**MVC Proje Kampı İşlemleri**

1. **Projedeki katmanların belirlenmesi ve oluşturulması**
2. **Entity Layer :** Projenin tabloların (sql tarafındaki tabloların) ve bu tabloların içerisindeki yer alacak sütunların tanımlanacağı yerdir.

**SQL Tarafında** **C# Tarafında**

Baslıklar(Tablo) Class

BaslikId (Sütun) Property

BaslikIsim (Sütun) Property

BaslikTarih (Sütun) Property

BaslikYazar (Sütun) Property

1. **Data Access Layer (DAL) :** CRUD (Create,Read,Update,Delete), Filter (Listeleme) işlemlerinin yapılacağı yerdir.
2. **Business Layer (Is Katmani) :** DAL katmanındaki veritabanı işlemlerini sunum katmanına yansıtmadan önce, yapılacak işlemlerin yer aldığı kısımdır. Örneğin, bir ürüne dair girilen değerler üzerindeki kısıtlamaların, yetkilerin yapılacağı alanlar burada tanımlanır. Mesela, bir ürünün adi 20 karakterden fazla olamaz vb.
3. **Presentation Layer – User Interface (UI) :** Kullanıcıların gördüğü alandır.
4. **Projedeki içeriğin belirlenmesi**

**Tablolar**

Heading (Başlık - Title)

Content (içerik)

Writer (Yazar)

Category (kategori)

Contact (iletişim)

About (Hakkımda)

1. **Katmanları oluşturma**
2. **EntityLayer**
3. Class1 dosyasının silinmesi
4. **Concrete** klasörü oluştur. Somut ifadeler bu klasörde tutulur

* Class’ların tanımlanması (About,Category,Contact,Content,Heading,Writer)
* Class’ların erişim belirleyicilerinin **public** yapılması
* Class’lar içerisine Property lerin oluşturulması
* İlişkili tabloların tanımlanması (**ICollection<ilişki\_içine\_alınacak\_sınıf**> Bir koleksiyon türündeki interface)
* EntityFramework paketinin yüklenmesi (EntityLayer🡪Manage NuGet Packages…🡪Browse🡪Entity Framework🡪Install🡪I Accept)
* Using ifadesinin sayfalara yüklenmesi (using System.ComponentModel.DataAnnotations;)
* Attribute’lerin Property lere tanımlanarak girdilerin kontrolünün yapılması (PrimaryKey 🡪[Key])

**About Tablosu**

[Key]

public int AboutId { get; set; }

[StringLength(500)]

public string AboutDetails1 { get; set; }

[StringLength(1000)]

public string AboutDetails2 { get; set; }

[StringLength(100)]

public string AboutImage1 { get; set; }

[StringLength(100)]

public string AboutImage2 { get; set; }

**Category Tablosu (Kategori ve Başlık ilişkili)**

[Key]

public int CategoryId { get; set; }

[StringLength(50)]

public string CategoryName { get; set; }

[StringLength(200)]

public string CategoryDescription { get; set; }

public bool CategoryStatus { get; set; }

public ICollection<Heading> Headings { get; set; } //Bire çok bir ilişki kurulacak (Heading alanı ile ilişki kuracak)

**Contact Tablosu**

[Key]

public int ContactId { get; set; }

[StringLength(50)]

public string UserName { get; set; }

[StringLength(50)]

public string UserMail { get; set; }

[StringLength(50)]

public string Subject { get; set; }

public string Message { get; set; }

**Content Tablosu**

[Key]

public int ContentId { get; set; }

[StringLength(1000)]

public string ContentValue { get; set; }

public DateTime ContentDate { get; set; }

public int HeadingId { get; set; } //**1.ForeignKey**

public virtual Heading Heading { get; set; }

public int? WriterId { get; set; } //**2.ForeignKey (int? :** bos bırakılabilir yapıyoruz**)**

public virtual Writer Writer { get; set; }

**Heading Tablosu (Baslik-Icerik Iliskisi)**

[Key]

public int HeadingId { get; set; }

[StringLength(50)]

public string HeadingName { get; set; }

public DateTime HeadingDate { get; set; }

public int CategoryId { get; set; } //**İlişkili tablonun anahtar sütunu** ile ayni isimde olacak!!! (İlişkili sınıf Category tablosu, en çok hata alınabilen alan)

public virtual Category Category { get; set; } //**Bir sınıftan değer alacak**

public int WriterId { get; set; }

public virtual Writer Writer { get; set; }

public ICollection<Content> Contents { get; set; } //Başlık alanı (Heading) da içerik alanı (content) ile ilişkili

**Writer Tablosu (Yazar – Başlık, Yazar – İçerik ilişkili)**

[Key]

public int WriterId { get; set; }

[StringLength(50)]

public string WriterName { get; set; }

[StringLength(50)]

public string WriterSurName { get; set; }

[StringLength(100)]

public string WriterImage { get; set; }

[StringLength(50)]

public string WriterMail { get; set; }

[StringLength(20)]

public string WriterPassword { get; set; }

public ICollection<Heading> Headings { get; set; } // Bir yazarın birden fazla başlığı olabilir

public ICollection<Content> Contents { get; set; } // Bir yazarın birden fazla içeriği olabilir

1. **DataAccessLayer**
2. Class1 dosyasının silinmesi
3. **Concrete** klasörü oluşturulur. Somut ifadeler bu klasörde tutulur

* **Context** isimli bir Class oluşturulması (Bu Sınıfın Amacı: burada tanımlanan property’ler birer tablo ismi olarak Sql de karşılık bulacak)

**Not: Context : Database tabloları ile proje class’larını bağlamak için köprüdür. DbContext de bunun için EntityFramework kütüphanesinde oluşturulmuş bir class‘dir.**

* Context Class erişim belirleyicisi **public** yapılmalı ve **DbContext sınıfından kalıtım** alınmalı
* EntityFramework paketinin yüklenmesi (DataAccessLayer🡪Manage NuGet Packages…🡪Browse🡪Entity Framework🡪Install🡪I Accept)
* Property lerin **DbSet<Entity\_Katmanındaki\_Sınıf\_Adlari> Çoğul\_Sınıf\_Adı** oluşturulması
* İlgili Katman Üzerinde Sağ Click 🡪Add Reference…🡪Projects🡪EntityLayer eklenmeli
* Using ifadesinin sayfaya yüklenmesi (using EntityLayer.Concrete;)

**Context:DbContext**

public DbSet<**About**> **Abouts** { get; set; }

public DbSet<**Category**> **Categories** { get; set; }

public DbSet<**Contact**> **Contacts** { get; set; }

public DbSet<**Content**> **Contents** { get; set; }

public DbSet<**Heading**> **Headings** { get; set; }

public DbSet<**Writer**> **Writers** { get; set; }

1. **MvcProjeKampi(UI)**
2. Web Config dosyasına **ConnectionsString** Eklenmesi (</system.web> sonuna)

<connectionStrings>

<add name="Context"

connectionString="data source=Veritabani\_Server\_Ad; initial

catalog=VeriTabanimizin\_Adi; integrated security=true;"

providerName="System.Data.SqlClient"/>

</connectionStrings>

Anlamları

Name="Context" : Bu isim tablolarımızın, veritabanına yansıtılacağı sınıfın ismi ile ayni olmak zorunda

data source= : (SqlServer 🡪 Server Name = „Data\_Source“)

catalog= :DbMvcKamp

integrated security= true (windows Autentication)

providerName= : Veritabanı\_türü

1. **BusinessLayer**

**Bu katmana ait bilgiler 7) ana başlık altında islenecektir.**

1. Class1 dosyasının silinmesi
2. **Migrations ve Veri Tabanının Oluşturulması**

**Migration:** C# ile Sql arasındaki köprüye denir.

**Migration İşlemleri :**

Visual Studio 🡪View🡪Package Manager Console

PM>enable-migrations

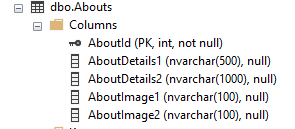
Açılan Configuration dosyası üzerinde:

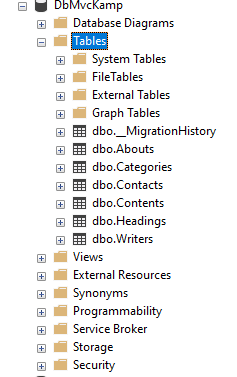
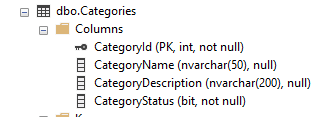
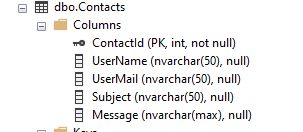
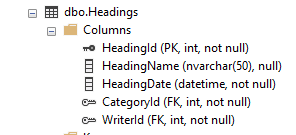
AutomaticMigrationsEnabled = true; olarak kaydedilir.

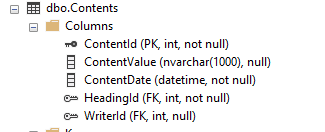
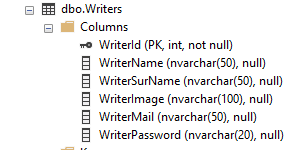
PM>Update-database çalıştırılır.

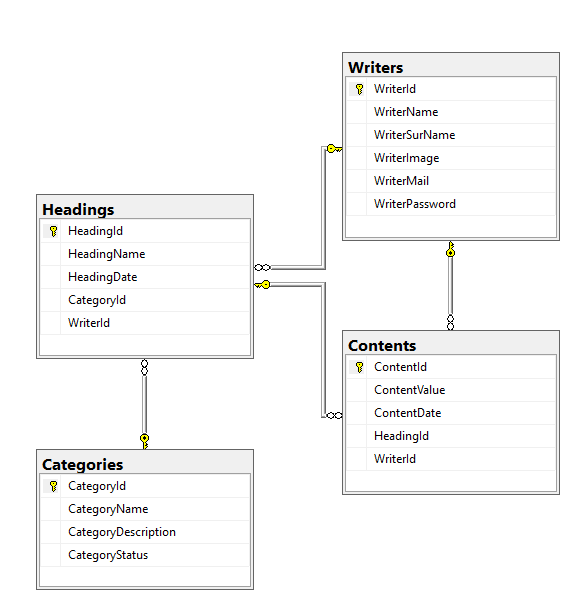
SQL server daki veritabanı, tabloları ve diyagramı kontrol edilir.

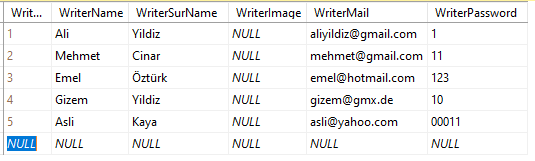
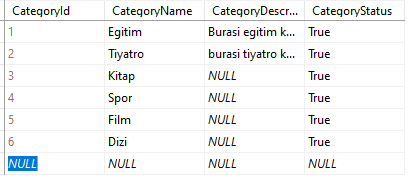
1. **Veri tabanına örnek verilerin girilmesi**

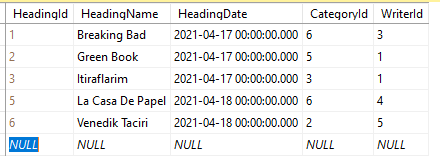


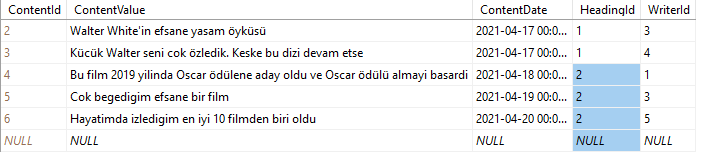






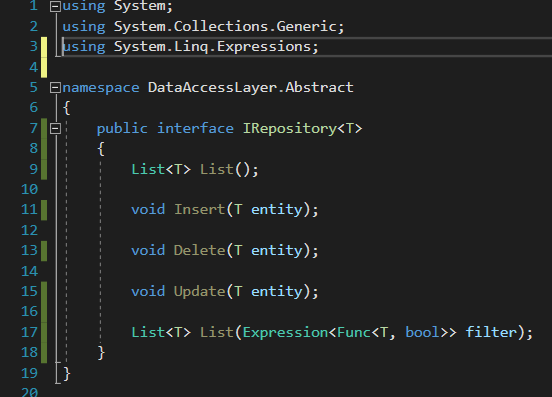




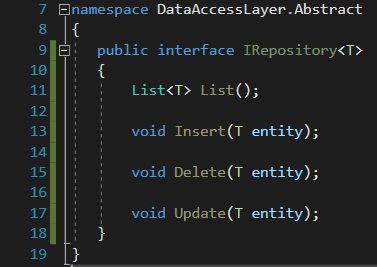


1. **DataAccessLayer Islemleri**
2. **Abstract** klasörü oluşturulur. Soyut ifadeler bu klasörde tutulur.
3. **IRepository Interface’nin olusturulmasi** 🡪**DRY Yapisi** için temel Interface

* Bu interface’in erişim belirleyicisi **public** yapılmalı
* CRUD (Create,Read(List),Update,Delete) işlemlerinin gerçekleşeceği metotlar bu katmanda tanımlanır.
* Generic List oluşturulması **(List<**Type**> Metod\_Adi();**) 🡪 (**List <T> List();**) 🡪Listeleme Metodu
* void Insert/Add (T entity) 🡪Ekleme Metodu
* void Update (T entity) 🡪Güncelleme Metodu
* void Delete (T entity) 🡪Silme Metodu
* List<T> List(Expression<Func<T, bool>> filter); 🡪 Sartli Listeleme Metodu
* Yukaridaki Expression’u kullanmak için **using System.Linq.Expressions;** ifadesi eklenmelidir.



**Not:** Interface’ler sadece deklare edilebilir, implemente edilemezler. Yani gövde içeriği olmayan bos metotlar olarak tanımlanır ve başka class’lara bu metotlar, implement edilirler. Gövde içerikleri de implement’e edildikleri sınıflarda doldurulurlar.



**Hatirlatma!**

**DRY (Don’t Repeat Yourself – Kendini Tekrar Etme) Yapısı**

Bir yerde birden fazla ayni isi yapan kodlar varsa, onları metotlara çevirin!

**Metod Yapisi**

Erisim\_Belirleyici Dönüs\_Türü Metot\_Adi (Parametre\_Listesi)

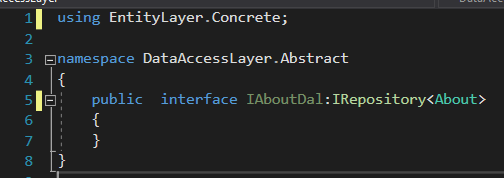
{

Metot Gövdesi

}

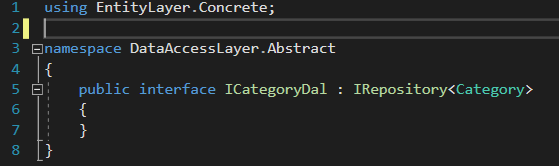
1. **IAboutDal Interface’nin olusturulmasi**

* Bu interface’in erişim belirleyicisi **public** yapılmalı
* IRepository<Entity\_Name> interface’inden kalıtım alınır 🡪 IAboutDal : **IRepository<About>**
* İlgili Entity (About)’nin kullanılabilmesi için **using EntityLayer.Concrete;** ifadesi kullanılmalıdır.



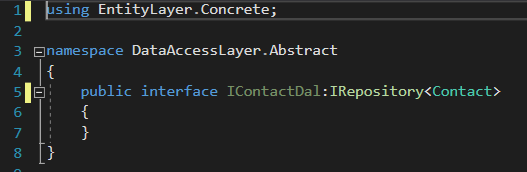
1. **ICategoryDal Interface’nin olusturulmasi**

* Bu interface’in erişim belirleyicisi **public** yapılmalı
* IRepository<Entity\_Name> interface’inden kalıtım alınır 🡪 ICategoryDal : **IRepository<About>**
* İlgili Entity (Category)’nin kullanılabilmesi için **using EntityLayer.Concrete;** ifadesi kullanılmalıdır.



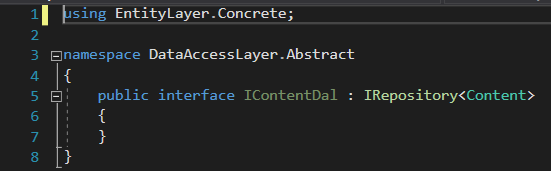
1. **IContactDal** **Interface’nin olusturulmasi**

* Bu interface’in erişim belirleyicisi **public** yapılmalı
* IRepository<Entity\_Name> interface’inden kalıtım alınır 🡪 IContactDal : **IRepository<Contact>**
* İlgili Entity (Contact)’nin kullanılabilmesi için **using EntityLayer.Concrete;** ifadesi kullanılmalıdır.



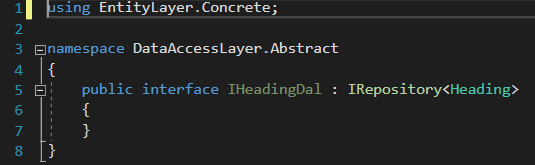
1. **IContentDal** **Interface’nin olusturulmasi**

* Bu interface’in erişim belirleyicisi **public** yapılmalı
* IRepository<Entity\_Name> interface’inden kalıtım alınır 🡪 IContentDal : **IRepository<Content>**
* İlgili Entity (Content)’nin kullanılabilmesi için **using EntityLayer.Concrete;** ifadesi kullanılmalıdır.



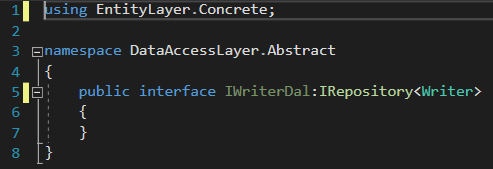
1. **IHeadingDal** **Interface’nin olusturulmasi**

* Bu interface’in erişim belirleyicisi **public** yapılmalı
* IRepository<Entity\_Name> interface’inden kalıtım alınır 🡪 IHeadingDal : **IRepository<Contact>**
* İlgili Entity (Heading)’nin kullanılabilmesi için **using EntityLayer.Concrete;** ifadesi kullanılmalıdır.



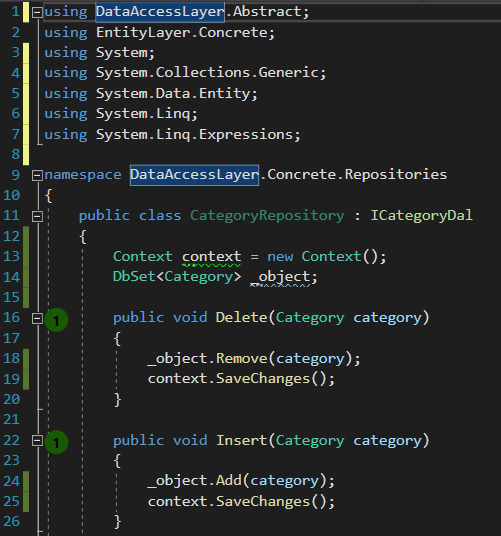
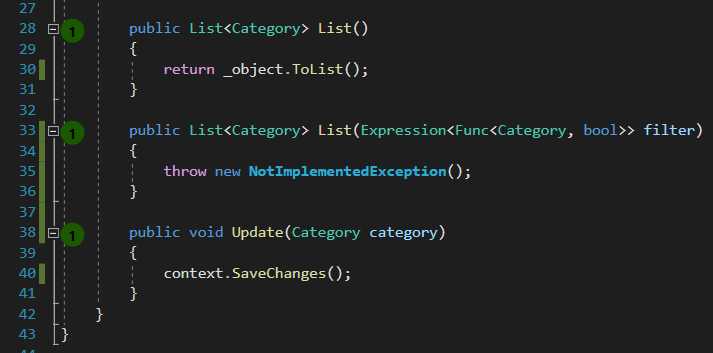
1. **IWriterDal** **Interface’nin olusturulmasi**

* Bu interface’in erişim belirleyicisi **public** yapılmalı
* IRepository<Entity\_Name> interface’inden kalıtım alınır 🡪 IWriterDal : **IRepository<Contact>**
* İlgili Entity (Writer)’nin kullanılabilmesi için **using EntityLayer.Concrete;** ifadesi kullanılmalıdır.



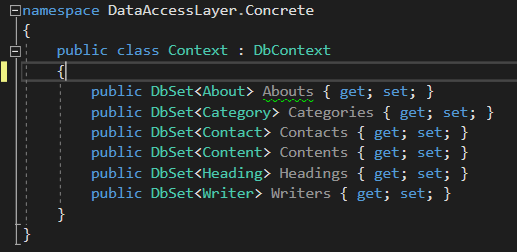
1. Concrete klasörü altına **Repositories** klasörü oluştur.
2. İlgili klasör içerisine bir **CategoryRepository** isminde bir Class oluşturulur (Bu yol **DRY Yapısına** göre geliştirilerek b) maddesindeki **Generic Repository yapısı** kullanılacaktır. a) maddesi sadece konuların mantığının anlaşılması için yapılmıştır. Aksi durumda her bir tablo için bu yol izlenecektir ki, bu da bizim için daha fazla is yükü getirecek ve zaman kaybına neden olacaktır.)

* İlgili klasör içerisine bir **CategoryRepository** isminde bir Class oluşturulur
* Bu Class’in erişim belirleyicisi **public** yapılmalı
* Abstact klasöründeki ilgili interface burada deklare edilmelidir. Bunun için önce ilgili interface using ifadesi ile çağrılmalı (**using DataAccessLayer.Abstact**), daha sonrasında da implementasyonu yapılmalıdır.
* Implemente edilen interface ait metotların gövdeleri burada ilgili işlemler için doldurulacaktır.
* Context class’indan yeni bir nesne türetilmelidir. (**Context context = new Context();**)
* DbSet<Entity\_Name> \_degisken; **🡪 DbSet<Category> \_object;** (DbSet sinifi icin **using System.Data.Entity** ifadesi eklenmeli)

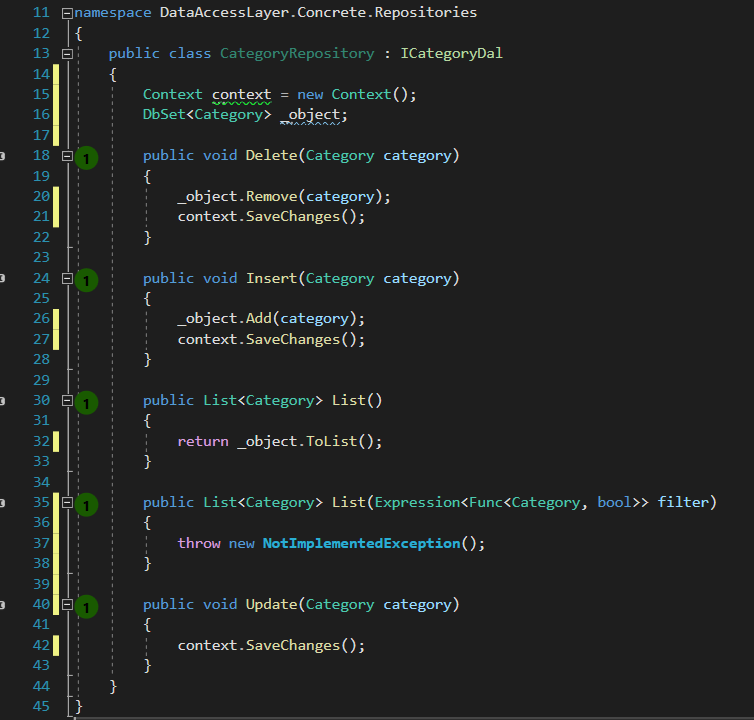
 

**Önemli Not: DbContext ve DbSet nedir, ne ise yarar?**

Sınıf ve özellikler belirlendikten sonra **System.Data.Entity** alanında yer alan **DbContext** sınıfından kalıtım alınarak veritabanı işlemlerini yapan sınıf oluşturulur.



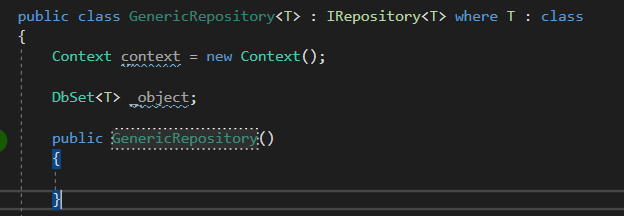
Veritabanı işlemlerinin yapıldığı sınıf içerisine model **DbSet** türünden özellik(property) olarak eklenir.Yani **DbSet** olarak tanımladığımız property (Category Entity) bizim veritabanımizdaki Category tablosunu temsil etmektedir.



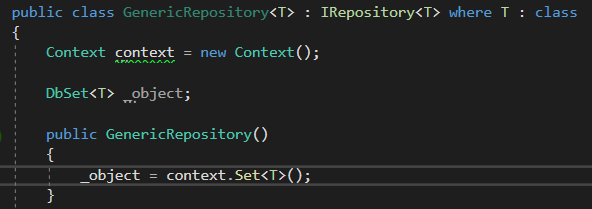
1. GenericRepository adında tüm bilgilerin bu sınıftan türetileceği bir sınıf oluşturulur.

* Bu Class’in erişim belirleyicisi **public** yapılmalı
* IRepository’i interface’nden kalıtım alınır
* Abstact klasöründeki ilgili interface burada deklare edilmelidir. Bunun için önce ilgili interface using ifadesi ile çağrılmalı (**using DataAccessLayer.Abstact**), daha sonrasında da implementasyonu yapılmalıdır.
* Class’in sonuna ilgili şart cümlesi yazılır. (public class GenericRepository<T> : IRepository<T> where T : class) ve burada tanımlanan T nesnesi bir class türünden olmalidir.
* Context class’indan yeni bir nesne türetilmelidir. ( Context context = new Context(); ).
* DbSet<Entity\_Name> \_degisken; **🡪 DbSet<T> \_object;** (DbSet sınıfı için **using System.Data.Entity** ifadesi eklenmeli)
* DbSet türündeki T değişkeni için bizim bir Yapıcı Metot (Constructor 🡪 ctor +tab+tab kısayolu ile) oluşturmamız gerekli.

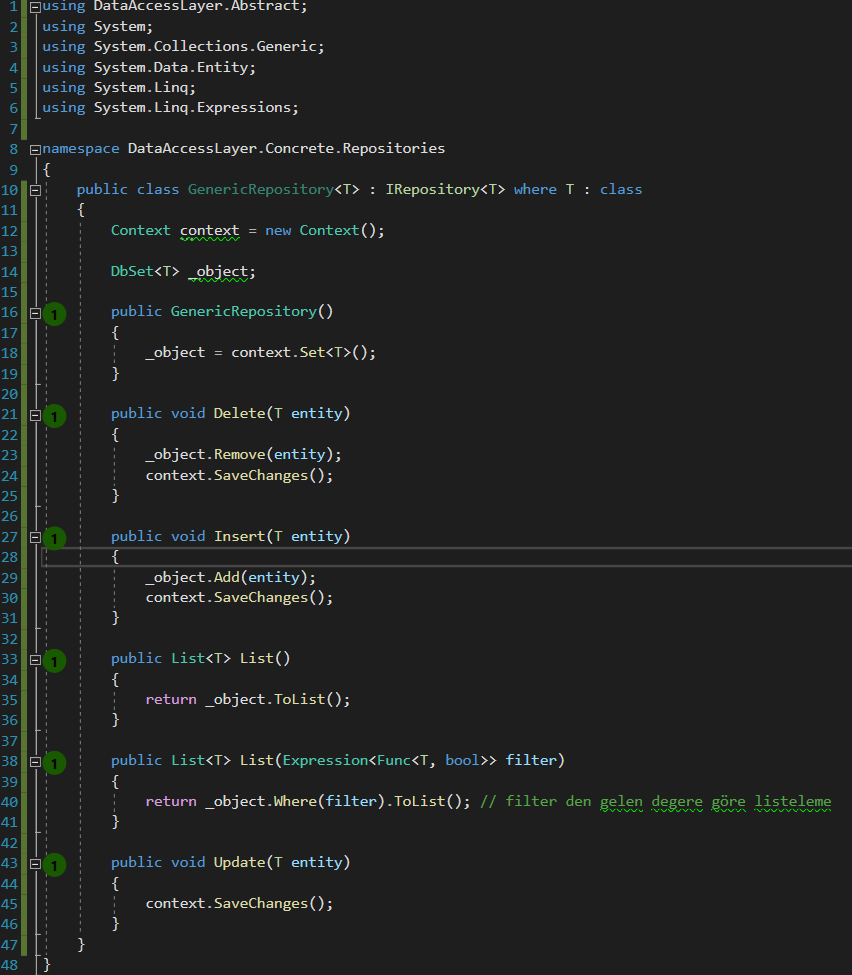
**Önemli Not:** Constructor’lar bulunduğu sinif ismi ile ayni isimde olmalidir. Parametre almalari opsiyoneldir.



* Bu Constructor’un gövdesine Context sınıfından olan bir T değeri ile DbSet sınıfından daha önce tanımladığımız **\_object** değişkenine bir değer ataması yaparız.



* IRepository’i interface’nden implemente edilen metotların gövdeleri doldurulur.



* Şimdilik DataAccessLayer işlemleri bitmiştir.

1. **BusinessLayer Islemleri**

Bu katmanda, isin genellikle geçerliliğinin ve kurallarının kontrolünün yapıldığı katmandır. UI katmanında kullanıcıların hatalı veri girmesinin önüne bu katmanda geçilir.

EntityLayer

DataAccessLayer(DAL)

BusinessLayer

PresantationLayer(UI)

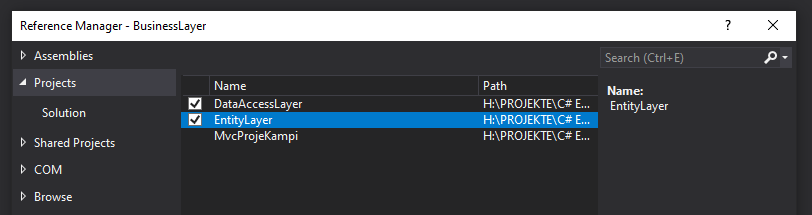
Sol taraftaki semada, hangi katmanın hangi katmanlara bağlı olduğu anlatılmıştır.

DataAccessLayer ile EntityLayer arasında bir bağ,

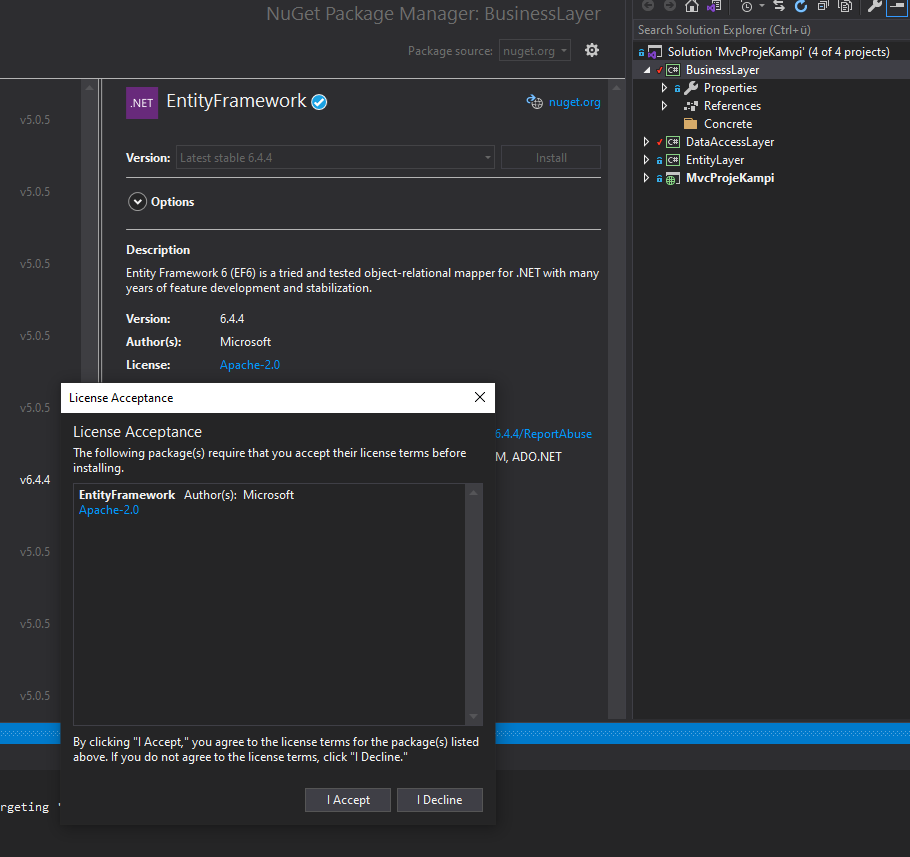
BusinessLayer ile hem DataAccessLayer ile hem de EntityLayer ile bağ vardır.

Presantation katmanı ise tüm katmanlarla birlikte çalışır.

1. **Concrete** klasörü oluşturulur. Somut ifadeler bu klasörde tutulur
2. İlgili referansların eklenmesi



1. İlgili katman üzerinden NugetPackage Manager ile EntityFramework paketi yüklenir.

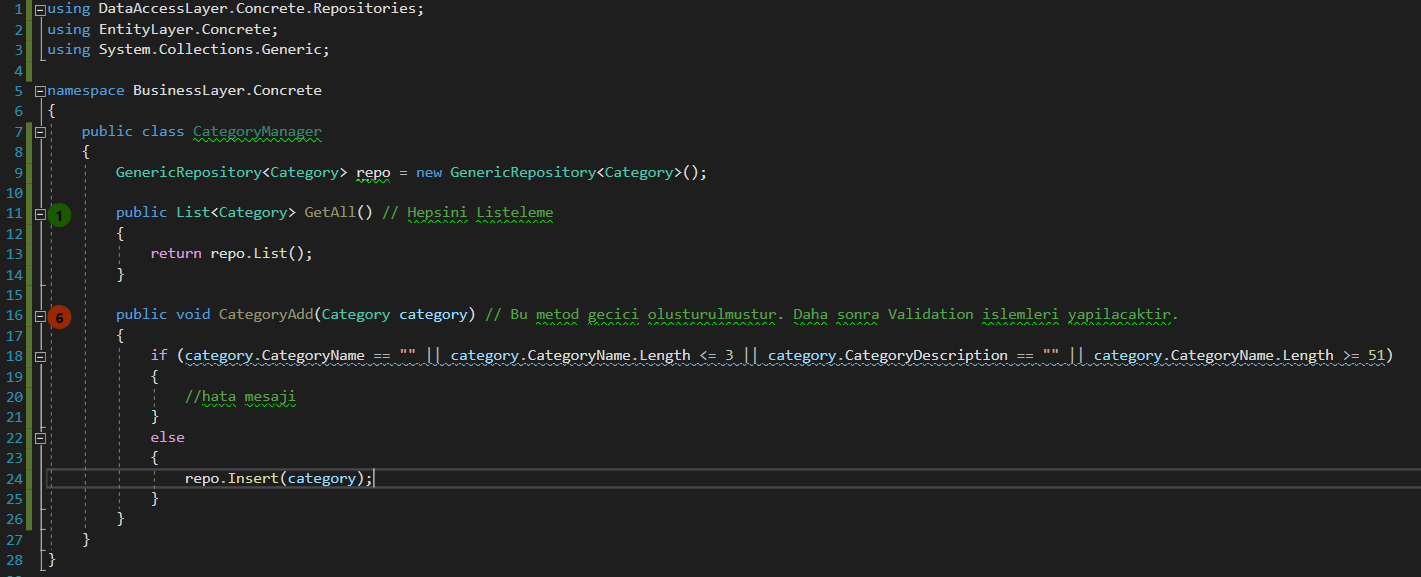


1. Bütün Entity’ler için birer BusinessLayer sınıfı oluşturulacak.
2. CategoryManager class’i oluşturulur

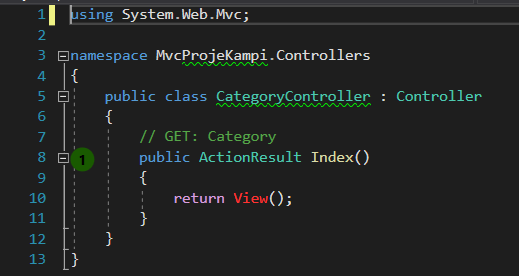
* Bu Class’in erişim belirleyicisi **public** yapılmalı
* GenericRepository<Entity> türünde bir nesne türetilir.
* İlgili katmanlara ait using ifadeleri eklenir (Yukarıdaki katmanların anlatıldığı şemayı baz alabilirsiniz.)

**using DataAccessLayer.Concrete.Repositories; using EntityLayer.Concrete;**

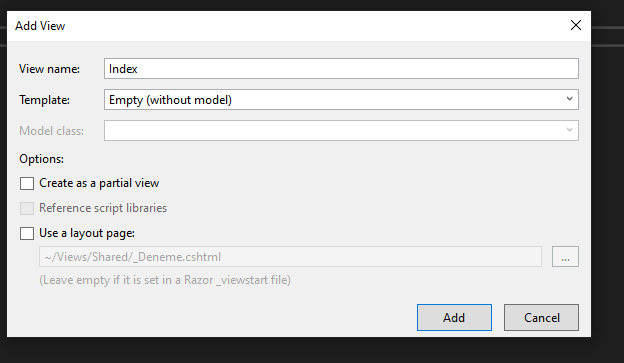
* Yapılacak işlemlere ait metotların tanımlanması.



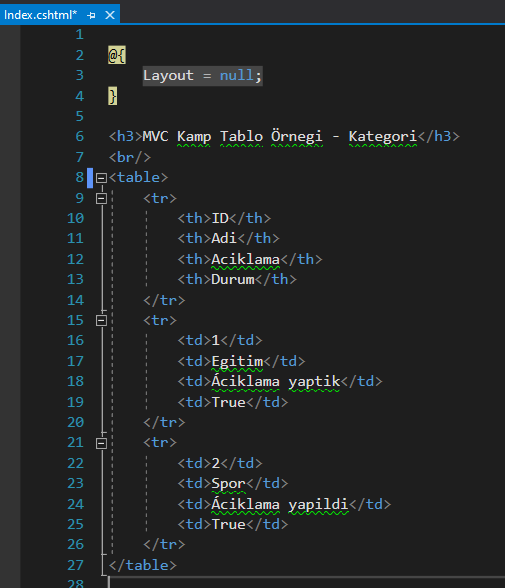
1. **Abstract** klasörü oluşturulur. Soyut ifadeler bu klasörde tutulur.
2. **Presentation Layer – User Interface (UI) :**
3. **Contoller’in eklenmesi**
4. **CategoryController Olusturulmasi**
5. **Controller klasörü içerisine bos bir MVC5 Contoller ve ismi CategoryController olarak oluşturulur. (Sadece bu kategoriye ait işlemler bu controller üzerinde tutulacaktır.)**



1. Index üzerinde sağ click yapılarak Add View den Index isminde herhangi bir Layout’a bağlı olmadan bos bir sayfa olsun.



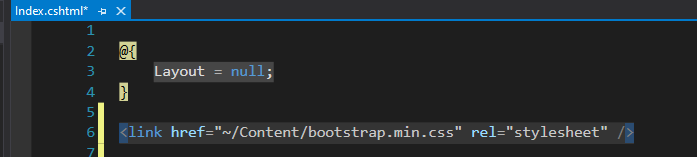
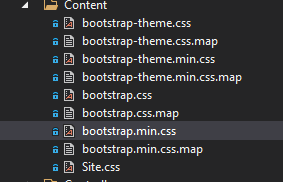
1. **Index sayfasının düzenlenmesi**
2. Gerekli tabloların eklenilmesi



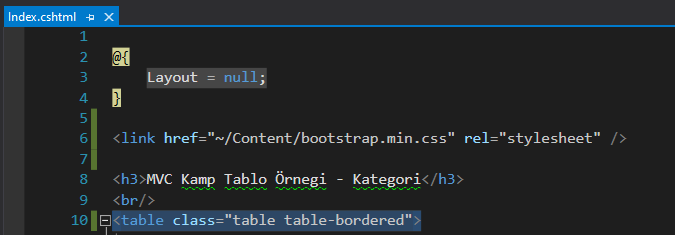
1. **Bootstrap bileşenlerinin eklenmesi.**

**Bootstrap : HTML tarafında daha güçlü ara yüzler oluşturmak için kullanılan bir bileşendir.**

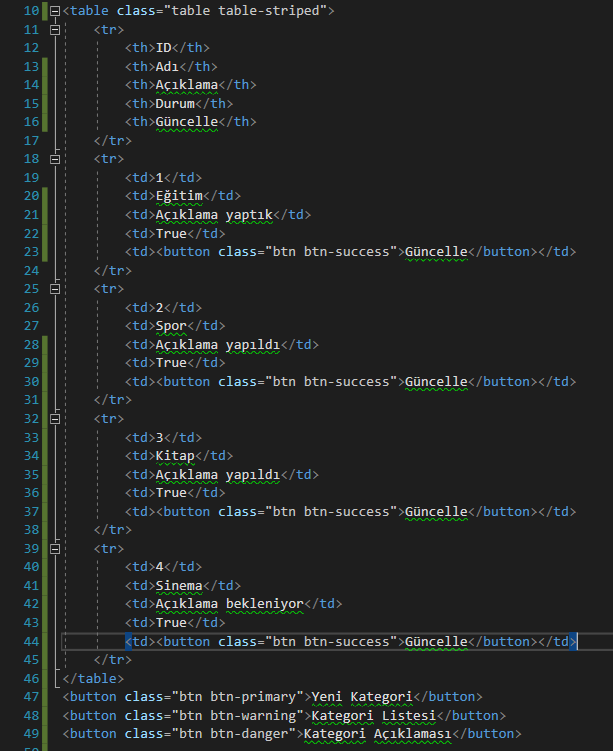
1. Presenatation katmanın altında yer alan Content klasöründeki bootstrap.min.css dosyası Mouse ile sürüklenip index sayfasının üst bölümüne eklenir. Ya da manuel olarak eklenir ( **<link href="~/Content/bootstrap.min.css" rel="stylesheet" />** ).

1. Bootstrap bileşenlerini çağırabilmek için class lardan yararlanırız. Tablomuzun görünümünü daha estetik yapabilmek için **<table class="table table-bordered">** seklinde yapılır. Sayfa yenilenerek tekrar Web Tarayıcı da açılır fark gözlenir.

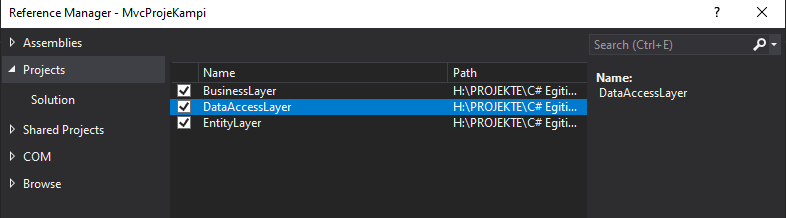


1. Index sayfasına ilgili butonların eklenmesi

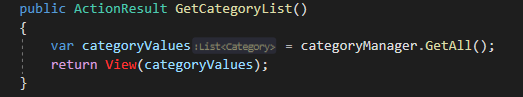
 

1. **References’lerin ilgili katmana eklenmesi**

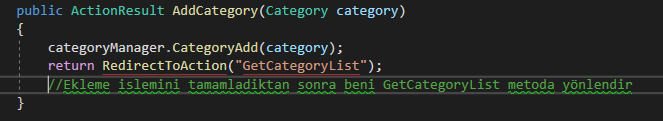
References üzerinden sağ tıklanarak tüm katmanlar bu katmana eklenir.



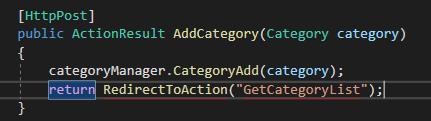
1. **İlgili katman üzerinden NuGet Packages aracılığı ile EntityFramework’ün eklenmesi**
2. **CategoryController sınıfı içerisindeki metotların oluşturulması ve içerilerinin doldurulması**
3. **BusinessLayer da oluşturulan sinifin (CategoryManager) cagrilmasi gerekir.**
4. Bu sınıfın kullanılması için using ifadesi eklenmelidir ( **using BusinessLayer.Concrete;** ).
5. Yeni bir ActionResult türünde bir metot ( **GetCategoryList()** ) tanımlıyoruz. Burada Kategorileri listeliyoruz
6. **GetCategoryList()** metodunun gövdesinin doldurulması



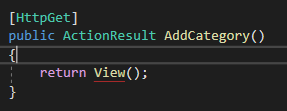
1. **View** üzerine sağ click **Add View** kısmından **GetCategoryList** adında **bağımsız bir Layout** oluşturuyoruz.
2. Yeni bir ActionResult türünde bir metot ( **AddCategory ()** ) tanımlıyoruz. Burada yeni Kategori ekliyoruz
3. **AddCategory ()** metodunun gövdesinin doldurulması



1. **AddCategory()**  üzerine sağ click **Add View** kısmından **GetCategoryList** adında **bağımsız bir Layout** oluşturuyoruz.
2. **HttpGet ve HttpPost Attribute’lerin eklenmesi**
3. AddCategory ActionResult’una **[HttpPost]** attribute eklenmesi. Sayfada herhangi bir butona basıldığında HttpPost işlemi gerçekleşir.



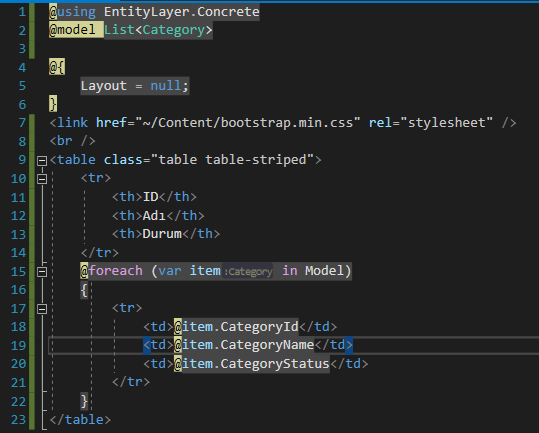
1. AddCategory ActionResult’una **[HttpGet]** attribute eklenmesi. Sayfada yüklendiğinde HttpGet işlemi gerçekleşir.



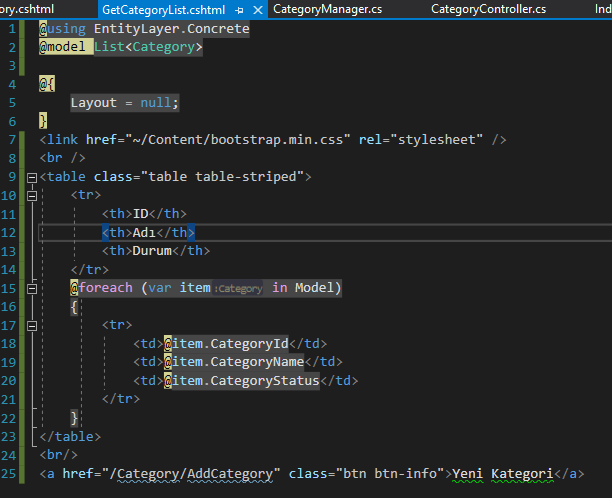
1. **View’lerin olusturulmasi ve düzenlenmesi**
2. **GetCategoryList View’in düzenlenmesi**
3. Bu view içerisindeki tüm HTML kodları temizlenir
4. Content klasörü içerisindeki Bootstrap bileşenlerini çağırabilmek için class’lardan yararlanırız. Tablomuzun görünümünü daha estetik yapabilmek için **<table class="table table-bordered">** seklinde yapılır.
5. Entity’nin kullanılabilmesi için using ifadesi eklenmelidir ( @**using Entity.Concrete;** ).
6. Bir model oluşturularak ilgili değerlerin tutulacağı bir List nesnesi eklenir ( @model **List<**Category**>** )

**Razor Syntax :** HTML kodları arasında C# kodlarını kullanabilmek için komutların önüne **@** işareti konur ve bu yazıma **Razor Syntax** denir.

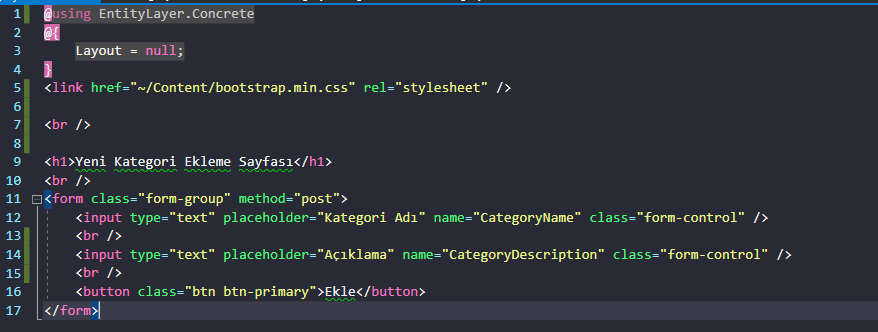
1. İlgili sayfa elemanlarının otomatik görüntülenmesi için Foreach döngüsünün kurulması.



1. Yönlendirme (Navigate) işleminin eklenmesi 🡪 <a href="/Category/AddCategory">Yeni Kategori</a>

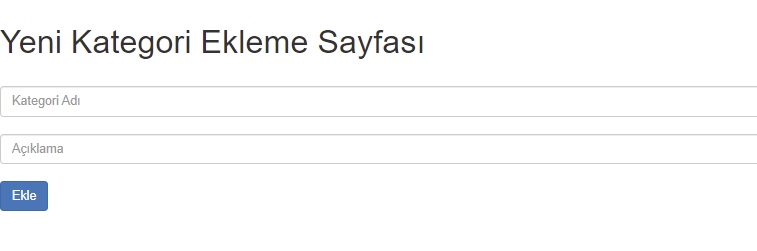


1. **AddCategory View’in düzenlenmesi**
2. Bu view içerisindeki tüm HTML kodları temizlenir
3. Content klasörü içerisindeki Bootstrap bileşenlerini çağırabilmek için class’lardan yararlanırız. Tablomuzun görünümünü daha estetik yapabilmek için **<table class="table table-bordered">** seklinde yapılır.
4. Entity’nin kullanılabilmesi için using ifadesi eklenmelidir ( @**using Entity.Concrete;** ).
5. Html elementlerinin eklenmesi



Html elementlerindeki “name” alanının karşılıkları veritabanındaki ilgili tablodaki sütun adları ile ayni olmalıdır.





Veritabanindaki categoryId‘ ler silme ve eklemelerden dolayı düzensiz ise bu komut ile sıralı hale getirebilirsiniz. (aradaki bos alanlar manuel olarak silinmelidir)

**DBCC CHECKIDENT (Categories,RESEED,0)**

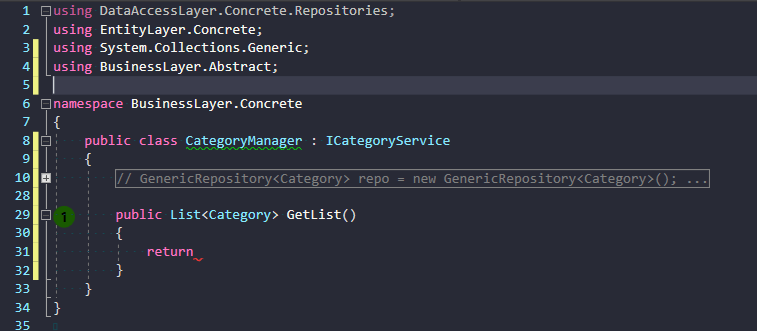
1. **BusinessLayer’e geri dönüş**
2. **Abstract** klasörü oluşturulur. Soyut ifadeler bu klasörde tutulur.
3. **ICategoryService Interface’nin olusturulmasi** 🡪Bu katmanda oluşturulanInterface’ler “Service” ya da “Servis” son eki alırlar.

* Bu interface’in erişim belirleyicisi **public** yapılmalı
* Generic List oluşturulması **(List<**Type**> Metod\_Adi();**) 🡪 (**List <Category> GetList();**) 🡪Listeleme Metodu
* using ifadesi eklenmelidir ( **using EntityLayer.Concrete;** ).

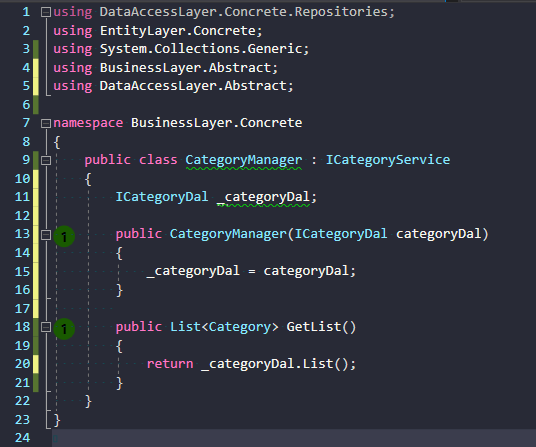
1. İlgili Class’lardaki alanların silinmesi ve ilgili değişikliklerin yapılması
2. **CategoryManager (BusinessLayer.Concrete.CategoryManager)**
3. Eski alanların yorum satiri haline getirilmesi



1. Content CategoryManager sınıfını **ICategoryService** den miras aldırma ve **ICategoryService interface’ini implemente** edilmesi



1. **ICategoryDal** türünde **\_categoryDal** adında değişken oluşturulması ve using ifadesinin eklenmesi 🡪 **using DataAccessLayer.Abstract**
2. **\_categoryDal** değişkenine değer ataması yapmak için bir **Constructor** (yapıcı metot) oluşturulması gerekmektedir. CategoryManager üzerinde 🡪 ctrl + . 🡪 Generate constructor…



Aşağıdaki Class ve Değişkenlerin referansları en iyi şekilde katmanlı yapıyı anlatmaktadır.

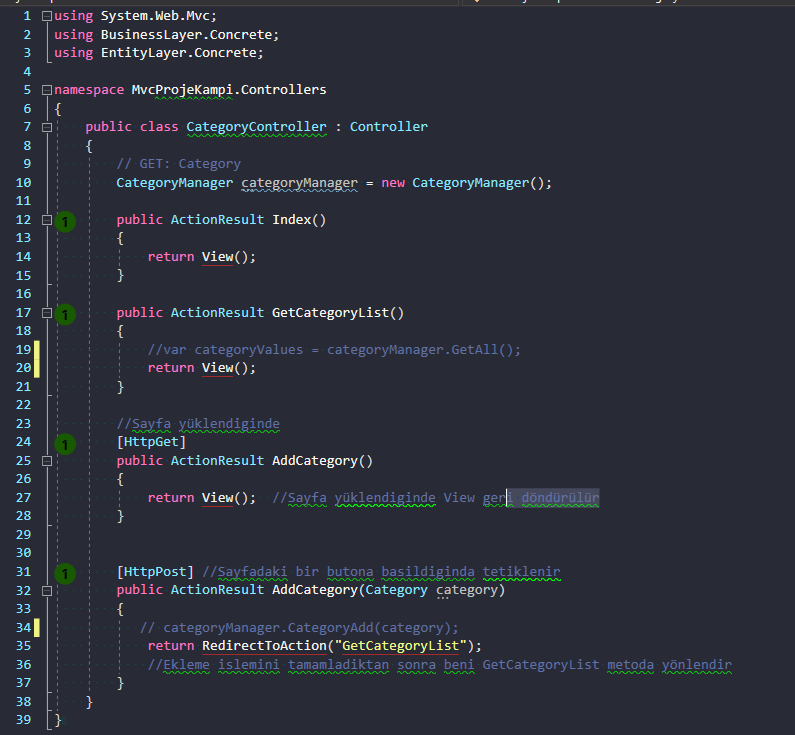
CategoryManager 🡪ICategoryService

\_categoryDal 🡪 ICategoriDal🡪IRepository 🡪List(),Insert(),Delete(),Update()

**NOT:** **Dependency Injection (DI) :** Bağımlılık oluşturacak parçaların ayrılıp, bunların dışardan verilmesiyle sistem içerisindeki bağımlılığı minimize etme işlemidir. Yani, temel olarak oluşturacağınız bir sınıf içerisinde başka bir sınıfın nesnesini kullanacaksanız new anahtar sözcüğüyle oluşturmamanız gerektiğini söyleyen bir yaklaşımdır. Gereken nesnenin ya Constructor’dan ya da Setter metoduyla parametre olarak alınması gerektiğini vurgulamaktadır. Böylece iki sınıfı birbirinden izole etmiş oluruz.

***Dependency Injection, bağımlılıkları soyutlamak demektir.***

1. **CategoryController (MvcProjeKampi.Controllers.CategoryController) da ilgili kısımların degistirilmesi.**



1. **DataAccessLayer’e geri dönüş**
2. **EntityFramework** klasörü oluşturulur.

* Entity Framework’e ait bazı DataAccessLayer sınıflarını burada oluşturacağız.

1. **To be Continued!**