# Asp.Net MVC Sadık Turan

<https://www.youtube.com/watch?v=vQwnOUAhh_g&list=PLXuv2PShkuHx8VavFvqVGWZ4HjQFgf5wo&index=2>

Solution: İçerisinde birçok platform yada proje barındırabilen platform.

ASP.NET WEB APPLİCATİON(.NET Framework) C# >Empty ama MVC seçili bir şekilde projeyi oluştur.

Controller aracılığıyla kullanıcın istekleri karşılanır.

**Controller:** Controller klosörünün içerisine Sağ Tık>Add>Controller ekle. MVC 5 Empty Controller ekle ve o kısımdaki “Default” yazan yere isim gir (controller kısmı olmak zorunda onun önüne isimi gir!).

İçersinde Action metot bulunan bir controller oluşturuldu. Bu kısımda geriye ne döndüreceksek(return) ActionResult kısmına return verinin datatype’ını yazıyoruz.

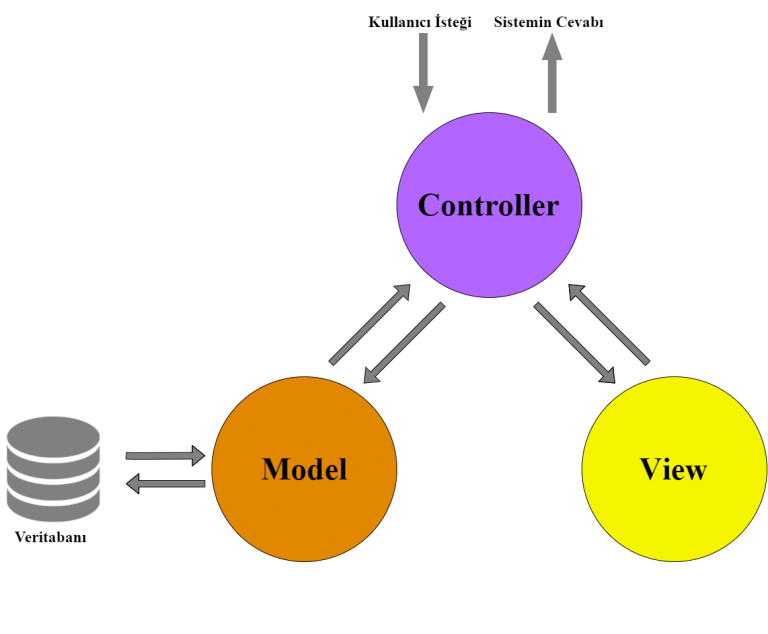
Return yaptığında dikkat et ilk varsayılan değer döndürülmektedir. App\_Start altındaki RouteConfig altında bulunmaktadır.

Localhost:1212/urun/urunDetay

Localhost:1212/Home/about

Localhost:1212/Home/contact

Gibi controller ile yukarıdaki gibi site URL yapılabilir.



View: Kendisine gelen verileri oluşturduğumuz bir yapı içerisinde derleyip bir HTML sayfayı oluşturur.

**View**  kısmında biz public ViewResult UrunDetay() burada UrunDetay Sağ Tık > add View diyerek bir cshtml sayfası oluşturuyoruz. Ve çalıştırdığımızda default olan View bize cshtml yani sourcecode kısmında bize html sayfası olarak kullanıcıya döner.

Bu kısımda bir theme’daki kodları kopyalayıp kendi oluşturduğumuza layout kısmını silmeden yapıştırıyoruz. Daha sonra oradaki css , js ve img klosör yollarını ctrl+h yani bul ve yaz kısmı ile doğru yol şeklinde değiştiriyoruz.

Layout:Bu kısım sitenin değişmeyecek kısımlarını tanımlıyoruz ve bu kısımları birçok sayfada kolayca kullanabiliyoruz **değişen kısım ise** RenderBody() işaretlediğimiz her yer değiştirilebilir alan oluyor. Sonuç layoutlarda css vesaire eklemiyoruz sadece RenderBody kısmını istediğimiz şekilde yapabiliyoruz.

**Layout ekleme:** View klasörü içerisine **Shared** isimli klasör oluşturmak. Shared sağ tık> add>New İtem>MVC 5 Layout şeklinde ekliyoruz.

2.Diğer ekleme ise View klasörüne View eklerken o kısımda layout kutusunu tik’lersek burada otomatikmak shared klasörü oluşacaktır.

Bu template’te nagivation ve footer kısmı için layout yapıyoruz. Bunun için layout sayfası oluşturduk. İçerik için Home/Index kopyaladık ve o’nun nav ve footer kısmı hariç içeriği sildik.

Bu kısımda bir layout oluşturduk.orada nav ve footer kısımları bulunmaktadır.CSS ve Js ler bulunuyor. İndex sayfamızı layout için nav ve footer kısmını siliyoruz.head kısmı body kısmı doctype html kısmını siliyoruz sadece o sayfaya özel değişebilir olan alanları yani MyCarousel , section gibi div elemlerini bırakıyoruz böylece bu işi layout yapmış oluyoruz.

Google ile template inleceden nav background özelliğini değiştiriyoruz daha sonra kendi css dosyamızın içerisinde özelliği bulup onun değerini değiştiriyoruz.

O sırada css dosyasından değiştirdikten sonra Run kapatmadan siteyi aç yenileme tuşuna bas ve(f12 açıkken yani incele varken) önbelleği boşat ve tam yeniden yükleye tıkla.

**Eğer**  views ‘lerimizin içerisini herhangibir layout seçmezsek.Bu viewsler otomatik olarak viewStart’taki olanı kullanır.

@{Layout=null } bu şekilde bir views kodlama yaparsak herhangi bir layout kullanmaz.

Yeni bir layout oluşturup ve yeni views oluşturuken layout kısmında ilgili layoutu seçersek bu layout kullanılacaktır.

ViewStart**: Bu kısım DEFAULT**  layout ayaralamamıza yarar.

PARTİAL VİEWS**:**Sayfaların belli bölümlerini parçalara ayırıp bu bölümleri tekrar başka sayfalarda kullanabilmek. Bir değişlikyapmak istediğimizde gidip her sayfa için değişlik yapmak yerine sadece partial kısmında değişiklik yaparak işlem kısaltılabilir.

Shared klasörünün sağ tık> içersinden mvc 5 partial page tıklıyoruz. Gelen sayfayı code yapıştırıp. **@Html.Partial(“name”) bu şekilde istediğimiz** sayfada partial çağırıp kullanabiliyoruz.

Linkler**:** Viewleri(html sayfalarını) birbirleriyle ilişkilendirme.”

“/“ anadizin çalıştırtılacak. Linkleri ayarlarken “bulunduğu controller ismi / bu controllerın altında hangi view is onun ismi. Özet “/Home/Contact” gibi.Önce anadizine çıkıyoruz ordan home seçiyor ve onun alıntındanda ilgili sayfayı yazıyoruz./home/privacy.

**Başka bir metot**  @Html.ActionLink(“Anasayfa”,”Index”,Home)

@Html.ActionLink(“Blog Listesi”,”Index”,”Blog”)

@Html.ActionLink(“Blog Detay”,”Details”,”Blog”)

1.parametre oluşcak olan linkin text’i

2.parametre action ismi

3.parametre controllerın ismi

///NOTE : Diyelimki Novels altında olan view kenarına türler diye bir alan tanımladık bunların herbiri için view tanımlamaktansa onlara 2 şekilde parametre ekleyebiliriz. Birincisi rauteValue ikincisi queryString bunları daha sonra göreceğiz

RAZOR: Viewler içerisinde koşul ifadeleri döngüler vs. view içerisinde HTML etiketleri ile içerik sunabiliyoruz.

@{} bu işaretlerinin arasına server kodlarımızı yazıyoruz. Bunların arasına string bir dizi vs. c# kodları yazabiliriz. Bunları daha sonra html kısmında @if (true) gibi koşul ifadesi yazabiliriz ama @ işareti koyuyoruz önüne.

Örnek ul elemenin içerisine bir @foreach döngüsü kurup onunda içerisine li elemnti koyar ve onunda içerisine @kurs derssek forceach olan item kadar dönecek ve bize li elementinin içerisine itemleri yazdıracak. İşte yukarıda yaptığımız kodlama diline RAZOR deniyor.Aslında yeni bir dil öğrenmiyoruz sadece c# kodlarının önüne @ işareti geliyor.

Gerçek seneryoda controller veritabanına bağlarnır ve bilgileri getirir ve getirilen bilgiler bir model içerisine yani sınıf içerisine aktarılır. Bu model view’e gönderilir.

MODEL**:** Bilgi taşıma işlemi model taşır ve bilgileri bir model içirişinde bir yapı oluşturursun.

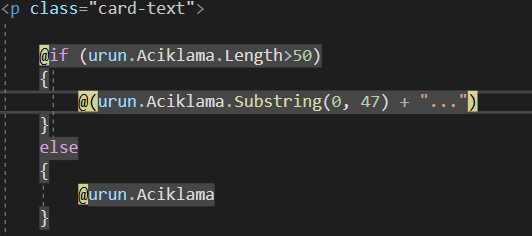
Öncelikler controller kısmında bir string dizi tanımladık ve bunu return View(kurslar); şeklinde view e döndürdük. Sayfanın bundan haberi olması lazım bu sırade devreye model giriyor . Haberi olması için” @model string[]” ( gelecek olan verinin tipini yazıoyorz.) Şimdi model tanımlayarak view’e gelecek olan veriyi bildirdik. Kullanırken @Model şeklinde kullanıyoruz. Eğer aynı satırda bir @ işareti varsa @ kullanılmaz sadece Model diyoruz.

**MODEL LİSTESİ:**Models klasörünün içerisine urun isimli bir class oluşturuyoruz. Gerekli tanımlamaları yaptıktan sonra kullanacağımız controllerda bunu List<Urun> diyerek çağırıyoruz ama çağırmak için bunlar ayrı namespace’te bulundukları için using kullanarak çağrılan using mvcDERS.Models; örnekteki gibi ekliyoruz. Yada list<Urun> dediğimizde gelen sarı lambadan ‘da using’i ekleyebiliriz.

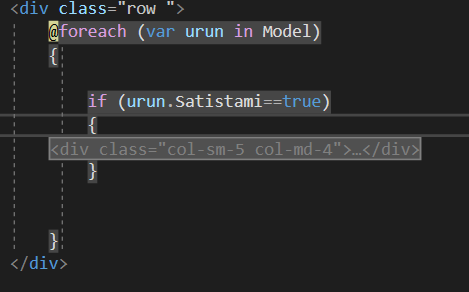
Daha sonra controller kısmında verileri dolduruyoruz ve bunu view kısmında göstermek için @model referance kullanıyoruz ama burada @model List<mvcDERS.Models.Urun> bu şekilde bir namespace +Urun ismini yazarak yada yukarıda using mvcDERS.Models; referance verip list kısmınada sadece List<Urun > yazarak kullanabiliriz.

View kısmında foreach içerisine bir card css componentinin bootstrapten direk alarak ekledik yerlerini doldurduk ve çalıştırdığımızda kaç tane urun varsa okadar bize card üretti.

Daha sonra açıklama kısmı için index razor ile bir kod yazuyoruz



Yukarıdaki kod eğer açıklama 50 karakterden büyükse sadece 47 karakteini alıp sonuna 3 nokta ekliyecek değilse normak açıklamayı ekleyecek. Bu şekilde Modeldan gelen verileri istediğimiz koşul ile index kısmında kodlayarak kullanıcıya döndürdük

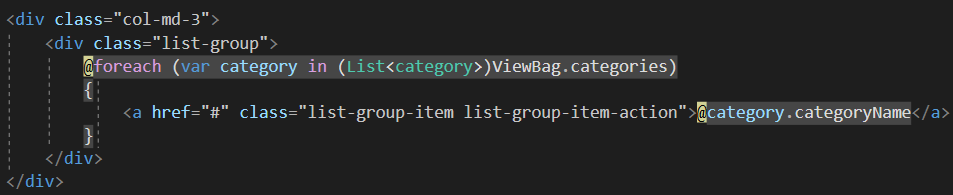


Yukarıdaki kodda eğer ürün satıştami true ise gösterecek değilse göstermeyecek.

ViewBag**:**İndex action metot içersindeki henhangibir parametreyi ilgili view taşımak için kullandığımız değişkendir. Daha önce view bir veri yolladığımızda sadece bir tipte gönderebiliyorduk. Bu kısımda viewBag bizim yardımımıza koşuyor.

ViewBag.urunSayisi = urunler.Count(); şeklinde controller kısmında tanımlıyoruz.

Bir kategori class oluşturuldu ve daha sonra orada tanımlamalar yapıldı. Conroller kısmında list tipinde veriler eklendi ve viewbag kullanıldı.Viewbag bunları genel bir tipe çevirir. Bu yüzden kullanacağımız viewde bunun liste oluduğunu belirtmeliyiz. Daha sonra @using ile viewbag yolunu belirttik.



Yukarıdaki şekilde listemizi oluşturduk.

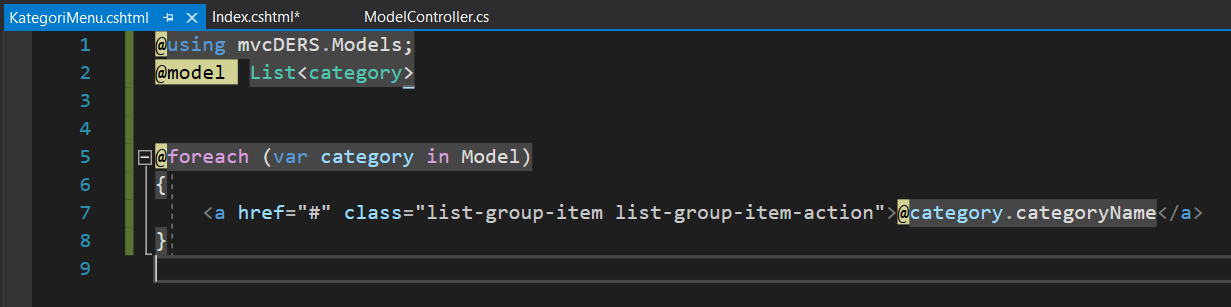
//NOTE: Aslında Yukarıda model listesi dersinde biz view içerinde true olanları göstertik ama doğru olan controller kısmında Linq ile sadece true olan verileri view yollamaktır.

MUTİPLE OBJECTS İN MODELS; Bir class’ın içersine hem kategori hemde urunler oluşturabiliriz. UrunKategoriModel isimli bir class oluşturuyoruz ve içerisine Urun ve kategori olmak üzere iki List oluşturuyoruz. Urunsayisi isimli bir değişken daha atıyoruz.

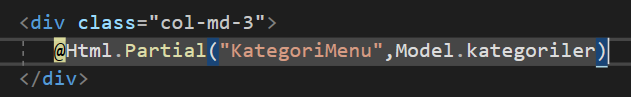
@model ve @Model farklı işlemleri vardır. Küçük harfli ile gelen verinin tipini en üste yazıyoruz. İçerik kısmında kullanırken @Model şeklinde kullanıyoruz.

Daha önce oluşturduğumuz listlerin bu model’a ekledi. Örnek: veritabanından gelseydi bu verileri model içerisine paketliyecek ve view’e yollacak. Bu şekilde verileri paketleyip view üzerinde istediğimiz gibi kullanabiliriz.

Partial View Model; **Shared**  klosöründe default layout kullanmayan ve partial view kısmı tikli bir view oluşturuyoruz.



Partial view oluşturuldu.



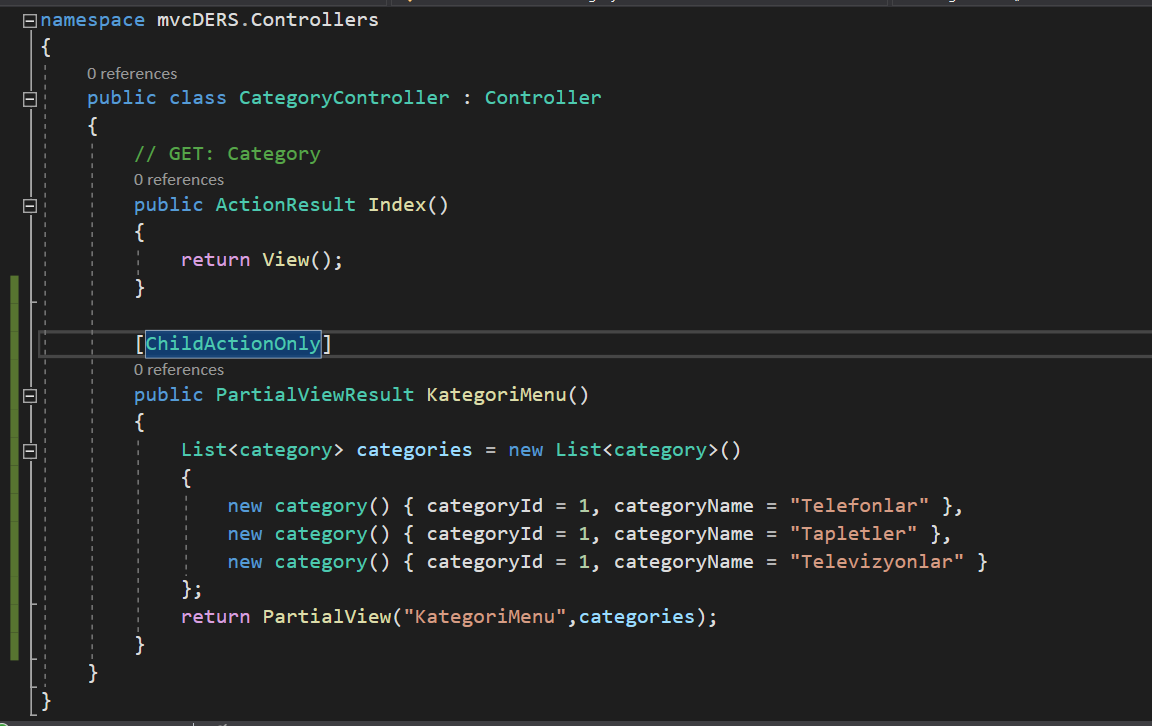
İndex kısmında model paketimizden gelen kategoriler listesini kategorimenu’de list şeklinde yollanıyor.

Yukarıda controller’dan gelen model UrunKategoriModel içerisindeki verileri html.partial ile kategoriMenu.cshtml partial view’ine gönderiyoruz. Orada List<category> şeklinde gelen verileri foreach ile yazdırıyoruz.

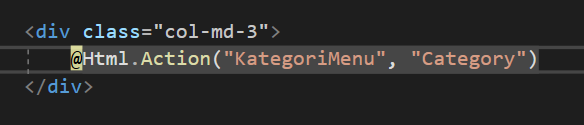
Bağımlı bir kullanım yaptık.yukarıda ayrı bir action metota tanımlarım. Biz bunu nerde yaparsak yapalım biz partial ‘a kategori göndermek zorundayız işte buna bağımlı dedi.

# -Partial Child Action

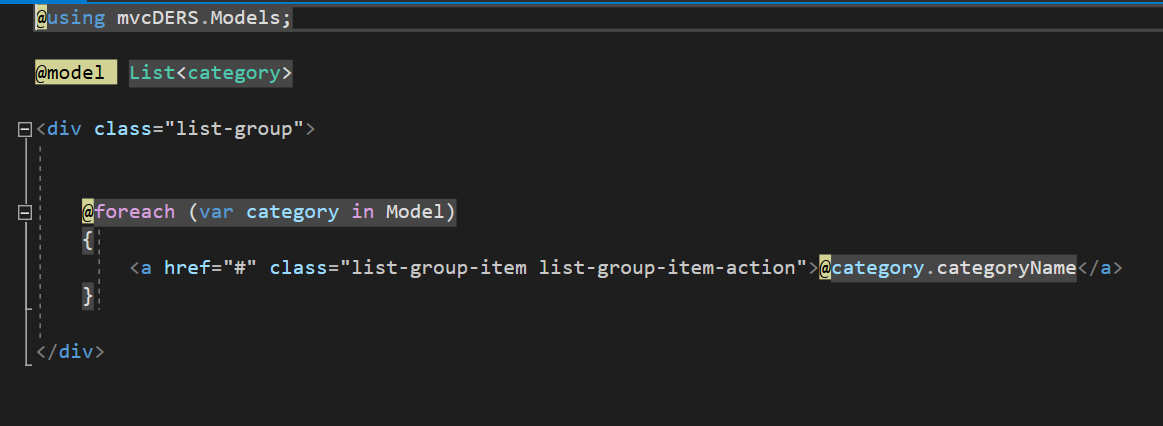
Partial view içinde bulunduğu action metotduna bağlı olduğunu verilerini o action metot üzerinden aldığını biliyoruz. Kendi başına çalışabilen bir partial view oluşturacağız.



Öncelikle bir controlle oluşturduk ve içerisine yukarıdaki metotdu tanımladık.



İndex kısmından partial view’e gönderdik.Yukarıdaki kodu nereye yazarsanız yazın orada menüler çıkar işte bağımsız olması budur.



Buda bizim partial view’imiz.

# SECTİON

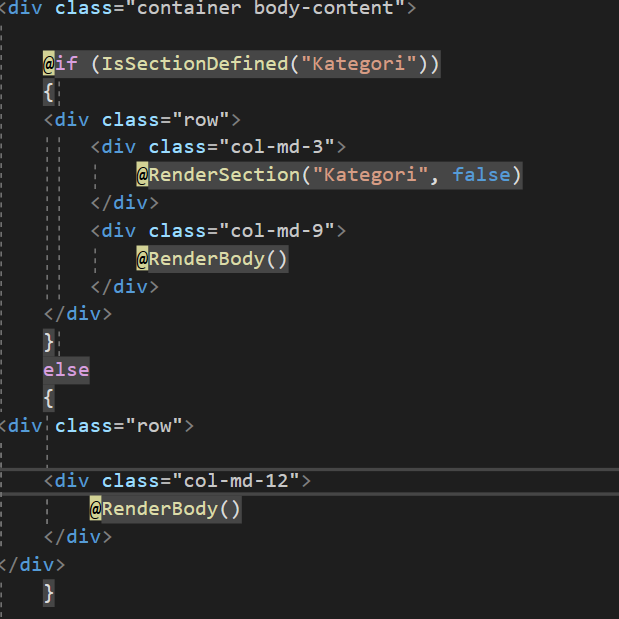
@RenderSection(“Kategori”)

Eğer yukarıdaki gibi kullanırsa her sayfada tanımlamak gerekir.

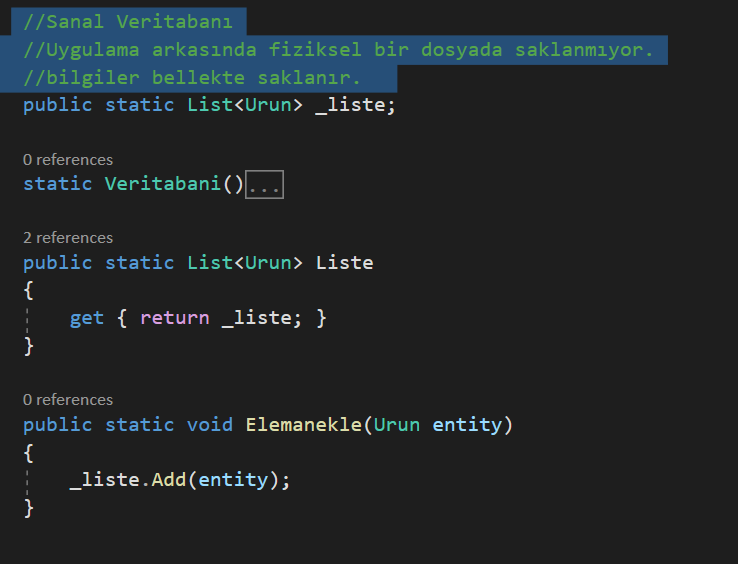
@RenderSection(“Kategori”,false) EĞER böyle yapmazsak hata alırız birçok.

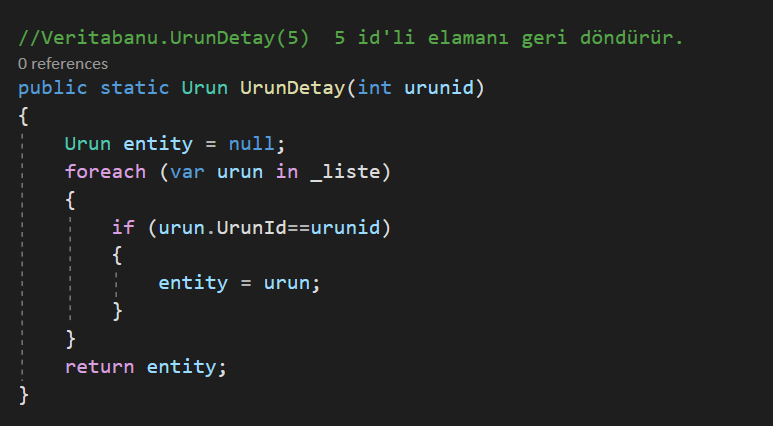
Bu şekilde section varsa tanımlanır yoksa bir sıkıntı olmaz.

ilgili sayfaya bu şekilde kategori çağırabiliriz.

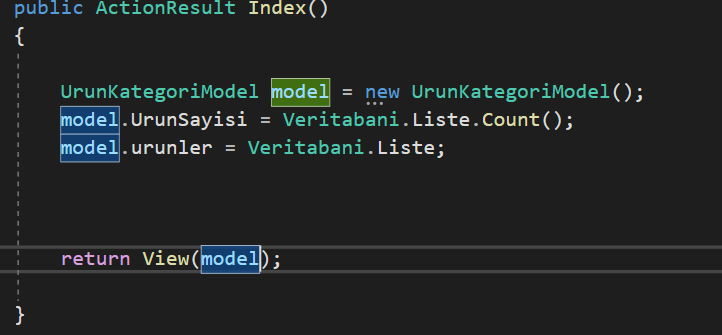
Yukarıda mesela eğer kategori section tanımlanmışsın sayfa 3-9 bölünür ama yoksa renderbody kısmı bütün sayfayı kaplar yani 12 kullanır.

# Static Veri Tabanı Sınıfı





Static elemanlarda bir instance yani nesne üretmeye gerek yok direk kullanıyoruz.



Kullanıma örnek.

# Controller Parametreleri

Şimdiye kadar controller kısmında verileri viewlere yolladık. Ama viewden gelen kullanıcı istekleri için işte bunun için parametleri öğreneceğiz.**View üzerinden controllere parametre göndermek. Querystring veya routevalue.**



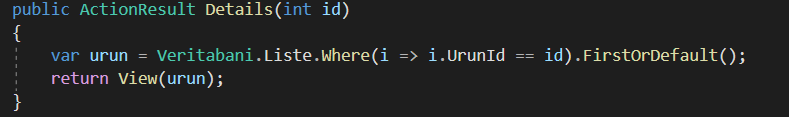
Model içerisindeki ürünler hepsi i’nin içerisine kopyalanıyor ve satıştamı true ‘mu diye kontrol edliyor true ise urunler içerisne aktarılıyor.Liste şekline çeviriyorz çünkü bizden liste bekleniyor..



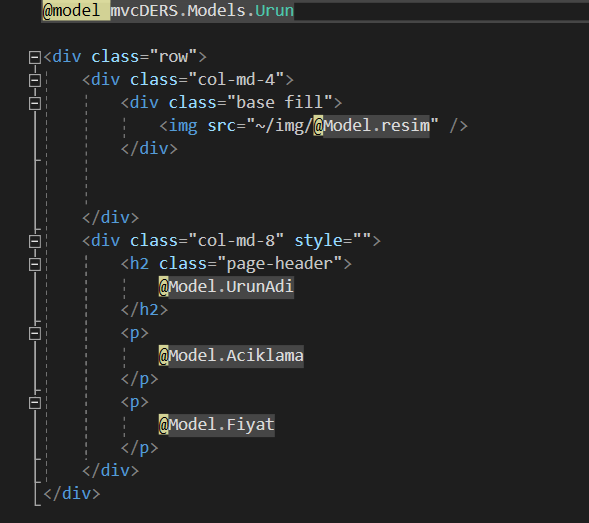
True olanların sayısını döndürür.



**RauteValue olarak :**İndex kısmında tıklanan urunun ıd’sini , Model controller altındaki details’e parametre olarak yolluyoruz yukarıdaki kod ile.



Gelen id parametre olarak alıp onu where ile liste içerisinde ilk eşleşen id’yi alıp onuda Details View’ine gönderir.



Details ize bize verinin bilgilerini döndürür.Yani butona basıldığında ürün bilgileri bu şekilde gelmiş olur.

**QueryString olarak:**

****

**Altarnatif:**

****

# Formlar

**Form aracılığıyla controller’a bilgi taşıyacağız. Bu** formu ilgili action metota gönderecez. Controller daha sonra bu form paketini açacak ve ilgili yerlere kayıt yapacak.

Forma action default olarak kendi sayfasını alır. buton bastığında verileri ekleyeceği sayfadır.

GET kullanırsan aşağıdaki gibi bir querystring şeklinde olur.

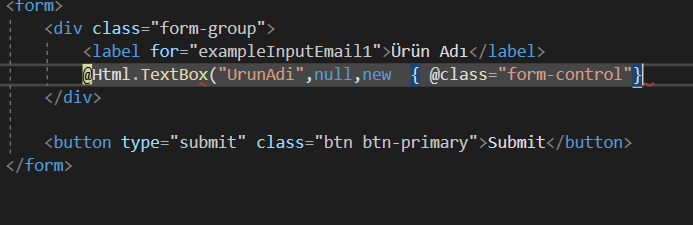


POST kullanırsan: Eğer paketlensin şifrelensin öyle gönderilsin istiyorsan kullan.

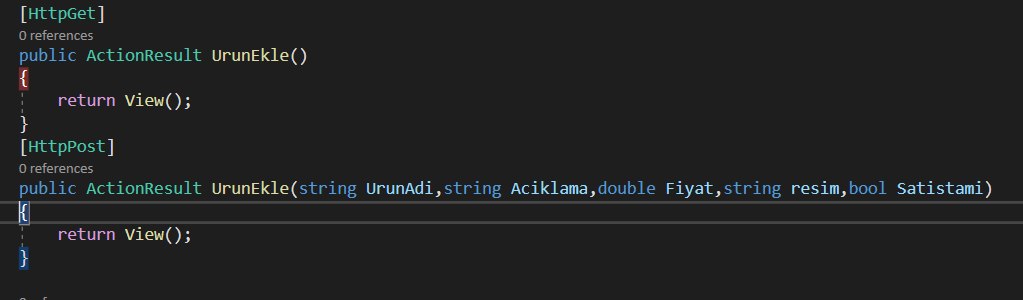


Bilgi kayıt formalarında POST kullanılır.ActionMetot/Controller,tipi yazılır.

Bu form Urun tipi paketleneceği için öncelikler Urun classında ismi neyse ona göre yapmalıyız. İkinci değer sormaktadır null dedik. Ve son olarakta bir class ataması yaptık.

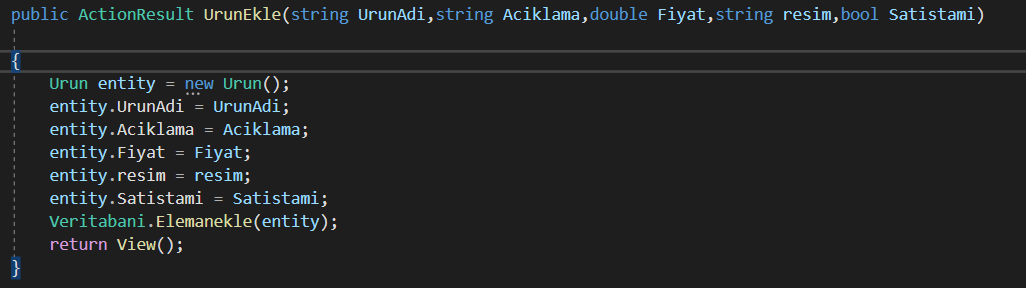


# Formu Post Etme

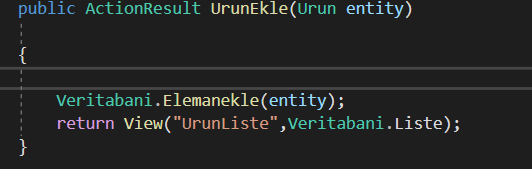


UrunEkle iki tane yapıyoruz çünkü ilk sayfanın boş halini çağıracam. İkincisinde sayfanın dolu halini çağıracam.

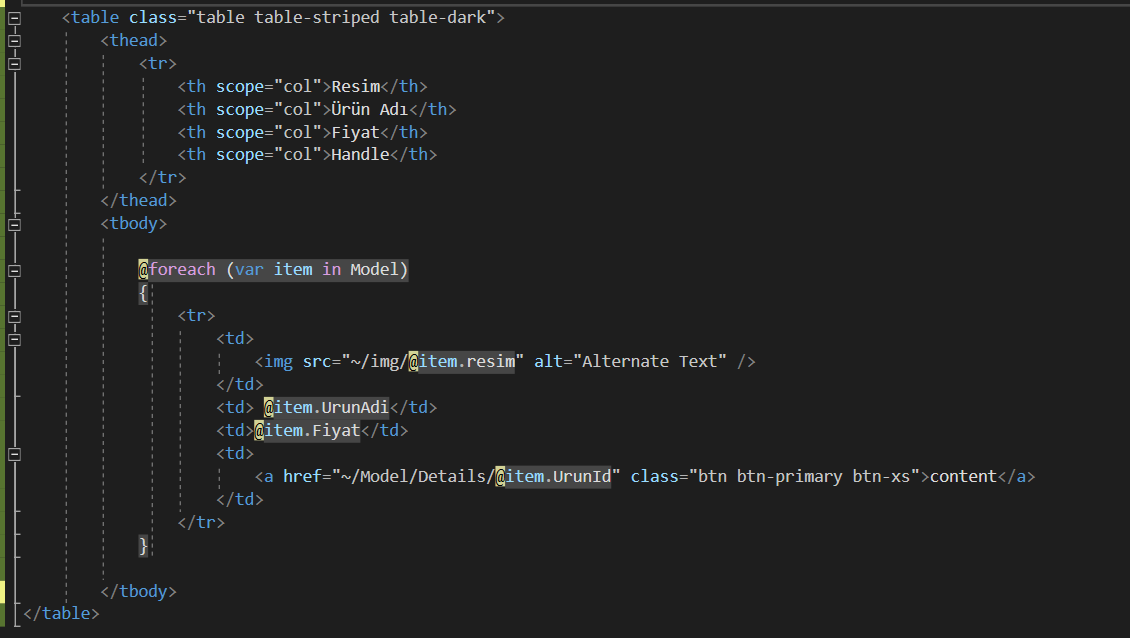
HttpPost kısmında bizim view’deki veriler nasıl buraya geliyor dersen orada textbox’lara vs. verdiğimiz değerler burada kaşılaştırılıyor aynı olanlar kopyalanıyor. Buna **model binding**  deniyor.



Yukarıdaki şekilde verileri ekleyebilirsin .



Yada yukarıdaki gibi model binding ile yapılabilir.Daha sonra ise UrunListe isimli bağımsız bir view geri döndürüyoruz.

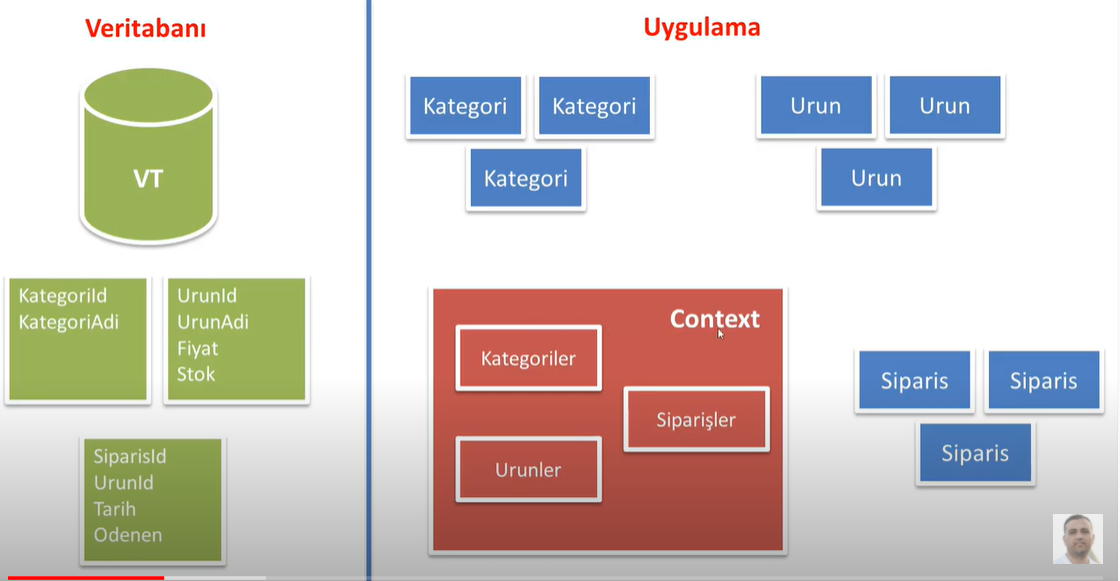


Oradaki yapıda yukarıdaki gibidir.

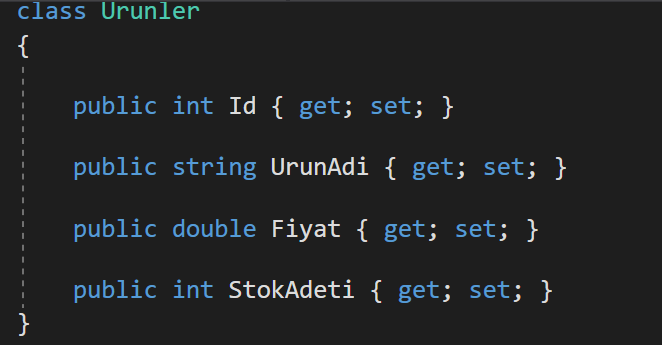
# Entity Framework Nedir?

Code ortamında sql kodlarını kolayca yazabiliyoruz. Veritabanına ulaşıp o bize sql kodlarına dönüştürüp bize istediğimizi getiriyor.

Projeye sağ tık > Manage Nuget packages > browse > entityFramework



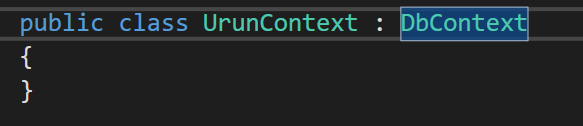
Yukarıdaki mavi şekiller entity(Kategori k1=new Kategori();) sınıfıdır bunlar context üzerinden bir işlem yaptığımızda bu veritabanına bağlanır ve işlemi yapıyor.



Bu bizim veritabanında kullanılan tablo gibi aklımıza gelsin. Şema ,Şablon vs

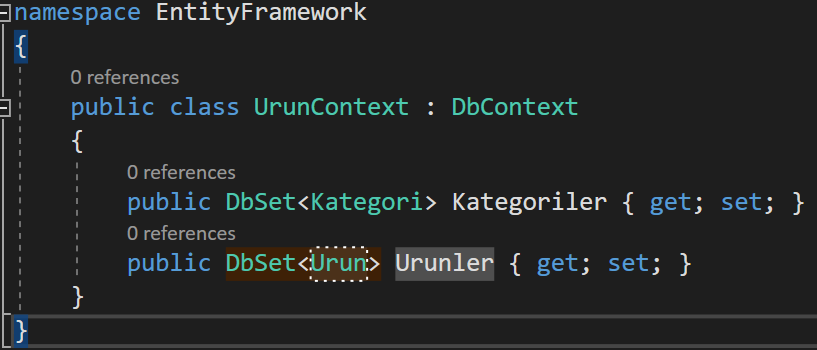
# Context

 Oluşturduğumuz classın sonuna context diyoruz ki bunun olduğunu anlayalım.

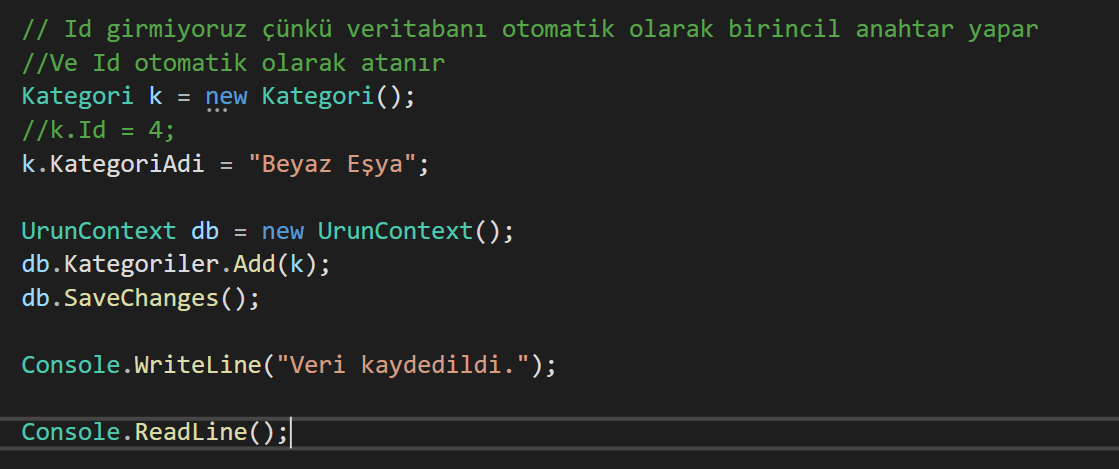


Yukarıdaki şekilde biz kendi classımıza DbContext özellikleri kullanabilmesi için **kalıtım** yapıyoruz. Ama o sırada hata çıkar ampüle tıklayıp namespace usin Data.Entity ekliyoruz.

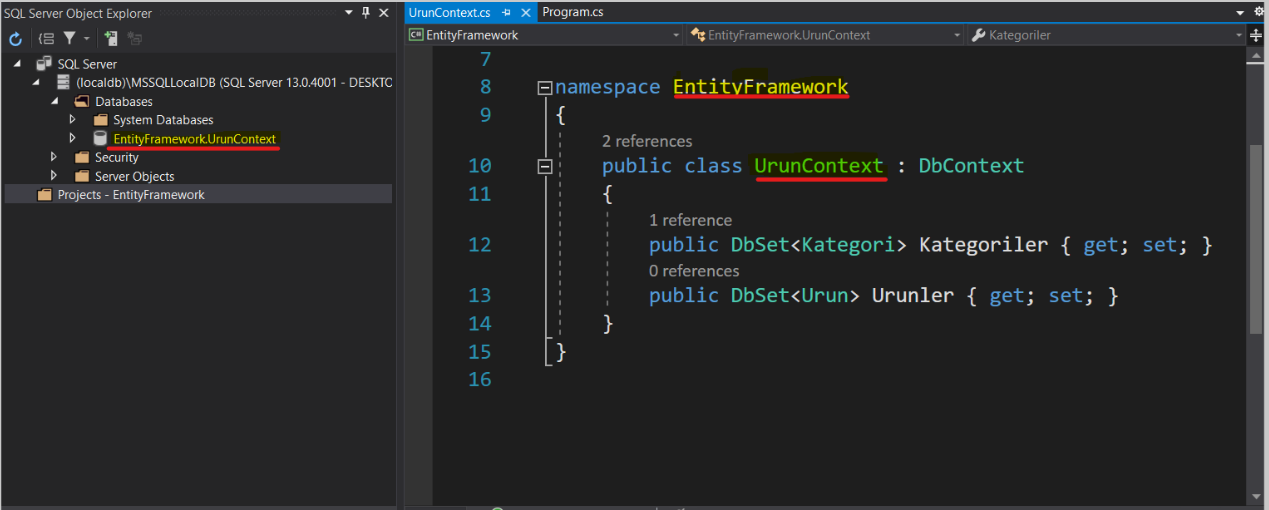
Kategori ve urun sınıfı ürettik biz bunları kalıcı olarak saklamak istiyorsak. Bu classları Context içerisine List olarak eklememiz gerekmektedir.

****

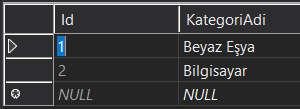
Yukarıda biz dbset değişkenli sadece kategori listesi koyabilidiğimiz yada urun bir dbset değişkenli bir list oluşturduk.Eğer veritabanı varsa onu kullanır yoksa oluşturur.



Yukarıda bir veri kaydettik ama veritabanımız olmadığı için önce varsayılan yerde veritabanı oluşturdu.



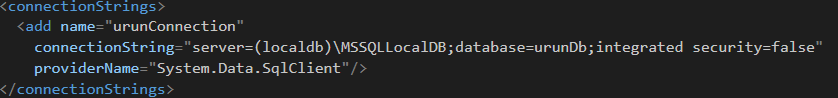
Bizim ilk başta oluşturduğumuz classların şeklinde tabo oluşturdu ve verilerimizi kaydetti.



# ConnectionString

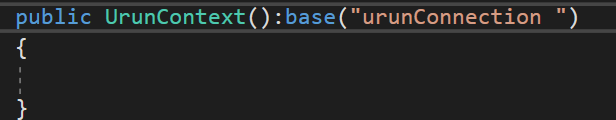
App.config veya web.config içerisinde ayarlayabiliyoruz.

.config uzantılı kısma connectionString yazıyoruz.



Config kısmında ayarı yukarıdaki gibi yaptık ama bu adresi uruncontext ile ilişkilendiremiz gerekiyor.

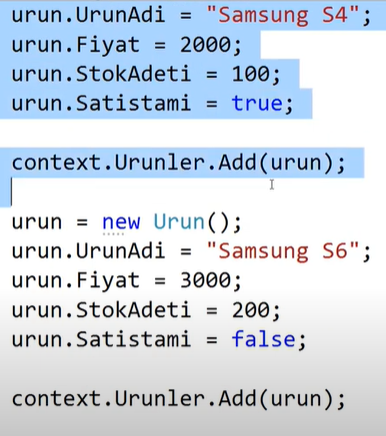
\*\*ctor constructure kısaltma.



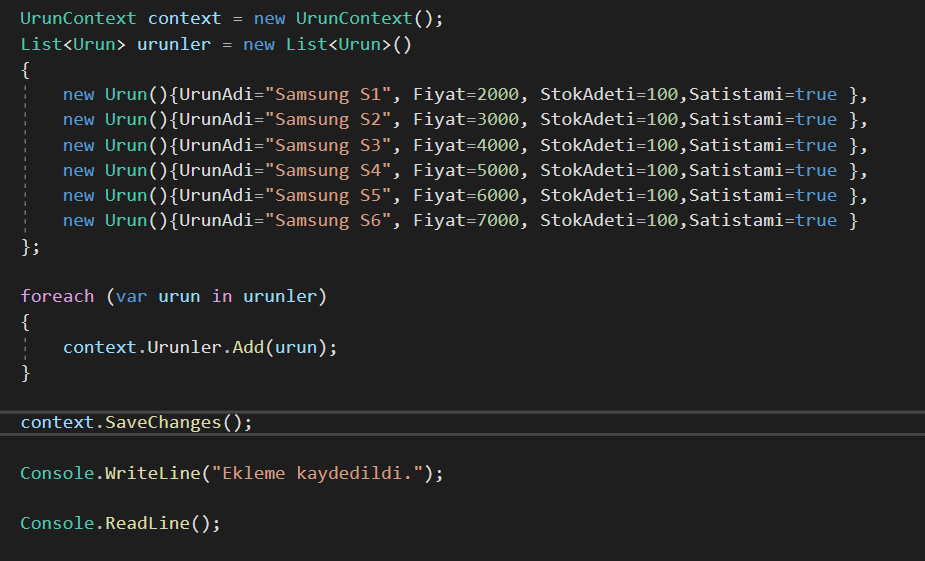
Yukarıdaki şekilde uruncontext class

# Entity Framework ile Kayıt Ekleme

Aşağıdaki şekilde nesne türetip ekleme işlemi yapılabilir.

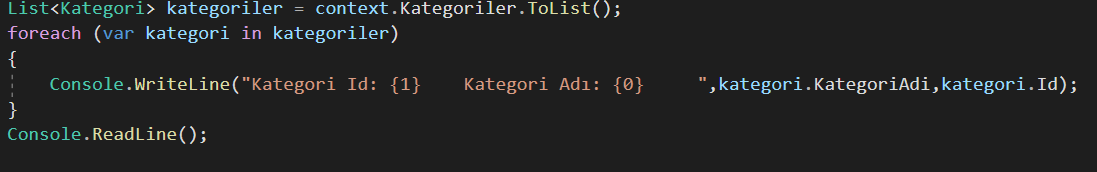


Yada liste içersine doldurup foreach ile döndük ve ekledik.

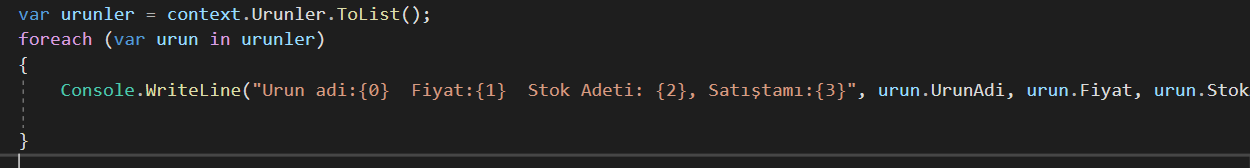


# Entity Framework ile Veri Sorgulama

Select yapmak için urunContext ‘ten bir nesne ile ilgili tabloyu çağırıyoruz.



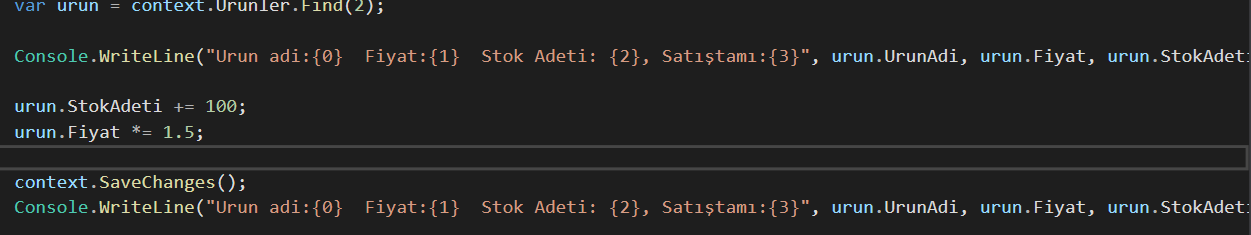
Bu şekilde bir verileri consoleda gösterdik. **List<Kategori>**  kısmını “var “ değişkeni yaparsak ona atadığımız ney ise otamatikman tipini ayarlayacaktır.



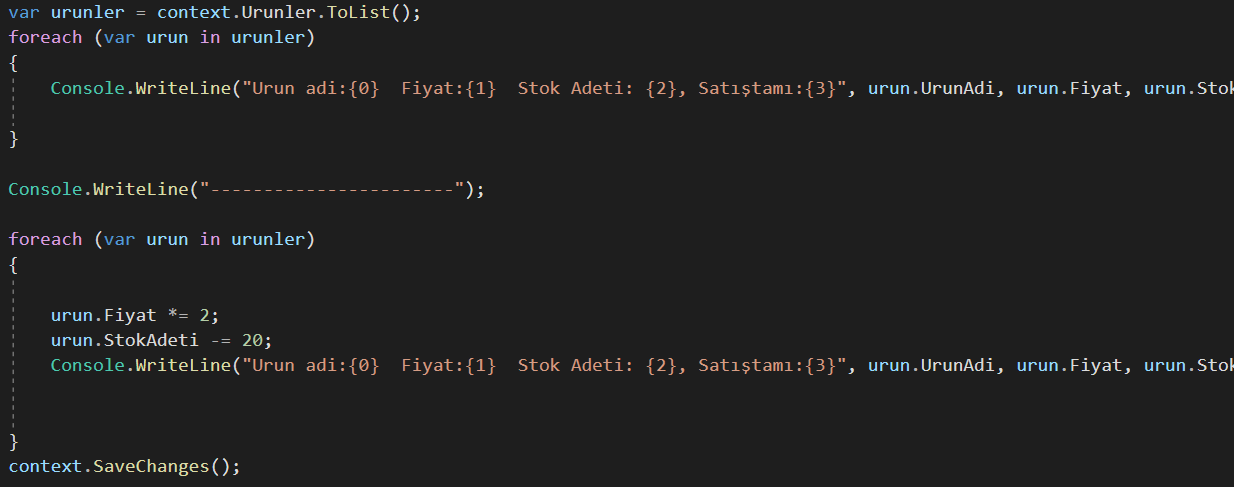
Aşağıdaki örnekte find ile sadece bir ürün olduğundan var list tipi değildir. Urun tipindedir.



# Entity Framework ile Veri Güncelleme

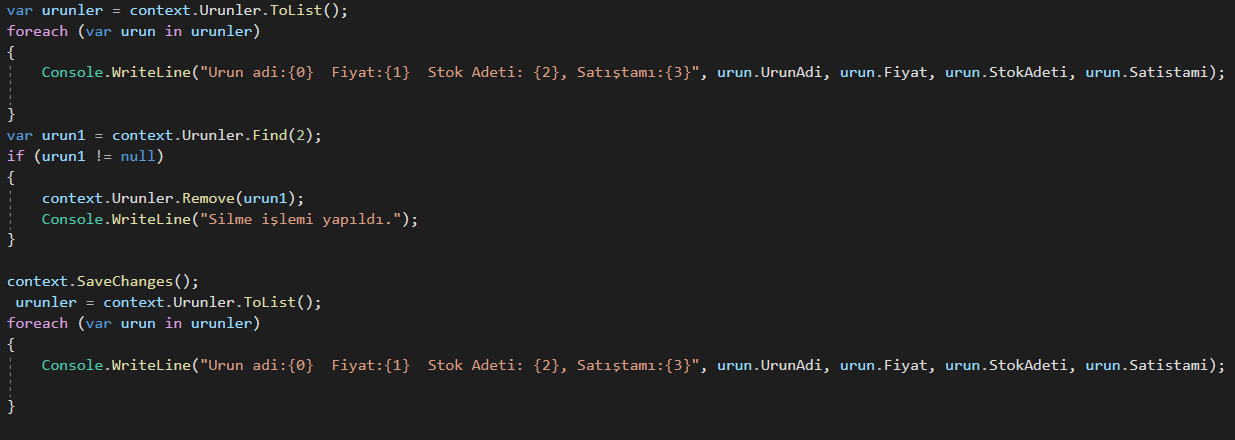


Yukarıdaki şekildeki gibi güncelleme işlemi yaptık yada



Bu şekilde ile güncelleme işlemi yapıldı.

# Entity Framework ile Veri Silme



Yukarıdaki şekildeki gibi önce silinmeden önceki bütün verileri yazdırdık daha sonra if ile kontrol ettik ve silindikten sonra bir daha yazdırdık.

\*\*\***not: savechanges**  kısmını bir döngüde kullanırsak eğer veriler çoksa birçok kez database gidip gelecektir buda işlemi uzatır .

# Entity Framework ile Initializer / Seed İşlemi

<https://www.entityframeworktutorial.net/code-first/database-initialization-strategy-in-code-first.aspx>

31.Derste bunu öğrendik bu kısımda bir class oluşturuduk.

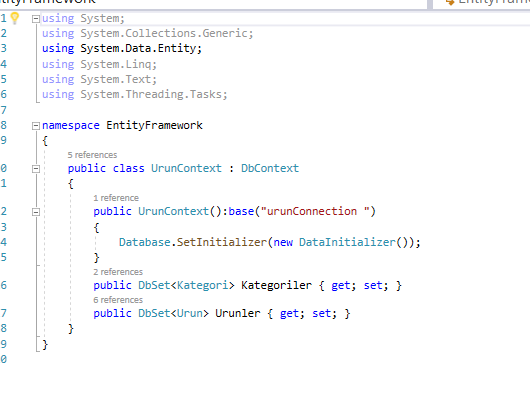
 isminde. Bu sınıfın içerisine



Yukarıdaki gibi  sınıfından DropCreateDatabaseIfModelChanges kendi DataInitializer sınıfımıza kalıtım yaptık. Bu şu işe yarıyor UrunContext belirttiğimiz için bu UrunContext içersinde bir değişiklik olduğunda database siliniyor ver birda oluşturuluyor. Tabi bunun için UrunContext kısmındada bir haber vermemiz gerekiyor.

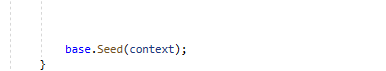
UrunContext

İçerisinde aşağıdaki gibi constructer içersinde database.setInıtiliazer yaptık yani biz UrunContext nerede çağırdığımızda yada nesne türettiğimizde ilk çalışacak olan kısımdır Constructer . Haberleşme Kısmınıda yaptık. Böylece biz eğer bu UrunContext içerisinde herhangini değişiklik yaptığımızda DataInıtializer program ilk çalıştığında database silecek ve tekrar oluşturacak böylece diğer uzun yöntemlere başvurmamıza gerek yok.



DataInitializer kısmnda bir kez daha geliyoruz. Override diyoruz ve gelen kısımdan Seed seçiyoruz ve otomatikman Seed override ediliyor.



Bu seed içerinde ise biz database sadece örneklik yani verilerimi ekliyoruz bu database her oluşturulduğunda bu verilerde otomatikman seed edilecek ve bizim bir kez daha uğraşmamız gerekmeyecekk.

# Entity Framework ile Tablolar arası ilişkiler yada classlar ile ilişki kurma

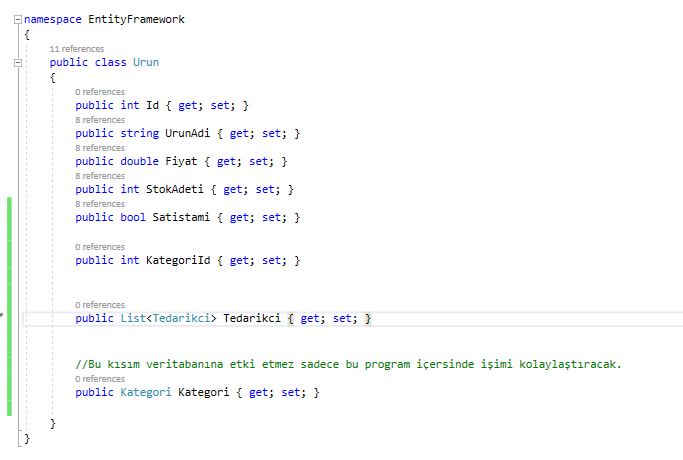
Bire çok bir ilişkilerde code-first ile bir ürün bir kategoriye ait olduğu için Urun classının içersinde oluştudurk bunu yaparkenki taktik

Önce ClassName +Id

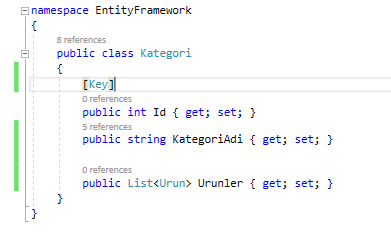
Yani Kategori+Id dedik yani Kategori sınıfının içersinden Id demiş oluyoruz.

Eğer Kategori sınıfda Id değilde başka bir isimlendirme yapıldıysa üstüne[Key] diyerek tanımlama yapıyoruz.Şu şekilde



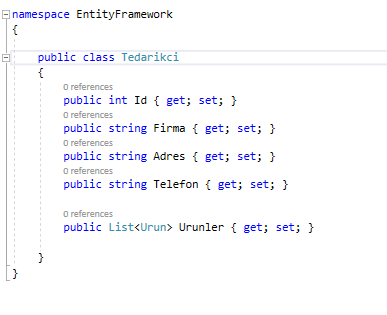


Kategori kısmında bir kategorinin sonsuz sayıda ürünü olabileceiği için bu urunleri Urun tipinde bir liste içersinde tuttuk.Böylece Kategori içerisnden ürünlere erişim sağlayabilceğiz örnek kategorisi =3 olan ürünleri getir gibi.



Çoka Çok ilişkide ise Bir tedarikçi classı yani tablosu oluşturduk.Bu da aşağıdaki şekildeki gibidir.Çoğa çok ilişkilerde ayrı bir class oluşturmamıza gerek yok. Bir tedarikçi birçok ürün sağlayabilir ve bir ürün birçok tedarikçi sağlayabilir bu yüzden bu kısımda bir urun tipinde liste oluşturuyoruz aynısınıda Urun kısmındada bir Tedarikci tipinde liste oluşturuyoruz. Biz programı çalıştırdığımızda arka tarafta 3.bir tabloyu entity otomatik olarak oluşturuyor.

Tedarikci



Urun