→ Merhaba yazısını silelim aşağıdaki şekilde komutları yazalım.

```
<ul>
<li> Laptop</li>
<li> Mouse</li>
<li> Keyboard</li>
</ul>
```

Not: HTML sadece sayfaya ilgilenen bir static bir yapıdır. Bir ürün eklemek için HTML kısmına gelip ürünü tanıtmak gereklidir. Angular ise istenen datada yer alan verileri sergiler

Not: Angular Nesne güdümlüdür. Her şey Class'lar üzerine kuruludur.

Adım-4: app.component.ts kısmına gelelim Component basit bir Class iken @Component yazarak component olduğunu bildiriyoruz ve tanıtmalar yapıyoruz. Kodu inceleyelim

Not: Component HTML 'in datasını yönettiğimiz yerdir

Not: Söslü parantez objedir

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'northwind';
}
```

// html kodlarını kullanabilmek için
// ben html 'nin datasını yönetecek Komponent olduğu anlamına gelir
// Bu Compnenet'in Css lerinin yerini ifade eder bir dizidir(köseli parantezle notasyon edilir)
// datası

Adım-5: Src içindeki **index.html** bakarsak `<app-root></app-root>` yapabilmemiz için yukarıdaki selector'u yazmış olmalıyız
Not:Angular dünyasına typescript kullanılır

```
export class AppComponent {  
  title :string= 'northwind';           // Java da tip tanımlamaya gerek yok. Typescript 'te datanın tipini tanımlayarak daha güvenli hale getirebiliriz  
  user:string ="Engin Demirog";  
}  
}
```

Adım-6: **app.component.html** açalım en üstte yazalım ve kaydedelim. Title uygulamada görünebilmesi için işlem yapalım

```
Welcome to {{title}}  
hello{{user}}
```

4200 html ye bakınca “**welcome to Northwind hello Engin Demirog**” yazar ve ardından ürünler yazar (yukarıda yazdığımız)

Adım-7: **app.component.ts** gelelim , bir ürün tanıtalım. Notasyona dikkat (C#dan farklı)

```
product1:any={productId:1, productName:'Bardak', categoryId:1, unitPrice:5} // any her şey olabilir demektir. Yazmasak ta olur
```

Not: string tek tırnak çift tırnak fark etmez

Adım-8: app.component.html gelelim. Birkaç ürün tanıtalım ve html 4200 de görelim

Adım-9: app.component.ts gelelim

Adim-10: app.component.html gelelim -

Açıklama: <li *ngFor="">"let product of products" // burası foreach döngüsüdür li' yi tekrarla demektir bu yazıma directive adı verilir dikkat F harfi büyük

Not: Ekranda çalıştığımız yerde sağa tıklayıp **"format document"** yaparsak notasyonu düzenler

Not: product listedir köşeli parantezle notasyon

Not: Yukarı yön tuşu son komutu ekrana getirir

Adim-10 app.component.html kodlari

```
<ul>Welcome to {{title}}</ul>
<ul>hello{{user}}</ul>

<ul>
  <li *ngFor="let product of products">{{product.productName}}</li>
</ul>
<router-outlet></router-outlet>
```

Adim-8 app.component.html kodlari

```
<ul>Welcome to {{title}}</ul>
<ul>hello{{user}}</ul>
<ul>{{product1.productName}}</ul>

<ul>
  <li> Laptop</li>
  <li> Mouse</li>
  <li> Keyboard</li>
</ul>
<router-outlet></router-outlet>
```

Adim-9 app.component.ts kodlari

```
import { Component } from '@angular/core';
```

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title: string = 'northwind';
  user: string = "Engin Demirog";
```

```
product1: any = { productId: 1, productName: 'Bardak', categoryId: 1, unitPrice: 5 }
product2: any = { productId: 2, productName: 'Laptop', categoryId: 1, unitPrice: 5 }
product3: any = { productId: 3, productName: 'Mouse', categoryId: 1, unitPrice: 5 }
product4: any = { productId: 4, productName: 'Keyboard', categoryId: 1, unitPrice: 5 }
product5: any = { productId: 5, productName: 'Camera', categoryId: 1, unitPrice: 5 }
```

```
products =[  
    this.product1,  
    this.product2,  
    this.product3,  
    this.product4,  
    this.product5  
];
```

Adım-11: **src** → **app** sağ tıkla **new folder=> directives** adı ver. Daha sonra direktiflerimizi buraya ekleyeceğiz

Adım-12: Kendi oluşturduğumuz “Components” klasöründe sağ tıkla ve “open in integrated terminal”

Açılan kod kısmına sırasıyla bu 3 kodu yazalım ve **Product, Category ve Navi** componentlerini oluşturalım

Bu duruda component klasörü altında bir componentin tüm bileşenleri oluşur

ng g component product // g generate anlamındadır. (-g (global) ile karıştırma) Product isminde component oluşturur

ng g component category

ng g component navi

Bakış: app → **app.module.ts** adlı klasörünü açalım. **Angular cla** otomatik olarak importları ekledi ve kullanma imkânı vermek için deklere etti

Bakış: app → **index.html** açalım . Burada component hiyerarşisi kurulur

Bakış:: app → **main.ts** hangi modülle işleme başlayacağı tanıtılar

Adım-13: **app.component.html** açalım . Son satır hariç hepsini silelim veya comment yapalım

Not: **Ctrl+k → Ctrl+c** : seçili satırlar Comment haline gelir veya **edit** menüsünden “**toggle block Comment**” seçilir

Adım-13 app.component.html kodları

```
<app-navi></app-navi>
<app-category></app-category>
<app-product></app-product>

<router-outlet></router-outlet>
```

4200 html ekranına çalıştığı bilgisi geldi

Adım-14: Hazır Kodlar : Getbootstrap.com // twitter firmasının CSS geliştirme tool kiti

(Alternatifi Google material)

Yazılan html'lerin daha güzel görünmesini sağlar. **Get started** tuşuna basınca menüler açılıyor

Doc kısmından **Components navbar** açalım

Kod kısmını kopyalayalım

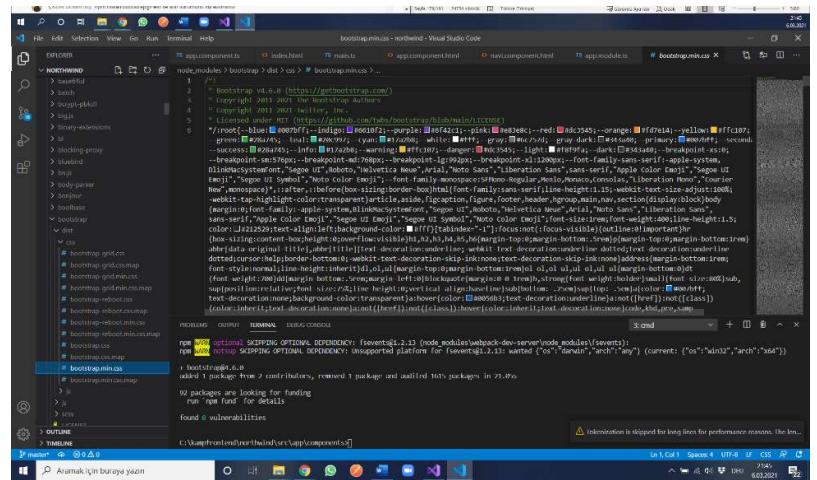
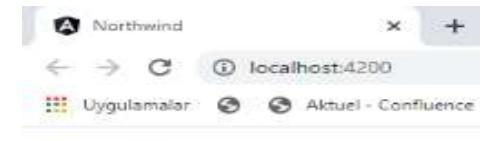
Kendi **navi.components.html** içine yapıştalıyalım (daha önce var olan tek satırı silelim)

4200 html halen düzgün görünmedi

Adım-15: Herhangi bir sayfa acılı iken aşağı kod kısmına “**Npm install bootstrap**” yazalıml ve yükleyelim

Nodemodules içine **bootstrap** klasörü eklenmiş. İcinde **bootstrap.min.cs** adlı dosyayı açalım

Not: min yazılı olan dosyalar kodların az yer tutması için yan yazılmış halidir



Adım-16: **Angular.json** 30. satırındaki **styles** kısmına ekleme yapacağız.

Burasi konfigürasyon yeridir (Backend projedeki WebAPI startup ve upjson kısmına benzer)

```
"styles": [  
    "./node.modules/bootstrap/dist/css/bootstrap.min.css",  
    "src/styles.css"  
],
```

Adım-17: Herhangi bir sayfa açıkken alt komut kısmından “**2:node**” seçelim

Ctrl+C ile programı durduralım **Y(Yes)** ile onay vermeliyiz

Sonra **ng serve – open** koduya tekrar çalıştıralım. Her şey düzeldi



Adım-18: **Src → app → components → app.components.ts** acalim. Product ile ilgili kısımları keselim

Src → app → components → Product → products.component.ts ye gelelim ve ürünleri ekleyelim

```
import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-product',  
  templateUrl: './product.component.html',  
  styleUrls: ['./product.component.css']  
})
```

```

export class ProductComponent implements OnInit {

product1: any = { productId: 1, productName: 'Bardak', categoryId: 1, unitPrice: 5 }
product2: any = { productId: 2, productName: 'Laptop', categoryId: 1, unitPrice: 5 }
product3: any = { productId: 3, productName: 'Mouse', categoryId: 1, unitPrice: 5 }
product4: any = { productId: 4, productName: 'Keyboard', categoryId: 1, unitPrice: 5 }
product5: any = { productId: 5, productName: 'Camera', categoryId: 1, unitPrice: 5 }

products =[  

  this.product1,  

  this.product2,  

  this.product3,  

  this.product4,  

  this.product5  

];
constructor() {}  

ngOnInit(): void {  

}  

}

```

Adım-19: Src → app → components → Product → products.component.html ye gelelim

```

<table class ="table">  

  <tr *ngFor="let product of products">  

    <td>{{product.productId}}</td>  

    <td>{{product.productName}}</td>  

    <td>{{product.categoryId}}</td>  

    <td>{{product.unitPrice}}</td>  

    <td>{{product.unitsInStock}}</td> // Veri tabanını karşılamak için tüm değerlere karşılık değişkenler tanımlıyoruz  

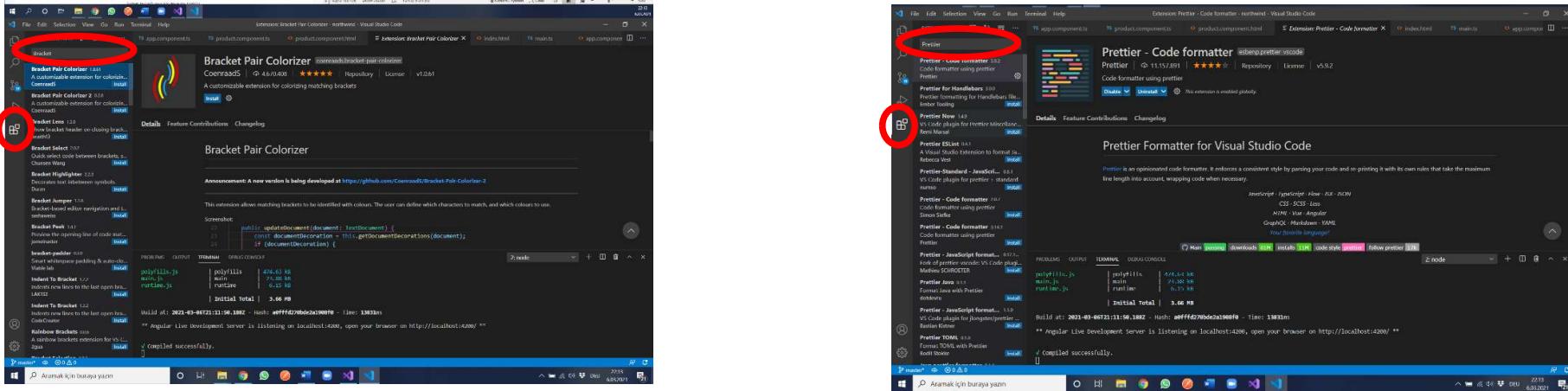
  </tr>  

</table>

```

Web Sayfamız: HTML 4200 'de ürünler düzenli sergilendi

Eklenti: Bracket Pair Colorizer ve Prettier Code Formatter



Not: Ctrl K+ctrl D kod formatlar. Veya çalışılan sayfa üzerinde sağa tıklayıp "format Document" seçilir veya sağ en alta prettier seçebiliriz

17.DERS (10.03.2021)

Konu: NAVBAR ayarı

Backend-Frontend Bağlantısı ve veri görüntüleme

Gecen hafta gerekli program ve eklentileri kurduk. Componentler üzerinde çalıştık, Bu derste ilk Navbar'i düzeltelim. Sonra veri tabanını bağlayalım

Adım-20: <https://getbootstrap.com/> da versiyon sıkıntısı olmuş. Ona dikkat edelim

Yeni bir terminal de (altta solda + (artı) işaretü ile new terminal açalım

npm install bootstrap@ v5.0.0-beta2 ➔ kodlarını yazalım ve navbar versiyonu güncelleyelim
ng serve --open ➔ bir tarayıcı da aç anlama geliyor (HTML 4200 açılır)

Navbar'i düzenleyelim. Categorileri sola ürünleri sağda olacak şekilde düzenleyelim

App-component.html'ye gelelim. Kodları **Container** adlı class içine alalım.

```
<div class = "container">                                         // barın ortalanması için
  <app-navi></app-navi>
<div class="row">
  <div class="col-md-3">                                         // Category kısmı için 3 birim yer tahsis ettik (henüz tanımlamadık)
    <app-category></app-category>
  </div>
  <div class = "col-md-9">                                         // Ürünler kısmı için 3 birim yer tahsis ettik
    <app-product></app-product>
  </div>
</div>
<router-outlet></router-outlet>
</div>
```

Not: Bootstrap 1 satırı 12 eşit parçaya böldüğünü varsayar.

Not: Creative.Com adlı sitede hazır web sayfaları var. Angular template'leri mevcut (ücretsiz de var)
İndirip “npm install” yapılmınca hazır kodlar programınıza eklenir

Getbootstrap sitesine gel Component içinde List group içinden ilk kodu alalım (dikkat versiyon sağ üstte 5.0 seçili olmalı)

Program Component → Category → Category.component.html içine kodları kopyalayalım (Category work yazısını silelim)

HTML 4200 son hali budur

1	Bardak	1	5
2	Laptop	1	5
3	Mouse	1	5
4	Keyboard	1	5
5	Camera	1	5

NEYI NICIN YAPIYORUZ: Visual Studio'da “MyFinalProject” adlı projemizi de açalım

Programı çalıştırıralım products getall yaptığımda datalar sergilenecektir.

Sergi ekranın en başında **Data** başlığı altında geliyor. En sonda da **success** durumu ve **mesaj** sergilenecektir. Sergi ekranında tüm verilen 1 süslü parantez içinde ise nesne olarak sergilendiği anlamına gelir. Kısacası sergilenen veriyi iyi okumalıyız. Frontend kısmında karşılığını tanıtmalıyız

Not: API yüklememiş olanlar (direkt frontend kısmında geçen hafta başlayanlar) Jsplaceholder.typicode.com içinde mevcut fake apileri kullanabilir

Adım-21: App içinde “Models” adlı klasör (new Folder) oluştur.

Onun altına da “product.ts” adlı yeni dosya (Dikkat: new File) oluştur ve ürün karşılığını tanıt. Tipler buraya özeldir. İnt yerine number tipi verilir. Notasyona, noktalama işaretlerine dikkat.

Interface olmasına rağmen burada başına “I” harfi koymaya gerek yoktur(Java da da koyulmaz)

```
export interface Product{  
    productId:number;  
    categoryId:number;  
    productName:string;  
    unitsInStock:number;  
    unitPrice:number;  
}
```

Adım 22: **Product.components.ts** ye gelelim ve product tipinin product Array olarak tanımlayalım. Ürün tanımlamalarını silelim. Artık verileri veri tabanından alacağız.

```
@Component({ selector: 'app-product',  
templateUrl: './product.component.html',  
styleUrls: ['./product.component.css']  
})  
  
export class ProductComponent implements OnInit {  
products:Product[] =[];  
    // burada tipi product array olarak tanımladık. Prod yazdıktan sonra ampül gibi bir şey çıktı oradan product  
    //seçersek yukarı using benzeri import kütüphane satırı otomatik eklenir  
    constructor() {}  
    ngOnInit(): void {  
    }  
}
```

Adım-23: Models icinde **productResponseModel.ts** adlı bir dosya (new file) oluştur. Postman Products.GetAll metodu çıktısı gibi tanımlayalım

Adım-24: Models icinde **responseModel.ts** adlı bir dosya (sağ tıkla new file) oluştur // success ve Message tüm nesneler için ortak olacak (product ve Category için)

Adım-25: Models icinde **productResponseModel.ts** (file) yeni hali

Adım-23: productResponseModel.ts

```
import { Product } from "./products";

export interface ProductResponseModel{
  data:Product [],
  success:boolean,
  message:string
}
```

Adım-24: responseModel.ts

```
import { Product } from "./products";

export interface ResponseModel{
  success:boolean,
  message:string
}
```

Adım-25: productResponseModel.ts

```
import { Product } from "./products";
import { ResponseModel } from "./responseModel";

export interface ProductResponseModel extends ResponseModel {
  data:Product []
}
```

Adım-26: Models icinde **category.ts** oluştur

Adım-27: Models icinde **categoryResponseModel.ts** oluştur. Category ve ResponseModel için yukarı import satırlarını ekleyelim (manuel veya çıkan sekmeden)

Adım-26: category.ts

```
export interface Category{
  categoryId:number;
  categoryName:string;
}
```

Adım-27: categoryResponseModel.ts

```
export interface CategoryResponseModel extends ResponseModel{
  data:Category[]
}
```

Not: 18.Hafta başında Category ve Response Model dosyalarını silip yerine herhangi bir değer döndüren bir dosya olusturacagız

Adım-28: **product.component.ts** açalım. Apideki datayı karşılayacağız

Dikkat edelim class tanımlamada **Implement OnInit** komutu var. En alta da onun metodu var. Bu basta çalışır HPPT Client ile bağlanacağımız o yüzden **import** etmeli ve Construction metodu ile **injection** tanımlamalıyız

Not: HTML açıkken F12'ye basılırsa HTML Comment/kod kısmı açılır

Not: javascript fonksiyon esası, c# class esaslıdır. Bazen this fonksiyonunu kullanırız (metod dışında olana değişkeni kullanmak için)

```
import { Component, OnInit } from '@angular/core';
import { ProductResponseModel } from 'src/app/models/productResponseModel';
import { Product } from 'src/app/models/products';
import {HttpClient} from '@angular/common/http';// apiye baglanıp backenddeki veriyi alabilmek için manuel ekledik
```

```
@Component({
  selector: 'app-product',
  templateUrl: './product.component.html',
  styleUrls: ['./product.component.css']
})
```

```
export class ProductComponent implements OnInit {
  products:Product[] =[];
  apiUrl= "https://localhost:44356/api/products/getall"
  constructor(private httpClient:HttpClient) {}
```

// burası backend API'nin html adresi GITHUB 'dan alırsan burayı uyarlamalısın
// http aktive etmek için injektion yapmaliyiz

```
ngOnInit(): void {
  this.getProducts();
}
```

```
getProducts(){
  this.httpClient
    .get<ProductResponseModel>(this.apiUrl)
    .subscribe((response)=>{
      this.products= response.data
    });
}
```

//postmanın get ile products.Getall ile aynı işlemi yapıyoruz. adresi cagırıyoruz
// gelen yanıtını eşleştiriyoruz
// eşleştirme sorunu var. Bunu yukarıda url adresi olduğu yerde tanımlamalıyız

Açıklama : Bu getProducts() metodu ne yapıyor. Tipi tanımla. Subscribe ol. Gelen Responsu karşıla

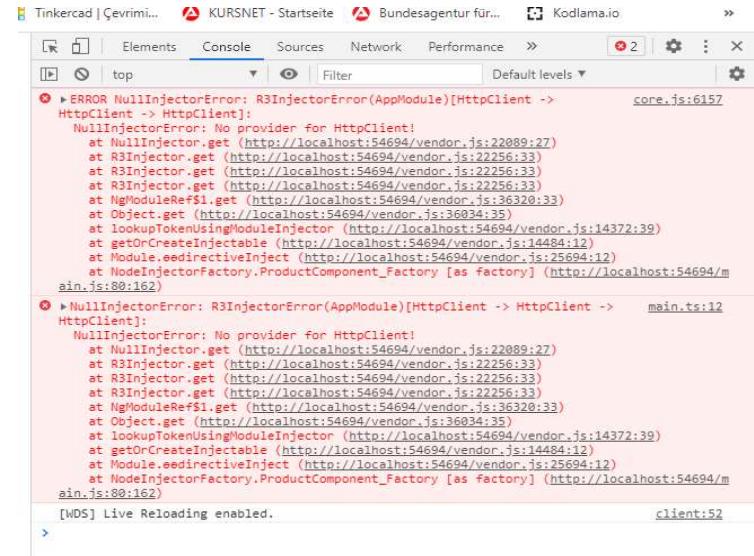
HATA: HTML 4200 acalim.Http ClientA kasilik geleln bir instance bulmadigi icin hata verdi. Burada da Backend da instance Modul benzeri tanimlama yapmaliyiz

ADIM-29: **app.module.ts** acalim. Declarations kismina girdi yapmaliyiz

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';
// backend api baglamak icin eklemeliyiz

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { ProductComponent } from './components/product/product.component';
import { CategoryComponent } from './components/category/category.component';
import { NaviComponent } from './components/navi/navi.component';

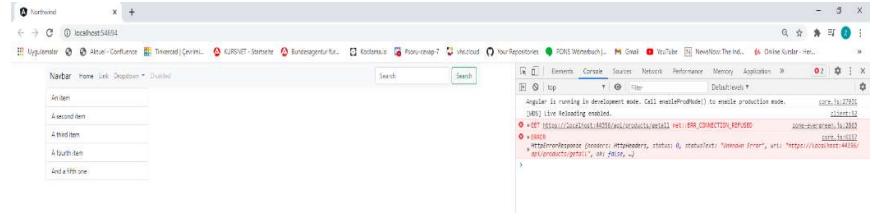
@NgModule({
  declarations: [
    AppComponent,
    ProductComponent,
    CategoryComponent,
    NaviComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```



[WDS] Live Reloading enabled.

client:52

HATA: HTML 4200 açalım. Başka bir hata verdi. Hatada “CORS” ifadesi varsa Backend içinde Frontend erişimi için Konfigürasyon yapılmadığı anlamına gelir. Backend HTML 44356 iken bu HTML 4200



Adım-30: Visual Studio'da **Backend** açalım **Startup'a açalım** Frontend erişimi için Konfigürasyon yapmalıyız

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to the container.
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddControllers();
        services.AddCors(); //Frontend'ten erişimi için Configurasyon yapmaliyiz 17.Hafta
        -----
        -----
    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        ----

        app.UseCors(builder=>builder.WithOrigins ("http://localhost:54694").AllowAnyHeader()); //Frontend'ten erişimi için Configurasyon yapmaliyiz 17.Hafta. Buradaki konumu önemli
        -----
    }
}
```

Adım 31: `product.component.html` gelelim ve sütun adlarını verelim

```
<table class ="table">
<thead>
  <th>Id</th>
  <th>Ürün Adı</th>
  <th>Kategori</th>
  <th>Fiyat</th>
  <th>Stok Adedi</th>
</thead>

<tr *ngFor="let product of products">
  <td>{{product.productId}}</td>
  <td>{{product.productName}}</td>
  <td>{{product.categoryId}}</td>
  <td>{{product.unitPrice}}</td>
  <td>{{product.unitsInStock}}</td>
</tr>
</table>
```

Koşturma (RUN): Önce Backend çalıştırıp sonra Frontend çalıştırılın.

Ürünler Web sayfasında sergilendi

Name	Id	Unit Price	Stock Level
Chai	1	18	17
Chang	2	19	13
Aniseed Syrup	3	13	20
General Tso's Chicken	4	20	50
Chef Anton's Gumbo Mix	5	21.35	0
Grandma's Boysenberry Spread	6	25	120
Uncle Bob's Organic Dried Pears	7	30	15
Mrs. Butcher's Superberry Sauce	8	40	6
Mishi Kobe Niku	9	97	29
Izla	10	31	31
Quince Paste	11	21	22
Océan Mandarine La Patate	12	30	66
Korbu	13	6	24
Tofu	14	23.25	35
Genen Shoyu	15	15.5	39
Pepperidge Farm Cookies	16	13.45	29
Alice Merton	17	39	0
Cameron Topics	18	62.5	42
Teatime Chocolate Biscuits	19	92	25
Swiss Vanillekipferl	20	45	40
Se Rosely's Gourmet	21	10	1
Gusto's Knekksekold	22	27	104
Turkish Delight	23	9	61
Rustico® Ricotta	24	45	20
Nutty Nut-Nougat-Creme	25	14	76
Gummiar Gummibärchen	26	31.23	15
Schöller Schokoäpfel	27	49.9	49
Rouleau Royal	28	40.9	26
Trappist Doublebois	29	132.75	0
Nord-Ost Mehlgerste	30	25.89	10
Gorgonzola Telino	31	123.5	9
Mademoiselle Fabiol	32	33	9
Montrachet	33	2.5	112
Sequoia Ale	34	14	111
Steiner's Saut	35	18	20
Hegel's Hegel	36	18	112
Grana Padano	37	28	71
Côte de Baye	38	203.3	37
Chartreuse verte	39	18	69
Budos Capri Melat	40	18.8	123
Almond Delight Crem Choclate	41	5.95	55
Singapuren Hokkien Fried Mee	42	14	26
Irish Coffee	43	48	17
Gochujang	44	19.45	27
Rogede Jeld	45	93	5
-----	x	x	44

Adım-32: `product.component.ts` açalım. Ortak olabilecek kodları (buradaki backend ısimi metodu)service adlı bir yere toplayıp tekrara yazmak yerine gerektiğinde oradan çağıracağız. Refactoring yapıyoruz (iyileştirme)

App içine **services** adlı yeni bir klasör oluştururalım. Sağ tıklayalım ve “open in integrated Terminal” seçerek Kod kısmına yazacağız

`ng g service product` → bu komutla birlikte `service` klasörü altında product için `product.service.spec.ts` ve `product.service.ts` adlı 2 file oluştı

product.service.ts bu sekilde tanimlanacak

```
import { Injectable } from '@angular/core';
import {HttpClient} from '@angular/common/http'; // apiye baglanip backendeki veriyi alabilmek icin manuel ekledik
import { ProductResponseModel } from 'src/app/models/productResponseModel';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class ProductService {

  apiUrl= "https://localhost:44356/api/products/getall" // burasi backend Apinin html adresi GITHUB 'dan alırsan burayı uyarlamalısın
  constructor(private httpClient:HttpClient) { }

  getProducts():Observable<ProductResponseModel> {           // observable import edilmeli
    return this.httpClient.get<ProductResponseModel>(this.apiUrl);
  }
}
```

Adım-33: product.component.ts son hali

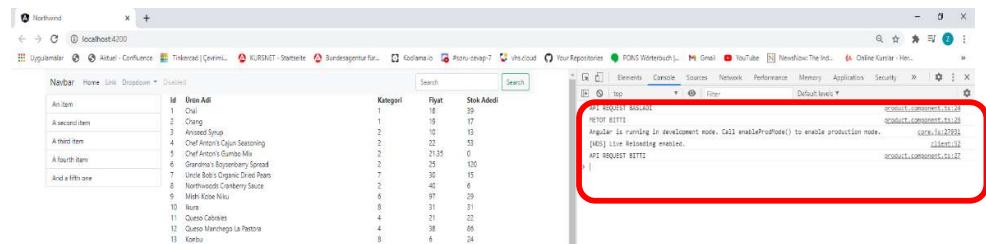
```
import { Component, OnInit } from '@angular/core';
import { ProductResponseModel } from 'src/app/models/productResponseModel';
import { Product } from 'src/app/models/product';
import { ProductService } from 'src/app/services/product.service';

@Component({
  selector: 'app-product',
  templateUrl: './product.component.html',
  styleUrls: ['./product.component.css']
})

export class ProductComponent implements OnInit {
  products:Product[] =[];

  constructor(private productService:ProductService) { } // ProductServie tanitalım ve import edelim
  ----
}
```

HTML 4200: F12 ile console acalim. Dikkat et Api başlıyor Metot bitiyor.
Api ise Daha sonra bitiyor. Es zamanlı bir çalışma. Javada durum böyledir



Adım-34: `product.component.ts` son hali. `DataLoaded` adlı değişken ekledik

import -----

```
@Component({
  selector: 'app-product',
  templateUrl: './product.component.html',
  styleUrls: ['./product.component.css']
})

export class ProductComponent implements OnInit {
  products:Product[] =[];
  dataLoaded = false; // sirali calismasi icin tanitiyoruz

  constructor(private productService:ProductService) { } // ProductService tanitalim ve import edelim

  ngOnInit(): void {
    this.getProducts();
  }

  getProducts(){

    this.productService.getProducts().subscribe(response=>{
      this.products = response.data
      this.dataLoaded=true; // bu kod sirali hale getirmek icin tanittik. Yukleme bitince
    })
  }
}
```

Adım-35: `product.component.html` son hali (en basa bootstrapden aldığımız kodu yükledik.) sadece `dataLoaded` false ise çalışması için if koşulu yazdık

Bootstrap sitésinden Component ➔ Spinner açalım. (veri yükleme tamamlanınca kadar yükleme döngü hareketi olsun istiyoruz)

```
<div *ngIf="dataLoaded==false" class="spinner-border text-primary" role="status">
  <span class="visually-hidden">Loading...</span>
</div>

<table *ngIf="dataLoaded==true" class ="table">    // ürün listesi yokken baslik görünmez
<thead>
  <th>Id</th>
  <th>Ürün Adı</th>
  <th>Kategori</th>
  <th>Fiyat</th>
  <th>Stok Adedi</th>
</thead>

<tr *ngFor="let product of products">
  <td>{{product.productId}}</td>
  <td>{{product.productName}}</td>
  <td>{{product.categoryId}}</td>
  <td>{{product.unitPrice}}</td>
  <td>{{product.unitsInStock}}</td>

</tr>
</table>
```

ADIM-36: Backend açalım. WEBAPI=>Controller icinde ProductController GetAll metodu içine bir satır komut ekleyeceğiz. 5 sn. (sistem milisaniye tanır 5000) bekleme konutu

```
[HttpGet ("getall")]
public IActionResult GetAll()
{
    //Dependency chain -- 

    Thread.Sleep(1000);           // ekledik 17.ders 1 sn uzatmak demek
    var result = _productService.GetAll();
    if (result.Success)
    {
        return Ok(result);
    }
    return BadRequest(result);
}
```

HTML 4200 belli bir süre (sonuç alan kadar) çalışıyor döngüsünde görülür ve ürünler sergilenir

NOT: Backendi olmayanlar için todo yazımı (Github'a kaydetti)

Not: Angular JS ve Angular farklıdır

18.DERS (13.03.2021)

Konu: Angular ile Frontend Tasarlama

Backend-Frontend Bağlantı kodlarında Refactoring (standartlastırma)

Secime göre ürün listeleme

Hazırlık: Backend (MyFinaProject) ve Frontend (KampFrontend) açalım ve koşturalım. Frontend için Alt komut kısmına “ng serve –open” yazacağız

Product ve Category için değerleri liste olarak döndüren ayrı ayrı Response Modeller tanımlamıştık ve her ikisi için de ortak dönen değerler (success ve mesaj) için Response Model tanımlamıştık ve onları Ortak Response Model'e inherit (extend) etmiştık. Burada Ortak değer döndüren ResponseModeller bu projeye özgüdür.

Ancak bir liste döndüren şekilde standart bir Response tanımlayabilir ve her projede kullanabileceğimiz bir yapı oluşturabiliriz. Kısacası bunları profesyonelleştirebiliriz. Böylece yeni tanımladığımız Response Model nedeniyle Product ve CategoryResponseModel'e gerek kalmayaahı için sileceğiz

Adım-37: Models sağ tıkla New File ile ListResponseModel.ts adlı bir dosya oluşturalım ve içine herhangi bir tipte (ürün, kategori, marka vb.) liste döndürecek şekilde tanımlayalım. Response Modele Ingerit etmemeyi unutmayalım ki Success ve mesaj değişkenlerini de karşılayabilelim

```
export interface ListResponseModel <T> extends ResponseModel {  
    data:T[];  
}
```

Adım-38: ProductResponseMode.ts ve CategoryResponseModel.ts 'ye gerek kalmadığı için silelim (üzerine gelip sağ tıkla)

Adım-39: `product.service.ts` adlı dosya hata verir. Çünkü orada `productResponseModel`'ı kullanmıştır: Kod içinde `ProductResponseModel` yerine `ListResponseModel` tanımlayalım. Import satırlarında da `ProductResponseModel` yerine `ListResponseModel`'ı tanımlayalım.

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http'; // apiye baglanip backendeki veriyi alabilmek icin manuel ekledik
import { Observable } from 'rxjs';
import { ListResponseModel } from '../models/ListResponseModel';
import { Product } from '../models/product';

@Injectable({
  providedIn: 'root'
})
export class ProductService {

  apiUrl= "https://localhost:44356/api/products/getall" // burasi backend Apinin html adresi
constructor(private httpClient:HttpClient) { }

getProducts():Observable<ListResponseModel<Product>> {           // observable import edilmeli
  return this.httpClient.get<ListResponseModel<Product>>(this.apiUrl);
}

}
```

Adım-40: `product.component.ts` adlı dosya hata verir. Import satırında yer alan ilgili satırı silelim

Not: Class tanımlamalarında newleme yapmalıyız ancak interfacelerde newlemeye gerek yoktur

Dikkat: Dosya isimleri küçük harfle, Class isimleri büyük harfle başlayacak .

Dosya ismi birleşik kelimelerden oluşuyorsa ilk kelime haricinde diğer kelimelerin bas harfleri büyük yazılabılır veya onlarda küçük yazılırlar ancak kelime aralarına tire (-) eklenir

Adım-41: category.component.ts Category icin import yapacağız

Adım-42: services klasörü üzerinde ağa tıklayalım ve “open in integrated Terminal” seçenek Kod kısmına yazacağız

ng g service category (skip tests) → bu komutla birlikte service klasörü altında product icin category.service.spec.ts ve category.service.ts adlı 2 file oluşturur. Spec.ts adlı uzantılı dosyaları oluşturmaz

Düzen spec.ts uzantılı dosyaları da bu projede kullanmadığımız için silebiliriz

Adım-43: category.service.ts adlı dosyayı anı product.service.ts gibi yazacağız (kodları ondan kopyalayalım)

```
import { Injectable } from '@angular/core';
import { HttpClient} from '@angular/common/http';// apiye baglanıp backendeki veriyi alabilmek icin manuel ekledik
import { Observable } from 'rxjs';
import { ListResponseModel } from '../models>ListResponseModel';
import { Category } from '../models/category';

@Injectable({
  providedIn: 'root'
})
export class CategoryService {

  apiUrl= "https://localhost:44356/api/categories/getall" // burası backend Apinin html adresi GITHUB 'dan alırsan burayı uyarlamalısın
  constructor(private httpClient:HttpClient) { }

  getCategories():Observable<ListResponseModel<Category>> {           // observable import edilmeli
    return this.httpClient.get<ListResponseModel<Category>>(this.apiUrl);
  }
}
```

Adım-44: Backend icinde CategoriesController daha önce hazırlamadık.

WebAPI=>Controller içine CategoriesController adlı bir controller ekleyelim

Injecton tanımlayacagız ve ampulden Using Business.Abstract komutunu ekleyelim

_categoryService henüz tanımlanmadı. Onun için metot yazacağınız veya ampulden “Intilialize field from Constructor” seçiliirse otomatik oluşur. Bende otomatik çıkmadı çünkü bir öğrenci email kaydıyla ücretiz kurulabilen Rescharper adlı eklenti gerekiyor



ProductsController GetAll Metotunu kopyalayalım

```
namespace WebAPI.Controllers

{
    [Route("api/[controller]")]
    [ApiController]
    public class CategoriesController : ControllerBase
    {
        ICategoryService _categoryService;
        public CategoriesController(ICategoryService categoryService)
        {
            _categoryService = categoryService;
        }

        [HttpGet("getall")]
        public IActionResult GetAll()
        {
            var result = _categoryService.GetAll();
            if (result.Success)
            {
                return Ok(result);
            }
            return BadRequest(result);
        }
    }
}
```

Adım-45: Backend icinde Business=DependencyResolvers=> Autofac=> AutofacBusinessModule adlı Classı açalım

CategoriesController icinde injection yaptığım CategoryService karşılığı olup olmadığını kontrol etmeliyiz. Daha önce tanımlamışız

Koşturma (Run): Backend calisti ve Kategorileri sergiledi

Adım-46: Frontend'e geri dönelim category.component.ts adlı dosyaya gelelim. Getcategories adlı metodu products.components.ts Den product icin olanı alalım ve uyarlayalım.

```
import { Component, OnInit } from '@angular/core';
import { Category } from 'src/app/models/category';
import { CategoryService } from 'src/app/services/category.service';

@Component({
  selector: 'app-category',
  templateUrl: './category.component.html',
  styleUrls: ['./category.component.css']
})
export class CategoryComponent implements OnInit {

  categories: Category[]=[];
  dataLoaded = false; // sirali calismasi icin tanit

  constructor(private categoryService:CategoryService) { } //private disardan ergusimi engeller

  ngOnInit(): void {
    this.getCategories(); // metodlerin calismasi icin burada cagirmaliyiz
  }

  getCategories(){

    this.categoryService.getCategories().subscribe(response=>{
      this.categories = response.data
      this.dataLoaded=true; // bu kod sirali hale getirmek icin tanittik. Yükleme bitince
    })
  }
}
```

Adım-47: category.component.html kodları yazalım

Dikkat: HTML'de "ul" liste adı "li" listenin elemanlarıdır. *ngFor içinde belirtilen liste elemanı kadar tekrarlanan for döngüsüdür

```
<ul class="list-group">
  <li *ngFor="let category of categories" class="list-group-item">{{category.categoryName}}</li>
</ul>
```

HTML 4200 artık categorileri de sergiliyor

The screenshot shows a web application interface. At the top, there's a navigation bar with links like 'Uygulamalar', 'Altınlı - Confluence', 'Ticariyet | Lync', 'BURSTTT - Stanssen', 'BurdurAnadolu TV', 'Kodlama.io', 'Avrupa Grubu 7', 'Your Resumes', 'PDDN Webdoku', 'Gmail', 'YouTube', and 'NewsNow The Ind...'. Below the navigation is a search bar with a 'Search' button.

LAYOUT: A blue callout points to the sidebar menu on the left, which contains a tree view of product categories: 'Beverages', 'Condiments', 'Confections', 'Dairy Products', 'Grains/Cereals', 'Meat/Poultry', 'Produce', and 'Seafood'. This sidebar is highlighted with a red box.

Görsel Kısım: A blue callout points to the main content area, which displays a table of products. The table has columns: 'Id', 'Ürün Adı', 'Kategori', 'Fiyat', and 'Stok Adedi'. The data includes items like 'Coca-Cola', 'Chang', 'Aniseed Syrup', 'Chef Anton's Cajun Seasoning', 'Chef Anton's Gumbo Mix', 'Ginger Paste', 'Uncle Bob's Organic Dried Pears', 'Northwoods Cranberry Sauce', 'Mishi Kobe Niku', 'Rouges', 'Queso Cabrero', 'Queso Manchego La Pastora', 'Konbu', 'Lata', 'Genen Shoyu', 'Pavlova', 'Alice Mufflers', 'Cinnamon Tops', 'Pfefferminzkoekjes', 'Sir Rodney's Marmalade', 'Sir Rodney's Scones', 'Gustaf's Knickerbock', 'Lakrids', 'Guaraná FantaStica', 'NuNuka Null-Knugget-Creme', 'Gummibärchen', 'Röslein Lebkuchen', 'Rössle Sauerkraut', 'Thüringer Rostbratenurst', 'Nord-Ost Mayonnaise', 'Scharfes Feuer Tello', 'Mexicopoppy Felicit', 'Gewlost', 'Sasquatch Ale', 'Pilsner Urquell', 'Integrit Sil', 'Greider Ilex', 'Côte de Blie', 'Côte de Blie', 'Boston Cream', 'Boston Curb Meet', 'Jack's New England Clam Chowder', 'Singaporean Hokkien Fried Mee', 'Iggy Coffee', 'Røgette Smør', 'Røgede sild', 'Spegefilet', 'Zaanse koeken', and 'Zaanse koeken'. The entire table area is highlighted with a red box.

FOOTER: A blue callout points to the footer area at the bottom of the page, which contains a 'Console' tab and a 'Who's New' tab. The 'Console' tab shows some developer logs, and the 'Who's New' tab lists recent updates.

Neyi nicin yapıyoruz: Artık HTML üzerinde Kategori secimi yapınca sadece o kategorideki ürünler sergilensin
Angular Rooting yapısından yararlanacağız

Adım-48: src→app→app.component.html dosyasını acalim

Burada görsel kisim veya layout kimi için bir ayrim henüz yok. İstene veriyi frontend programında yazarak sergiledik
Artık kullanıcının seçimine göre sergilenmesini sağlayacağız. Biz görsel kışım olarak tanımladığımız yukarıda ürünleri sabit olarak sergilediğimiz kismi bu maksatla tanımlayacağız.

Kodda en allta duran rooter-outlet ile farklı sayfalar varmış gibi sayfanın ilgili bölmesinin isten doğrultuda değişmesini sağlayan komut.
Bu rooter yapısını nerde düzenliyoruz? Biz angular ilk oluştururken rooting yapısını da eklemiştik. Tüm dosyaların içinde rooter'in yapısına ilişkin dosyalar var

```
<div class = "container">
  <app-navi></app-navi>

  <div class="row">
    <div class="col-md-3">
      <app-category></app-category>
    </div>
    <div class = "col-md-9">
      <router-outlet></router-outlet>          // önceden burada product sabır bir şekilde segilenecektir
    </div>
  </div>
</div>
```

Adım-49: app-routing.module.ts adlı dosyayı acalim

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { CategoryComponent } from './components/category/category.component';
import { ProductComponent } from './components/product/product.component';
```

```

const routes: Routes = [
  {path:"", pathMatch:"full", component:ProductComponent},    // hicbirsey yazilmazsa productcomponent sergilensin
  {path:"products", component:ProductComponent},    // HTML adina /products yazilrisa productcomponent sergilensin
  {path:"categories", component:CategoryComponent} // HTML adina /categories yazilrisa categoriescomponent sergilensin
];
// sabitler

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

pathMatch:"full" ➔ eski versiyonlarda bu komutu yazmadığımızda boş seçimle aynı olan başka bir seçim olduğunda kafası karışıyordu

Adım-50: category.component.html'yi açalım Tıklama durumunda aktivasyon tanımlamalıyız (metod çağrıma)

```

<ul class="list-group">
  <li (click)="setCurrentCategory(category)" *ngFor="let category of categories" class="list-group-item">{{category.categoryName}}</li>
</ul>

```

Adım-51: category.component.ts yi açalım Tıklama durumunda aktivasyon olacak metodu tanımlayalım

```

setCurrentCategory(category:Category){
  console.log(category.categoryName)
}

```

HTML4200'de kategorilerin üzerine tıklandığında F12 ile ekrana gelen konsolda seçilen kategori adı sergileniyor. Demek ki çalışıyor

Adım-52: category.component.ts yi açalım Tıklama durumunda aktivasyon olacak metodu tanımlayalım

```
import { Component, OnInit } from '@angular/core';
import { Category } from 'src/app/models/category';
import { CategoryService } from 'src/app/services/category.service';

@Component({
  selector: 'app-category',
  templateUrl: './category.component.html',
  styleUrls: ['./category.component.css']
})
export class CategoryComponent implements OnInit {

  categories: Category[]=[];
  dataLoaded = false; // sirali calismasi icin tanit
  currentCategory:Category={categoryId:0,categoryName:""}; //Degerleri rastgele atiyoruzki hata vermesin (intialize hatasi) veya Nota bakin

  constructor(private categoryService:CategoryService) { } //private disardan erisimi engeller

  ngOnInit(): void {
    this.getCategories(); // metodlerin calismasi icin burada cagirmaliyiz
  }

  getCategories(){

    this.categoryService.getCategories().subscribe(response=>{
      this.categories = response.data
      this.dataLoaded=true; // bu kod sirali hale getirmek icin tanittik. Yükleme bitince
    })
  }

  setCurrentCategory(category:Category){
    this.currentCategory=category;
    //console.log(category.categoryName)
  }

}

//NOT: newleme hatasini önlemek için tsconfig.json da ayar yaparız."strictPropertyInitialization":false tanımlarız
```

Adım-53: category.component.html'yi acalim. Tıklana kategori secimi belirtilsin

```
<ul class="list-group">
  <li (click)="setCurrentCategory(category)" *ngFor="let category of categories"
    class="list-group-item"><{{category.categoryName}}>/li>
  </ul>
  <h5> {{currentCategory.categoryName}} sectiniz</h5>
```

HTML 4200 artık seçilen kategori kategorilerinin altında görülmeyecek.
En başta default tanımlanmadığı için başlangıç hatası verdi. Bunu düzeltelim

```
<ul class="list-group">
  <li (click)="setCurrentCategory(category)" *ngFor="let category of categories"
    class="list-group-item"><{{category.categoryName}}>/li>
  </ul>
  <h5 *ngIf="currentCategory"> {{currentCategory.categoryName}} sectiniz</h5>
```

Aşağıdaki kodlamayla tüm kategoriler aktiv(seçili) olarak belirir. CSS değiştiriyoruz

```
<ul class="list-group">
  <li (click)="setCurrentCategory(category)" *ngFor="let category of categories"
    class="list-group-item active"><{{category.categoryName}}>/li>
  </ul>
  <h5 *ngIf="currentCategory"> {{currentCategory.categoryName}} sectiniz</h5>
```

The screenshot shows a navigation bar with links like 'Actual - Confluence', 'Tümleme | Çeviri...', 'KÜRSAYI - Statische', 'Bundesagentur für...', 'Kodlama... | #hausenverw-7', 'wheelsand...', 'Your Repositories', and 'PDF'. Below the navigation is a table titled 'Kategoriler' (Categories) with columns 'Id', 'Düzen Adı' (Display Name), 'Kategori' (Category), 'Fiyat' (Price), and 'Stock Adedi' (Stock Quantity). The table lists various food items categorized by type (Beverages, Condiments, etc.).

		Kategori	Fiyat	Stock Adedi
1	Chai	Drinks	15	39
1	Chang	Drinks	19	17
2	Apple juice Syrup	Condiments	20	28
4	Chef Anton's Cajun Seasoning	Condiments	22	5
5	Chef Anton's Gumbo Mix	Condiments	25	120
7	Uncle Bob's Organic Dried Pears	Gums/Candies	30	15
8	NutriGrain Cereal Bars	Snacks	40	6
9	Mishi Koba Namae	Snacks	45	20
10	Bonito Crab Meat	Seafood	50	21
11	Oregano	Spices/Herbs	55	23
12	Queso Manchego La Pastora	Seafood	58	24
13	Queso Cabrales	Seafood	60	24
14	Tofu	Seafood	65	23
15	Deli - Phoebe	Seafood	72	23
16	Parmesan	Seafood	75	20
17	Alice Mutton	Seafood	78	0
18	Cinnamon Toffee	Seafood	80	42
19	Tea Time Chocolate Biscuits	Snacks	82.5	25
20	Sir Rodney's Marmalade	Condiments	85	45
21	Sir Rodney's Scones	Condiments	90	104
22	Gluglug's Knobkrieb	Condiments	95	61
23	Tumbleweed	Seafood	105	20
24	Glutafon Fantastika	Seafood	115	20
25	NutriCe Nutri-Nougat-Creme	Seafood	125	78
26	Uncle Bob's Organic Green Beans	Gums/Candies	132.5	15
27	Schöpp Schoko-Küchlein	Seafood	135	49
28	Thüringer Fleischwaren	Seafood	137.5	20
29	Thüringer Fleischwaren	Seafood	152.5	0
30	Uncle Bob's Organic Green Beans	Gums/Candies	160	0
31	Lengenbach's Lebkuchen	Seafood	172.5	0
32	Glutafon Knobkrieb	Seafood	175	0
33	Glutafon Knobkrieb	Seafood	180	0
34	Glutafon Knobkrieb	Seafood	185	0
35	Glutafon Knobkrieb	Seafood	190	0
36	Côte de Boeuf	Meat/Poultry	205	17
37	Uncle Bob's Organic Green Beans	Gums/Candies	212.5	15
38	Côte de Boeuf	Meat/Poultry	215	15
39	Uncle Bob's Organic Green Beans	Gums/Candies	222.5	15
40	Boston Crab Meat	Seafood	230	123
41	Jack's New England Clam Chowder	Seafood	235	85
42	Glutafon Knobkrieb Fried Rice	Seafood	237.5	28
43	Ipan Coffee	Drinks	240	17
44	Glutafon Knobkrieb	Seafood	245	27
45	Rigatoni	Seafood	250	5
46	Spaghetti	Seafood	255	95
47	Zähne Keksen	Seafood	260	20
48	Chocolate	Seafood	262.5	15

The screenshot shows a navigation bar with links like 'Actual - Confluence', 'Tümleme | Çeviri...', 'KÜRSAYI - Statische', 'Bundesagentur für...', 'Kodlama... | #hausenverw-7', 'wheelsand...', 'Your Repositories', and 'PDF'. Below the navigation is a table titled 'Kategoriler' (Categories) with columns 'Id', 'Düzen Adı' (Display Name), 'Kategori' (Category), 'Fiyat' (Price), and 'Stock Adedi' (Stock Quantity). The table lists various food items categorized by type (Beverages, Condiments, etc.).

		Kategori	Fiyat	Stock Adedi
1	Chai	Drinks	15	39
1	Chang	Drinks	19	17
2	Apple juice Syrup	Condiments	20	28
4	Chef Anton's Cajun Seasoning	Condiments	22	5
5	Chef Anton's Gumbo Mix	Condiments	25	120
7	Uncle Bob's Organic Dried Pears	Gums/Candies	30	15
8	NutriGrain Cereal Bars	Snacks	40	6
9	Mishi Koba Namae	Snacks	45	20
10	Bonito Crab Meat	Seafood	50	21
11	Oregano	Spices/Herbs	55	23
12	Queso Manchego La Pastora	Seafood	58	24
13	Queso Cabrales	Seafood	60	24
14	Tofu	Seafood	65	23
15	Deli - Phoebe	Seafood	72	23
16	Parmesan	Seafood	75	20
17	Alice Mutton	Seafood	78	0
18	Cinnamon Toffee	Seafood	80	42
19	Tea Time Chocolate Biscuits	Snacks	82.5	25
20	Sir Rodney's Marmalade	Condiments	85	45
21	Sir Rodney's Scones	Condiments	90	104
22	Gluglug's Knobkrieb	Condiments	95	61
23	Tumbleweed	Seafood	105	20
24	Glutafon Fantastika	Seafood	115	20
25	NutriCe Nutri-Nougat-Creme	Seafood	125	78
26	Uncle Bob's Organic Green Beans	Gums/Candies	132.5	15
27	Schöpp Schoko-Küchlein	Seafood	135	49
28	Rosie Saucenrat	Seafood	145.5	26
29	Thüringer Fleischwaren	Seafood	152.5	0
30	Nord-Ost Matjeshering	Seafood	155	10
31	Glutafon Knobkrieb	Seafood	157.5	40
32	Mescarpone Fettball	Seafood	160	25
33	Glutafon Knobkrieb	Seafood	162.5	40
34	Glutafon Knobkrieb	Seafood	165	3
35	Glutafon Knobkrieb	Seafood	170	20
36	Stawberry Sult	Seafood	175	112
37	Imaqot Sult	Seafood	180	20
38	Côte de Boeuf	Meat/Poultry	205	17
39	Uncle Bob's Organic Green Beans	Gums/Candies	212.5	15
40	Boston Crab Meat	Seafood	230	123
41	Jack's New England Clam Chowder	Seafood	235	85
42	Glutafon Knobkrieb Fried Rice	Seafood	237.5	28
43	Ipan Coffee	Drinks	240	17
44	Glutafon Knobkrieb	Seafood	245	27
45	Rigatoni	Seafood	250	5
46	Spaghetti	Seafood	255	95
47	Zähne Keksen	Seafood	260	20
48	Chocolate	Seafood	262.5	15

Ancak biz sadece seçilen kategori aktiv (seçili) olacak şekilde görünmesini saglayalım .Css Class degitirelim. Metot tanimlamalıyız

Adım-54: Category.component.ts içine seçili Category aktiv gözükmeyi sağlayan metot yazalım

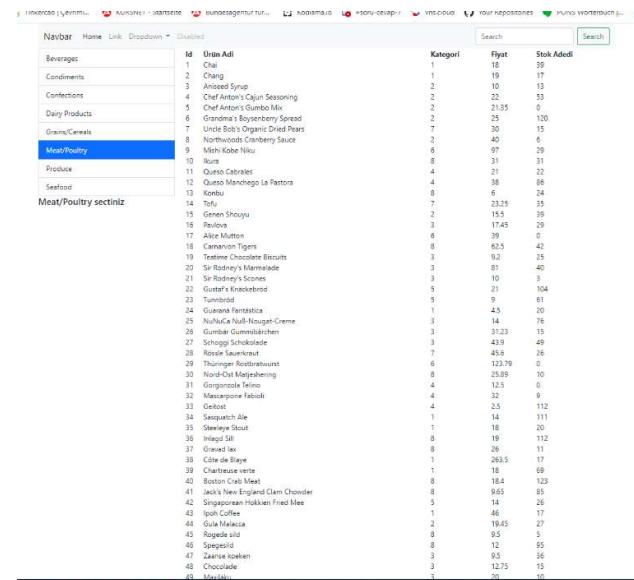
```
----  
getCurrentCategoryClass(category:Category){  
  if (category==this.currentCategory){  
    return "list-group-item active"  
  }  
  else {  
    return "list-group-item"  
  }  
}
```

Adım-55: Category.component.html içine seçili Category aktif gözükmeyi sağlayan metot çağırıyalım

```
<ul class="list-group">  
  <li (click)="setCurrentCategory(category)" *ngFor="let category of categories"  
    [class]="getCurrentCategoryClass(category)">{{category.categoryName}}</li>  
  </ul>  
  
<h5 *ngIf="currentCategory"> {{currentCategory.categoryName}} sectiniz</h5>
```

Dikkat: .class adı koseli paranteze alındı

HTML 4200 artık seçilen kategori aktiv hale geliyor



	Ürün Adı	Kategori	Flyat	Stok Adedi
1	Chai	1	18	39
2	Chang	1	19	17
3	Almond Syrup	2	10	13
4	Chef Anton's Cajun Seasoning	2	22	53
5	Chef Anton's Gumbo Mix	2	21.35	0
6	Grand Marnier® Syrup	2	25	100
7	Lundberg® Organic Dried Pears	7	20	15
8	Northeastern Cranberry Sauce	2	40	6
9	Mishi Kobe Niku	6	97	29
10	Quaker Oats	8	31	31
11	Quaker Oatbrak	4	21	22
12	Queso Manchego La Pastora	4	38	88
13	Korbu	8	6	24
14	Tofu	7	23.25	35
15	Genen Shoyu	2	20	39
16	Pavlova	3	17.45	29
17	Alice Mutton	6	39	0
18	Canneton Tigret	8	65.5	42
19	Grand Marnier® Biscuits	3	24	25
20	Sir Rodney's Marmalade	3	81	40
21	Sir Rodney's Scones	3	10	3
22	Gustaf's Knäckebrot	5	21	104
23	Pringles	5	9	81
24	Guarana Fantaşika	1	4.5	20
25	Nutella® Nut-Hazelnut-Creme	3	14	76
26	Gumba Grilledhähnchen	3	31.23	15
27	Spicy Chicken Wings	3	24	49
28	Rosée Saerkraut	7	45.8	26
29	Thüringer Rostbratenwurst	6	123.79	0
30	Nord-Ost Marzipanring	8	25.89	10
31	Corporate Térino	4	15.5	120
32	Montezuma® Feudal	4	1	9
33	Geitzer	4	2.5	112
34	Seasquash Ale	1	14	111
35	Steinweg Roulé	1	18	20
36	Apple Sild	8	19	112
37	General Tea	8	26	11
38	Côte de Biaye	1	263.5	17
39	Chartreuse Verte	1	18	69
40	Almond Choc Melt	8	14	120
41	Jack's New England Clam Chowder	8	9.65	85
42	Singaporean Hokkien Fried Mee	5	14	26
43	Ipoh Coffee	1	46	17
44	Ukrop's Pickles	2	10.25	27
45	Røgeblad Sild	8	9.5	5
46	Spegeplad	8	12	95
47	Zaanse koeken	3	9.5	36
48	Chocolate	3	12.75	15
49	Yogurt	1	20	17

Adım-56: Category.component.html içine seçili Category aktiv gözükmesinin yanında görsel kısımda sadece seçilen categorydeki ürünlerin sergilenebilmesini sağlayalım

```
<ul class="list-group">
  <li (click)="setCurrentCategory(category)"
    routerLink="/products/category/{{category.categoryId}}"
    *ngFor="let category of categories"
    [class]="getCurrentCategoryClass(category)">{{category.categoryName}}</li>
</ul>

<h5 *ngIf="currentCategory"> {{currentCategory.categoryName}} sectiniz</h5>
```

Burası Tıklandığında aktive olması gereken değişkeni tanımladı

Adım-57: Geriye Route 'u aktive etmek kaldı. Yani sadece aktive edilen Kategorideki router linke istinaden ürünler veri tabanından alınacak ve görsel kısımda sergilenecek

App-routing.module.ts acalim

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { CategoryComponent } from './components/category/category.component';
import { ProductComponent } from './components/product/product.component';

const routes: Routes = [
  {path:"",pathMatch:"full", component:ProductComponent},    // hicbirsey yazilmazsa productcomponent sergilensin
  {path:"products", component:ProductComponent},    // HTML adina /products yazilrisa productcomponent sergilensin
  {path:"products/category/:categoryId", component:ProductComponent} // secilen kategorideki ürünler sergilenecek

];    // sabitler

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Adım-58: Backend'i açalım ProductController içine gerekli kodu yazacağımız- Niyet güdümlü programlama yaparız. GetbyCategoryId adlı metodу CategoryService ve CategoryManager içinde tanımlamış olsaydık yapacaktık ancak zaten var

```
[HttpGet("getbycategory")]
public IActionResult GetByCategory(int categoryId)
{
    var result = _productService.GetAllByCategoryId(categoryId);
    if (result.Success)
    {
        return Ok(result);
    }
    return BadRequest(result);
}
```

Koşturma (Run) : Backend kosturusak yazdığımız metodun çalıştığını görürüz
Bunun için Host'a bu kodu yazarsınız => https://localhost:44356/api/products/getbycategory?categoryId=1
Ya da Postman Get kısmında yazarsınız

Adım-59: Backend'e kategori seçimine göre görüntüleyen metodу Frontend'de karşılaşacağız

Burada linkin değişmeyen ilk kısmını biracakacağız. Diğer devam kısmını metoda özgü tanımlayacağız

Sevices → product.service.ts açalım

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http'; // apiye bağlanıp backendeki veriyi alabilmek için manuel ekledik
import { Observable } from 'rxjs';
import { ListResponseModel } from '../models/ListResponseModel';
import { Product } from '../models/product';

@Injectable({
  providedIn: 'root'
```

```

})
export class ProductService {

apiUrl= "https://localhost:44356/api/" // burasi backend Apinin html adresinin degismeyen kismi
constructor(private httpClient:HttpClient) { }

getProducts():Observable<ListResponseModel<Product>> {           // observable import edilmeli
  let newPath = this.apiUrl+ "products/getall" // apiUrl standart kismi yukarıda kaldi. metoda ögür son kismini tanimlayacagiz
  return this.httpClient.get<ListResponseModel<Product>>(newPath);
}

getProductsByCategory(categoryId:number):Observable<ListResponseModel<Product>> {           // observable import edilmeli
  let newPath = this.apiUrl+ "products/getbycategory?categoryId="+categoryId // selected kategoriye özgü url tanimlama
  return this.httpClient.get<ListResponseModel<Product>>(newPath);
}
}

```

Not: **let** keyword: fonksiyon icersinden degiskekn tanımlamak için kullanılır

Adım-60: APP ➔ Component=> Products ➔ product.component.ts

```

import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { Product } from 'src/app/models/product';
import { ProductService } from 'src/app/services/product.service';

-----

export class ProductComponent implements OnInit {
  products:Product[] =[];
  dataLoaded = false; // sirali calismasi icin tanitiyoruz

  constructor(private productService:ProductService,           // ProductService tanitalim ve import edelim
             private activatedRoute:ActivatedRoute) { }      // hazir bir framework kullaniyoruz

```

```

ngOnInit(): void {
  this.activatedRoute.params.subscribe(params=>{
    if (params["categoryId"]){
      this.getProductsByCategory(params["categoryId"]);
    }
    else{
      this.getProducts();
    }
  }) //kategori secimi olup olmadığı sorgulamak için
}

getProducts(){

this.productService.getProducts().subscribe(response=>{
  this.products = response.data
  this.dataLoaded=true; // bu kod sıralı hale getirmek için tanıttık. Yükleme bitince
})

getProductsByCategory(categoryId:number){
  this.productService.getProductsByCategory(categoryId).subscribe(response=>{
  this.products = response.data
  this.dataLoaded=true; // bu kod sıralı hale getirmek için tanıttık. Yükleme bitince
})
}
}

```

Observable tanımlamalarda subscribe olarak erişebiliriz. Burada Kategori secimi olup olmadığını sorgulamak için kullandık

Not: Componentler dataları yönetmekte ilgilenir, dataları HTML kullanır

Neyi niçin yapıyoruz

Secilen Category'eki ürünlerini görüntüledikten sonra tekrar tüm ürünleri görüntülemek istersek; GetcurrentCategoryClass gibi kategori secimi bossa tüm ürünler döndürebilir. Bir router link tanımlarız ve yeni classımızı bu router ile set ederiz.

Adım-61: APP → Component=> Products→ Category.component.ts

```
getAllCategoryClass(){
  if (!this.currentCategory){
    return "list-group-item active"
  }
  else {
    return "list-group-item"
  }
}
```

Adım-62: APP → Component=> Category→category.component.html

```
<ul class="list-group">
  <li [class]="getAllCategoryClass()" routerLink = "/products">Tüm Ürünler</li>
  <li (click)="(category setCurrentCategory)"
    routerLink="/products/category/{{category.categoryId}}"
    *ngFor="let category of categories"
    [class]="getCurrentCategoryClass(category)">{{category.categoryName}}</li>
</ul>

<h5 *ngIf="currentCategory"> {{currentCategory.categoryName}} sectiniz</h5>
```

Simdi de Tüm ürünlerin görüntülenmesi seçildiği zaman Current Category temizlenmelidir

Eklenti : Herhangi bir terminalde aşağıdaki kodla jquery kurulumu yapalım . Navbardaki acılım sekmeleri çalışır hale gelebilmesi için. Bir sonraki derste sepeṭe ürün ekleme ve silme işlemleri için navbara bir kısım kullanacağız.

npm install jquery ➔

Adım-63: **Angular.json** içinde scripts kısmına aşağıdaki eklemeyi yapalım
Burası projenin genelinde geçerli ayarlamaları yaptığımız kısımdır. Mode modules'in jquery implementationunu eklemiş olduk

```
"scripts": [  
    "./node_modules/jquery/dist/jquery.min.js",  
    "./node_modules/bootstrap/dist/js/bootstrap.min.js"  
]
```

Bu tür eklemelerde proje yeniden başlatılmalıdır

19.DERS (17.03.2021)

Konu: Angular ile Frontend Hazırlama

PIPES (veriyi farklı görüntüleme işlemi)

Sepete ürün ekleme ve silme

TANIM: Pipes elimizdeki veriyi görsel olarak daha farklı görüntülemek için kullanılır. Sergilemek maksatlı olduğu için .Html içinde kullanılır

Adım-64: App altına Pipes adlı klasör oluştur. Pipes sağ tıkla [“Open in integrated terminal”](#)

ng g pipe vatAdded → terminal içine bu kodu yazınca pipes içinde vatAdded adlı yapıyı kuruyor. Gerekli dosyaları oluşturuyor.

Not: Projemizdespec.ts uzantılı dosyaları silebiliriz. İsimiz yok

Adım-65: products.component.html 'ye veri tabanından alınan fiyat bilgisini kullanarak KDV'li fiyatını sergileyen yeni bir sütün ekleyelim

```
<div *ngIf="dataLoaded==false" class="spinner-border text-primary" role="status">
  <span class="visually-hidden">Loading...</span>
</div>

<table *ngIf="dataLoaded==true" class ="table">
<thead>
  <th>Id</th>
  <th>Ürün Adı</th>
  <th>Kategori</th>
  <th>Fiyat</th>
  <th>KDV'li Fiyat</th>    //veri tabanından gelen fiyat bilgisi üzerinden KDV'li fiyatı hesaplayıp ilaveten görüntüleyeceğiz
  <th>Stok Adedi</th>
</thead>
<tbody>

  <tr *ngFor="let product of products">
```

```

<td>{{product.productId}}</td>
<td>{{product.productName | uppercase}}</td> // büyük harfle görüntülenir
<td>{{product.categoryId | lowercase}}</td> // küçük harfle görüntülenir
<td>{{product.unitPrice}}</td>
<td>{{product.unitPrice | vatAdded:18 | currency}}</td> // ilave bilgi
<td>{{product.unitsInStock}}</td>

</tr>
</tbody>
</table>

```

Adım-66: vat-added.pipe.ts

```

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'vatAdded'
})
export class VatAddedPipe implements PipeTransform {

  transform(value: number, rate:number): number {
    return value+(value*rate/100);
  }
}

```

pipe sablonu açıklaması

transform(value: giren veri tipi, gönderdiğimiz parametreler dizi şeklindedir): çıkan veri tipi {
 return deger; }

	Kategori	Fiyat	Stok Adedi
1	[product.productName uppercase]	\$12.50	39
2	[product.productName uppercase]	\$24.00	17
3	[product.productName uppercase]	\$11.80	13
4	[product.productName uppercase]	\$13.95	53
5	[product.productName uppercase]	\$2.50	9
6	[product.productName uppercase]	\$39.50	100
7	[product.productName uppercase]	\$35.40	15
8	[product.productName uppercase]	\$47.20	8
9	[product.productName uppercase]	\$14.66	29
10	[product.productName uppercase]	\$36.58	31
11	[product.productName uppercase]	\$24.78	22
12	[product.productName uppercase]	\$44.84	66
13	[product.productName uppercase]	\$7.50	24
14	[product.productName uppercase]	\$23.25	35
15	[product.productName uppercase]	\$18.29	39
16	[product.productName uppercase]	\$30.59	29
17	[product.productName uppercase]	\$17.45	10
18	[product.productName uppercase]	\$30.00	0
19	[product.productName uppercase]	\$25.75	42
20	[product.productName uppercase]	\$10.88	25
21	[product.productName uppercase]	\$95.58	40
22	[product.productName uppercase]	\$10.00	3
23	[product.productName uppercase]	\$24.78	104
24	[product.productName uppercase]	\$10.62	51
25	[product.productName uppercase]	\$3.31	20
26	[product.productName uppercase]	\$30.50	78
27	[product.productName uppercase]	\$38.85	15
28	[product.productName uppercase]	\$31.00	49
29	[product.productName uppercase]	\$45.60	26
30	[product.productName uppercase]	\$12.50	9
31	[product.productName uppercase]	\$35.55	10
32	[product.productName uppercase]	\$14.75	0
33	[product.productName uppercase]	\$27.76	9
34	[product.productName uppercase]	\$1.95	112
35	[product.productName uppercase]	\$12.50	111
36	[product.productName uppercase]	\$37.24	20
37	[product.productName uppercase]	\$22.42	112
38	[product.productName uppercase]	\$30.68	11
39	[product.productName uppercase]	\$20.50	17
40	[product.productName uppercase]	\$37.24	69
41	[product.productName uppercase]	\$21.71	123
42	[product.productName uppercase]	\$11.39	65
43	[product.productName uppercase]	\$12.50	26
44	[product.productName uppercase]	\$34.28	17
45	[product.productName uppercase]	\$22.95	27
46	[product.productName uppercase]	\$11.21	5
47	[product.productName uppercase]	\$14.75	95

Adım-67: **Product.component.html** içine Getbootstraps sitesinden forms → form control içinden ilk div kopyalayalım
ilk <div> </div>'den sonraya yapıştırıyalım ve uyarlayalım

```
<div class="mb-3">
  <label for="filterText" class="form-label">Ürün ara</label>
  <input type="text" [(ngModel)] class="form-control" id="filterText" placeholder="arama ifadesi giriniz">
</div>

<div *ngIf ="filterText" class= "alert alert-success"> // filter text boossa görünmez arama yaoilirsa arama kelimeleri HTML'de yeşil alan
  içinde görünür
    {{filterText}} aradınız
</div>
```

Açıklama: <input type="text" [(ngModel)] → banana(muz) model demek Filtertext değişkeni ile etkileşim halinde demektir. HTML'DEN veya buradan bu değer degistriginde dieri de etkilenir

Adım-68: Product.Component.ts

```
export class ProductComponent implements OnInit {
  products:Product[] =[];
  dataLoaded = false; // sirali calismasi icin tanitiyoruz
  filterText="" ; // girilen arama texti ile placehoder ile bunu ilişkilendirmeliyiz
```

Ürünler					
ID	Ürün Adı	Kategori	Fiyat	KDV'lı Fiyat	Stok Adedi
1	[product.productName uppercase]	18	\$11.43	\$11.43	39
2	[product.productName uppercase]	19	\$32.43	\$32.43	17
3	[product.productName uppercase]	20	\$11.80	\$11.80	13
4	[product.productName uppercase]	22	\$25.98	\$25.98	53
5	[product.productName uppercase]	2	\$21.35	\$25.19	0
6	[product.productName uppercase]	23	\$38.50	\$38.50	120
7	[product.productName uppercase]	30	\$35.40	\$35.40	19
8	[product.productName uppercase]	40	\$47.20	\$47.20	6
9	[product.productName uppercase]	97	\$14.46	\$14.46	29
10	[product.productName uppercase]	8	\$34.58	\$34.58	31
11	[product.productName uppercase]	31	\$24.78	\$24.78	22
12	[product.productName uppercase]	21	\$24.78	\$24.78	86
13	[product.productName uppercase]	4	\$44.84	\$44.84	86
14	[product.productName uppercase]	8	\$7.08	\$7.08	24
15	[product.productName uppercase]	7	\$33.44	\$33.44	39
16	[product.productName uppercase]	2	\$18.29	\$18.29	39
	[product.productName uppercase]	3	\$17.45	\$20.59	29

Adım-69: app.module.ts içine [(ngmodel)] için import satırı eklemeliyiz (yükarıdaki hata gider)

```
import{FormsModule} from "@angular/forms"

imports: [
  BrowserModule,
  AppRoutingModule,
  HttpClientModule, // // backend api baglamak icin eklemeliyiz
  FormsModule
],
```

Adım-70: App ➔ pipes terminal kısmına (2:cmd) kod yazarak dosya oluşturacağız

Ng g pipe filterPipe ➔ filterPipe adlı dosya oluşturacağız

Adım-71: filter.pipe.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';
import { Product } from '../models/product';

@Pipe({
  name: 'filterPipe'
})
export class FilterPipePipe implements PipeTransform {

  transform(value: Product[], filterText: string): Product[] {
    filterText = filterText?filterText.toLocaleLowerCase():""
    return filterText?value
      .filter((p:Product)=>p.productName.toLocaleLowerCase().indexOf(filterText)!==-1)
      :value;
  }
}

// tüm ürünleri foreach gibi tek tek gezer ve tüm ürün adlarını küçük harfe çevirir ve kontrol eder. Eşleşeni alır, yoksa değer geri döner
```

Açıklama: filterText var mı, yok mu kontrolü yaparak (soru işaretü) kod tüm karakterleri küçük harfe çevirir. Yoksa null değeri atar

```
filterText = filterText?filterText.toLocaleLowerCase():"null"
```

Adım-72: **Product.component.html**

```
<tr *ngFor="let product of products | filterPipe:filterText">
  <td>{{product.productId}}</td>
  <td>{{product.productName | uppercase}}</td>
  <td>{{product.categoryId}}</td>
  <td>{{product.unitPrice}}</td>
  <td>{{product.unitPrice|vatAdded:18 | currency }}</td>
  <td>{{product.unitsInStock}}</td>
</tr>
```

Adım-73: **Product.component.html**

```
<table *ngIf="dataLoaded==true" class ="table">
<thead>
  <th>Id</th>
  <th>Ürün Adı</th>
  <th>Kategori</th>
  <th>Fiyat</th>
  <th>KDV'li Fiyat</th>
  <th>Stok Adedi</th>
  <th></th>
</thead>
<tbody>
<tr *ngFor="let product of products | filterPipe:filterText">
  <td>{{product.productId}}</td>
  <td>{{product.productName | uppercase}}</td>
  <td>{{product.categoryId}}</td>
  <td>{{product.unitPrice}}</td>
  <td>{{product.unitPrice|vatAdded:18 | currency }}</td>
  <td>{{product.unitsInStock}}</td>
  <td><button (click)="addToCart(product)" type="button" class ="btn btn-success">Sepete Ekle</button></td>
</tr>
</tbody>
</table>
```

Adım-74: `Product.component.ts` Metod tanımlayalım. Daha sonra tam olarak tanımlayacağız

```
addToCart(product:Product){  
}
```

Eklenti :Ana terminal haricinde başka bir terminale bu kodu yazıp 2 eklienti kurulumu yapalım

```
npm install ngx-toastr
```

```
npm install @angular/animations
```

Adım-75: `angular.json styles` içine kod ekleyelim

```
"styles": [  
    "./node_modules/bootstrap/dist/css/bootstrap.min.css",  
    "./node_modules/ngx-toastr/toastr.css", // ekledik 19.ders  
    "src/styles.css"  
,
```

Adım-76: `App.module.ts` içine

```
import{BrowserAnimationsModule} from "@angular/platform-browser/animations" // ekledik 19.ders  
import {ToastrModule} from "ngx-toastr"; // ekledik 19.ders  
  
imports: [  
    BrowserModule,  
    AppRoutingModule,  
    HttpClientModule, // // backend api baglamak için eklemeliyiz  
    BrowserAnimationsModule,  
    ToastrModule.forRoot({ // ekledik 19.ders  
        positionClass:"toast-bottom-right"  
    }),  
    FormsModule  
,
```

Adım-77: Product.component.ts

```
constructor(private productService:ProductService,           // ProductService tanıtalım ve import edelim
            private activatedRoute:ActivatedRoute, private toastrService:ToastrService) { }           // hazır bir framework kullanıyoruz

---  
  
addToCart(product:Product){
  this.toastrService.success("Sepete eklendi",product.productName)
}
```

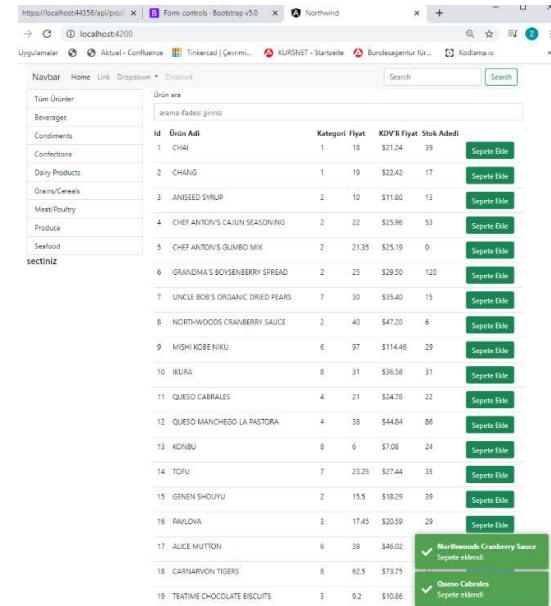
NEYI NICIN YAPIYORUZ

HMTL 4200'de ürünün yanında sepete ekle butonuna basınca ürünü sepete eklendiği bilgisini veriyor.

Artık seçilen bilgileri gerçekten sepete ekleyelim. Bunu 2 türlü yapılabılır. Birincisi client (ilgili HTML'de) saklamak, ikincisi veri tabanına eklemek. Veri tabanına ekleyince başka cihazlardan aynı hesaba erişebilmek için satıcı işlem yapabilmesi için kullanabilir

Bir ürün için 2 defa sepete eklenince ürün sayısı 2 olarak sepete eklenir

Önce sepet elemanı oluşturalım. Ürün adı ve adedi olacak şekilde



Id	Ürün Adı	Kategori	Fiyat	KDV'lı Fiyat	Stok Adedi	Sepete Ekle
1	CHAI	1	18	\$21.24	39	Sepete Ekle
2	CHANG	1	19	\$22.40	17	Sepete Ekle
3	ANISEED SYRUP	2	10	\$11.00	13	Sepete Ekle
4	CHEF ANTON'S CAJUN SEASONING	2	22	\$25.98	53	Sepete Ekle
5	CHEF ANTON'S GUMBO MIX	2	21.35	\$25.19	0	Sepete Ekle
6	GRANDMA'S BOYSENBERRY SPREAD	2	25	\$29.50	120	Sepete Ekle
7	UNCLE BOB'S ORGANIC DRIED PEARS	7	30	\$35.40	15	Sepete Ekle
8	NORTHWOODS CRANBERRY SAUCE	2	40	\$47.20	6	Sepete Ekle
9	MISHI KOBE NIJU	6	97	\$114.46	29	Sepete Ekle
10	IKURA	8	31	\$36.58	31	Sepete Ekle
11	QUESO CABRALES	4	21	\$24.70	22	Sepete Ekle
12	QUESO MANCHEGO LA PASTORA	4	38	\$44.84	86	Sepete Ekle
13	KONBU	8	6	\$7.08	24	Sepete Ekle
14	TOFU	7	23.25	\$27.44	35	Sepete Ekle
15	GENEN SHOUYU	2	15.5	\$18.29	39	Sepete Ekle
16	PAVLOVA	3	17.45	\$20.59	29	Sepete Ekle
17	AUICE MUTTON	6	39	\$46.02	✓ Northwoods Cranberry Sauce Sepete eklendi	
18	CARNARVON TIGERS	8	62.5	\$73.75	✓ Queso Cabrales Sepete eklendi	
19	TEATIME CHOCOLATE BISCUITS	3	9.2	\$10.88		

Adım-78: Models. New file ile cartitem.ts adı ver CartItem yazınca alta çıkanlardan secince import satırı otomatik eklenir

```
import{Product} from "./product";  
  
export class CartItem{  
  product:Product;  
  quantity:Number;  
}
```

Adım-79: Models sağ tıkla **New file** ile **cartItems.ts** (dikkat "s" takılı cogul ad) adı ver. Kod yazınca CartItem yazınca alta çıkanlardan secince import satırı otomatik eklenir

```
import { CartItem } from "./cartItem";
export const CartItems:CartItem []=[];
```

Adım-80: components sağ tıkla “**open in integrated terminal**” seç ve aşağıdaki kodla cart-summary'nin ilgili dosyalarını kur

Ng g component cart-summary→

Adım-81: Navi.component.html içinde ki dropdown sekmesi için olan kodu kesip **cart-summary.component.html** içine kopyala. Yerine de aşağıdaki kodu yaz

```
<app-cart-summary>
</app-cart-summary>
```

Adım-82: **cart-summary.component.html** içinde kodları uyarlayalım. Ayrıca Getbootstraps sitesinden components→ Badge içinden ilk div kopyalayalım

```
<li *ngIf="cartItems.length>0" class="nav-item dropdown">
  <a class="nav-link dropdown-toggle" id="navbarDropdown" role="button" data-bs-toggle="dropdown" aria-expanded="false">
    Sepetiniz
  </a>
  <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
    <li *ngFor="let cartItem of cartItems"> <a
      class="dropdown-item"> <span class="badge bg-secondary">{{cartItem.quantity}} </span>{{cartItem.product.productName}}</a></li>
    <li><hr class="dropdown-divider"></li>
    <li><a class="dropdown-item">Sepete git</a></li>
  </ul>
</li>
```

ADIM: cart-summary.component.ts

```
import { Component, OnInit } from '@angular/core';
import { CartItem } from 'src/app/models/cartItem';

@Component({
  selector: 'app-cart-summary',
  templateUrl: './cart-summary.component.html',
  styleUrls: ['./cart-summary.component.css']
})
export class CartSummaryComponent implements OnInit {
  cartItems:CartItem[]=[]; // 19.ders
  constructor() { }

  ngOnInit(): void {
  }
}
```

ADIM: Product.component.ts

```
.....
import { ToastrService } from 'ngx-toastr';
import { CartService } from 'src/app/services/cart.service';

-----
// girilen arama texti ile placehoder ile bunu iliskilendirmeliyiz
constructor(private productService:ProductService,
  private activatedRoute:ActivatedRoute,
  private toastrService:ToastrService,
  private cartService:CartService ) {} // hazir framework
.....
addToCart(product:Product){
  this.toastrService.success("Sepete eklendi",product.productName)
  this.cartService.addToCart(product);
}
}
```

ADIM: services sağ tıkla “open in integrated terminal” sec ve service altına aşağıdaki kodla cart için ilgili dosyaları oluştur.

ng g service cart ➔

cart.service.ts

```
import { Injectable } from '@angular/core';
import { CartItem } from '../models/cartItem';
import { CartItems } from '../models/cartItems';
import { Product } from '../models/product';

@Injectable({
  providedIn: 'root'
})
export class CartService {

  constructor() { }

  addToCart(product:Product){

    let item =CartItems.find(c=>c.product.productId==product.productId);
    if (item){ // ürünü nepette ise miktarı 1 artır
      item.quantity+=1;
    }else{
      let cartItem = new CartItem();
      cartItem.product = product;
      cartItem.quantity=1;
      CartItems.push(cartItem);
    }
  }

  list():CartItem[]{
    return CartItems;
  }
}
```

Adım-83: Cart-summary.component.ts

```
import { Component, OnInit } from '@angular/core';
import { CartItem } from 'src/app/models/cartitem';
import { CartService } from 'src/app/services/cart.service';

@Component({
  selector: 'app-cart-summary',
  templateUrl: './cart-summary.component.html',
  styleUrls: ['./cart-summary.component.css']
})
export class CartSummaryComponent implements OnInit {

  cartItems:CartItem[]=[]; // 19.ders
  constructor(private cartService:CartService) { }

  ngOnInit(): void {
    this.getCart();
  }

  getCart(){
    this.cartItems = this.cartService.list();
  }
}
```

Adım-84: cartService.ts ekleme yapalım Sepete eklediğimiz bir ürünü silme işlemi yapalım.

```
-----
removeFromCart(product:Product){
  let item = CartItems.find(c=>c.product.productId==product.productId);
  CartItems.splice(CartItems.indexOf(item),1);
}
-----
```

Adım-85: Tsconfig.json içine aşağıdaki satır ekleyelim ki yukarıda item tipine hata vermesin.

```
"strictNullChecks": false, // removeFromCart metodik kapsamında ekledik ekledik.
```

Adım-86: Cart.summary.componenet.ts içine sepetteki istenen ürünü silen metodu ekleyelim.

```
import { Component, OnInit } from '@angular/core';
import { CartItem } from 'src/app/models/cartItem';
import { Product } from 'src/app/models/product';
import { CartService } from 'src/app/services/cart.service';

@Component({
  selector: 'app-cart-summary',
  templateUrl: './cart-summary.component.html',
  styleUrls: ['./cart-summary.component.css']
})
export class CartSummaryComponent implements OnInit {

  cartItems:CartItem[]=[]; // 19.ders
  constructor(private cartService:CartService, private toastrService:ToastrService) { }

  ngOnInit(): void {
    this.getCart();
  }

  getCart(){
    this.cartItems = this.cartService.list();
  }

  removeFromCart(product:Product){
    this.cartService.removeFromCart(product);
    this.toastrService.error("Silindi",product.productName + " sepeten silindi")
  }
}
```

Adım-87: cart-summary-component.html içinde sil butonu tanımladığımız yerde silme metodunu çağıralım ve gerçekten sepetten sildirelim.

```
<li *ngIf="cartItems.length>0" class="nav-item dropdown">
  <a class="nav-link dropdown-toggle" id="navbarDropdown" role="button" data-bs-toggle="dropdown" aria-expanded="false">
    Sepetiniz
  </a>
  <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
    <li *ngFor="let cartItem of cartItems"> <a
      class="dropdown-item">
      <span class="badge bg-
secondary">{{cartItem.quantity}} </span>{{cartItem.product.productName}} <span (click) ="removeFromCart (cartItem.product)" class="badge bg-
secondary">Sil</span></a></li>
    <li><hr class="dropdown-divider"></li>
    <li><a class="dropdown-item">Sepete git</a></li>
  </ul>
</li>
```

HTML 4200 seçilen ürünleri sepete ekleme ve sepetten silme işlemlerini başarıyla gerçekleştiriyor.

Angular içinde tanımlı (built-in) pipe'lar

Elimizdeki datayı bazen olduğu haliyle değil, değiştirerek kullanmayı tercih ederiz. Yani veriyi farklı şekillerde göstermek isteriz. Bunun için pipe'ları kullanınız. Pipe'lar " |pipeName " şeklinde kullanılır. Angular içinde tanımlı olan pipe'ler olduğu gibi, kendimiz de pipe yazabiliriz. Kendimiz yazdığımız pipe'lar içerisinde parametre gönderebiliriz. Bunu da " |pipeName:parameter1 " şeklinde kullanırız. Birden fazla pipe'ı yan yana kullanabiliriz. Angular içinde tanımlı olan pipe'lar aşağıdaki gibidir.

TitleCasePipe: Verilen string değerin başlık formatı şeklinde gösterilmesini sağlar.

AsyncPipe: Observable veya Promiselerin yayındığı son değerin gösterilmesini sağlar.

JsonPipe: Verilen objenin arayüzde json yapısında gösterilmesini sağlar.

UpperCasePipe: Verilen string değerin tamamının büyük harfle gösterilmesini sağlar.

LowerCasePipe: Verilen string değerin tamamının küçük harfle gösterilmesini sağlar.

DatePipe: Verilen tarih değerinin, gönderilen parametrelere göre gösterilmesini sağlar.

CurrencyPipe: Verilen sayı değerinin para birimi olarak gösterilmesini sağlar.

DecimalPipe: Verilen sayı değerinin ondalıklı gösterilmesini sağlar.

KeyValuePipe: Object veya Maplerin ngFor ile dönülmesini sağlar.

PercentPipe: Verilen sayı değerinin yüzde biçiminde gösterilmesini sağlar.

JavaScript map, filter, reduce fonksiyonları

MAP FONKSIYONU Bir dizinin her elemanına aynı işi yapmak istediğimiz zaman kullanırız. Verilen elemanlara istenilen fonksiyon uygulanır ve yeni bir dizi elde edilir.

FILTER FONKSIYONU Bir dizinin her elemanını verilen mantıksal şart ile istenilen özelliklere uygunluğunu kontrol eder, bu şartları sağlayan elemanlardan oluşan yeni bir dizi oluşturur ve diziyi geri döndürür.

REDUCE FONKSIYONU Bir dizinin her elemanı için uygulanan fonksiyondan tek bir sonuç almak için kullanılır. Örneğin dizinin içinde en büyük sayıyı bulma, dizinin elemanlarının toplamını bulma gibi.

Konu: Reactive Forms

Tanım: Ürün-Kategori eklemek vb işlemleri yapmak için kullanıyoruz. HTML tabanlı ve Angular tabanlı 2 türde hazırlanabilir, Su an Angular daha kolay ve güncel

Adım-88: `src→app→Components` sağ tıkla “open in integrated Terminal” ile açılan Terminale
`ng g component product-add→` koduya gerekli dosyaları oluşturalım

Adım-89: `src→app→models→app.modules.ts` açalım İmport satırına ve import modülüne Reactive Formların çalışması için koda yazalım

```
import {FormsModule, ReactiveFormsModule} from "@angular/forms" // eklemeliyim
...
imports: [
  BrowserModule,
  AppRoutingModule,
  HttpClientModule,
  FormsModule,
  BrowserAnimationsModule,
  ReactiveFormsModule, // eklemeliyim
  ToastrModule.forRoot({
    positionClass:"toast-bottom-right"
})
```

Adım-90: src → app → Components → product-add.components.ts açalım. Import satırına ekleme yapalım

FormBuilder : Reaktiv Formun servisidir. Bir Form HTML ile burada yazdığımız kodları Componentlerle eşleştirdiğimiz kısım

Backend içinde Fluentvalidation ile kuralları yazıyorduk ve ilgili nesnelerle eşleştiriyorduk. Benzer işlem

Peki neden bir daha Frontend içinde yapıyoruz. Frontend opsiyoneldir. Ancak Kullanıcı rahatlığı için önemlidir

FormControl: Kullanıcının değer girebileceği Formlar

```
import { Component, OnInit } from '@angular/core';
import { FormGroup, FormBuilder, FormControl, Validators } from "@angular/forms"      // eklemeliyim
import { ToastrService } from 'ngx-toastr';
import { ProductService } from 'src/app/services/product.service';

@Component({
  selector: 'app-product-add',
  templateUrl: './product-add.component.html',
  styleUrls: ['./product-add.component.css']
})
export class ProductAddComponent implements OnInit {

  productAddForm : FormGroup;                                //FormControl için eklemeliyim
  constructor(private formBuilder:FormBuilder) { }           // FormBuilder için eklemeliyim

  ngOnInit(): void {
    this.createProductAddForm();
  }

  // ürün eklerken Formda olması istenen değerleri yazacağız. Değişkenlerin adı eklenmek istenen nesne (Veritabanı) ile aynı olsun
  // MAP edilmesi istenen değerler ve geçerli validationlar yazılır
  createProductAddForm(){
    this.productAddForm = this.formBuilder.group({
      productName:[ "",Validators.required],                // ilk tırnak içinde default değeri tanımlanır. Burada boş
      unitPrice: [ "",Validators.required],
      unitsInStock:[ "", Validators.required],
      categoryId:[ "",Validators.required]
    })
  }
}
```

Adım-91: src → app → Components → product-add.components.html açalım. Değer alacak bütün alanları oluşturalım

```
<div class="content">
  <div class="col-md-12"> <!--html içinde yerini (sütünunu) tanımlıyoruz-->
    <div class="card">
      <div class="card-header"><h5 class="title">Ürün ekle</h5></div> <!--veri giriş hücresinin punto ayarları -->
      <div class="card-body">
        <form [formGroup]="productAddForm"> <!--Değer okuyacağım formatta olduğunu tanımlıyorum-->

        <!-- productName için tanımlama yapalım-->
        <div class="mb-3"> <!-- etiketlerin görsel başlık tanımlaması (sık görünüm) -->
          <label for="productName">Ürün adı</label> <!-- etiket adı (kutunun üzerinde görünecek) -->
          <div class="form-group"> <!-- okuyacağım ilk etiketi tanımlıyorum -->
            <input type="text" <!-- veri tipini tanımlamayı unutmayalım -->
            id="productName"
            formControlName="productName" class="form-control" <!-- ilişkilendirmeyi yapan kısım-->
            placeholder="Ürün adı"/> <!-- veri girişi yapılacak kutu içinde görünecek yazı -->
          </div>
        </div>
        <!-- categoryId için tanımlama yapalım-->
        <div class="mb-3">
          <label for="categoryId">Kategori</label>
          <div class="form-group">
            <input type="number"
            id="categoryId"
            formControlName="categoryId" class="form-control" placeholder="Kategori"/>
          </div>
        </div>

        <!-- unitsInStock için tanımlama yapalım-->
        <div class="mb-3">
          <label for="unitsInStock">Stok Adedi</label>
          <div class="form-group">
            <input type="number"
            id="unitsInStock"
            formControlName="unitsInStock" class="form-control" placeholder="Stok adedi"/>
          </div>
        </div>
```

```

<!-- unitPrice için tanımlama yapalım-->
    </div>
    <div class="mb-3">
        <label for="unitPrice">Birim Fiyatı</label>
        <div class="form-group">
            <input type="number" id="unitPrice"
                formControlName="unitPrice" class="form-control" placeholder="Fiyatı"/>
        </div>
    </div>

    </form>
</div>
<div class="card-footer">      <!-- cart body bittiğten sonra ekle butonu için tanımlama yapalım-->
    <button class="btn btn-fill btn-primary" (click)="add()">Ekle</button>
</div>
</div>

</div>
</div>

```

Adım-92: App-routing.module.ts açalım. Ürün eklemek istendiğinde ilgili componenti çalıştıracak (ekleme arayüzü ekrana gelecek) Path ekleyelim ve import edelim

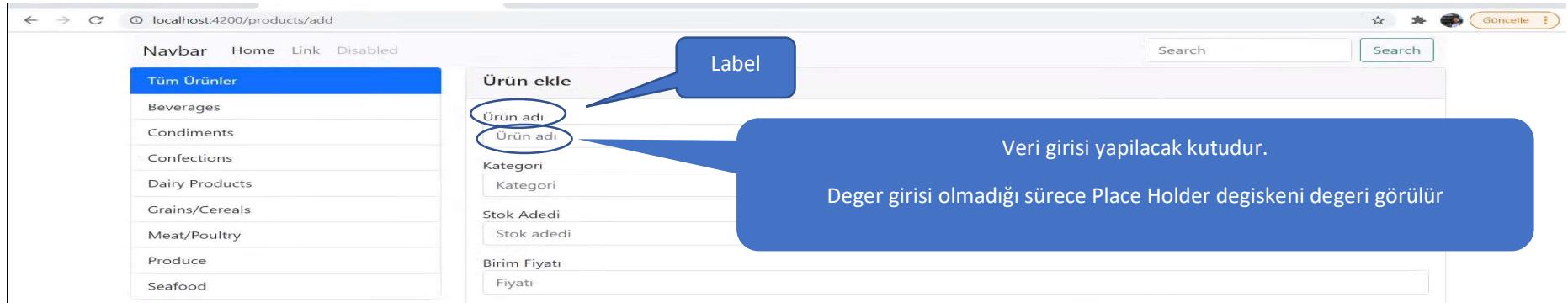
```

import { ProductAddComponent } from './components/product-add/product-add.component';
---

const routes: Routes = [
    {path:"", pathMatch:"full", component:ProductComponent},
    {path:"products", component:ProductComponent},
    {path:"products/category/:categoryId", component:ProductComponent},
    {path:"products/add", component:ProductAddComponent}
];
---

```

Program çalışıyorsa durdurup (CTRL+C ardından Y (yes)) tekrar yayına alalım (ng serve --open) HTML 4200 servis adını Ürün ekleme kısmının açılması için yazalım "localhost/4200/products/add" ve ekleme sayfası açıldı



Yukarıda HTML görüntüsü card-flooter ile add butonu çağrırdığımız ancak henüz tanımladığımız için görülmeyebilir, çünkü hata verir.
Add metodunu tanımlayalım

Adım-93: src → app → Components → **product-add.components.ts** açalım. Add metodunu oluşturacağız. Add Observable bir obje olduğu için subscribe ile girisi yapmalıyım

```
---  
  <!--add metodu için productService ve durum bildirimi için Toastr enjekte etmeliyim-->  
  constructor(private formBuilder:FormBuilder,  
    private productService:ProductService, private toastrService:ToastrService) { }  
---  
  
  <!--valide edecek, geçerli ise işlem yapacak, toastr hazır kodlarla işlem sonuçlarını bildireceğiz-->  
  add(){  
    if(this.productAddForm.valid){  
      let productModel = Object.assign({},this.productAddForm.value) <!--productmodel için oluşan classin değerleri girilen değerleri alıyor-->  
      this.productService.add(productModel).subscribe(response=>{  
        this.toastrService.success(response.message,"Başarılı")  
      })  
    }  
  }
```

```

        }else{
            this.toastrService.error("Formunuz eksik","Dikkat")
        }
    }
}

```

Adım-94: src→app→services→ product.service.ts açalım. Bu bir Interface'dır.

Linki tanımlayacağız. Bu metodun başında html ilk kısmı (sabit) tanımlamıştık. Burada ilave kısmını (products/add) tanımlayacağız ve hangi nesne ile işlem yapacaksa onu (burada Product) belirteceğiz

```

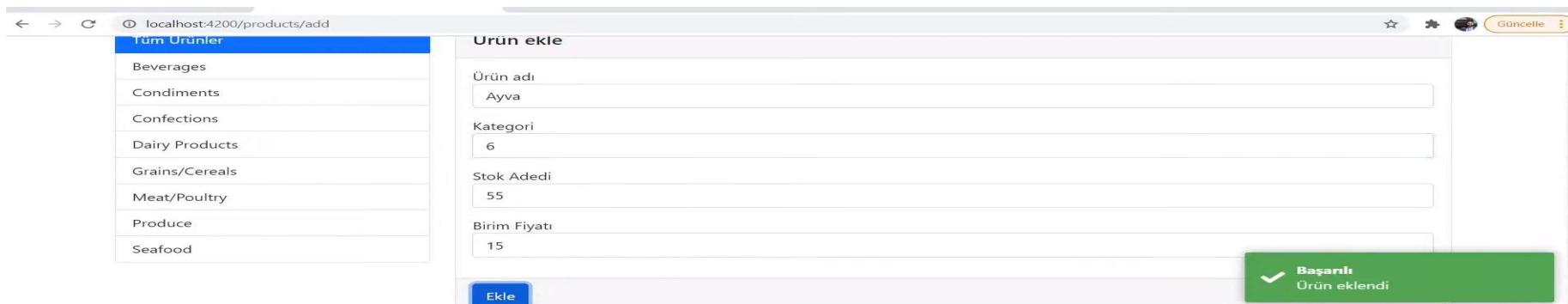
add(product:Product){
    return this.httpClient.post(this.apiUrl+"products/add",product)
}

```

Adım-95: Backend→Concrete=> ProductManager içinde “[SecuredOperation("product.add,admin")]” satırını şimdilik iptal edelim Çünkü Frontent altyapısı henüz hazır değil

```
//[SecuredOperation("product.add,admin")]
```

Adım-96: HMTL 42000 ekle üzerinden Backend içinde Business=>ValidationRules=> FluentValidation=> ProductValidation içinde tanımlı kurallara uygun olacak şekilde ürün bilgilerini girelim ve ürün ekleyelim. Veri tabanına baktığımızda da ürünün eklendiğini gördük



Neyi niçin yapıyoruz: Ekleme durumunu bildiren basari veya hata Mesajı ekleyelim. Yukarıda sadece HTML(Frontend) için görülecek bir hata yazdırmıştık. Simdi backend 'de validation kuralları sonucu basari ve hata mesajlarını frontende yansıtınca düzenlemeye yapacağız
Zaten mesaj degiskeni içeren ortak bir ResponseModel vardı. Ondan faydalanalım.
Backend'de de düzenleme gerekiyor.

Adım-97: Backend içindeki Product Add metodlarını inceleyelim ve gerekirse düzeltelim

WebAPI=>Controller → ProductController içinde Add metodunda bir sorun yok. Basari ve Hata durumu tanımlı.
Business=>Concrete → ProductManager içinde Add metoduna bakalım

Merkezi bir yere API'de bir hata olursa ne yapılacağına ilişkin kodları yazalım. Bu durum ExceptionMiddleware adlı kısma yazılır
Bütün kodları try-catch içine alıyor. Hata olursa Handle yapılacak kodlar var

Core → Extentions içine ExceptionMiddleware adlı bir Class oluştur ve Engin Demirog GITHUB'ından NetCoreBackend projesinden aynı yerden (Core-Extensions-ExceptionMiddleware) kodları kopyala. Gerekli Using komut satır çözümlemeleri yap. Sarı renkli düzenlemeleri yap

```
using System;
using System.Collections.Generic;
using System.Net;
using System.Text;
using System.Threading.Tasks;
using FluentValidation;
using FluentValidation.Results;
using Microsoft.AspNetCore.Http;

namespace Core.Extensions
{
    public class ExceptionMiddleware
    {
        private RequestDelegate _next;

        public ExceptionMiddleware(RequestDelegate next)
        {
            _next = next;
        }
    }
}
```

```

public async Task InvokeAsync(HttpContext httpContext)
{
    try
    {
        await _next(httpContext);
    }
    catch (Exception e)
    {
        await HandleExceptionAsync(httpContext, e);
    }
}
private Task HandleExceptionAsync(HttpContext httpContext, Exception e)
{
    httpContext.Response.ContentType = "application/json";
    httpContext.Response.StatusCode = (int) HttpStatusCode.InternalServerError; // burada hata sistem hatası 500 olarak belirtilmiş

    string message = "Internal Server Error";
    IEnumerable<ValidationFailure> errors; // hata listesi oluşturacağız
    if (e.GetType() == typeof(ValidationException))
    {
        message = e.Message;
        errors = ((ValidationException)e).Errors; // hataların hepsini yakaladık ve listeye aldık
        httpContext.Response.StatusCode = 400; // hata statü kodunu da kötü sonuç olacak şekilde düzeltelim (500'den 400'e)

        // validation'a uygun bir hata listesi oluşturduk ve ekledik
        return httpContext.Response.WriteAsync(new ValidationErrorDetails
        {
            StatusCode = 400,
            Message = message,
            Errors = errors
        }.ToString()); // Json'a çevirmek için
    }
    return httpContext.Response.WriteAsync(new ErrorDetails // burası sistemsel hatayı belirtir. Validation Hata istesi içermez
    {
        StatusCode = httpContext.Response.StatusCode,
        Message = message
    }.ToString());
}
}

```

Adım-98: Backend → Core → Extentions içine **ErrorDetails** adlı bir Class oluştur ve Engin Demirog GITHUB'ından NetCoreBackend projesinden aynı yerden (Core-Extensions-**ErrorDetails**) kodları kopyala.

JsonConvert için ampülden Install Package Newtonsoft.Json ile eklenti kuralım ve using komut satırını ekleyelim

```
using System;
using System.Collections.Generic;
using System.Text;
using FluentValidation.Results;
using Newtonsoft.Json;

namespace Core.Extensions
{
    public class ErrorDetails
    {
        public string Message { get; set; }
        public int StatusCode { get; set; }
        public override string ToString()
        {
            return JsonConvert.SerializeObject(this);
        }
    }
    // hata kodu oluşturan metodу tanımlayalım. Yukarıdakilere ilaveten Validation hata listesi var
    public class ValidationErrorDetails : ErrorDetails
    {
        public IEnumerable<ValidationFailure> Errors { get; set; }
    }
}
```

Adım-99: Backend → Core → Extentions içine **ExceptionMiddlewareExtensions** adlı bir Class oluştur ve Engin Demirog GITHUB'ından NetCoreBackend projesinden aynı yerden (Core-Extensions-**ExceptionMiddlewareExtensions**) kodları kopyala. IApplicationBuilder için Ampülden Using komut satırını çöz

Kendi Midleware'imizi ekleyeceğiz

```

using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.AspNetCore.Builder;

namespace Core.Extensions
{
    public static class ExceptionMiddlewareExtensions
    {
        public static void ConfigureCustomExceptionMiddleware(this IApplicationBuilder app) //StartUp'ta zaten kullanılan bir yaşam döngüsü
        {
            app.UseMiddleware<ExceptionMiddleware>(); //Startup'ta olan yaşam döngüsüne hata olup olmadığını gözeten ekleme yapıyoruz
        }
    }
}

```

Adım-100: **Backend** → **WebAPI Startup** içindeki mevcut metoda **ExceptionMiddleware** çalışması için komut ekliyoruz

```

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    app.ConfigureCustomExceptionMiddleware(); // 20.derste ekledik. Frontend veri tabanı işleminin barası/hata durumu için
    app.UseCors(builder=>builder.WithOrigins ("http://localhost:4200").AllowAnyHeader()); //Frontend'ten erişimi için Configurasyon
yapmalıyız 17.Hafta. Buradaki konumu önemli
    app.UseHttpsRedirection();
    app.UseRouting();
    app.UseAuthentication(); // ekledik 14. ders
    app.UseAuthorization();
    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllers();
    });
}

```

Adım-101: src→app→services→ product.service.ts açalım. Add metodu hazır metod olan ResponseModel tipinde tanımlayalım ki mesaj değişkenini kullanalım

```
add(product:Product):Observable<ResponseModel>{
  return this.httpClient.post<ResponseModel>(this.apiUrl+"products/add",product)
}
```

Adım-102: src→app→Components→ product-add.components.ts açalım. Add metodunu düzenleyelim.

```
<!--valide edecek, geçerli ise işlem yapacak, toastr hazır kodlarla işlem sonuçlarını bildireceğiz-->
add(){
  if(this.productAddForm.valid){
    let productModel = Object.assign({},this.productAddForm.value) <!--productmodel için oluşan classin değerleri girilen değerleri alıyor-->
    this.productService.add(productModel).subscribe(response=>{
      this.toastrService.success(response.message,"Başarılı") <!--response degiskeni ile backend basari mesajı da görüntülenir-->
    },responseError=>{
      <!--responseError degiskeni ile backend hata mesajı da görüntülenir-->
      if(responseError.error.Errors.length>0){ <!--Hata listesi 0'dan büyükse hataları al ve frontente sergile -->
        for (let i = 0; i <responseError.error.Errors.length; i++) {
          this.toastrService.error(responseError.error.Errors[i].ErrorMessage,"Doğrulama hatası")
        }
      }
    })
  }else{
    this.toastrService.error("Formunuz eksik","Dikkat")
  }
}
```

HTML 4200 Ürün ekleme sayfasında validation hataları kullanıcıya sergilenmektedir

localhost:4200/products/add

Uygulamalar Altbel - Confluence Tricerat | Çeviri... KURSNET - Startseite Bundesagentur für... Kodlamaio #soru-cevap-7 vits.cloud Your Repositories PONS Wörterbuch... NevishNow: The Ind... Online Kurstar - Her... Çeviri - Google Sear...

Navbar Home DASHBOARD

Search Search

Tüm Ürünler

- Beverages
- Condiments
- Confections
- Dairy Products
- Grain/Cereals
- Meat/Poultry
- Produce
- Seafood

Ürün ekle

Ürün adı: d

Kategori: 3

Stok Adedi: 1

Birim Fiyatı: 1

Ekle

Doğrulama hatası Ürün A harfi ile başlamalı

Doğrulama hatası 'Product Name': 2 karakterden büyük veya ept olmalıdır. 1 karakter girdiniz.

21.DERS (24.03.2021)

Konu: Login ve Register

Adım-103: **src**→**app**→**Component** sağ tıkla **open in integrated new terminal** ile açılan yeni bir terminalde login component olusuralım
ng g component login→

Not: yanlış yerde oluşturmuşsak o dosyayı taşıyabilir/silebiliriz, yalnız app.modul.ts dosyasında import satırı ve metodu içinde o oluşan dosyanın uzantısını (import satırı ve import metodunda 2 yerde) değiştirmeli veya silmeliyiz

Adım-104: **logincomponent.html**

Getbootstrap sitesinden **examples** sekmesi içinde “**sign in**” kısmını bulalım. Açılan sayfada sağ tıkla sayfa kaynağını görüntüle seçenek kodlara erişelim en sonda yer alan Body kodlarını **login.componenet.html** içine yapıştıracağız. Product-add.component.html dosyasında olduğu gibi tek bir metoda bağlı veri giriş formları tek bir from grubu altında tanımlanmadır

```
<body class="text-center">

<main class="form-signin">
  <form [formGroup]="loginForm">
    <h1 class="h3 mb-3 fw-normal">Giris Yap</h1>
    <!-->aynı metoda bağlı formları gruppala-->
    <div class="form-floating">
      <input
        formControlName="email"
        type="email"
        class="form-control"
        id="floatingInput"
        placeholder="name@example.com">
      <label for="floatingInput">Email address</label>
    </div>
    <!-->Formu değişkene atama-->
```

```

<div class="form-floating">
  <input
    formControlName="password"                                     <!-->Formu değişkene atama-->
    type="password"
    class="form-control"
    id="floatingPassword"
    placeholder="Password">
    <label for="floatingPassword">Parola</label>
  </div>

  <div class="checkbox mb-3">
    <label>
      <input type="checkbox" value="remember-me"> Remember
    </label>
  </div>
  <button class="w-100 btn btn-lg btn-primary" (click)="login()">           <!-->buton ile metodу aktive etme-->
    Giriş Yap</button>
  <p class="mt-5 mb-3 text-muted">&copy; 2017-2021</p>
</form>
</main>
</body>

```

Adım-105: [Login.component.css](#)

Bootstrap içinde biraz önce açtığımız sign in kodları içinde signin.css adlı bir link var. Onu tıklayıp açalım. Css kodlarına erişelim ve kopyalayalım

```

html,
body {
  height: 100%;
}

body {
  display: flex;
  align-items: center;
  padding-top: 40px;
  padding-bottom: 40px;
}

```

```
.form-signin {  
    width: 100%;  
    max-width: 330px;  
    padding: 15px;  
    margin: auto;  
}  
  
.form-signin .checkbox {  
    font-weight: 400;  
}  
  
.form-signin .form-floating:focus-within {  
    z-index: 2;  
}  
  
.form-signin input[type="email"] {  
    margin-bottom: -1px;  
    border-bottom-right-radius: 0;  
    border-bottom-left-radius: 0;  
}  
  
.form-signin input[type="password"] {  
    margin-bottom: 10px;  
    border-top-left-radius: 0;  
    border-top-right-radius: 0;  
}  
  
.bd-placeholder-img {  
    font-size: 1.125rem;  
    text-anchor: middle;  
    -webkit-user-select: none;  
    -moz-user-select: none;  
    user-select: none;  
}  
  
@media (min-width: 768px) {  
    .bd-placeholder-img-lg {  
        font-size: 3.5rem;  
    }  
}
```

Adım-106: Login.component.ts

```
import { Component, OnInit } from '@angular/core';
import { FormControl, Validators, FormBuilder, FormGroup} from "@angular/forms";
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {
  loginForm:FormGroup;
  constructor(private formBuilder:FormBuilder) { }

  ngOnInit(): void {
    this.createLoginForm();
  }

  createLoginForm(){
    this.loginForm = this.formBuilder.group({
      email:[ "",Validators.required],
      password:[ "",Validators.required]
    })
  }
  login(){
    if(this.loginForm.valid){
      console.log(this.loginForm.value);
    }
  }
}
```

Neyi ne için yapıyoruz: Simdi Login olmak istendiğinde bilgileri API'ye göndermeliyiz. **Backend** ile doğru değişkenler üzerinden etkileşim kurmalıyız

Adım-107: Models sağ tıkla **tokenModel.ts** adlı new file oluştur. Dikkat file ismi küçük harfle başlıyor ancak interface ismi büyük harfle baslıyor

```
export interface TokenModel{
  token:string;
  expiration:string;}
```

Adım-108: Models sağ tıkla `loginModel.ts` adlı new file oluştur

```
export interface LoginModel{  
    email:string;  
    password:string;  
}
```

Adım-109: App ➔ services sağ tıkla open in integrated terminal

Ng g service auth ➔

Adım-110: Services ➔ auth.service.ts

```
import { HttpClient } from '@angular/common/http';  
import { Injectable } from '@angular/core';  
import { LoginModel } from '../models/loginModel';  
  
@Injectable({  
    providedIn: 'root'  
})  
export class AuthService {  
    apiUrl= "https://localhost:44356/api/auth/"; // burası backend Apinin html adresi, sondaki taksim işaretini unutma  
    constructor(private httpClient:HttpClient) { }  
  
    login(loginModel:LoginModel){  
        return this.httpClient.post (this.apiUrl + "login", loginModel)  
    }  
    // register durumunu kontrol eden bir yapı kuralım.  
    isAuthenticated (){  
        if(localStorage.getItem("token")){  
            return true;  
        }  
        else {  
            return false;  
        }  
    }  
}
```

Adım-111: Login.component.ts

```
import { Component, OnInit } from '@angular/core';
import { FormControl, Validators, FormBuilder, FormGroup} from "@angular/forms";
import { AuthService } from 'src/app/services/auth.service';
import { ToastrService } from 'ngx-toastr';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {
  loginForm:FormGroup
  constructor(private formBuilder:FormBuilder,
  private authService:AuthService,
  private toastrService:ToastrService) { }
  ngOnInit(): void {
    this.createLoginForm();
  }
  createLoginForm(){
    this.loginForm = this.formBuilder.group({
      email:[ "",Validators.required],
      password:[ "",Validators.required]
    })
  }
  login(){
    if(this.loginForm.valid){
      let loginModel= Object.assign({},this.loginForm.value)
      this.authService.login(loginModel).subscribe(response=>{
        this.toastrService.info(response.message)
        localStorage.setItem("token",response.data.token)
      },responseError=>{
        this.toastrService.error(responseError.error)
      })
    }
  }
}
```

Not:java fonksiyon temelliidir. Class gibi davranan fonksiyonlar vardır

Adım-112: Services→auth.service.ts

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { LoginModel } from '../models/loginModel';
import { SingleResponseModel } from '../models/singleResponseModel';
import { TokenModel } from '../models/tokenModel';

@Injectable({
  providedIn: 'root'
})
export class AuthService {
  apiUrl= "https://localhost:44356/api/auth"; // burası backend Apinin html adresi
  constructor(private httpClient:HttpClient) { }

  login(loginModel:LoginModel){
    return this.httpClient.post<SingleResponseModel<TokenModel>>(this.apiUrl + "login", loginModel)
  }

  // register durumunu kontrol eden bir yapı kuralım.
  isAuthenticated (){
    if(localStorage.getItem("token")){
      return true;
    }
    else {
      return false;
    }
  }
}
```

Adım-113: Models→sag tikla singleResponseModel.ts adlı new file oluştur (ResponseModel sadece success ve mesaj okuyordu. Token bilgisini okuyamazdı)

```
import {ResponseModel } from "./responseModel";
// responseModel'i Token'i da ıersibilecek şekilde genişlettik
export interface SingleResponseModel<T> extends ResponseModel{
  data:T
}
```

Adım-114: **Backend**→**WebAPI**→**authController** içinde bir satırı düzelttik

```
[HttpPost("login")]
public ActionResult Login(UserForLoginDto userForLoginDto)
{
    var userToLogin = _authService.Login(userForLoginDto);
    if (!userToLogin.Success)
    {
        return BadRequest(userToLogin.Message);
    }
    var result = _authService.CreateAccessToken(userToLogin.Data);
    if (result.Success)
    {
        return Ok(result); /////////////////////////////////////////////////////////////////// 21.derste düzelttik
    }
    return BadRequest(result.Message);
}
```

Adım-115: **Backend**→**Business**→**Concrete**→**ProductManager** içinde add metodu üzerinde ekleme yetkisi kontrol yapan aspect satrini yeniden aktive edelim

```
[SecuredOperation("product.add,admin")] // Frontent kapsamında 21.derste tekrar aktiv ettik
```

NEYI NICN YAPIYORUZ: Artık sadece yetkililere ürün ekleme yetkisini frontend kısmında da tanımlamalıyız.

Frontend kısmında yapılan kullanıcı girişi sonucu oluşan Token'i hafızaya almıştık. Simdi kontrol yapmalı ve ona göre eklemeliyiz

Adım-116: **src**→**app new folder interceptors** oluştur. **Open in integrated terminal** ile auth **interceptor** dosyaları oluştur, burası bütün http isteklerimizi intercept edecek kısımdır. Middleware yazmıştık gecen hafta, aynısını buraya da tanımlayabiliriz

ng g interceptor auth→

Adım-117: `src→app→auth.interceptors.ts`

```
import { Injectable } from '@angular/core';
import {
  HttpRequest,
  HttpHandler,
  HttpEvent,
  HttpInterceptor
} from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable()
export class AuthInterceptor implements HttpInterceptor {

  constructor() {}

  intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<unknown>> {
    let token = localStorage.getItem("token");
    let newRequest : HttpRequest<any>;           //islemi yakalamaya calisiyoruz. tip tanimlama iki nota üst üste isareti ile
    // kullanıcının yaptığı işlemini klonlarız
    newRequest = request.clone({
      headers: request.headers.set ("Authorization", "Bearer " + token) //yakaladığı bilgiyi yerlestirme sekli dikaat bearer sonrası bosluk var
    })
    return next.handle(newrequest);
  }
}
```

Kodu inceleyelim: Request== bizim yaptığımız işlem, Next: bizim bir sonraki aşamadaki işlem

Adım-118: `app.module.ts` // obje formatında service injection yapıyoruz

```
import {HttpClientModule, HTTP_INTERCEPTORS} from '@angular/common/http'; //Dikkat
import { AuthInterceptor } from './interceptors/auth.interceptor';

providers: [
  {provide:HTTP_INTERCEPTORS, useClass:AuthInterceptor, multi:true}
],
```

NEYI NICIN YAPIYORUZ: login olmayan kişi ekleme yapamıyor ancak add sayfasına ertsiyor. Simdi login olmadan ürün ekleme sayfasına ertsimek için işlem yapan (şimdilik HTML adresini yazan, sonra ekle butonuna basan) birisini login ekranına yönlendirelim

Adım-119: `src→app new folder guards` oluştur. Open in integrated terminal
`ng g guard login→`

Default olan canActivate seçerek yüklemeye devam edelim

<code>>(*) CanActivate</code>	// aktiv etme yetkisi
<code>() CanActivateChild</code>	// başka bir child çağırılabilme etkisi
<code>() CanDeactivate</code>	// kapatma yetkisi
<code>() CanLoad</code>	// yükleme yetkisi

Guard Tanım: Angularda guard bir işlemi yapıp yapamamasın mı desteği veren bir yapıdır

Adım-120: `login.guard.ts`

```
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, UrlTree, Router } from '@angular/router';
import { ToastrService } from 'ngx-toastr';
import { Observable } from 'rxjs';
import { AuthService } from '../services/auth.service';

@Injectable({
  providedIn: 'root'
})
export class LoginGuard implements CanActivate {

  constructor (private authService:AuthService,      // giriş yapmamış
  private toastrService:ToastrService,          // bilgi
  private router:Router){                      // router erişmek veya reddetmek için

}
}
```

```

canActivate(
  route: ActivatedRouteSnapshot,
  state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
if (this.authService.isAuthenticated()){
  return true;
} else{
  this.router.navigate(["login"])
  this.toastrService.info("Sisteme giriş Yapmalısınız")
  return false;
}
}
}

```

Adım-121: App-routing.module.ts

```

import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { CategoryComponent } from './components/category/category.component';
import { ProductAddComponent } from './components/product-add/product-add.component';
import { ProductComponent } from './components/product/product.component';
import { LoginComponent } from './components/login/login.component';
import { LoginGuard } from './guards/login.guard';

const routes: Routes = [
  {path:"",pathMatch:"full", component:ProductComponent},
  {path:"products", component:ProductComponent},
  {path:"products/category/:categoryId", component:ProductComponent},
  {path:"products/add", component:ProductAddComponent, canActivate:[LoginGuard]},
  {path:"login", component:LoginComponent}
];

```

HMTL4200 İstenen her şeyi sorunsuz yapıyor

Navbar kısmına rooter link ile user girisi ve ürün ekleme sayfalarına link verebiliriz

İLAVE BİLGİLER

Erişim düzenleyiciler (Access modifiers) Private /Public / protected /internal (Default) /protected internal

Erişim düzenleyici dediğimiz anahtar kelimelerin ne işe yaradığı hakkında genel bir tanım yapalım. Bir tipin kendisine veya o tipe ait üyelerine (metod, özellik ya da olay) nasıl erişileceğini daha doğrusu, nereden (hangi kod bloğundan) erişilebileceğini belirleyen kelimelere “erişim düzenleyiciler” diyoruz. İşte şimdî bu kelimeleri, tek tek anlatmaya sıra geldi.

- 1. Private** : En minik erişim düzenleyicisidir.
- 2. Public** : En genel ve sınırsız erişim düzenleyicisidir

3. protected

private’den farkı, protected olarak tanımlanmış alan ya da metodlar, miras olarak aktarılabilirler. Yani, “bu üyeye kesinlikle dışardan ulaşılmasın, ama miras bırakılsın” diyorsanız, o üye protected olmalı

4. internal (DefAUTL)

“Dahili” anlamına gelmektedir. Yalnızca bulunduğu assembly’den erişilebilir. Burada assembly ifadesinden kasıt, projenin kendisidir. Yani, bir kütüphane (.dll) oluşturuyorsanız, internal bir sınıfa sadece o kütüphaneden ulaşabilirsiniz. Bu erişim düzenleyicisi, sınıf üyelerinde de kullanılabilir. Onda da etkisi aynıdır. Bir sınıfın erişim düzenleyicisi belirtilmemezse, varsayılan olarak internal kabul edilir.

5. protected internal

Yalnızca sınıf üyelerine uygulanır. Kelimelerin arasına “ya da” konulunca anlaşılması daha kolay olacaktır. Erişim yapılan yere göre “internal” ya da “protected” davranışır. Doğal olarak assembly dışından erişilmeye çalışıldığındá internal, aynı assembly’den erişilmeye çalışıldığındá ise protected davranışır.

SQL Veri Tipleri

Veri Tipi	Açıklama
1) Metinsel Veri Tipleri	
char	Unicode'u desteklemeyip char(n) şeklinde kullanılırlar. 8000 karaktere kadar değer alabilirler. Belirtilenden(n) az karakter girilse dahi giriş yapılan boyut kadar yer kaplar. Veri giriş boyutları benzer, sabit olan veri kümelerinde kullanılması önerilir.
nchar	Unicode(uluslararası karakter setini, tanımlı tüm alfabeleri içerir)destekler. Chardan farklı olarak maksimum 4000 karaktere kadar değer alabilir.
varchar	Chardan farklı olarak verinin boyutu kadar yere kaplar. 8000 karaktere kadar depolama yapar.Birbirinden farklı uzunlukta veri girişi yapılacağı zaman kullanılması önerilir. varchar(MAX) kullanımı ile 2GB'a kadar depolama yapılabilir.
nvarchar	Verinin boyutu kadar yer kaplar. Varchardan farklı olarak unicode'u destekler.4000 karaktere kadar değer alabilir.
text	Belirtilenden az değer girilse bile boyutu kadar yer kaplar.2GB'a kadar Metinsel veri depolar.Unicode'u desteklemez.
ntext	Text'den farklı olarak girilen karakter boyutu kadar yer kaplar ve unicode'u destekler.
2) Binary(İkilik) Veri Tipleri	
binar	1 ve 0 ları temsil eden ikilik taban veri tipidir. Sabit uzunluklu veri tiplerinde kullanılırlar.8000 byte'a kadar depolama yapabilir.
varbinary	Binary'den farklı olarak girilen karakter kadar yer kaplar. Bu yüzden uzunlıkların değişken olduğu durumlarda tercih edilir.
image	Resim dosyalarını saklamak için kullanılır. En fazla 2 GB'a kadar veri depolar. Bunun yerine varbinary(MAX) kullanılması tercih edilir.
3) Sayısal Veri Tipleri	
bit	Bir byte uzunlığında tam sayı veri tipidir. Genellikle evet/hayır şeklinde mantıksal bilgileri tutmak için kullanılır.
int	4 byte büyüğünde, -2 milyar /+2 milyar arasında değer tutabilen tam sayı veri tipidir.
bigint	8 byte büyüğünde -2^{63} ve 2^{63} arasında değer tutabilen tam sayı veri tipidir.
smallint	2 byte büyüğünde -32.768 ve 32.768 arası değer alabilen tam sayı veri tipidir.
tinyint	1 byte büyüğüğe sahip, 0–255 arası tam sayı veriler için kullanılan tam sayı veri tipidir.
decimal, numeric:	İkisinin de kullanımı aynıdır. Bu veri tipinde saklanacak sayının basamak sayısı tanımlanabilir. Veri tipi boyutu belirtilen basamak sayılarına göre değişkenlik gösterebilir.-38 ve +38 basamak arası verileri depolayabilir. -10^{38} , 10^{38} arası ondalık ve tam sayı türünde veri saklayabilir.

Veri Tipi	Açıklama
4) Parasal Veri Tipleri	
money	8 byte boyutunda, yaklaşık -2^{64} ile 2^{64} arasında parasal değerleri tutmak için kullanılır. 4 basamağa kadar duyarlı ondalık tipli verileri saklar.
smallmoney	4 byte uzunluğunda yaklaşık 214.000 ile 214.000 arası parasal değerleri tutmak için kullanılır. Money tipinde olduğu gibi 4 basamağa kadar duyarlı ondalık tipli verileri saklarken kullanılır.
5) Tarih-Zaman Veri Tipleri	
date	Tarihleri YYYY-AA-GG (yıl-ay-gün) formatında saklar. 3 byte uzunluğunda veri tipidir.
smalldatetime	Tarih ve zaman verilerini yıl-ay-gün ve saat-dakika-saniye-salise şeklinde saklar. 4 byte uzunluğunda veri tipidir.
datetime	YYYY-AA-GG şeklinde tarih ve zaman verilerini tutan 8 byte uzunluğunda veri tipidir. 1 Ocak 1753–31 Aralık 9999 arası veriler için kullanılır.
datetime2	Datetime'dan farklı olarak 1 Ocak 0001–31 Aralık 9999 tarihleri arası verileri tutan ekstra olarak salise hassasiyeti daha yüksektir. Kapladığı alan salise hassasiyetine göre 6–8 byte arası değişimdir.
time	Sadece saat verilerini saat-dakika-saniye-salise (7 basamaklı) şeklinde saklayan, boyutu kullanıcı tarafından değiştirilebilen 3–5 byte arası yer kaplayan veri tipidir.
datetimeoffset	Ülkelere göre değişen zaman farkını hesaplayıp tutarken kullanılır.
6) Diğer Veri Tipleri	
sql_variant	Sayı, metin, Binary gibi farklı veri tiplerini depolamak için kullanılan veri tipidir. Yani bir sütun ya da fonksiyonda birden fazla veri tipi kullanmamız gerekiyorunda tercih etmeliyiz.
xml	XML türünde veri saklamak için kullanılır. Kapasitesi 2 GB'dır. Bellekteki boyutu, saklanan XML verisine göre değişkenlik gösterir.
geometry	Öklid koordinat sistemine ait verileri tutmak için kullanılır. Geometrik şekillerin en-boy-yükseklik verilerini saklar.
timestamp	Tabloya kayıt eklendiğinde, güncellendiğinde Binary türünde özel değer alan veri tipidir.
uniqueidentifier	16 byte uzunluğunda benzersiz GUID tipinde veri tutar. İki GUID birbirinden tamamen farklıdır eşit olamazlar.
hierarchyid	Ağaç veri modeli ve ya hiyerarşik olarak sınıflandırılmış verileri saklamak için kullanılır.
geography	Coğrafi koordinat ve GPS verilerini tutmak için kullanılır.

Backend Notları

1. Komutları toplu Comment yapma veya geri alma

Ctrl k + Ctrl c ===== seçilen komutlar Comment (Yorum satırı) haline gelir. Seçili satırların başına “//” işaretleri eklenir

Ctrl k + Ctrl u===== seçilen Comment (Yorum satırı) komut haline gelir. Seçilen satırların başındaki “//” işaretleri kalkar.

Ctrl k + Ctrl d===== boşluksuz yazılan kodları otomatik düzenler

2. Değişkenler büyük harfle tanımlanır

3. **get** komutu bu metottaki değişkenlerin başka yerlerde okunabileceğini, **set** komutu ise yazdırılabilceğini belirtir

get ve set ayarları Java'da metodlarla yapılırken C#'da properties ile yapılır.

Program içinde sadece kullanıyorsak **get** bloğu çalışır

Örnek : Console.WriteLine(Urun.Adi);

Program içinde eşitleme ile veya datadan okuyup eşitleme ile veri alınıyorsa **set** bloğu çalışır

Örnek : Urun.Adi="elma";

4. **Birden fazla programın olduğu bir projede sadece üzerinde çalıştığı programın koşturulması:** Visual Studio 'da “**Solution Explorer**” sayfasındaki (sağ taraf) koşturma istediğiniz programın üzerinde sağa tıklayıp “**Set a Startup Projekt**” seçilir.

5. Bir projede bir Class 'a başka bir Class tipinde değişken gönderilirse ve o Class değişkeninde bir değer değiştirilse. O Class değişkeninin değeri her yerde kalıcı olarak değişir. Class ve Array 'lar referans tiptir

Ancak int, Double, string vb. değişkenler benzer şekilde bir Class 'a gönderilirse o Class içinde değeri değiştirilse bile ana yerdeki değişken değeri değişmez. Bir önceki derste benzer bir problem vardır. Sayıların değeri ile ilgili. Bunlar değer tiptir. Benzer mantık....

6. Bir metottan dönen bir değer yoksa **void** yazılır, Aksi takdirde dönen değer yazılır.

```
namespace classes
{
    class Customer
    {
        public int Id { get; set; }

        private string _firstName;
        public string FirstName
        {
            get { return _firstName; }
            set { _firstName = value; }
        }

        public string LastName { get; set; }
        public string City { get; set; }
    }
}
```

7. Programda hazır kütüphane metotları kullanılacaksa ilgili kütüphane oluşturulan Class/ Controller vs. en üst satırında using komutu ile tanımlanmalıdır. Ya da kullanılmak istenen metot adı yazıldığında çıkacak ampulden seçilen ilgili komut satırı en üste otomatik eklenir. Örneğin **List** yapıları kullanılacaksa programın en üstüne “**using System.Collections.Generic**” komutu ile kütüphane eklenir. Programa **list** yazısı yazılırsa gözüken ampul tıklanınca o komut seçilirse otomatik eklenir
8. Yukarıda benzer şekilde Programa bazı komutlar yazıldığına/seçildiğinde ekranda ampul belirebilir. Ampul tıklandığında bazı hazır işlemler için seçenekler gözükmür (metot ekle, ayrı bir class olarak tanımla vb.)
9. Bir işlem yapılmayacak, sadece kayıt tutulacak değişkenler sadece rakam bile içerde **string** olarak tanımlamak daha faydalıdır
10. Eğer ki bir nesnede bir değeri kullanmak zorunda değilse ama tanımlamışsan hata yapıyorsun demektir
11. Bir Class aynı anda iki Interface Class ‘a Interface edilebilir. İyi bir Class içerisinde sadece 1 metot olmalıdır. Class ‘lar çiplak bırakılmamalıdır. Mutlaka bir başka Class ‘a veya Interface ‘a inherit edilmelidir. Bir Class inheritance veya implementation almayıorsa ilerde problem yaşama ihtimalin vardır.
Istisna: Utilities Classları çiplak kalabilir. Normalde bir Interface oluşturup bunu ona inherit edebilirdik ancak **overdesign** olurdu.
12. `Console.WriteLine("-----METOTLAR-----"); //kısa yolu "cw" yapıp 2xTab`
`public int MyProperty { get; set; } //kısa yolu "prob" yapıp 2xTab`
13. Bir projede özellikleri kaydeden Class ‘lar ver operasyon Class ‘ları ayrı ayrı yazılmalıdır. Operasyon Class ‘ları da Interface Class ile tanımlanmalıdır.
14. Formasyonda veri tabanları silindir şeklinde resmedilir
15. Bir **entity Manager** (Örneğin **ProductManager**) kendisi hariç başka entity için (örneğin Category için) oluşturulmuşdal şeklinde Class ‘i enjekte edemez, ancak başka bir **Service interface** enjekte edebiliriz
16. Core katmanı diğer katmanları referans almaz. Bağımsız olmalıdır
17. Build menüsünden **Built Solution** yapalım. Hata olup olmadığını kontrol edelim.
Build Succeeded görürsek işler yolunda gidiyor demektir.
18. **Başka bir veri tabanını, başka bir program kullanan bir sisteme entegre olabilmek için bu şekilde Her katmanı değiştirebiliriz**
`public class EfCategoryDal : EfEntityRepositoryBase<Product, NorthwindContext>, ICategoryDal { }`

- 19.** Bir is sınıfı başka sınıfları new 'lemez. Onun yerine kendisi alt çizgi ile yeni bir Class tanımlar. Sonra ampulden using satırını ekler ve **generate constructor** yaparak burada çalışacağın bir Class üreten metot oluşturur
- 20.** **Adlandırmalar** anlamlı olmalı (Mümkünse İngilizce).
Static değişkenlerin adlandırması küçük harfle başlamalı, (Kısaltma olmasın, yeni bilgisayarlarda Complexity bu isimlendirmelerden artık çok etkilenmez)
Dinamik değişkenlerin (dizi, liste, class vb) adlandırması (birleşik kelimelerde her kelime bas harfi) büyük harfle
- 21.** **Code Refactoring**
C# Nesnel bir dildir)
1 Fonksiyon sadece 1 işi yapacak şekilde oluşturulmalıdır. Fonksiyon içinde 1'den fazla fonksiyonu çağrılabilsin. Adlandırmalarda yapılacak işi anlatır. Ayrıca alt fonksiyonlardan biri başka yerlerde lazım olduğunda bunu orada da çağrılabilsin
Denge de önemlidir. **Overdesign** olmasın ancak işlemler mümkün mertebe en küçük birim halinde tanıtılsın
- 22.** **Yorumlar:** Fonksiyon başlıklarının üstüne genel yorum yazılabilir ancak fonksiyon içerisinde Kurumsal Hafıza için yapılan bu işlem yanlış yerde yapılıyor demektir.
- 23.** Bir aspecti her şeye çalışmasın istersek **Core => Utilities** Klasörü içindeki **Interceptors** Class içine koyarız
- 24.** **Statik Yapılar:** Bu şekilde tanımlanan statik yapılara diğer bir Class'tan erişim newlemeyle değil, intelligenz ile mümkün olur. Referans degisen olan Class'in içinde statik bir yapı kurmuş olduk. Statik yapılar özellikle standart işlemleri(operasyonları) oluşturacagımız class'in (Utilities) içindeki operasyonları diğer Classlar içinde gerektiğinde çağrıldığımızda newlemeye gerek olmadan kullanmamıza imkan tanır. Bu veri program resetlenene kadar hafızada durur ve her gerektiğinde kullanılır. Tekrar tekrar cegirilmeye gerek kalmaz.Bu şekilde program ve hafıza daha verimli kullanılır. C#'da constructor bloklar static yapıda calısırken Java'da calısmaz. Bunun için Java'da static adlı bir yapı kurulmalıdır. Java'da Classlar Static tanımlanamaz iken C#'da mümkünür. Java'da sadece inner Class (metod) static olarak tanımlanı
- 25.** **DependencyInjection** yöntemi= Temel olarak bağımlılıkların kontrolü ve yönetimi için kullanılmaktadır. Dependency Injection tekniğinde bağımlılık oluşturacak parçalarının ayrılp, bunların sisteme dışarıdan verilmesi (enjekte edilmesi) ile meydana gelir. Bir katman diğer bir katmandaki ilgili/gerekli yapının sadece referans tutucu olan abstract kısmındaki yapıya, yani Interface'ine bağlanmalıdır.
- 26.** **DINAMIK LISTELEME:** Arraylar(Diziler) statik yapıda, list (listeler) dinamik yapıdadır. Arraylara sonradan ani bir değişken eklenemez. Ancak listelere ekleme yapılabilir. Java'da arrayList adlı bir yapı kullanırız. C#'da list yapısı gibi.

27. CRUD (Create Read Update Delete) Veri tabanı işlemlerine verilen kısaltma ad. Bu işlemleri yapan metodlara da bu ad verilmesi uygundur
28. **EntityFramework ve BaseRepository.** CRUD işlemleri için kullanılan hazır kaynak kodlar.
29. **Regular Expressions** (Regex veya Regexp / Düzenli / Kurallı İfadeler) modern programlama dillerinin neredeyse tamamında yer bulan, aynı söz dizimine (syntax) sahip olan, genellikle harflerden oluşan karakterler dizisinin (katar/ string) belirtilen kurallar çerçevesinde kısa yoldan ve esnek bir biçimde belirlenmesini sağlayan bir yapıdır.
30. **Micro Servis sistemlerin eklenmesi:** Mernis benzeri var olan Micro Servis Sistemlerin kendi projemize ekleyeceğiz. Alınan projeyi degistiremeyiz. sadece ekleyip kullanabiliriz. Kendi Interfacemizi ve Adapte için bir Class oluşturarak adapte etmeliyiz.
31. **Polymorphism:** Newleme imkani olmayan Interface ve inheritance süper Class içindeki metoda ilgili Class'tan olusturulan nesneyi göndererek işlem yapmasına imkan tanırız. Referans tutucu olan Interface ve süper Class'lar bu şekilde çalışabilir.
32. **OVERLOADING:** Bir method veya constructor'un farklı parametrelerle çalışacak şekilde tanımlamasını denir. Bir Class'a farklı parametreler gönderilebilir. Çalışması istenen her duruma göre o classla aynı anda constructorlar oluşturulur. aldığı değerler farklı olduğu için newleme yapıldığında girilerek değerleri karşılamaya uygun olan metod çalışır. Bu duruma override adı verilir. Bu durumu imzaya göre ilgili constructor metodun çalışması şeklinde ifade ediyoruz.
33. **Override:** Soyut Class'ta tanımlı bir metodу extend edilen classlarda farklılaştırarak (aynisi isimizi görmüyor ancak temel prensip aynı) tekrar yazmaya Override etmek denir.
34. **INHERITANCE(Abstract):** nesneleri kategorize etmek maksadıyla kullanılır. Ortak özellik veya ortak işlemleri ayrı bir Class halinde tanımlayıp bu özelliği kullanacak diğer Class'ların bu Class'a inherit ederek işlemi yapmasına imkan veren yapıdır. Ortak özelliğini içeren Base sınıf olayı inherit'dir.
35. **Interface :** Ortak Classe yer alan bir özellik diğer tüm Classlarda kullanılacaksa ancak özelliğin kullanımında farklılıklar varsa kullanılır. INTERFACE BIR REFERANS TUTUCUDUR. Interfaceleri birbirinin alternatifi olan ancak kod içerikleri farklı olan durumlar için kullanırız. Bu örnekte kredi türlerinin hepsinden farklı şekilde de olsa hesaplamalar vardır. C#(.Net) Interface tanımlada ismin başına "I" harfi ekleniyordu. Interface şablondur. Metodlar boş bir şekilde tanımlıdır. Implement eden Class'lar override ile metoda kodları ekler.

36. Constructor, kesinlikle bir metottur. Ama herhangi bir değer döndüren metot değildir. Ya da void metot olarak da düşünülemez. Constructor, yalnızca üyesi bulunduğu class'dan nesne üretimi sırasında çalışacak olan metoddur. (newlendiginde Heap'te oluşma esnasında) Constructor, classdan instance alınırken çalışır ve amacı, class üyelerinizin değerlerini ayarlayarak nesne referansına geçirir. Constructor yapısı bir class'in içindeki metodların doğru calisması için gerekli parametreleri programcinin girmesini zorunlu kıracak yapıyı kurmamızı da sağlar. Böylece muhtemel hatalar engellenmiş olur. Constructor'da girilmesi zorunlu parametreleri belirtip class içindeki private tanımladığımız aynı tipteki degiskene esleriz. Böylece is metodları o parametresi kullanarak çalışır.

37. Intelligence: Kod programlarında kodun devamında nokta sonrası olabilecek muhtemel özellikler(hazır veya kullanıcı tanımlı class, metod, degisken vb) gösteren pencereye verilen addır.

Frontend Notları

1. Ferfef

2.

Ödev Proje: Araba Kiralama Sistemi

7. Ders Ödevi: ReCapProject adlı yeni bir Proje oluşturun. (Tekrar ve geliştirme projesi)

Entities, DataAccess, Business ve Console katmanlarını oluşturunuz.

Bir araba nesnesi oluşturunuz. "Car"

Özellik olarak : **Id, BrandId, ColorId, ModelYear, DailyPrice, Description** alanlarını ekleyiniz. (Brand = Marka)

InMemory formatta **GetById, GetAll, Add, Update, Delete** operasyonlarını yazınız.

Consolda test ediniz.

Önemli: Copy-Paste yasak fakat kamp projesinden destek almak serbest.

Kodlarınızı Github'a aktarıp paylaşınız. İncelediğiniz arkadaşlarınıza yıldız vermeyi unutmayın.

8. Ders Ödevi: Car nesnesine ek olarak;

1) Brand ve Color nesneleri ekleyiniz(Entity)

Brand-->Id,Name

Color-->Id,Name

2) Sql Server tarafından yeni bir veri tabanı kurunuz. Cars, Brands, Colors tablolarını oluşturunuz. (Araştırma)

3) Sisteme Generic IEntityRepository altyapısı yazınız.

4) Car, Brand ve Color nesneleri için Entity Framework altyapısını yazınız.

5) GetCarsByBrandId, GetCarsByColorId servislerini yazınız.

6) Sisteme yeni araba eklendiğinde aşağıdaki kuralları çalıştırınız.

Araba ismi minimum 2 karakter olmalıdır

Araba günlük fiyatı 0'dan büyük olmalıdır.

9. Ders Ödevi

- CarRental Projenizde Core katmanı oluşturunuz.
- IEntity, IDto, IEntityRepository, EfEntityRepositoryBase dosyalarınızı 9. gün dersindeki gibi oluşturup ekleyiniz.
- Car, Brand, Color sınıflarınız için tüm CRUD operasyonlarını hazır hale getiriniz.
- Console'da Tüm CRUD operasyonlarınızı Car, Brand, Model nesneleriniz için test ediniz. GetAll, GetById, Insert, Update, Delete.
- Arabaları şu bilgiler olacak şekilde listeleyiniz. CarName, BrandName, ColorName, DailyPrice. (İpucu : IDto oluşturup 3 tabloya join yazınız)
- Kodlarınızı Github hesabınızda paylaşır link veriniz
- Başkalarının kodlarını inceleyiniz. Beğenirseniz yıldız veriniz.

Not : İsteyenler Northwind projesindeki Core katmanını da ekleyebilir ama pekiştirmek için yeniden yazmanız öneririm. Bu şekilde yapmak isteyenler CarRental/Solution Explorer Sağ Tık / Add /Existing Project/ Northwind içindeki Core klasöründe Core.csproj dosyasını ekleyebilirler. Bu şekilde yapanlar aşağıdaki 3. adımdan devam edebilirler.

10. Ders Ödevi Core katmanında Results yapılandırması yapınız.

Daha önce geliştirdiğiniz tüm Business sınıflarını bu yapıya göre refactor (kodu iyileştirme) ediniz.

Sonra;

- Kullanıcılar tablosu oluşturunuz. Users-->Id, FirstName, LastName, Email, Password
- Müşteriler tablosu oluşturunuz. Customers-->UserId, CompanyName
*****Kullanıcılar ve müşteriler ilişkilidir.
- Arabanın kiralama bilgisini tutan tablo oluşturunuz. Rentals-->Id, CarId, CustomerId, RentDate(Kiralama Tarihi), ReturnDate(Teslim Tarihi). Araba teslim edilmemişse ReturnDate null'dır.
- Projenizde bu entity'leri oluşturunuz.
- CRUD operasyonlarını yazınız.
- Yeni müşteriler ekleyiniz.
- Arabayı kiralama imkanını kodlayınız. Rental-->Add
- Arabanın kiralanabilmesi için arabanın teslim edilmesi gerekmektedir.

11. Ders Ödevi CarRental projenizde;

- WebAPI katmanını kurunuz.
- Business katmanındaki tüm servislerin Api karşılığını yazınız.
- Postman'de test ediniz.

12. Ders Ödevi Car Rental Projenize Autofac desteği ekleyiniz.

13. Ders Ödevi RentACar projenizde;

- CarImages (Araba Resimleri) tablosu oluşturunuz. (Id,CarId,ImagePath,Date) Bir arabanın birden fazla resmi olabilir.
- Api üzerinden arabaya resim ekleyecek sistemi yazınız.
- Resimler projeniz içerisinde bir klasörde tutulacaktır. Resimler yükleniği isimle değil, kendi vereceğiniz GUID ile dosyalanacaktır.
- Resim silme, güncelleme yetenekleri ekleyiniz.
- Bir arabanın en fazla 5 resmi olabilir.
- Resmin eklendiği tarih sistem tarafından atanacaktır.
- Bir arabaya ait resimleri listeleme imkânı oluşturunuz. (Liste)
- Eğer bir arabaya ait resim yoksa, default bir resim gösteriniz. Bu resim şirket logonuz olabilir. (Tek elemanlı liste)

Resim ekleme metotları için Microsoft.ASPNETCORE.HTTP kurulmalıdır (Business, DataAccess ve Entities Katmanları için)

- Kaynaklar

- <https://www.c-sharpcorner.com/UploadFile/mahesh/create-a-file-using-fileinfo-in-C-Sharp/>

<https://docs.microsoft.com/tr-tr/dotnet/api/system.io.file?view=net-5.0>

<https://docs.microsoft.com/tr-tr/dotnet/api/system.guid.newguid?view=net-5.0>

<https://docs.microsoft.com/tr-tr/aspnet/core/mvc/advanced/custom-model-binding?view=aspnetcore-5.0>

14. Ders Ödevi

NOT:*****Daha önce Entity katmanında oluşturduğum User nesnesini sildim ve Core içinde User nesnesi oluştururdum. Cümkü User tüm projelerde kullanılabilecek bir nesne. Ona baglı olarak IUserService ve UserManager ve WebAPi katmanındaki UserCOntroller, IUserDal,EfUSerDal vb. degisti Yanında Auth adlı nesne için gerekli Service,Manager. DTO vb. oluşturuldu User basit veri tabanı işlemlerini yapacak, Auth ise giriş konrolerini yapacak UserController silindi. Yerine AuthController işi görecek

15. Ders Ödevi Cache, Transaction ve Performance aspectlerini ekleyiniz.

17. Ders Ödevi

- Angular projesi oluşturunuz
- Bootstrap entegrasyonu yapınız
- Markaları listeleyiniz
- Renkleri listeleyiniz
- Müşterileri listeleyiniz
- Arabaları listeleyiniz. (Arabaları listelerken BrandId yerine BrandName, ColorId yerine ColorName şeklinde gösteriniz)
- Kiralamaları listeleyiniz (Rentals) CarId yerine BrandName, CustomerId yerine FirstName + LastName şeklinde gösteriniz

18. Ders Ödevi

- Brand listesinde herhangi bir marka seçildiğinde, o markaya ait arabaları listeleyiniz.
- Color listesinde herhangi bir renk seçildiğinde, o renge ait arabaları listeleyiniz.
- Car listesinde bir arabaya tıklandığında o arabaya ait detay sayfası oluşturunuz. Bu sayfada bu araca ait resimleri de gösteriniz.

19. Ders Ödevi

- Car, Brand, Color için pipe ile arama desteği ekleyiniz.
- Car sayfasına 2 adet açılır kutu ekleyiniz. Html-Select Option. Bu açılır kutularda sırasıyla Marka ve Renk listeleyiniz.
- Açıılır kutuların yanına "Filtrele" butonu ekleyiniz.
- Filtrele butonuna tıklandığında apiden ilgili filtreye uygun arabaları listeleyiniz.
- Araba detay sayfasında "Kirala" butonu ekleyiniz. Bu aracı kiralayabilecek sistemi yazınız. Araba hali hazırda başkası tarafından seçilen tarih aralığında kiralanmışsa, kiralama işlemi yapmayınız.
- Kiralama işleminde tarihler seçildikten sonra, yeni bir sayfada kredi kartıyla ödeme desteği getiriniz.
- Ödeme işlemi için api'de sahte bir banka servisi yazınız.
- Tüm işlemler için Toastr desteği ekleyiniz.

20. Ders Ödevi

- Backend Custom Error Middleware ekleyip fluent validation için refactoring yapınız.
- Reactive Forms kullanarak Brand, Color, Car Ekleme sayfalarını oluşturunuz.
- Brand,Color,Car listesinde güncelleme butonu ekleyiniz. Tıklanan ilgili elemanın detay sayfasına yönlendirerek güncelleme imkanı veriniz.
- Toast desteği veriniz.