

17.09.2021

**MAĞAZA STOK UYGULAMASI**  
**DÖKÜMANI**

**Barış ARSLAN**

# İÇİNDEKİLER

## Mesai Takip Programı

MVC Mantığı	3
Database Bağlantısı	3
Veri Tabanında Değişiklik	7
Veri Tabanı Aktarımı	8

## Yapım Aşamaları

Veri Tabanı Tasarımı	11
Proje Oluşturulması	11
Master Page – Navigation Bar	11
Controller Oluşturulması	12
Veri Ekleme	13
Silme İşlemleri	14
Güncelleme İşlemleri – DropDownList’e Veri Çekilmesi	14
Validasyon İşlemleri	15
Sayfalama	16
Alert Oluşturma	16
Pop-Up Modal	17
Search Bar	18
Jquery Eklentisi	19

## MVC Mantığı

MVC daha önceden Object Oriented tabanında manuel olarak yapılan işlemleri hazır olarak sunmaktadır. Veri tabanı bağlantısı gerçekleştirildikten sonra Entity katmanında, veri tabanındaki tablolarımıza ait Model dosyaları oluşturulmaktadır. Model dosyalarımız aslında veri tabanındaki her bir tablomuzun projeye sınıf olarak eklendiği dosyalardır. İlgili tablonun içinde bulunan sütunlar değişken olarak ait olduğu sınıfın içerisinde tanımlanmaktadır. Projenin iş mantığının oluşturulduğu, doğrulama ve veri işlemlerinin gerçekleştiği bölümdür.

Veri tabanına erişim, veri tabanı ilişkileri gibi data ile ilgili işlemlerle birlikte Entity Framework, Linq to Sql, NHibernate, ADO.Net gibi Framework leri içerisinde bulunduran katmandır yani data(veri) işlemleri bu katmanda gerçekleşir.

View, MVC’de projenin arayüzlerinin oluşturulduğu bölümdür. Bu bölümde projenin kullanıcılara sunulacak olan HTML dosyaları yer almaktadır.

Controller ise MVC’de projenin iç süreçlerini kontrol eden bölümdür. Bu bölümde View ile Model arasındaki bağlantı kurulur. Kullanıcılardan gelen istekler (request) Controller’larda değerlendirilir, istemin detayına göre hangi işlemlerin yapılacağı ve kullanıcıya hangi View’ın döneceği (response) belirtilir.

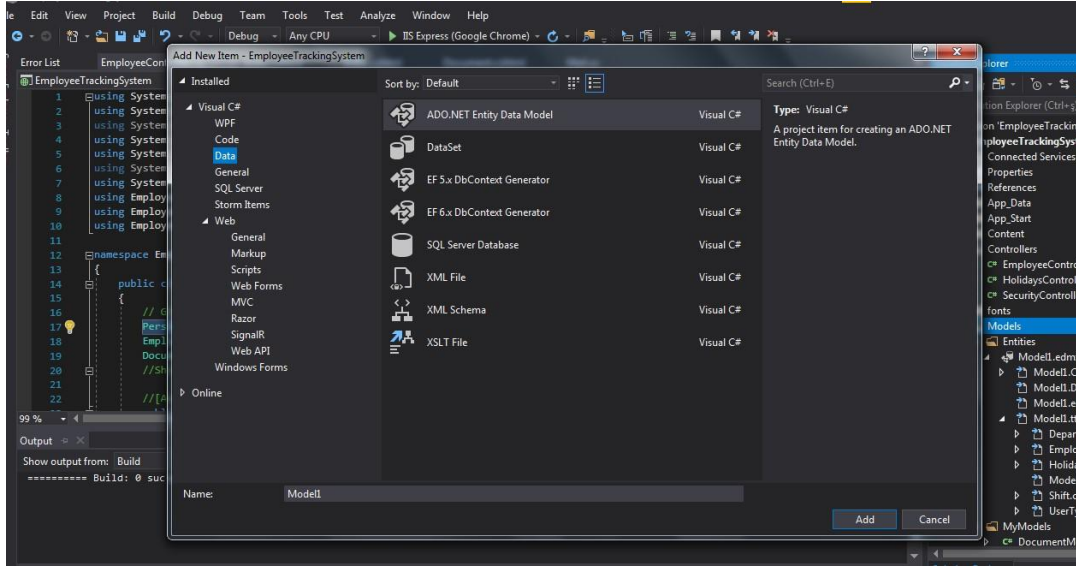
MVC projelerinde sayfa içerisinde bulunan belirli kısımların sabitlenmesi Layout adı verilen yapılar ile sağlanmaktadır. Sayfalar arası geçişlerde sabit kalacak olan kısımlar Layout içerisinde tanımlanmakta, bir görünüm oluşturulmaktadır.

Master Page oluşturulmasının sebebi, master page üzerinde birden fazla sayfanın görüntülenmesini daha hızlı ve dinamik bir şekilde sağlamaya çalışmaktır. Diğer sayfalar için paylaşılan, düzen ve işlevsellik içeren bir şablon sağlar. İçerik sayfaları tarafından geçersiz kılınacak içerikler için sabit yer tutucular sağlar. Örneğin Youtube sayfasında arama çubuğunun sabit olarak yukarıda kalması, sol tarafta menü bölmesinin sürekli yer alması buna örnektir.

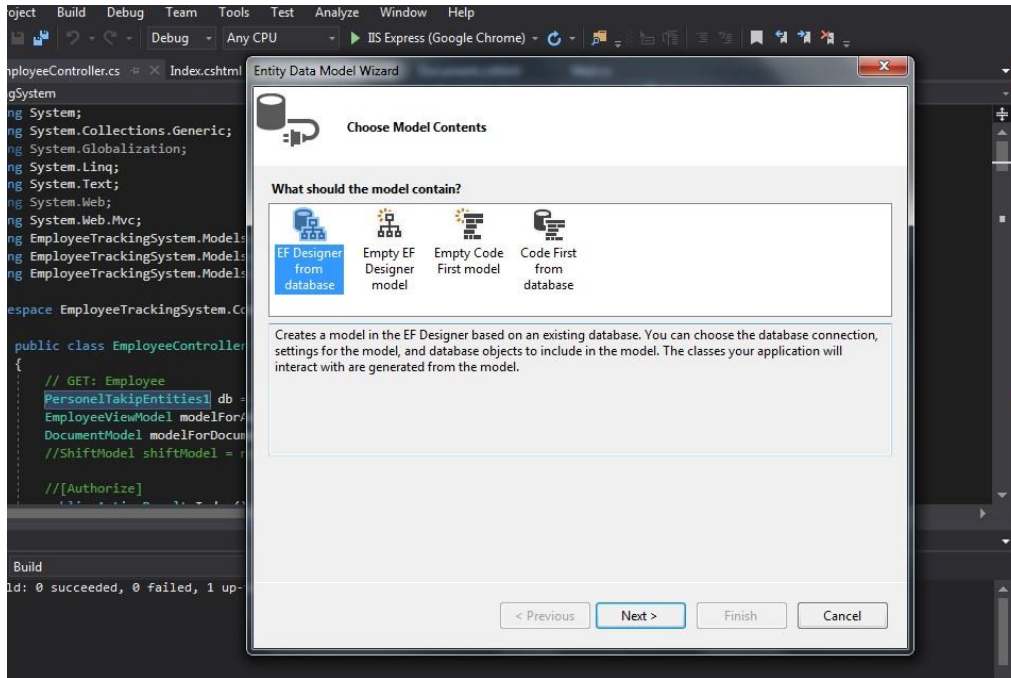
Master Page “Shared” klasörü içerisinde tanımlanan Layout ile oluşturulmaktadır. Bu projede bulunan MainLayout dosyası içerisinde Navigation Bar tanımlanmıştır. Bu sayede sayfalar arası geçişlerde Navigation Bar her sayfada sabit olarak kullanılmaktadır.

## Database Bağlantısı

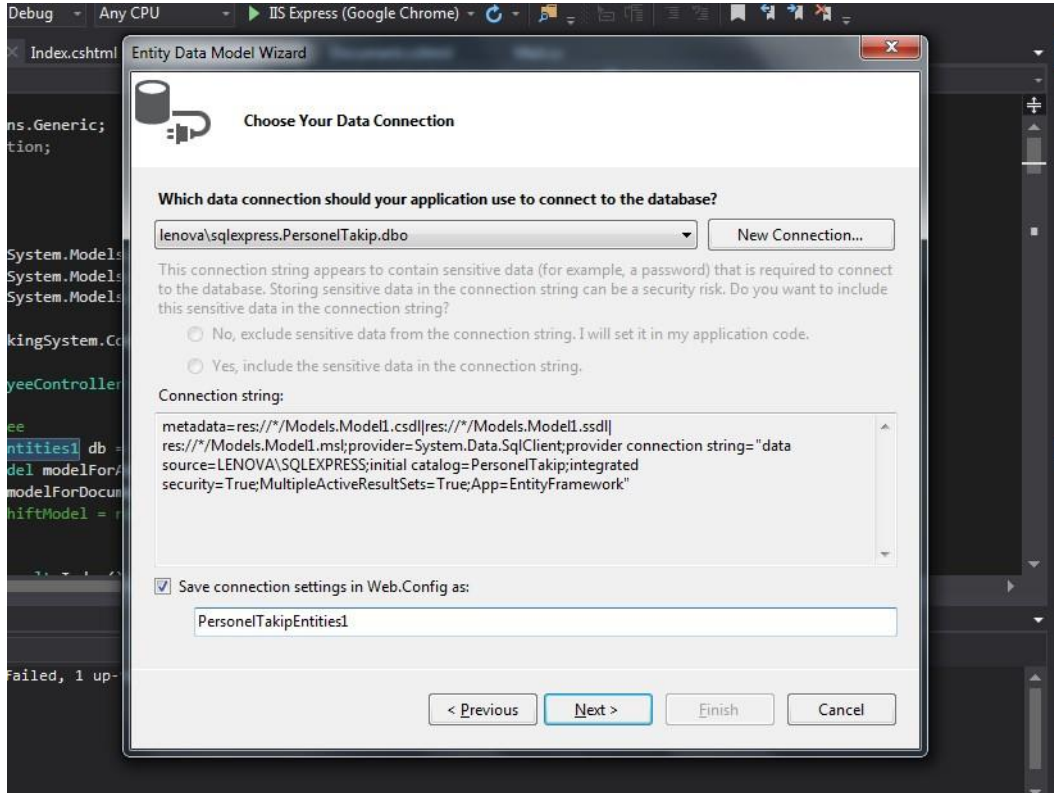
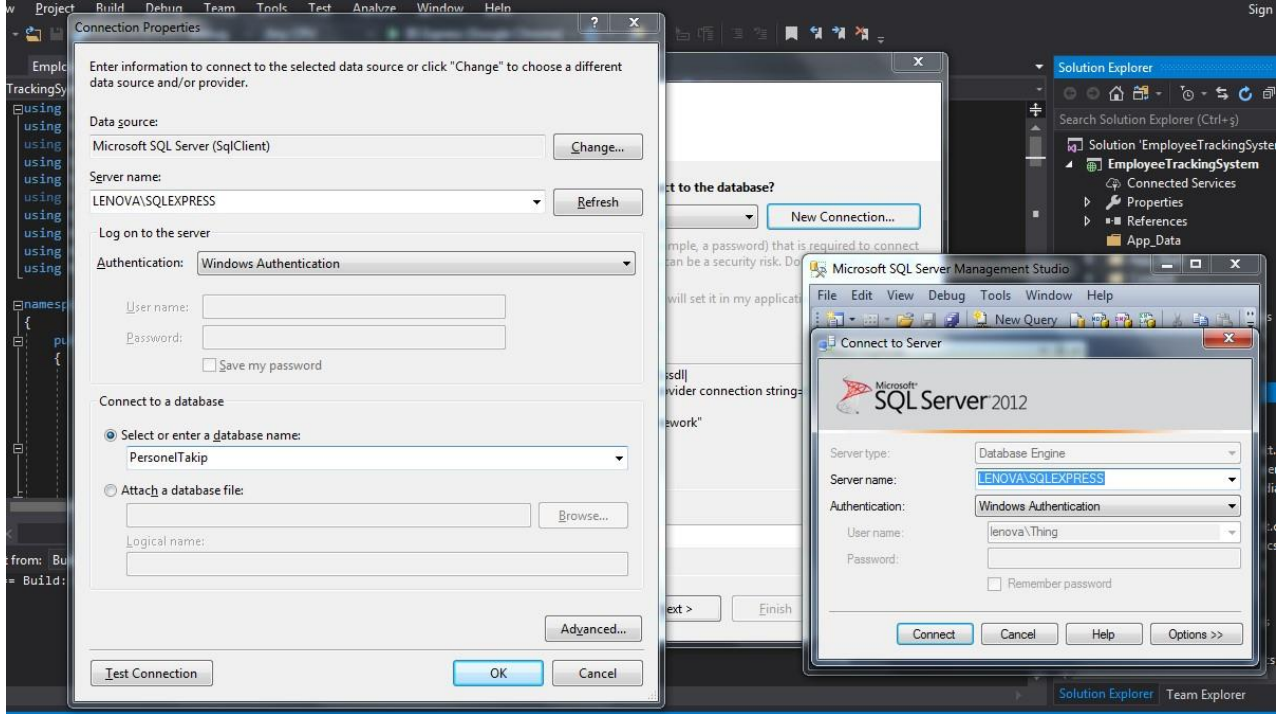
MsSql Server’da tutulan bilgiler, MVC projesi üzerinde Entity Framework ile kontrol edilmektedir. Amacımız veri tabanına Entity Framework kullanarak bağlanmaktır. “Solution Explorer” penceresinden “Models” Klasörüne sağ tıklayarak sırasıyla “Add” ve “New Item” seçenekleri seçilmekte. Sonrasında soldaki menüden “Data” seçilerek “ADO.NET Entity Data Model” seçeneğini seçilip, modele istenilen ad verilmekte.



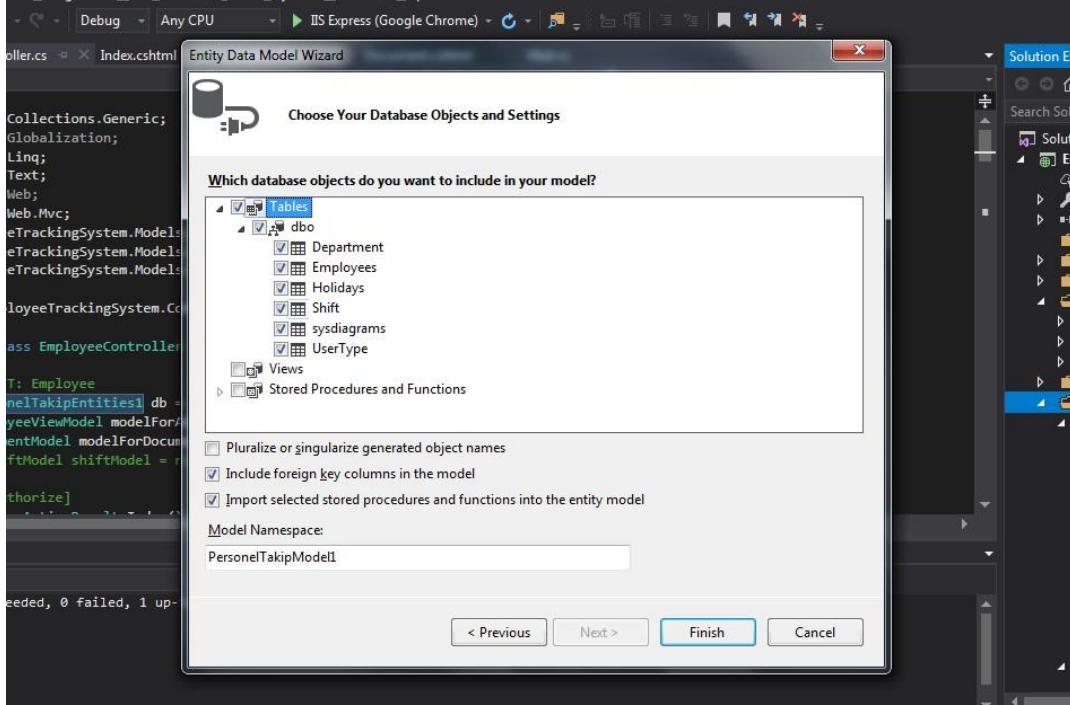
Bir sonraki ekranda karşımıza 4 seçenek gelmektedir. Burada “EF Designer From Data” seçeneği ile devam edilmektedir.



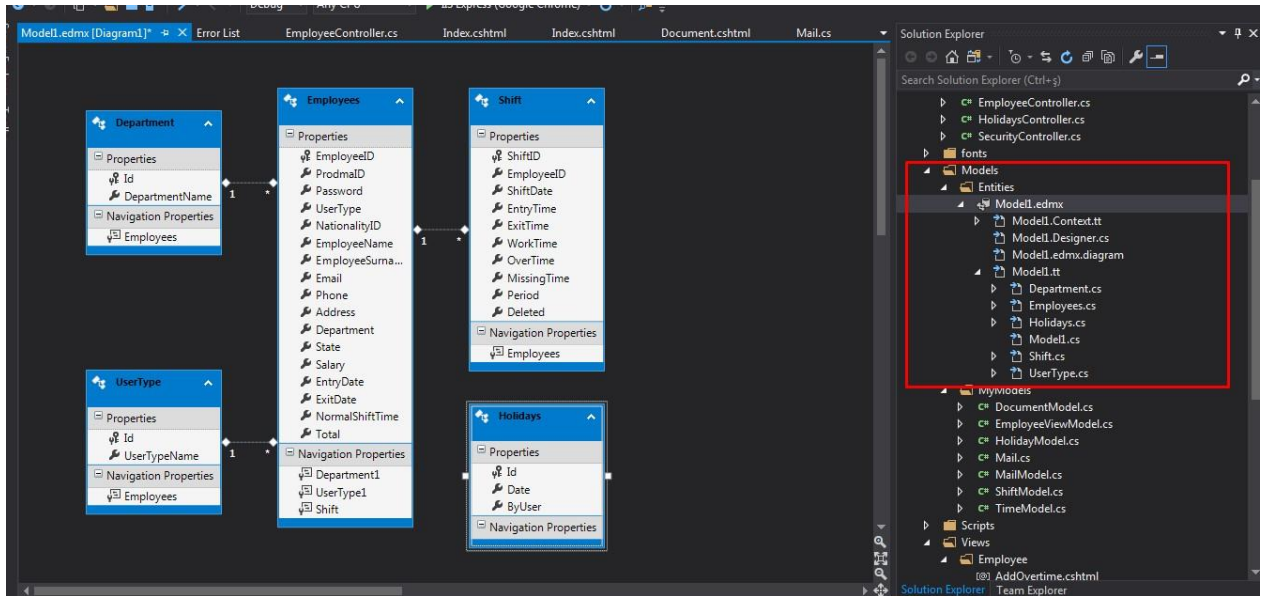
“New Connection” seçeneğini seçerek yeni bir veri tabanı bağlantısı oluşturuluyor. Bağlantı Özellikleri penceresinde “Server Name”, veri tabanı adı ve bağlantı bilgileri girilerek “OK” butonuna tıklanmakta.



Projede kullanılmak istenen isim bilgisi girilmekte. Projede “PersonelTakip1” ismi ile kullanılmaktadır. Sonrasında veri tabanında yer alan tablolardan projeye eklenilmesi istenen tablolar seçilerek işlem sonlandırılmaktadır.

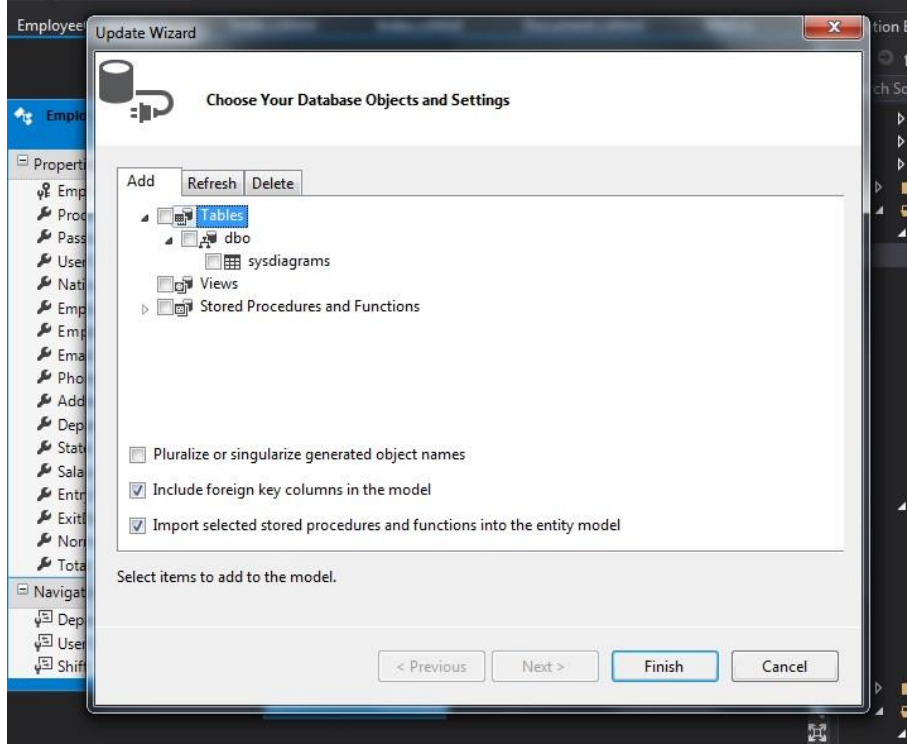


Proje içerisine model'lara ait diagram ve model dosyaları başarılı şekilde eklenmiştir.



## Veri Tabanında Değişiklik

Veri tabanında bir değişiklik yapıldığında yeniden model oluşturulmasına gerek yoktur. “Models” klosöründe bulunan “Model1.edmx” dosyasına sağ tıklararak model diagramına ulaşılmaktadır. Bu ekranda sağ tıklanarak “Update Model from Database” seçeneğine tıklanmasının ardında şu ekran açılmaktadır:



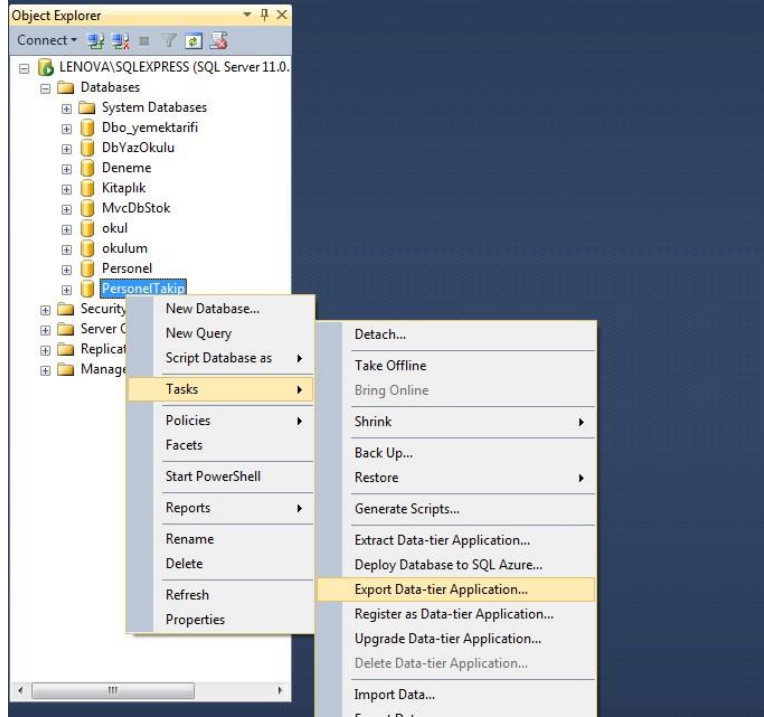
Bu ekran üzerinde eğer veri tabanına eklenmiş yeni bir tablo mevcut ise “Add” bölümünden seçilerek model yapısına eklenmektedir. Güncelleme işlemi gerçekleştirilmiş ise “Refresh” bölümü altından yine aynı şekilde seçilerek model üzerinde güncellenmektedir. Değişikliklerin kaydedilmesinin ardından proje içerisinde bulunan model dosyaları güncellenmektedir.

\*Eğer model dosyaları silinmiş gibi görünürse yani “Solution Explorer” altında gözükmeyse, model diagramında bulunan tüm tablolar silinmelidir. Sonrasında yukarıdaki anlatılan şekilde model güncellenmelidir. “Add” seçeneği altından silinen model dosyaları yeniden eklendiğinde sorun ortadan kalkacaktır.

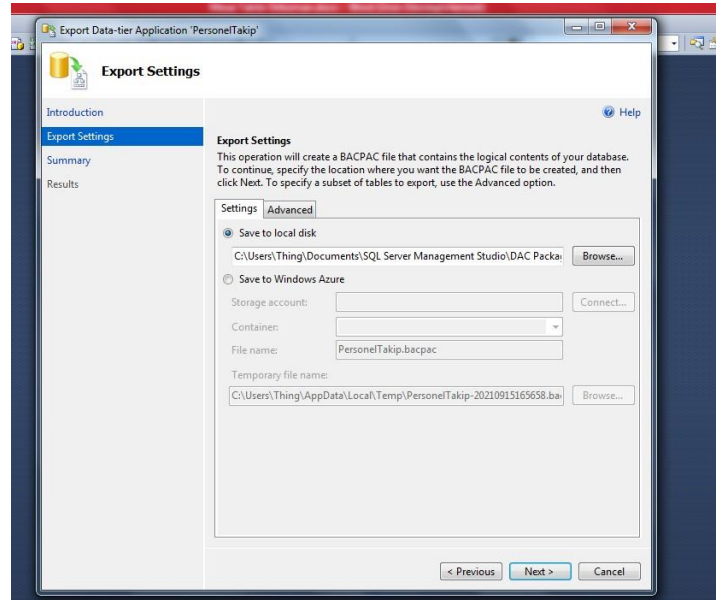


## Veri Tabanı Aktarımı

Veri tabanına sağ tıklayarak “Export Data-tier”i seçiyoruz.

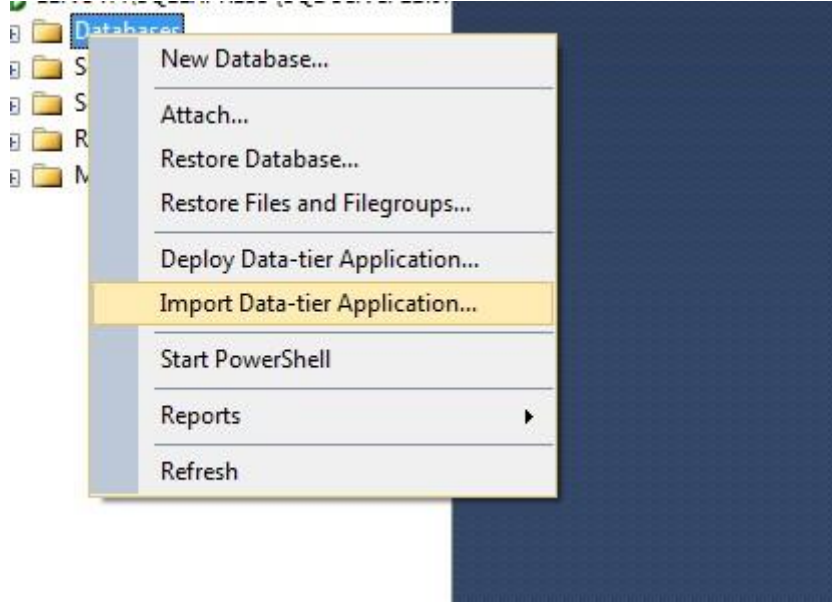


Sonrasında kaydedilecek dizini seçiyoruz.

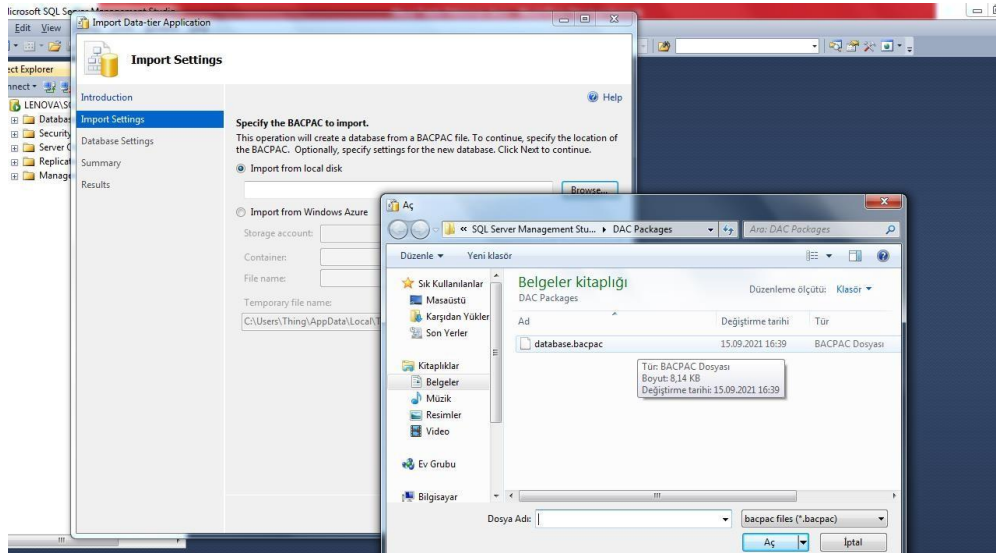




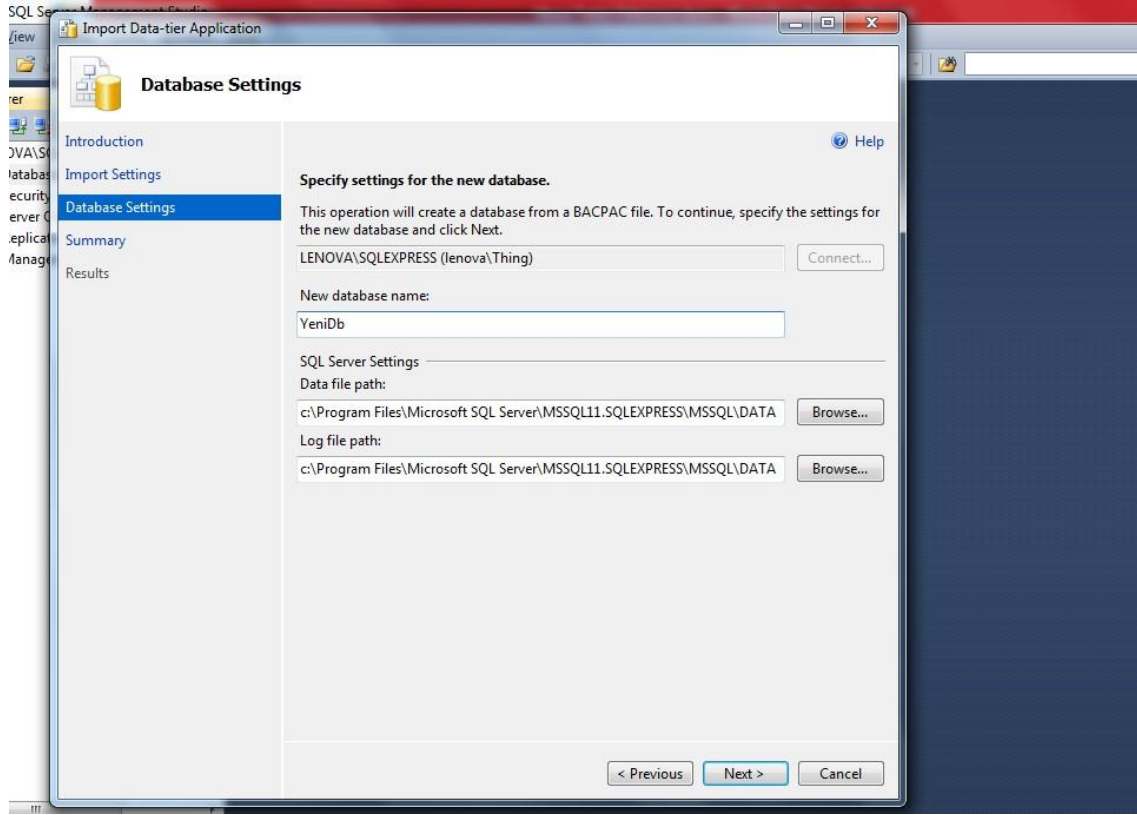
Next Next dedikten sonra “.bacpac” uzantılı dosya dizine kaydediliyor. Veri tabanını yeniden oluşturmak istersek Databse üzerinde sağ tıklayarak “Import Data-tier”i seçiyoruz.



“.bacpac” uzantılı dosya seçilerek devam ediliyor.



İlerledikten sonra yeni isim verilerek işlemler tamamlanıyor.

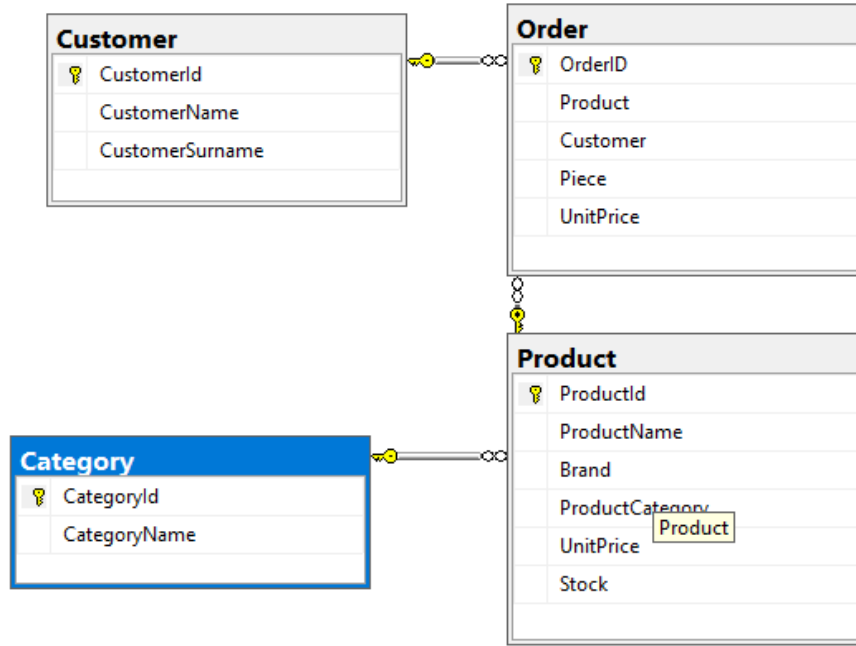


# YAPIM AŞAMALARI

## Veritabanı Tasarımı

Bu projede belirlenen amaç, bir alışveriş sitesinin yönetici panelini tasarlamaktır. Çok fazla detaya girmeden,

- Kayıtlı ürünlere ait, kategori, fiyat, stok vb bilgilerin tutulduğu ürünler tablosu
- Ürünlerin bulunduğu kategoriye ait kategori tablosu,
- Kayıtlı kullanıcıların bilgilerinin tutulduğu müşteri tablosu,
- Satış bilgilerinin tutulduğu sipariş tablosu olmak üzere dört adet tablo oluşturuldu. Tablolar arası ilişkiler kuruldu.



## Proje Oluşturulması

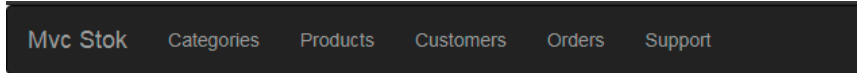
Visual Studio'da ASP.Net MVC projesi oluşturuldu ve veri tabanını bağlama işlemleri gerçekleştirildi. MVC daha önceden Object Oriented tabanında manuel olarak yapılan işlemleri hazır olarak sunmakta. Veri tabanı bağlantısı gerçekleştirildikten sonra Entity katmanında, veri tabanındaki tablolarımıza ait Model dosyaları oluşturuldu. Model dosyalarımız aslında veri tabanındaki her bir tablomuzun projeye sınıf olarak eklendiği dosyalardır. İlgili tablonun içinde bulunan sütunlar değişken olarak ait olduğu sınıfın içerisinde tanımlanmaktadır. Projenin iş mantığının oluşturulduğu, doğrulama ve veri işlemlerinin gerçekleştiği bölümdür.

## Master Page – Navigation Bar

ASP.NET üzerinden web sitesi tasarlarlarken sabit kalan bölümlerin tasarlandığı Master Page yapısı mevcuttu. MVC projelerinde ise sayfa içerisinde bulunan belirli kısımların sabitlenmesi, Master Page yapısını kurmak için kullanılan yapıya Layout adı verilmekte. Sayfalar arası geçişlerde sabit kalacak olan kısımlar Layout içerisinde tanımlanmakta.

Layout dışında ise oluşturulan View klasörleri bulunmakta. MVC’de projenin arayüzlerinin oluşturulduğu bölümdür. Kullanıcılara sunulacak olan HTML dosyaları içermektedir. Bu klasörler içerisinde Layout’larda tasarlanan sabit bölümler ile birlikte görüntülenecek olana değişken sayfalar bulunmakta. Layout içerisinde tanımlanan Navigation Bar ile sayfalar arası geçiş sağlandığında görüntülenecek olan sayfalar bu klasörlerin içerisinde oluşturulmakta.

Navigation bar, oluşturulan MainLayout üzerinde tanımlandı ve sayfalar arası yönlendirmeler bu Navigation Bar ile sağlandı.



## Controller Oluşturulması

Controller klasörleri ise içerisinde proje içerisindeki süreçleri kontrol eden bölümlerdir. View ile Model arasındaki bağlantı sağlanır. Kullanıcıdan gelen istekler doğrultusunda hangi işlemlerin yapıp geriye hani View’ın döneceğini belirtir. Bu yönlendirmeler de Controller’ların içerisinde bulunan ActionResult yapıları ile sağlanır. Her ActionResult, yönlendirme işlemi için bir View’a sahiptir. ActionResult çağırıldığında kendi View dosyasını geri döndürür.

Kategorileri listeleme işlemi için öncelikle Controller oluşturuldu. Oluşturulan Controller içerisinde Entity katmanında bulunan Model tanımlandı. Sonrasında listeleme işlemi için oluşturulan, tanımlanan Model kullanılarak ActionResult içerisinde veri çekme işlemi gerçekleştirildi. Ardından ilgili View, çekilen veriler ile birlikte döndürüldü.

```
MvcDbStokEntities db = new MvcDbStokEntities();

public ActionResult Index()
{
    var values = db.Category.ToList();
    return View(values);
}
```

Index.html içerisinde bir tablo oluşturuldu. MVC tarafında backend kodları “@” işareti ile yazılabilmektedir. Bu dosya içerisinde de Model tanımlandı ve foreach döngüsü kullanılarak Controller’dan iletilen veri burada tablo sütunlarına yerleştirildi.

```
@foreach(var category in Model) {
    <tr>
        <td>@category.CategoryId</td>
        <td>@category.CategoryName</td>
    </tr>
}
```

Ürünlerin ve müşterilerinin listeme işlemlerini gerçekleştirmek için aynı işlemler tekrarlandı. Controller oluşturuldu. ActionResult tanımlandı. ActionResult üzerinde oluşturulan View içerisinde Model tanımlandı, tablo oluşturuldu. Foreach döngüsü ile veriler listelendi.

## Veri Ekleme

Yeni girilecek veriler için ekleme butonları oluşturuldu. Bu işlem için iki çeşit ActionResult tanımlandı. Ekleme butonları tıklandığında Controller'da bulunan yönlendirici ActionResult tetiklenmekte. İlgili View'a yönlendirmekte. İlgili View içerisinde bulunan alanlar doldurulduktan sonra Ekle butonuna tıklandığında ise ekleme işlemi gerçekleştirilmekte. Bunun ayrımı atılan istek çeşidi ile sağlanmakta. Eğer yeni müşteri, kategori vs. eklemek için butona tıklanırsa, ilgili view bize getirilecek ve burada 'HttpGet' isteği oluşturulmakta. Bu View içerisinde veriler doldurulduktan sonra ise ekle butonuna tıklandığında 'HttpPost' isteği oluşturulmakta ve veri girişi sağlanmakta. Bu iki ActionResult farklı, oluşturulan iki istek çeşidi ile sağlanmakta.

```
[HttpGet]
public ActionResult NewCustomer(){
    return View();
}

[HttpPost]
public ActionResult NewCustomer(Customer customer1)
{
    if (!ModelState.IsValid)
    {
        return View("NewCustomer");
    }
    db.Customer.Add(customer1);
    db.SaveChanges();
    return RedirectToAction("Index");
}
```

Yeni veri girişi için gerekli View'a yönlendirme aşağıdaki html kodu ile gerçekleştirildi. 'href' içerisinde belirtilen uzantıdaki ilk veri hangi Controller'a gideceği, ikinci veri ise o Controller içerisindeki hangi ActionResult'ın çağırılacağı. Burada 'HttpGet' isteği oluşturulduğu için ilgili View'a yönlendirme işlemi gerçekleştirilmekte.

```
<a href="/Product/NewProduct/" class="btn btn-primary">Add New Product</a>
```

Doldurulan bilgilerin parametre olarak geçilmesi ise .NET'in sağlamış olduğu SQL benzeri bir sorgulama çeşidi olan LINQ ile gerçekleştirildi. İlgili View'a öncelikle ait olunan Model eklendi. Ardından Html içerisinde bir form oluşturuldu ve form'un tetikleyeceği metod belirtildi. Sonrasında veri girişinin sağlandı.

```
@Html.TextBoxFor(m => m.CustomerName, new { @class = "form-control" })
```

TextBox vb. elemanı içerisinde LINQ sorgusu ile girilen verinin, ait olduğu model'in hangi property'si olduğu belirtildi. Butona tıklandığında ise 'HttpPost' metodu çağırıldı ve ActionResult parametre geçilen veriyi veri tabanına ekledi.

Ekleme işlemi sırasında kategori seçimi için DropDownList kullanıldı. DropDownList'e kategorilerin çekilme işlemi ve veri gönderilirken seçilen kategoriye ait id değerinin gönderilmesi de LINQ sorgusu ile gerçekleştirildi. Bir liste oluşturuldu ve yazılan sorgu sonucu gelecek veri bu listeye atandı. DropDownList'e kategori verisinin adı, değerine de id değeri atandı. Seçim sırasında seçilen text'e ait id değeri iletilmekte. Ekleme işlemleri böyle gerçekleştirildi.

## Silme İşlemleri

Listelenen her üç tablo için silme ve güncelleme işlemleri yapıldı. Öncelikle Controller içerisinde silme ve güncelleme işlemleri için parametre alan ActionResult'lar tanımlandı. Daha sonra view içerisinde listelenen her bir item için güncelleme ve silme butonları oluşturuldu. Oluşturulan butonlara link ataması yapıldı. İşlemin gerçekleştirilebilmesi için silme ve güncelleme butonları, kendileri için oluşturulan ActionResultlara yönlendirildi. İşlem yapılacak item için ise id bilgisi yönlendirme sırasında ActionResulta parametre olarak gönderildi. ActionResult içerisinde bir değişken oluşturuldu. Bu ActionResult'lar içerisinde aldıkları parametreler ile 'db' değişkeni üzerinden işlem gerçekleştirmekte. Parametre geçilen Find() metodu sayesinde ilgili veri bulunup silinmekte. Sonrasında tekrar item'ların listelendiği Index.html döndürülmekte. Silme işlemi bu şekilde gerçekleştirilmekte.

```
var product = db.Product.Find(id);
db.Product.Remove(product);
db.SaveChanges();
return RedirectToAction("Index");
```

Category ID	Category Name	Update	Delete
1	Beyaz Eşya	<a href="#">Update</a>	<a href="#">Delete</a>
2	Küçük Ev Aletleri	<a href="#">Update</a>	<a href="#">Delete</a>
3	Bilgisayar	<a href="#">Update</a>	<a href="#">Delete</a>
4	Bilgisayar Aksesuarları	<a href="#">Update</a>	<a href="#">Delete</a>

## Güncelleme İşlemleri – DropDownList'e Veri Çekilmesi

Güncelleme işlemi için parametre geçilen id bilgisi ile birlikte gerekli view kullanıcıya döndürülmekte. Tıklanan item'a ait bilgiler değiştirilmek üzere listelenmekte. Bu listeleme işlemi yine parametre geçilen id bilgisi ile gerçekleşmekte. Find() metoduna geçilen id bilgisi ile eşleşen item döndürülmekte ve View'a gönderilmekte.

```
var customer = db.Customer.Find(id);
return View("GetCustomer", customer);
```

Aynı şekilde Find() metodu ile gönderilen id verisi sayesinde tablo elemanlarının bulunduğu liste içerisinde LINQ sorgusu gerçekleştirilmekte. Bu sorgu ürünler için kategori bilgisinin çekilip, DropDownList'e aktarılmasında kullanılmakta. Eşleşen id'ye ait kategori bilgisi değişkene aktarılmakta ve döndürülecek View'a parametre olarak ürün bilgileri gönderilmekte. View açıldığında ise form elemanlarına ürün bilgileri yazdırılmakta.

```
var product = db.Product.Find(id);
List<SelectListItem> values = (from i in db.Category.ToList()
                               select new SelectListItem{
                                   Text = i.CategoryName,
                                   Value = i.CategoryId.ToString()
                               }).ToList();

ViewBag.val = values;
return View("GetProduct", product);
```

Kullanıcı bilgileri güncelledikten sonra girilen veriler tekrar parametre olarak alınmakta ve bu ActionResult içerisinde örnekte görüldüğü gibi girdiği veriler, Find() metoduna id'si parametre olarak geçilen item'ın bilgileri ile değiştirilmekte

```
var ct = db.Category.Find(category.CategoryId);
ct.CategoryName = category.CategoryName;
db.SaveChanges();
return RedirectToAction("Index");
```

Product ID

1

Product Name

Çamaşır Makinesi

Brand

Arçelik

Category

Beyaz Eşya

Unit Price

1235,00

Stock

15

Save

## Validasyon İşlemleri

Kullanıcının veri girişi sırasında Validation kontrolleri de sağlandı. Kullanıcının boş veri girmesi, girilen verinin belirli aralıkta olması vb kontrol işlemleri gerçekleştirildi.

Entity katmanında bulunan model'lar içerisindeki property'lere 'Required' anatasyonu kullanılarak kurallar tanımlandı. Belirtilen aralıkta olması ise 'StringLength' anatasyonu ile gerçekleşti. View içerisinde ise burda tanımlanan kurallar uygulamaya geçirildi. Tanımlanan hata mesajı, html kodu içerisinde belirtilerek kırmızı renkte yazdırıldı. Müşteri adının boş geçilmesi ve belirlenen aralık dışında olması bu şekilde engellendi.

```
[Required(ErrorMessage="Name can not blank.")]
[StringLength(50,ErrorMessage="Max 50 character")]
public string CustomerName { get; set; }

@Html.TextBoxFor(m => m.CustomerName, new { @class = "form-control" })
@Html.ValidationMessageFor(m => m.CustomerName, "", new { @style = "color:red"
})
```

Doğrulama işlemleri daha basit bir şekilde html kod bloğu içerisinde de gerçekleştirilebilmekte. Örneğin kullanıcıdan marka bilgisi alınırken hem 'required' özelliği kullanılarak boş geçilmesi engelleniyor, hem de 'maxlength' ile en fazla girilebilecek karakter sayısı belirleniyor. Daha fazla girilmesine izin verilmiyor. 'Placeholder' ise kullanıcı için bilgilendirici işlev görmekte.



```
<input type="text" class="form-control" name="Brand" required="" placeholder="Brand" maxlength="50" />
```

**Customer Name**

  
Max 50 character


**Customer Surname**

  
Surname can not blank.

[Add New Customer](#)

**Product Name**

**Brand**

 Lütfen bu alanı doldurun.

## Sayfalama

Item listelemelerinde sayfalama işlemi gerçekleştirildi. Öncelikle projeye NuGet Package Manager üzerinden bir 'PagedList.Mvc' adında bir paket eklendi. Bu eklenti kütüphanesi Controller'lar içersine eklendi. Daha sonra ilk olarak yüklenen Index.html'in ActionResult'ında aşağıdaki değişiklik yapıldı. Listeleme işlemi için kullanılan 'ToPagedList()' metodu iki parametre almakta. Bunlardan birincisi kaçınıcı sayfadan okunmaya başlanacağı, diğeri ise her sayfada kaç tane değer yer alacağını belirtiyor.

```
var values = db.Category.ToList().ToPagedList(1, 4);  
return View(values);
```

Index.html içerisinde gelen veri List formatında tanımlanmıştı. Onu da PagedList formatında tanımladıktan sonra listeleme işlemi istenen sayfalama formatında gerçekleşmekte.

```
@using PagedList  
@using PagedList.Mvc  
@model PagedList.IPagedList<Category>
```

Category ID	Category Name	Update	Delete
5	Elektronik	<a href="#">Update</a>	<a href="#">Delete</a>
12	Tv	<a href="#">Update</a>	<a href="#">Delete</a>
13	Mutfak Ürünleri	<a href="#">Update</a>	<a href="#">Delete</a>
14	Kişisel Bakım	<a href="#">Update</a>	<a href="#">Delete</a>

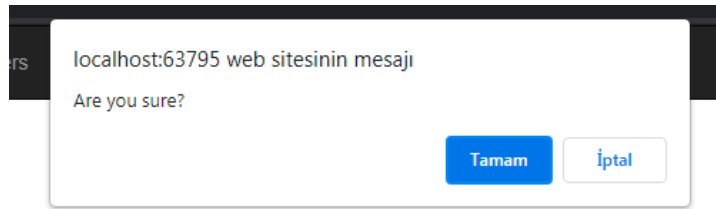
## Alert Oluşturma

Buton tıklanmalarına bildirim eklendi. Silme güncelleme ve ekleme işlemleri gerçekleştirildiğinde kullanıcıya bildirim gönderilmekte. Bunun için butona bir id verildi ve ilgili view üzerinde tanımlanan script içerisinde bildirim oluşturuldu.

```
<script>
    $('#btnAddMusteri').click(function(){
        alert("Customer Added.")
    });
</script>
```

Projeye alert kullanımı ile silme işlemi de eklendi. Kullanıcıdan silme işlemi sırasında son kez onay alınmakta ve onay verildiği takdirde silme işlemi gerçekleştirilmekte. Burada da bir `Html.ActionLink` tanımlandı ve içerisinde ilk olarak buton adı, sonrasında Controller içerisinde tanımlanan `ActionResult` belirtildi. Sonrasında tıklanan item'in id'si alındı. Tıklanması halinde bildirim mesajı belirtildi. Onay verilirse işlem gerçekleştirilecek, onaylanmaz ise iptal edilecek.

```
@Html.ActionLink("Delete Transact", "DeleteCustomer", new {id= c.CustomerId }, new
{ @class = "btn btn-warning", onclick = "return confirm('Are you sure?')" })
```



## Pop-Up Modal

PopUp Modal kullanarak sipariş detay sayfası oluşturuldu. Öncelikle modal'ı açacak buton düzenlenmekte. Buton geçişi ve hedefi 'data-toggle' ve 'data-target' ile belirleniyor.

```
<button type="button" class="btn btn-primary" data-toggle="modal" data-
target="#Modal1">Sale</button>
```

Daha sonra Modal dizaynına başlandı. İç içe 'div'ler oluşturularak modal yapısı kuruldu.

```
<div class="modal" id="Modal1">
    <div class="modal-dialog">
        <div class="modal-content">
```

Sınıfları belirtilen modal'lardan header içerisinde başlık tanımlandı.

```
        <div class="modal-header">
            <h2 class="modal-title" style="text-align:center">Order
Detail</h2>
        </div>
```

Daha sonrasında bir form oluşturularak içerisinde Modal'ın gövdesi tanımlandı. Buraya yerleştirilen `TextBox`'lar ile sipariş bilgileri kullanıcıdan alındı. 'form' etiketi içerisinde tanımlanan 'method' ve 'action' özellikleri ile post işlemi sırasında tetiklenmesi ve tetiklendiğinde `OrderController` içerisinde bulunan `NewOrder` adlı `ActionResult`'ın çağırılması sağlandı. Böylelikle girilen veriler, `ActionResult` içerisindeki kod bloğu sayesinde veri tabanına kaydedilebildi.

```

        <form method="post" action="/Order/NewOrder" class="form form-control"
        style="height:auto;">
            <div class="modal-body">
                <label>Product Name</label>
                @Html.TextBoxFor(m => m.Product, new { @class = "form-
control" })

```

Bir de onayla ve iptal butonları eklendi. İptal işlemi ise buton elemanın 'data-dismiss' özelliğiyle sağlandı. Modal kapatıldı.

```

        <button type="submit" class="btn btn-danger" data-
dismiss="modal">Cancel</button>

```

Search :

## Search Bar

Manuel olarak arama paneli oluşturuldu. Öncelikle controller içerisindeki listeleme işleminin gerçekleştirildiği Index ActionResult içerisinde değişiklikler yapıldı. Bu ActionResult artık string veri türünde bir parametre almakta. Varsayılan olarak tabloda listelenecek veri seti tüm müşterilerdir. Ancak parametre olarak gönderilen string ifade null veya boş değilse, parametre olarak girilen ifadenin veritabanında eşleştiği değerler getirilmekte.

```

public ActionResult Index(string p)
{
    var values = from v in db.Customer select v;
    if (!string.IsNullOrEmpty(p))
    {
        values = values.Where(m => m.CustomerName.Contains(p));
    }
    return View(values.ToList());
}

```

View içerisinde yapılmış değişiklik, Html.BeginForm oluşturulmasıdır. İçerisinde parametre olarak geçilecek string ifadenin girilebilmesi için TextBox ve istek oluşturabilmek için yerleştirilmiş bir buton. İstek oluşturduğunda varsa sahip olduğu parametre ile CustomerController içerisindeki Index ActionResult'a yönlendirilmekte. Daha sonrasında işlemler gerçekleştirilip varsa eşleşen değer, yoksa boş veri döndürülmekte.

```
@using (Html.BeginForm("Index", "Customer", FormMethod.Get))
{
    <p>
        <b>Search :</b> @Html.TextBox("p")
        <input type="submit" value="Search"/>
    </p>
}
```

## Jquery Eklentisi

Show  entries

Search:

Prod	Product Name	Brand	Category	Unit Price	Stock	Update	Delete
1	Çamaşır Makinesi	Arçelik	Beyaz Eşya	1235,00	15	<button>Update</button>	<button>Delete</button>
2	Bulaşık Makinesi	Beko	Beyaz Eşya	1350,00	10	<button>Update</button>	<button>Delete</button>
8	Buzdolabı	Regal	Beyaz Eşya	2700,00	5	<button>Update</button>	<button>Delete</button>
13	Derin Dondurucu	UĞUR	Beyaz Eşya	1200,00	3	<button>Update</button>	<button>Delete</button>

Showing 1 to 4 of 4 entries (filtered from 13 total entries)

Previous **1** Next

Son olarak var olan projeye NuGet Package Manager yardımıyla Datatable JQuery Eklentisi eklenerek listeleme işlemlerinde kullanılan tablolar daha kullanışlı hale getirildi.

Öncelikle View içerisinde tablonun 'id' değeri girildi. İndirilen eklentinin script dosyaları yine view içerisine script olarak eklendi. Son olarak küçük bir script komutu ile 'id' bilgisi verilen tablo üzerinde eklenti uygulandı.

Sol üst tarafta belirtilen kısımdan sayfa içerisinde kaç adet verinin listeleneceği seçilebilmekte. Arama yapabilmemiz için sağ üst köşede belirtilen eleman mevcut. Filtreleme işlemleri için kullanılabilmekte. Sağ altta sayfalama işlemi gerçekleştirildiğinde sayfalar arası geçiş için sayfa butonları mevcut. Tablonun her sütununda ise büyükten küçüğe veya küçükten büyüğe olmak üzere sıralama yapılabilmekte.