Veysel Uğur KIZMAZ



5. BASKI

OKU. İZLE. DİNLE. ÖĞREN!

ASP.NET MVC 5

İNTERAKTİF EĞİTİM SETİ HEDİYE!

KODLAB®
www.kodlab.com

ASP.NET MVC3 RAZOR

Volkan Aktaş

Tahsin Sevinç

KODLAB®
Yayın Dağıtım Yazılım ve Eğitim Hizmetleri San. ve Tic. Ltd. Şti.

KODLAB® 59

ASP.NET MVC3 RAZOR

VOLKAN AKTAŞ - TAHSİN SEVİNÇ

ISBN 978-605-4205-65-3
Yayıncılık Sertifika No: 13206

1. Baskı: Ekim 2011

Yayın Yönetmeni: Uğur Gelişken
Sayfa Düzeni: Mehmet Öztürk
Satış: Hüseyin Üstünel
Baskı: ÜÇ-ER Ofset Matbaa Yayın Reklam ve Amb. San. Tic. Ltd. Şti. Tel: (212) 629 03 15

Bu kitabın bütün yayın hakları Kodlab Yayın Dağıtım Yazılım ve Eğitim Hizmetleri San. ve Tic. Ltd. Şti.'ne aittir. Yayınevimizin yazılı izni olmaksızın kısmen veya tamamen alıntı yapılamaz, kopya çekilemez, çoğaltılamaz ve yayınlanamaz.

KODLAB Yayın Dağıtım Yazılım ve Eğitim Hizmetleri San. ve Tic. Ltd. Şti.
Alemdar Mah. Salkım Söğüt Sok. Ağa İş Hanı
No: 20 Kat:1 D: 3 Sultanahmet / İSTANBUL

tel: (212) 514 55 66 | **fax:** (212) 514 66 61
e-posta: bilgi@kodlab.com | **web:** www.kodlab.com

Volkan Aktaş

1976 Mersin doğumlu olup aslen Diyarbakır Erganili'dir. Lise öğrenimini Mersin Atatürk Endüstri Meslek Lisesi'nde Teknik Lise Bilgisayar Bölümü'nden, üniversite öğrenimini ise Gazi Üniversitesi Bilgisayar Öğretmenliği Bölümü'nden mezun olarak tamamlamıştır. 1999-2003 yılları arasında Şanlıurfa Endüstri Meslek Lisesi'nde bilgisayar öğretmeni olarak görev yapmış olup, 2003 yılından bu yana Mersin'de görev yapmaktadır. 2010 yılından itibaren de Mersin Kadri Şaman MTSO Teknik ve Endüstri Meslek Lisesi'nde görev yapmaktadır.

Visual Basic .NET, C# , ASP.NET, VBA, Python dilleri ile aktif olarak uygulamalar geliştirmekte olup bu dillerle yazdığı proje ve makalelerini *www.volkanaktas.com*, *www.vbnetnedir.com* ve *http://www.cyber-warrior.org* sitelerinde yayınlamaktadır.

Yazarımızın KODLAB'tan daha önce VB.NET 2010, Her Yönüyle C# 4.0 ve Visual Studio 2010 kitapları çıkmıştır.

Tahsin Sevinç

1987 Mersin doğumlu olup aslen Balıkesir Havranlı'dır. Üniversite eğitimini Mersin Üniversitesi Tarsus Teknik Eğitim Fakültesi Bilgisayar Öğretmenliği'nden mezun olarak tamamlamıştır. 2008-2009 Eğitim Öğretim yılında Özel Samanyolu Hasan Tanık İlköğretim Okulu'nda bilgisayar öğretmenliği yapmıştır. Bir sene öğretmenlik görevinden sonra, 2011 yılında özel bir yazılım şirketinde Web Developer olarak çalışmaya başlamıştır.

Programlaya 2004 yılında başlamıştır. C# ve ASP.NET dilleri ile birçok uygulama geliştirmiş olup, son olarak Üniversite Yönetim Sistemi Projesi'nde görev almaktadır. Yazdığı projelerin birçoğunu ve makalelerini *http://tahsin.sevinch.net* ve *http://www.cyber-warrior.org* sitelerinde yayınlamaktadır.

Yazarımız, Bilişim Güvenliği ve Bilişim Suçlarına Karşı Mücadele Derneği tarafından verilen Certified Web Security Expert unvanına sahiptir.

TEŞEKKÜR

Bu kitabın yazım sürecinde bizlere destek olan ailelerimize teşekkür ederiz.

Ayrıca içerik yönünden katkıda bulunan Mustafa KARAKURT'a, manevi desteklerini eksik etmeyen İrfan MERGAN, Bünyamin KARAHAN, Hale KATMER, İsmail Hakkı GÜVEN, Enver ARSLAN, İbrahim Erhan AŞI, Mustafa YAŞAR, Emrah ÇAPAR, Yakup Buğra BOZKURT, Seda TOKER, Elif BAŞARAL, Said KOTAY, Rabia KOTAY, Arif Emre AKTAŞ, Sevim KOTAY, Murat AKTAŞ, Hüseyin YILKIN ile adlarını burada sayamadığımız dostlarımız ve öğrencilerimize teşekkür ederiz.

ÖNSÖZ

MVC (**M***odel* **V***iew* **C***ontroller*) Framework, dinamik web siteleri, web uygulamaları ve web servisleri yazmak amacıyla geliştirilmiş olan bir web uygulamaları arayüzüdür. Microsoft firması tarafından 2007 yılı itibari ile geliştirilmeye başlanan MVC yeni bir tasarım kalıbı değildir. 80'li yıllardan beri farklı dil ve teknolojilerde kullanılmıştır.

MVC Framework, ASP.NET'in eksikliklerini kapatmak amacıyla geliştirilmiş bir sistemdir. MVC asıl gücünü, uygulamaları Model, View ve Controller olmak üzere üç farklı katmanda gerçekleştirilmesinden alır. Bu sayede bir web uygulaması üç kısım halinde parçalara bölünerek yürütülür. Her bir katmanda yürütülen işlemler diğer katmanlardan bağımsız biçimde gerçekleştirilir.

Visual Studio .NET içerisinde VB.NET ya da C# dillerini kullanarak MVC projeleri geliştirebilirsiniz.

Bu kitapta anlatılan MVC versiyonu olarak MVC'nin son sürümü olan MVC 3 Razor versiyonu tercih edilmiştir. Bununla, MVC'yi yeni öğrenmeye başlayan bir okuyucunun en son versiyonundan başlayarak öğrenmeye başlaması amaçlanmıştır.

Razor, biçimlendirme kodları ile Server Side kodlarını karıştırmak için geliştirilmiştir. Diğer bir ifadeyle web sayfalarına sunucu tabanlı kod gömmek için geliştirilmiş bir sözdizimidir. Yapı olarak Server Side dediğimiz sunucu taraflı kodlama dilleri olan ASP ve PHP dillerine benzemektedir. Razor ile oluşturulmuş bir web sayfasında Server Side ve Client Side kodları bulunmaktadır.

Okurlarımızın en azından temel düzeyde web tasarım ve web programlama bilgisine sahip olmaları, kitabımızdan en iyi şekilde istifade etmelerini sağlayacaktır. Her ne kadar kitabımızda konu içlerinde geçtikçe temel bilgilere yer versek de CSS, C#, ASP.NET gibi konuların her birinin ayrı birer kitap konusu olduğu aşikardır.

Kitapta elden geldiği kadar çok örneğe yer vermeye çalıştık. Hemen her konu örneklerle desteklenerek anlatılmıştır. Anlatılan her örnek denenmiş ve çalıştığı görüldükten sonra kitaba eklenmiştir.

Kitabımızın, tüm okuyucularımıza faydalı olmasını temenni ederiz.

VOLKAN AKTAŞ
TAHSİN SEVİNÇ

Bu kitabı, kardeşim Sevinç Uncu, eşi Murat Uncu ve çocuklarına ithaf ediyorum.

Volkan Aktaş

Bu kitabı, kitabın yazım sürecinde desteğini eksik etmeyen babama, anneme ve eşime ithaf ediyorum.

Tahsin Sevinç

İÇİNDEKİLER

1 MVC FRAMEWORK'E GİRİŞ 1
ASP.NET Sitelerinin Çalışma Sistemi 1
MVC Framework'un Çalışma Sistemi 10
ASP.NET ile MVC'nin Karşılaştırılması 15

2 MVC SAYFA TASARIMI 17
Layout Uygulama Örneği 17
RenderSection Metodu 29
RenderPage Metodu 33
Partial View 34

3 RAZOR VIEW ENGINE 39
View Engine 39
Razor View Engine 41
@ Operatörünün Kullanılması 41
Çok Satırlı Kod Blokları Oluşturma 42
Değişken Tanımlama ve Kullanımı 45
Boşluk Kullanımı 46
Sunucu Taraflı Yorum ve Açıklama Eklemek 47
Kontrol Yapıları ve Döngüler 47
Dosya ve Klasörlerin Yollarının Kodlarla Kullanımı 49
~ Operatörünün Kullanımı 50
Server.MapPath Metodu 51
Href Metodu 51

4 CONTROLLER 53
Controller'ın İşlevi 53
Action Kavramı 53
Controller Oluşturmak 54
Action Oluşturmak 55
Controller'dan View'e Veri Aktarımı 55

5 MODEL VE VERİTABANI İŞLEMLERİ 59
Model Katmanı 59
Model Validation 59

Modelleme Yaklaşımları 61
Veritabanı Kavramı 62
LINQ To SQL ile Veritabanı İşlemleri 63
Entity Framework ile Veritabanı İşlemleri 76
Entity Framework ile Code First Modelleme 89

6 MVC ROUTING 97
URL Yönlendirme 97
Global.asax.cs Dosyasının Varsayılan Kod Yapısı 99
Route Tanımlama 101
Route Tanımlarken Dikkat Edilmesi Gereken Kurallar 104
Route Tanımlama Örnekleri 106
Route için Kısıtlamalar Oluşturma 107
Düzenli İfadeler Kullanarak Kısıtlama Oluşturma 108
Http Metodu Kullanarak Kısıtlama Oluşturma 112
Özel Kısıtlamalar Oluşturma 113
Gelen URL İsteklerinin Tamamının Eşleşmesi 119
Catch-All Parametresi ile 404-Sayfa Bulunamadı Oluşturmak 120
Route Ignore Etmek 122
NotEqual Kısıtlaması Oluşturmak 123
MVC Areas Routing 124

7 HELPER 129
Html Helper Metotları 129
ActionLink 129
BeginForm 130
Label 131
TextBox 131
TextArea 131
CheckBox 132
RadioButton 132
DropDownList 133
ListBox 134
Password 135
ValidationSummary 138
ValidationMessage 140

Web Helper Metotları 156
Chart 156
WebMail 160
Crypto 160
WebImage 165
WebGrid 175
Microsoft Web Helper Metotları 184
Bing 187
Facebook 188
Twitter 203
LinkShare 214
Video 215
Analytics 221
Recaptcha 225
PayPal Helper 228
Kullanıcı Tanımlı Helper Oluşturmak 237
Static Metot 237
Extension Metot 241

8 FİLTRELER 245
Authorizaion Filter 247
Exception Filter 259
Action ve Result Filter 265
Filtrelerde Sıralama 268

9 AJAX 273
MVC ve Ajax 273
MVC 3 Ajax İşlemleri 274
Ajax.ActionLink AjaxHelper 274
Ajax.BeginForm 277

10 MVC VE SILVERLIGHT 283

11 JQUERY 293
JQuery'yi Tanımak 293
Temel Seçiciler 294
Seçicisi 294

. Class Seçicisi 295
[ ] Öz Nitelik Seçicisi 296
Element Seçicisi 296
: Form Elemanı Seçicisi 297
Animasyon Efektleri 298
Gösterip Gizleme 298
Görünürlüğünü Değiştirme 298
SlideUp/SlideDown 299
JQuery UI 300
JSON 305

12 MVC 3 PROJESİNİN YAYINLANMASI 309

13 HABER PORTALI UYGULAMASI 315
Model Yapısı 315
Haber ve Kategori Ekleme 316
Global.asax Sayfamızda Route Ayarları 319
Haberlerin Görüntülenmesi 320

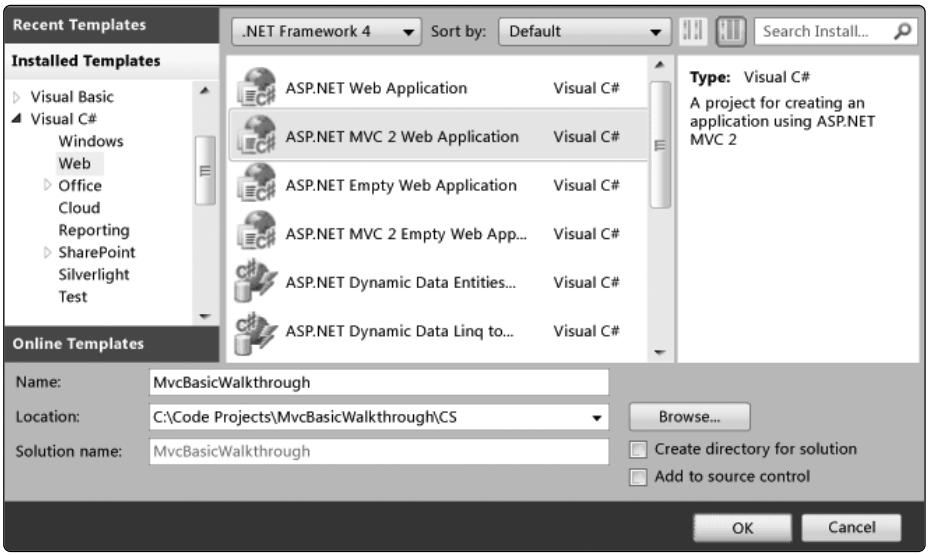
MVC FRAMEWORK'E GİRİŞ

1

MVC (**M***odel* **V***iew* **C***ontroller*) Framework; dinamik web siteleri, web uygulamaları ve web servisleri yazmak amacıyla geliştirilmiş olan bir web uygulamaları arayüzüdür. Microsoft firması tarafından 2007 yılı itibari ile geliştirilmeye başlanan **MVC Framework**'un geliştirilme süreci aşağıdaki gibidir:

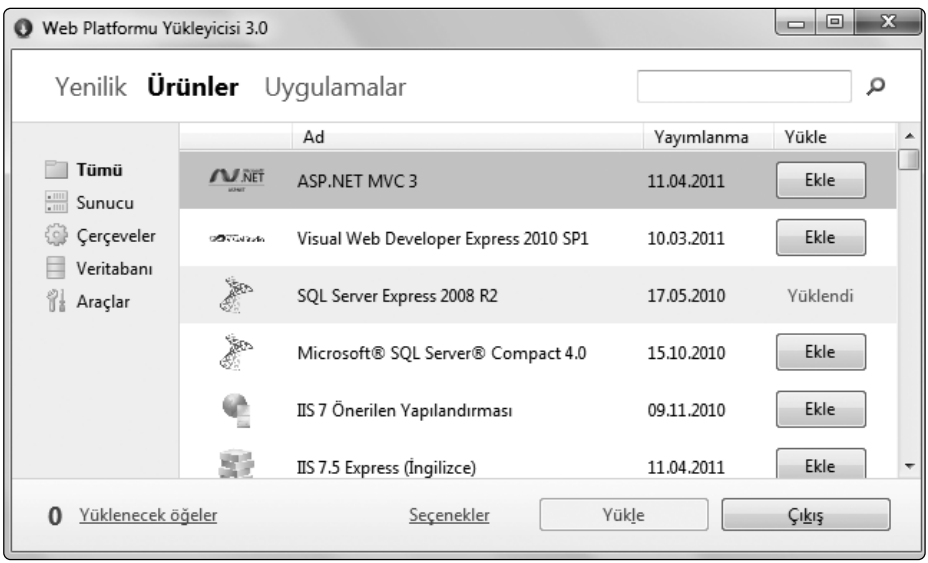
- 10 Aralık 2007 ASP.NET MVC CTP
- 13 Mart 2009 ASP.NET MVC 1.0
- 10 Mart 2010 ASP.NET MVC 2.0
- 13 Ocak 2011 ASP.NET MVC 3.0
- 11 Nisan 2011 ASP.NET MVC 3.1

MVC Framework 2 versiyonu, Visual Studio 2010 versiyonu ile varsayılan olarak yüklü olarak gelmektedir.

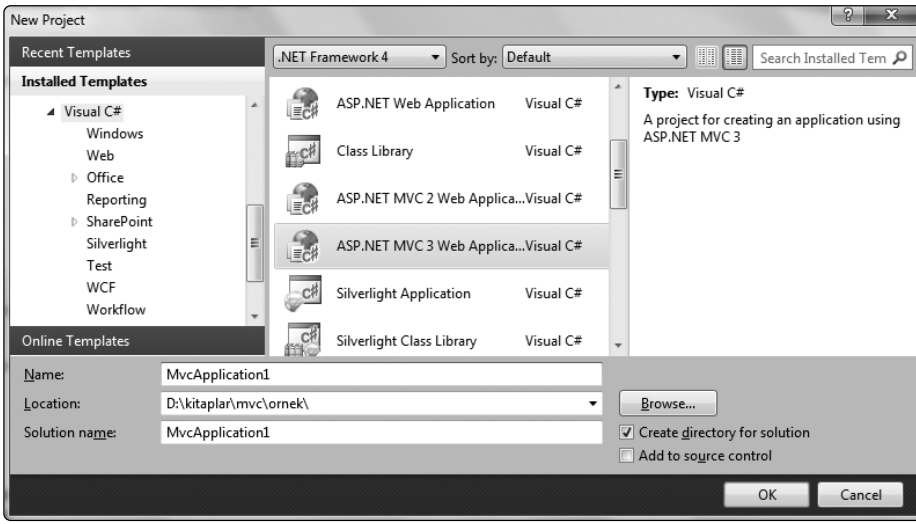


MVC 3 sürümünü Visual Studio içerisine yüklemek için ise;

http://www.asp.net/mvc/mvc3 adresini kullanarak yükleme dosyasını indiriyoruz.



İndirdiğimiz dosyayı çalıştırıp karşımıza gelen **Web Platform Installer** penceresi içerisinden **ASP.NET MVC 3** uygulamasını seçerek Visual Studio 2010 içerisine yüklüyoruz.



MVC Framework yeni bir tasarım kalıbı değildir. 80'li yıllardan beri farklı dil ve teknolojilerde kullanılmıştır. Microsoft'un geliştirdiği MVC versiyonu, ASP.NET tabanlı olarak çalışan ve geliştiricilerin **Model**, **View** ve **Controller** adlı üç katmanı kullanarak uygulama geliştirebilecekleri bir arayüzdür. MVC Framework ASP.NET'in eksikliklerini kapatmak amacıyla geliştirilmiş bir sistemdir. Bu nedenle öncelikle ASP.NET'in çalışma sistemini inceleyip, ardından MVC'nin çalışma sistemini inceleyerek birbiri ile karşılaştıralım. Bu şekilde MVC'nin geliştirilmesine neden ihtiyaç duyulduğunu daha iyi anlayabiliriz.

ASP.NET Sitelerinin Çalışma Sistemi

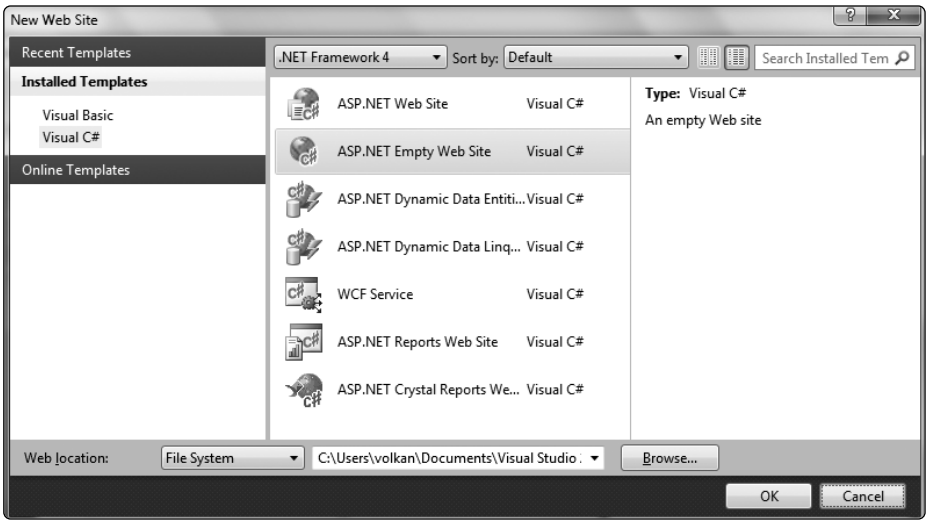
2002 yılında ASP.NET yayınlandığında web programlamada yeni bir çığır açmıştı. Masaüstü programlama ile ilgilenen pek çok programcı; ASP.NET'in sağladığı ve daha önce yalnızca Visual Basic, VB.NET, Delphi gibi dillerde bulunan olay güdümlü programlama, sürükle-bırak gibi kavramları web programlamada kullanma imkanına kavuştular. Bu da pek çok masaüstü programcısının aynı zamanda web programcısı olmasını da sağladı. Masaüstü programı geliştirir gibi web uygulamaları gelişti-

rilmeye başlandı. Microsoft'un sağladığı bu kolaylık beraberinde pek çok problemi de getirdi. Özellikle **ViewState**, **PostBack** gibi yapılar web teknolojisine uygun değildi. MVC ise ASP.NET'i tamamlayan ve eksikliklerini kapatan bir teknoloji olarak geliştirilmiştir. Dolayısı ile ASP.NET'den MVC'ye geçiş yapanların da işlerinin ne kadar kolaylaştıklarını gördüklerinde geçiş sürecinin hızlanacağını düşünüyoruz.

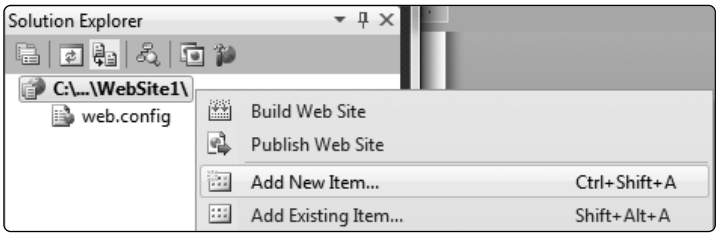
ASP.NET siteleri içerisinde kullanılan web formlarında oluşturulmuş olan kontrollere ait veriler **ViewState** adı verilen bir mekanizma ile **Hidden Input**'larla saklanmaktadır. Ancak kontrollerin sayısı arttıkça hem Hidden Input'ların boyutu artmakta hem de bu esnada oluşturulan html çıktıları karmaşıklaşmaktadır. Bu da istemci ile sunucu arasındaki trafiğin artmasına ve sayfaların yavaşlamasına neden olmaktadır.

ASP.NET sitelerinin çalışma sistemini basit bir web uygulaması eşliğinde inceleyelim.

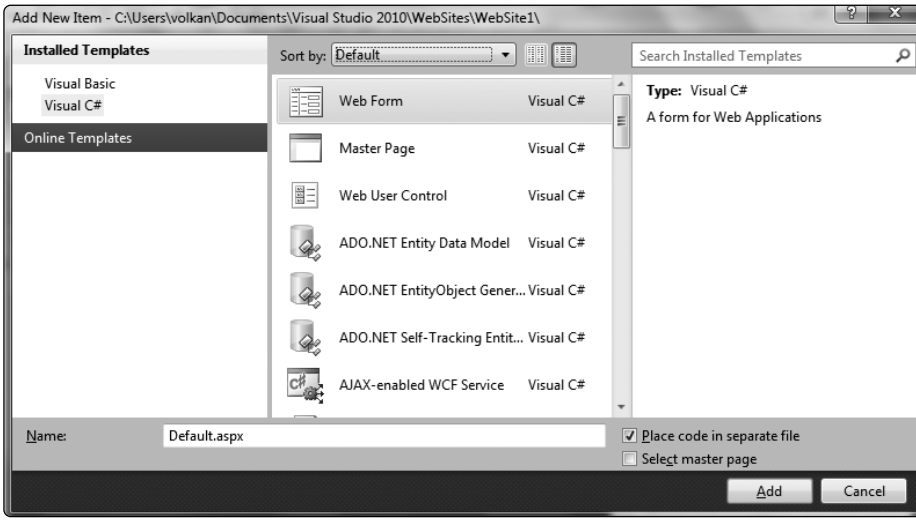
Visual Studio arabirimi içerisinde **File / New Web Site** menüsü ya da **`Shift+Alt+N`** kısayol tuşları ile gelen **New Web Site** penceresinden **ASP. NET Empty Web Site** seçeneğini seçerek boş bir ASP.NET web site projesi açıyoruz.



Solution Explorer içerisinde site ismine farenin sağ tuşu ile tıklayıp, **Add New Item** seçeneğini kullanarak...



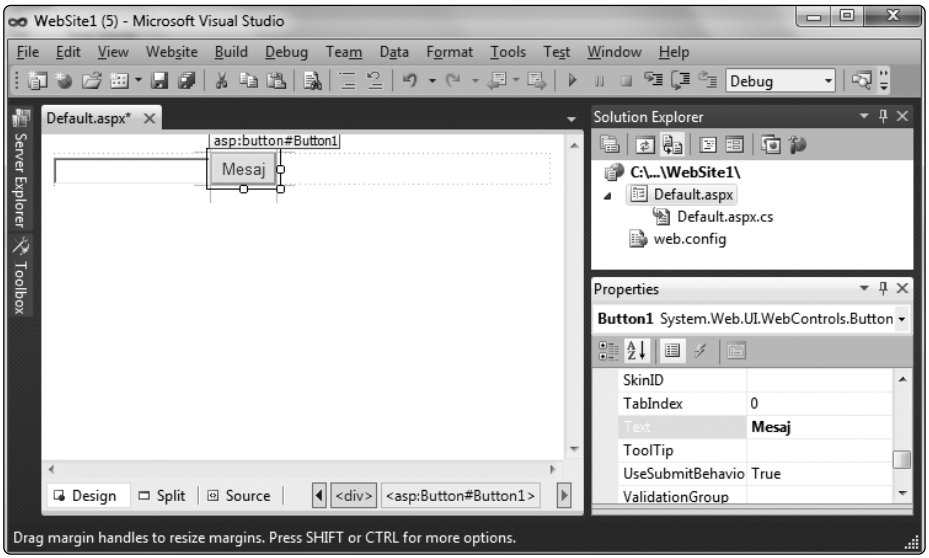
... ya da **Web Site** menüsünden **Add New Item** seçeneğini kullanarak veya **`Ctrl+Shift+A`** kısayol tuşları ile sitemiz içerisine yeni bir **Web Form** ekliyoruz.



Bu işlemin ardından **Default.aspx** sayfası eklenecektir. Farklı bir sayfa belirlemediğiniz sürece **Default.aspx** sayfası ana sayfanızdır. **Default.aspx** sayfasına tasarım görünümünde iken ekranın sol tarafında bulunan **Toolbox** içerisinden bir adet **TextBox** ve bir adet de **Button** kontrolü ekleyelim.

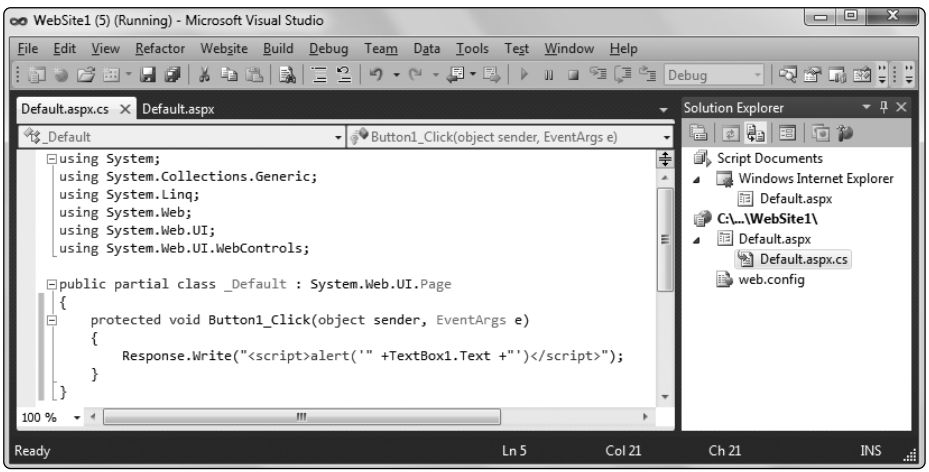


Bu işlemin ardından **Button** kontrolü seçili iken **Properties** penceresinden **Text** özelliğini **Mesaj** olarak değiştirelim.



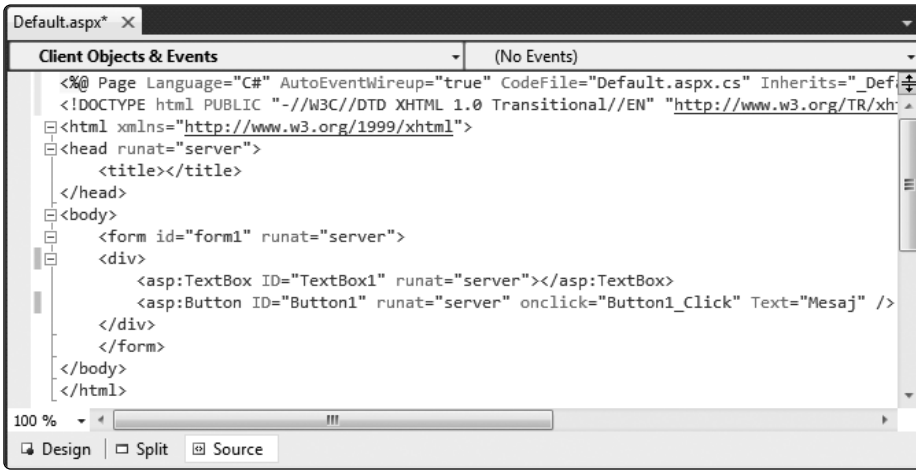
Tasarım ile ilgili tüm işlemler **Default.aspx** sayfasının tasarım görünümünde gerçekleştirilmektedir. C# kod ortamına geçerek **Mesaj** butonu için bir kod yazalım. Kod satırlarını **Default.aspx.cs** dosyasına yazacağız. Kodumuzu butonun tıklanma olayına yazacağımız için butona çift tıklamamız yeterlidir.

Kod satırlarımız aşağıdaki gibi olacak:

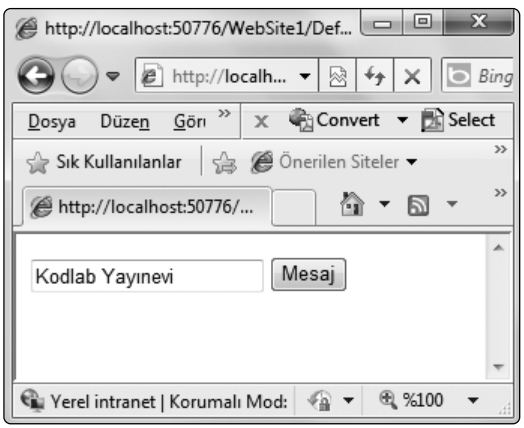


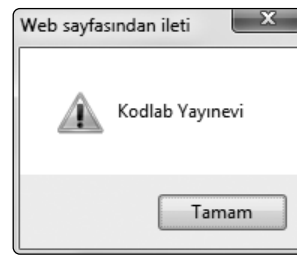
`Response.Write` ile client taraflı olarak ekrana **TextBox**'ın içeriğini bir mesaj penceresi içerisinde yazdırdık.

Uygulamamızın **aspx** kodunu inceleyelim. Bunun için kod penceresinin alt tarafında bulunan **Source** butonuna tıklayarak geçiş yapabiliriz.



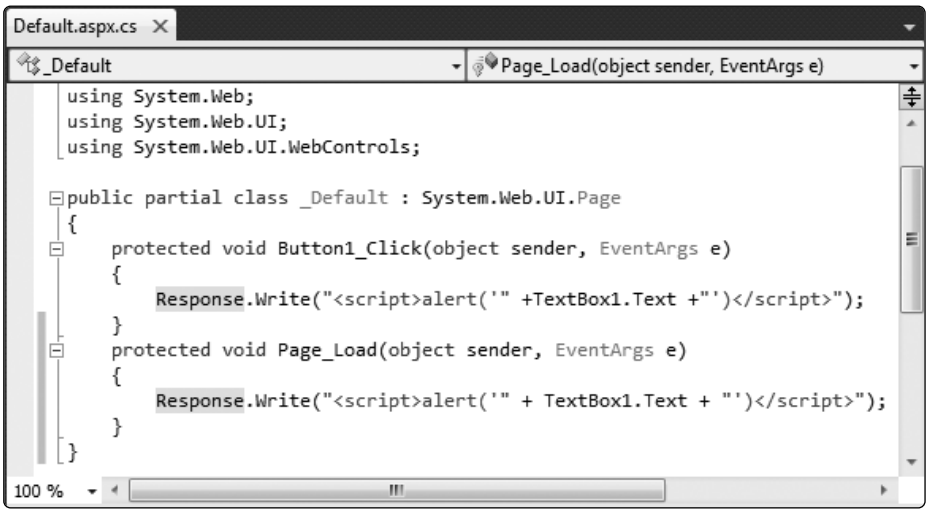
Sayfamızın **body** bloğunda bir form içerisinde **TextBox** ve **Button** kontrollerinin tanımlandığını görüyoruz. Örneğimizi **Debug** menüsünden **Start Debugging**, **F5** kısayol tuşu ile ya da **Default.aspx** sayfasına farenin sağ tuşu ile tıklayarak **View in Browser** seçeneği ile çalıştırabiliriz.





TextBox içerisinde yazıyı yazıp ardından **Mesaj** butonuna tıkladığımızda Textbox içerisindeki yazı mesaj penceresi içerisine görüntülenecektir.

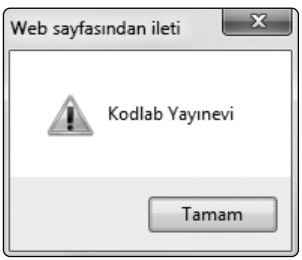
Örneğimizi bir adım daha geliştirelim. Tasarım görünümünde iken forma çift tıklayalım ve formun `Load` olayını görüntüleyelim.



Uygulamamızı derlediğimizde sayfa ilk yüklendiğinde ekrana boş bir mesaj penceresi gelmektedir. Pencere içeriğinin boş olması henüz TextBox içerisine bir değer girmemiş olmamızdan kaynaklanmaktadır.



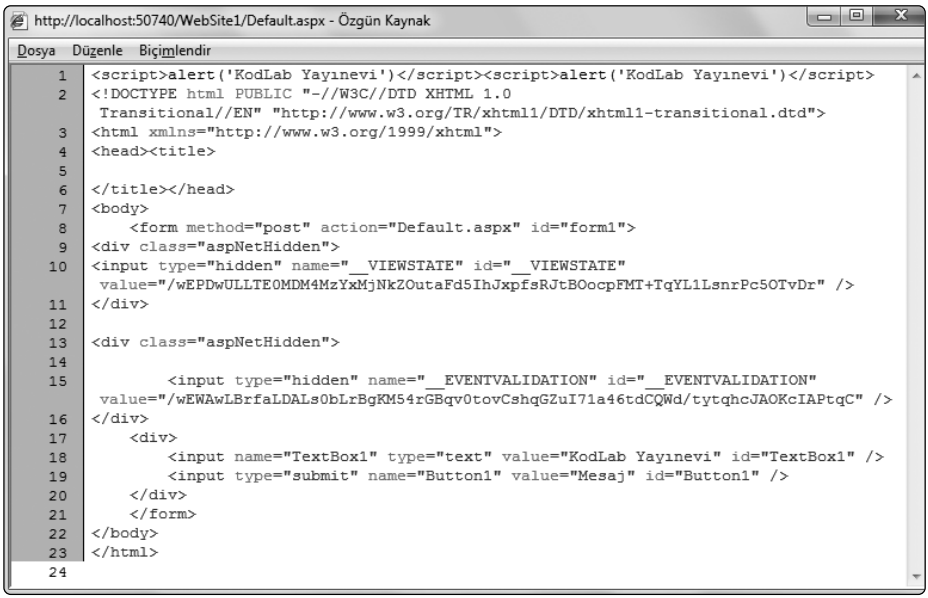
TextBox içerisine değer girip butona tıklayalım. Ardından sayfayı yenileyelim. Ekrana, TextBox içerisine girdiğimiz metnin yer aldığı mesaj penceresi gelecektir.



Ancak sayfayı her yenilediğimizde karşımıza yine mesaj penceresi gelecektir. Kullanıcı, **TextBox** içerisine yazı yazıp **Tamam** butonuna tıkladığında, tarayıcı verileri kendine **PostBack** metoduyla gönderiyor. ASP.NET Framework, gelen bu verilerden gerçekleşen olayın butona tıklama olayı olduğunu anlıyor ve `click_button` ola-

yını çalıştırıyor. İşlem tamamlandığında formda girilmiş olan veriler **Hidden Input**'larla saklanıyor ve kullanıcı sayfayı yenilediğinde daha önceden girilmiş olan veriler aynen karşısına geliyor (*ViewState metodu*).

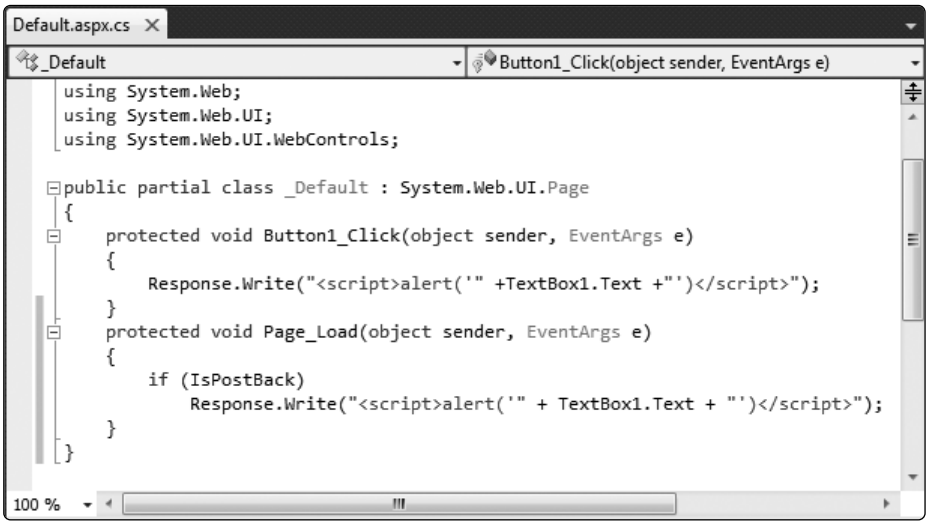
Kod kaynağını görüntüleyip HTML kodunu inceleyelim.



Kullanıcı tarafında girilen verilerin **Hidden Input**'lar içerisinde tutulduğunu kodumuzda görebiliyoruz.

Örneğimizde, verilerin daha önceden **PostBack** yapılıp yapılmadığını kontrol ederek eğer PostBack yapılmışsa mesaj penceresinin görüntülenmesini sağlayabiliriz. Bu sayede sayfanın ilk yüklenişinde, yani henüz bir PostBack işlemi yapılmadan mesaj penceresini görüntülenmesini engelleyerek, içeriği boş olan bir mesaj penceresi görüntülenmesini engelleyebiliriz.

Default.aspx.cs sayfasının kodlarını aşağıdaki gibi değiştirelim.



Uygulamamızı yeniden çalıştırdığımızda, sayfanın ilk yüklenişinde ekrana bir mesaj penceresi gelmeyip, butona tıklamamızın ardından gerçekleşen PostBack işleminin ardından, sayfanın sonraki yüklenişlerinde ekrana mesaj penceresi gelecektir.

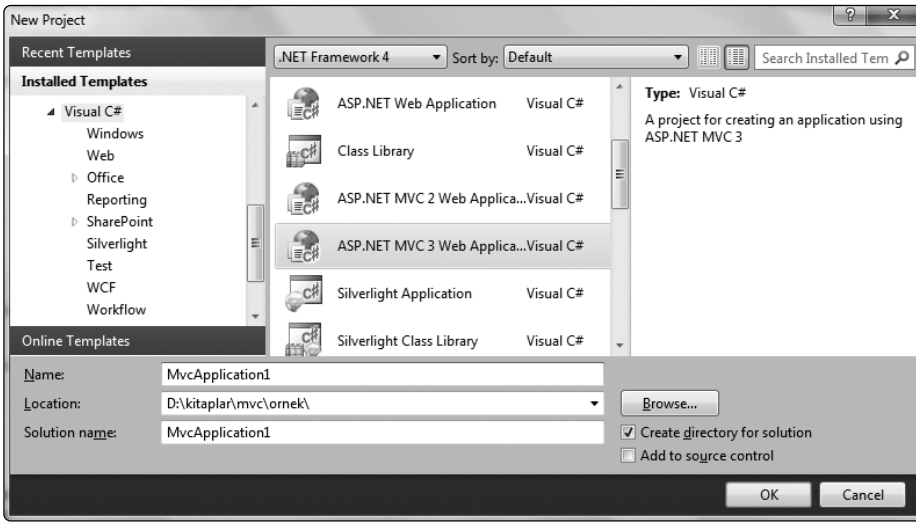
MVC Framework'un Çalışma Sistemi

ASP.NET, klasik masaüstü programlamadan web programlamaya geçiş sürecinde yeni bir çığır açmış olsa da web formlarında bulunan kontroller üzerinde kullanıcıların gerçekleştirdiği olaylara dayalı olarak gerçekleşen kodlama sistemi, görsel katmanlar ile kod katmanlarının iç içe girmesine neden olmuştur.

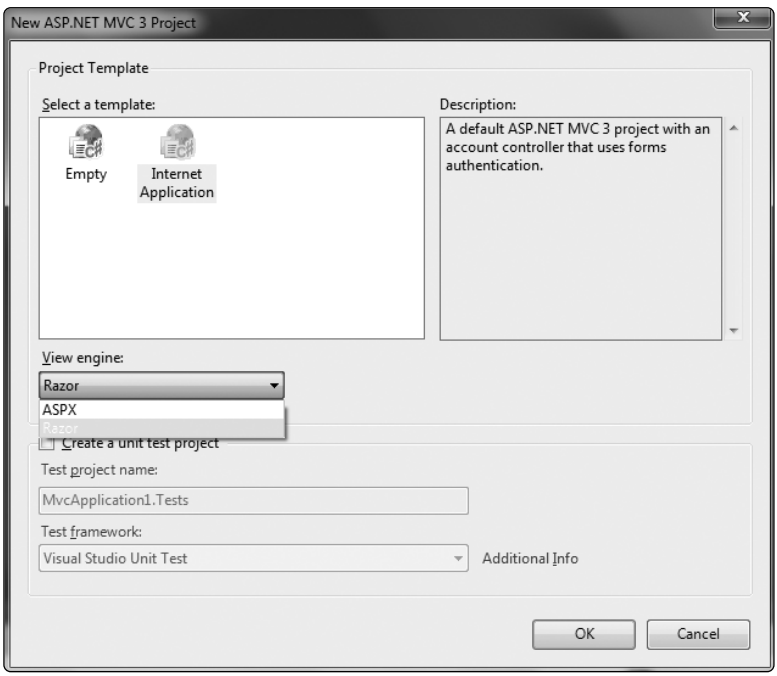
MVC Framework içerisinde ise uygulamalar **Model**, **View** ve **Controller** olmak üzere üç farklı katmanda gerçekleştirilir.

MVC Framework'un çalışma sistemini basit bir web uygulaması eşliğinde inceleyelim.

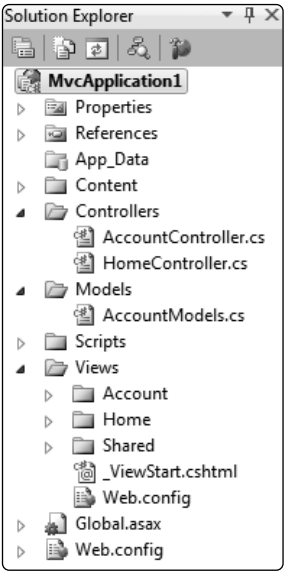
Visual Studio arabirimi içerisinde **File / New Project** menüsü ya da **`Ctrl+Shift+N`** kısayol tuşları ile gelen **New Project** penceresinden **ASP. NET MVC 3 Web Application** seçeneğini seçerek yeni bir MVC 3 web uygulaması açıyoruz.



Bu işlemin ardından karşımıza gelen **Project Template** penceresinden **Internet Application** seçeneğini seçiyoruz.



Internet Application seçeneği, içerisinde basitçe bir kullanıcı giriş sistemi barındıran örnek bir web uygulamasıdır. İçeriğini incelemek amacıyla bu proje tipini seçiyoruz.



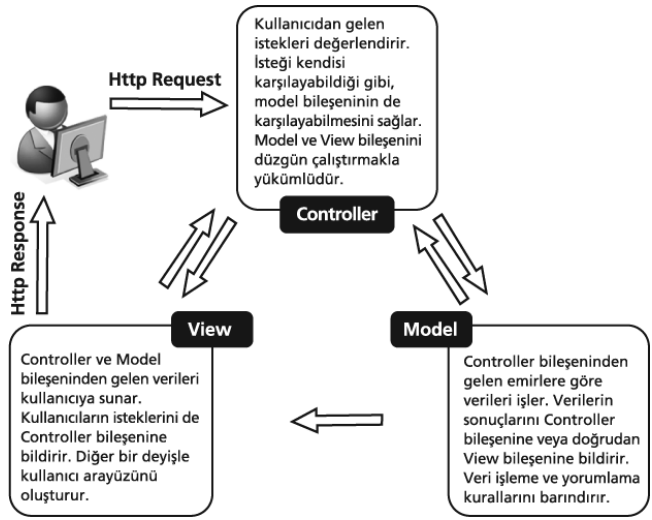
View Engine bölümünde ise kullanıcı tarafında görüntülenecek olan sayfa formatını seçiyoruz. **ASPX** ve **Razor** olmak üzere iki seçeneğimiz mevcut. **ASPX**; klasik web formlarını temsil ederken, **Razor** ise MVC 3 ile birlikte gelen bir biçimdir. **Razor View Engine** ile artık **HTML** ve **C#** ya da **VB.NET** kodlarını iç içe kullanabilmekteyiz. Razor'un ayrıntılarına daha sonra değineceğiz.

Projemizin açılmasının ardından **Solution Explorer** içerisini incelediğimizde toplam altı adet klasör yer aldığını görürüz. Bunlar MVC yapısına ait olup varsayılan olarak gelen klasörlerdir.

Solution Explorer içerisinde yer alan bu klasörleri inceleyelim. Klasörleri kısaca aşağıdaki tablo ile özetleyebiliriz.

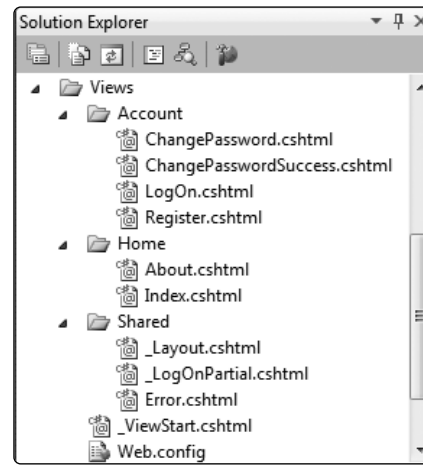
Klasör	Açıklama
Controllers	Kullanıcıdan gelen isteklere göre **Model** ve **View** üzerinde gerçekleştirilecek işlemleri yönetir.
Models	Web uygulamalarının verilerini ve yapacağı davranışları yönetir.
Views	Kullanıcı tarafında görüntülenecek olan sayfaların yer aldığı klasördür.
Scripts	JavaScript ve Ajax dosyaları gibi içeriklerin yer aldığı klasördür.
Content	CSS ve imaj dosyaları gibi JavaScript dosyaları dışındaki içeriklerin yer aldığı klasördür.
App_Data	Veritabanı dosyalarının yer aldığı klasördür.

Models, **Views**, **Controllers** adlı klasörler; MVC Framework'un ana yapısını teşkil etmektedir. **Model View Controller** yapısı sayesinde bir web uygulaması üç kısım halinde parçalara bölünerek yürütülür. Her bir katmanda yürütülen işlemler, diğer katmanlardan bağımsız biçimde gerçekleştirilir.



Model; uygulamanın veri işleme ve yorumlama kurallarını barındırır. Web uygulamasının verilerini ve yapacağı davranışları yönetir. Yaptığı işlemlerin sonuçlarını Controller bileşenine bildirmekle yükümlüdür.

Kullanıcıdan gelen isteklere cevap verir. **Models** klasörü içerisinde veritabanı işlemlerini yürütmek için gerekli sorgular ve kodlar mevcuttur. Örneğimizde kullanıcı login işlemleri için `AccountModels` isminde bir Model oluşturulmuştur.



View; kullanıcı arabirimlerini barındırır. Kullanıcıya verilerin nasıl sunulacağı bu bölümde tasarlanır. Kullanıcının isteklerini **Controller** bileşenine iletir. **Views** içerisinde oluşturulan klasörlerden **Home** ve **Shared** klasörleri MVC içerisinde standart olarak yer alan klasörlerdir.

Home klasörü içerisinde genellikle index sayfası yer alırken, **Shared** klasörü içerisinde ise standart olarak Layout sayfası bulunmaktadır. **Layout**; ASP.NET içerisinde bulunan **MasterPage**'in görevini yapmaktadır. Yani sayfalarda sabit olan içerikleri Layout içerisinde tanımlarız. Bir nevi şablon görevi yapmaktadır.

Layout ile ilgili tanımlama **_ViewStart.cshtml** sayfası içerisinde yapılmaktadır. Örnek uygulamamızdaki sayfayı açarsak içerisinde...

```
@{
     Layout = "~/Views/Shared/_Layout.cshtml";
}
```

... şeklinde yol tanımlaması yapıldığını görebiliriz.

Controller; kullanıcının yapacağı girdi işlemlerine göre bu iki katmanı düzgün çalıştırmakla yükümlüdür. Kullanıcı sayfadan bir istekte bulunduğunda (*request*, *input*) Controller bunu denetler ve Model de gelen işlemleri yürütür. Daha sonra View de kullanıcıya isteğinin karşılığını gösterir.

Controller bir denetçi gibi çalışır. Kullanıcının fare ve klavye girişlerini yorumlar. Kullanıcıdan gelen isteklere göre uygulamanın hangi davranışı sergileyeceği burada tanımlanır. Kullanıcıya verilecek cevap için arayüzü belirler. **Model** ve **View** bileşenlerinin yapacakları işlevleri belirler.

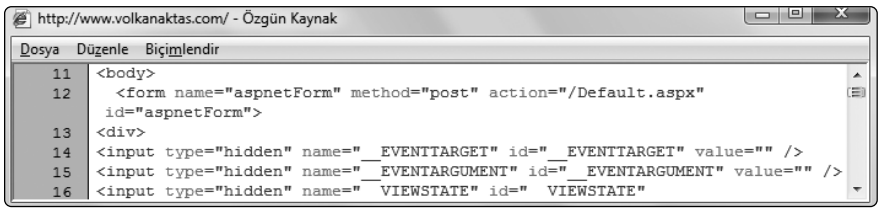
Controllers klasörü içerisinde **Account** işlemleri için kullanacağımız `AccountController` adlı bir sınıf ve ana sayfada gerçekleştirilecek işlemler için kullanacağımız `HomeController` adlı bir sınıf yer alır. Bu sınıflar içerisinde gerçekleştirilecek işlemlere dair metotlar yer alır.

Bu metotlar ile kullanıcı tarafından gelen isteklere karşılık gerekli işlemler yürütülür ve **Model** ve **View** katmanları yönlendirilir. Örnek uygulamamızı çalıştırdığımızda anasayfa aşağıdaki gibi görüntülenir.



ASP.NET İLE MVC'NİN KARŞILAŞTIRILMASI

- ASP.NET içerisinde kullanılan **ViewState** yapısı ile web formlar içerisinde kullanılan kontrollerle ilgili bilgiler ve kullanıcıların bu kontrolleri kullanarak gerçekleştirdiği olaylara dair bilgiler, **Hidden Input**'lar içerisinde saklanmaktadır. Siteye bağlanan kişi sayısı arttıkça Hidden Input'lar içerisinde saklanan verilerin boyutu da artmakta ve bu da siteye bağlanma hızının yavaşlamasına neden olmaktadır.



Yukarıda örnek olarak ASP.NET ile hazırlanmış olan kendi bloğumdaki **Hidden Input**'lar görünmektedir.

MVC Framework yapısında ise **Hidden Input** yoktur. Aşağıda MVC Framework'un örnek uygulamasının index sayfası kaynak kodları görüntülenmektedir. Görüldüğü gibi Hidden Input kullanılmamıştır.



Hidden Input yapısının kullanılmamasından dolayı, kullanıcı trafiğinin artması ile ASP.NET sayfalarında yaşanabilecek bağlantı hızı düşüklüğü problemi MVC sitelerde yaşanmaz.

- ASP.NET'in dezavantajlarından biri de `isPostBack` kontrolü yapmak zorunda olmanız. Örneğin; kullanıcı bir form açtı ve formda input alanlarını doldurduktan sonra gönder butonuna tıkladı. Sayfa açılırken `page_load` olayı da tetiklenecektir. Bu durumda sayfa ilk defa mı açıldığı, yoksa PostBack mi yapılmış diye kontrol yapılmazsa; `load` olayındaki işlem iki kere tekrar edilmiş olur. Başka bir örnek daha verecek olursak; otomatik güvenlik kodu üreten bir sayfaya girdiğimizde, karşımıza çıkan güvenlik kodunu girip tekrar gönder tuşuna bastığımızda, sayfanın ilk defa mı açıldığı yoksa PostBack mi yapıldığı belli olmadığı için, olay tekrar tetiklenecektir ve sayfa yine güvenlik kodu üretecektir. Bu durumda kullanıcının girdiği kodun hiçbir anlamı kalmayacaktır. Çünkü kod değişmiştir.

  MVC Framework içerisinde ise **PostBack** ve **ViewState** yoktur. Bu nedenle web uygulamalarının çalışma aşamalarında geliştiricilere daha fazla kontrol imkanı sağlamaktadır.
- MVC Framework, uygulamamızı **Model View Controller** bileşenlerine bölerek karmaşıklığı önler ve yönetmeyi kolaylaştırır. MVC mimarisinde tasarım katmanları ve mantıksal katmanlar ayrı olduğu için geliştirilmeye ve ekip çalışmasına daha uygundur. Tasarımı biri yaparken mantıksal katmanları başkası yapabilir.
- MVC Framework yapısında sunucu tabanlı formlar kullanmaz. Uygulama geliştirirken tam etkin denetim için idealdir.
- MVC Framework test edilebilirliği açısından daha iyidir. MVC Framework ile birlikte **TDD** (**T***est* **D***riven* **D***evelopment*) desteği gelmektedir.
- ASP.NET ile beraber tanıştığımız **MasterPage** özelliği MVC 2'de aynen karşımıza çıktığı gibi **MVC 3 Razor**'da daha da geliştirilmiş haliyle Layout sayfalarıyla karşımıza çıkmaktadır.
- MVC Framework içerisinde ASP.NET'de karşılaştığımız **Authentication** ve **Membership** özelliklerini kullanabiliyoruz.
- ASP.NET içerisinde dahili olarak bir **URL Routing** mekanizması yoktu. Ek kodlamalarla **URL Rewriting** şeklinde ASP.NET sitelerine bu destek eklenmekteydi. Ancak MVC içerisinde bu mekanizma mevcut. MVC içerisinde bu yapı oldukça güçlendirilmiş bir biçimde kolayca kullanılabilmektedir. Sayfalara, sayfa uzantılarını yazmadan erişmek mümkündür. Bu sayede Google gibi arama motorlarının MVC ile yapılmış web sayfalarını daha kolay bir şekilde indekslemesi sağlanmakta. Bu hali ile MVC güçlü bir SEO desteğine sahip bulunmaktadır.

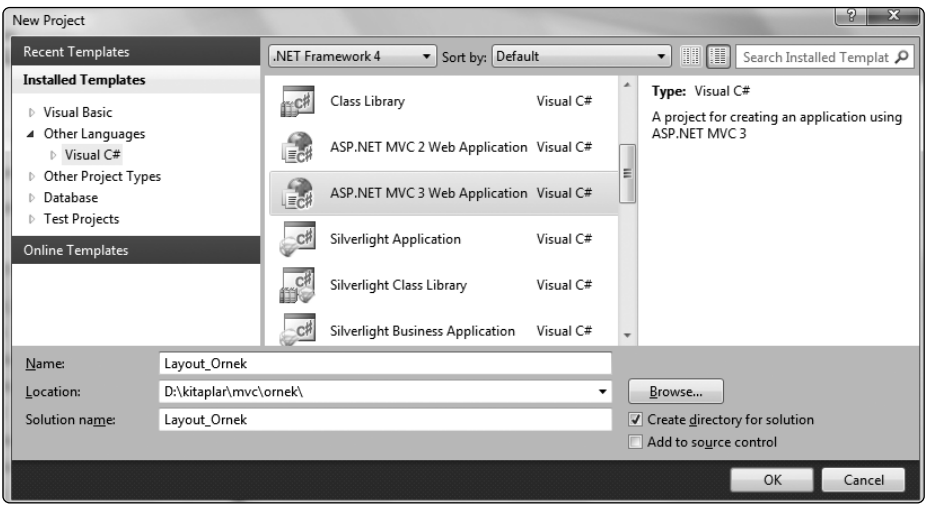
MVC SAYFA TASARIMI

2

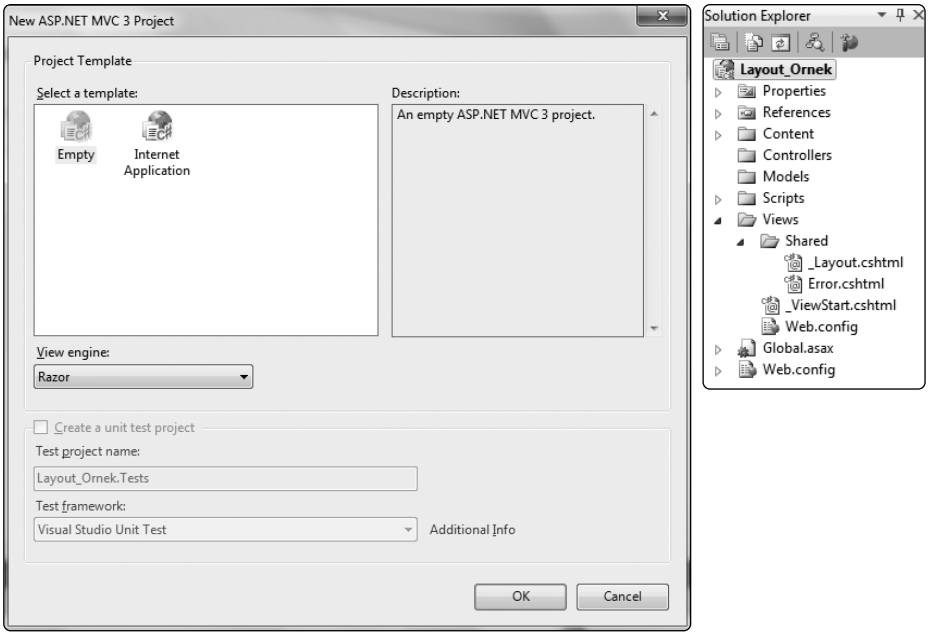
MVC (**M***odel* **V***iew* **C***ontroller*) içerisinde tasarladığımız sayfalarımızı çağırırken her sayfada sabit kalıp, değişmeyen içerik ve tasarımları ayrı bölümler ya da sayfalar olarak tasarlayıp direkt olarak bu sayfaları çağırabileceğimiz gibi, bu içerik ve tasarımları Layout sayfaları içerisinde de (ASP.NET içerisindeki MasterPage yapısı) tanımlayarak her sayfadan erişilmesini sağlayabiliriz. Değişen içerikleri ise **View** adı verilen sayfalar içerisinde görüntüleyebiliriz. View'ler içerisinde kullanıcıya verilerin nasıl sunulacağı tasarlanır. Kullanıcının isteklerini **Controller** bileşenine iletir. Layout için örnek bir uygulama yapalım.

Layout Uygulama Örneği

Yeni bir **ASP.NET MVC 3 Web Application** oluşturuyoruz.



Bu işlemin ardından karşımıza gelen **Project Template** penceresinden **Empty** seçeneğini seçiyoruz. **View Engine** olarak da **Razor View Engine**'ı seçiyoruz.



Boş bir **MVC 3 Razor** projesini bu şekilde oluşturmuş olduk. **Solution Explorer** içerisini incelediğimizde klasör yapısının yukarıda sağ yandaki resimde olduğu gibi oluşturulduğunu görürüz. **Layout** sayfası **Views** içerisinde **Shared** klasörü altında **_Layout.cshtml** adıyla oluşturulurken, **_ViewStart.cshtml** dosyası da **Views** içerisinde oluşturulur.

_ViewStart.cshtml dosyası içerisinde Layout dosyasının yolu belirtilmektedir.



Layout sayfası içeriği varsayılan olarak aşağıdaki gibi oluşturulur.

_Layout.cshtml

Client Objects & Events | (No Events)

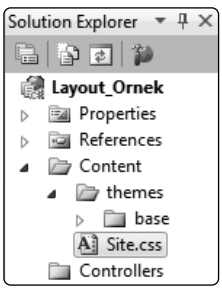
```
<!DOCTYPE html>
<html>
<head>
    <title>@ViewBag.Title</title>
    <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet" type="text/css" />
    <script src="@Url.Content("~/Scripts/jquery-1.4.4.min.js")" type="text/javascript"></script>
</head>

<body>
    @RenderBody()
</body>
</html>
```

`ViewBag.Title` kod satırı ile sayfa başlığını görüntülüyoruz. `ViewBag`, `Controller` sınıfındaki verileri alarak **View** içerisinde görüntülemek amacıyla kullanılan bir metottur. Anlık olarak oluşturulur, işlemin bitmesinin ardından hafızadan silinir.



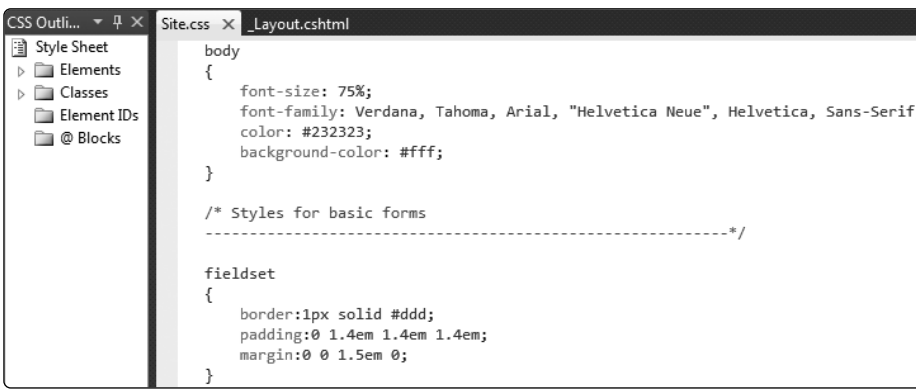
MVC 3 öncesi sürümlerde `ViewBag` metodu, `ViewTitle["Görüntülenecek olan içerik"]` şeklinde kullanılmaktaydı. MVC 3, halen bu metoda destek vermektedir.



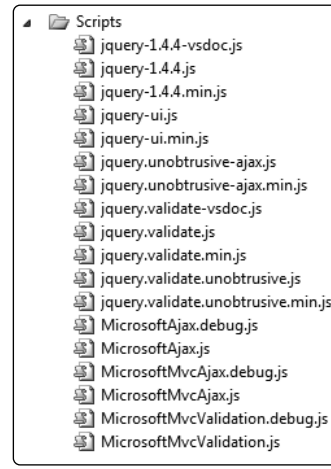
`<link href="@Url.Content...` satırında, sayfada görüntülenecek olan nesnelerin görsel özelliklerine dair tanımlamaları içeren **CSS** dosyamızın yolu belirtilmektedir.

CSS dosyası **Content** klasörü altında **Site.css** ismi ile yer almaktadır.

Site.css dosyası içeriği varsayılan olarak aşağıda görüldüğü gibidir.



İçeriğe dair bazı varsayılan görsel tanımlamalar **Site.css** dosyası içerisinde tanımlanmıştır.



`<script src="@Url.Content...` satırında, sayfada kullanılacak olan **JQuery** script dosyasının yolu belirtilmektedir. MVC içerisinde yalnızca JQuery değil, Ajax kütüphaneleri de yüklü olarak gelmektedir. Bu kütüphaneler **Scripts** klasörü altında yer alır.

Layout sayfamızda `<body> </body>` tag'ı arasında yer alan `RenderBody` metodu, Layout sayfası temelli olarak tanımlanmış olan **View Page**'i görüntüler ve body içeriğini belirtilen View Page ile doldurur.

Layout başına bir adet `RenderBody` metodu kullanılır. `RenderBody` metodu, ASP.NET içerisinde kullanılan `ContentPlaceHolder` server kontrolüne benzemektedir.

Layout dosya içerisindeki kod satırlarını incelediğinizde `@` karakteri gözünüze çarpmıştır. `@` operatörü ile sunucu taraflı kod yazımına başlanacağını MVC Framework'a bildiririz.

```
@{
        Kod Satırları
}
```

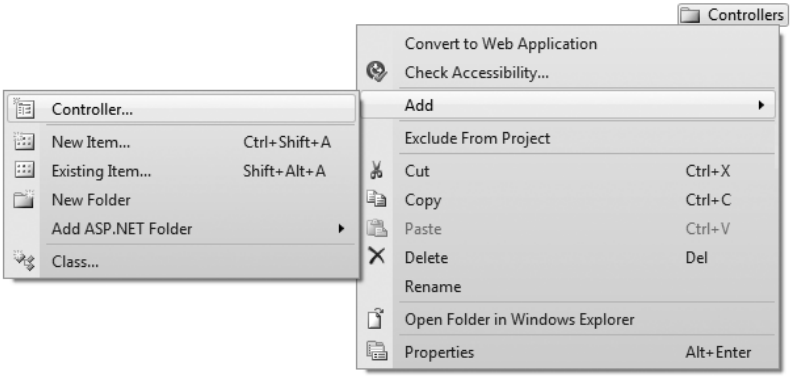
Bu kullanımda `{` ile `}` karakterleri arasındaki satırlar, sunucu taraflı kod satırları olarak kabul edilir.

`@` operatörü; bu hali ile ASP.NET yapısında ve MVC'nin önceki sürümlerinde kullanılan `<% %>` yapısının yerini almaktadır. Ayrıca her kod satırı için `<% %>` karakterlerinin kullanılmasına da gerek kalmamaktadır.

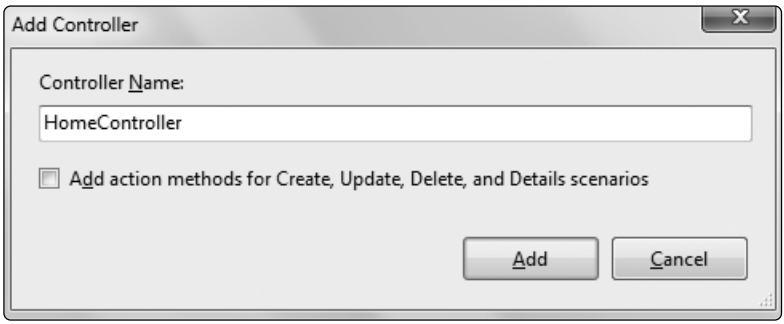
`@` operatörü; bir değişken isminden önce kullanıldığında, değişkenin değerini görüntüler. `@kullanıcı_sayısı` gibi. Metottan önce kullanıldığında ise metodun çalıştırılmasını sağlar. Örneğin; şu anki zamanı görüntülemek için kullanılan `DateTime.Now` metodunu, `@DateTime.Now` şeklinde kullanarak çalıştırabiliriz.

Artık uygulamamız için gereken View ve Controller nesnelerini oluşturabiliriz. Ön-

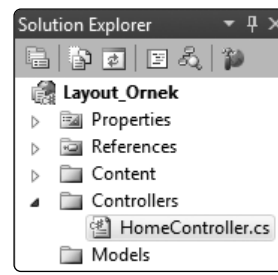
celikle yeni bir Controller oluşturacağız. Bunun için **Controllers** klasörüne farenin sağ tuşu ile tıklayarak **Add / Controller** menüsünü seçelim.



Karşımıza gelen pencerede Controller'ın ismini `HomeController` olarak belirliyoruz.



Penceredeki **CheckBox** işaretlendiğinde belirtilen metotların otomatik olarak oluşturulması sağlanır. Ancak örnek uygulamamızda veritabanı işlemlerini henüz gerçekleştirmeyeceğimiz için CheckBox'ı işaretlemiyoruz. `HomeController`, varsayılan Controller ismi olduğu için onu tercih ediyoruz. Bu Controller sayesinde ana sayfamızı oluşturup görüntüleyeceğiz.



Oluşturmuş olduğumuz `HomeController` isimli Controller, Controllers klasörü altına eklenecektir.

HomeController.cs*

Layout_Ornek.Controllers.HomeController

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
namespace Layout_Ornek.Controllers
{
    public class HomeController : Controller
    {
        //
        // GET: /Home/
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

Oluşturmuş olduğumuz `HomeController`'ın kodu yandaki gibidir.

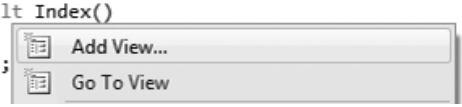
Kodumuzda `ActionResult` ile Index metodunda View'e geri dönüş yapılması sağlanmaktadır. Bir Action metodu kullanıcının isteklerine göre işlemleri gerçekleştirir ve geriye bir ActionResult döndürür. ActionResult, bir Action metodu adına gerçekleştirilecek komutları temsil eder.

Aşağıdaki tipler `ActionResult`'dan türer.

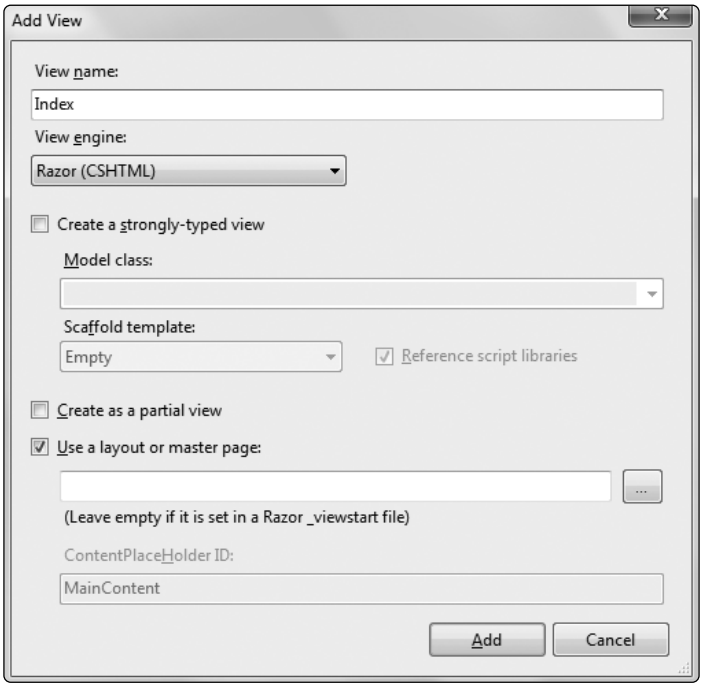
- **`ContentResult:`** Bir Action metot sonucunda geriye dönen kullanıcı tanımlı içerik türüdür.
- **`EmptyResult:`** Bir Action metot sonucunda geriye dönen boş değeri ifade eder.
- **`FileResult:`** Dosya içeriği göstermek için kullanılır.
- **`HttpUnauthorizedResult:`** Yetkisiz http istekleri sonucunda geriye dönen değeri ifade eder.
- **`JavaScriptResult:`** JavaScript içeriği yanıtı gönderir.
- **`JsonResult:`** JSON olarak biçimlendirilmiş içerik yanıtı gönderir.
- **`RedirectResult:`** Bir Action metodun gerçekleştirilmesi sonucunda belirtilen URI'ye yönlendirilir.
- **`RedirectToRouteResult:`** Bir Route tablosundaki değerler yardımıyla yönlendirilir.
- **`ViewResultBase:`** Bir View'in nasıl oluşturulduğuna dair sonuçları içerir.

`HomeController` isimli Controller'ı oluşturduktan sonra artık oluşturulan bu Controller'a, **View Template** ataması yapabiliriz. Bunun için `HomeController` isimli Controller içerisinde bulunan `Index` isimli metodumuza farenin sağ tuşunun tıklanması ile açılan menüden **Add View** seçeneğine tıklıyoruz.

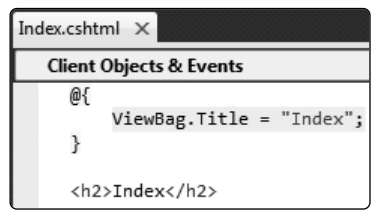
```
public class HomeController : Controller
{
    //
    // GET: /Home/
    public ActionResult Index()
    {
        return View();
    }
```



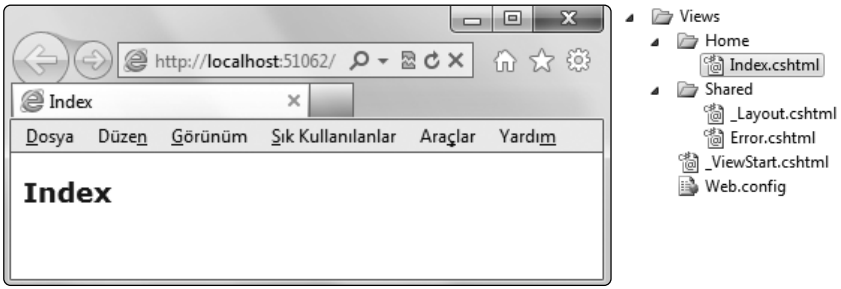
Karşımıza gelen pencerede Controller içerisindeki metot ismi **View Name** kısmında otomatik olarak yazılmaktadır.



Add butonuna tıklayarak **Views** altında, **HomeController** isimli Controller ile aynı isimde olarak **Home** klasörü oluşturulur ve bu klasör içerisinde **Index.cshtml** isimli index dosyamız, Razor tabanlı olarak oluşturulur.



Index.cshtml isimli sayfamızı açtığımızda içeriğinin şu şekilde olduğunu görürüz.



Bu hali ile web uygulamamızı çalıştırdığımızda index sayfası içerisinde yalnızca *"Index"* içeriğini görürüz.

Gerek Index sayfamızda gerekse diğer sayfalarda Layout sayfasının yolunu belirtmek gerekmektedir. Dolayısı ile biz de Index sayfasının kodunu...

```
@{
     ViewBag.Title = "Index";
     Layout = "~/Views/Shared/_Layout.cshtml";
}
```

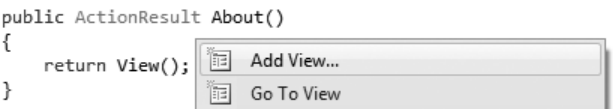
... şeklinde değiştiriyoruz.

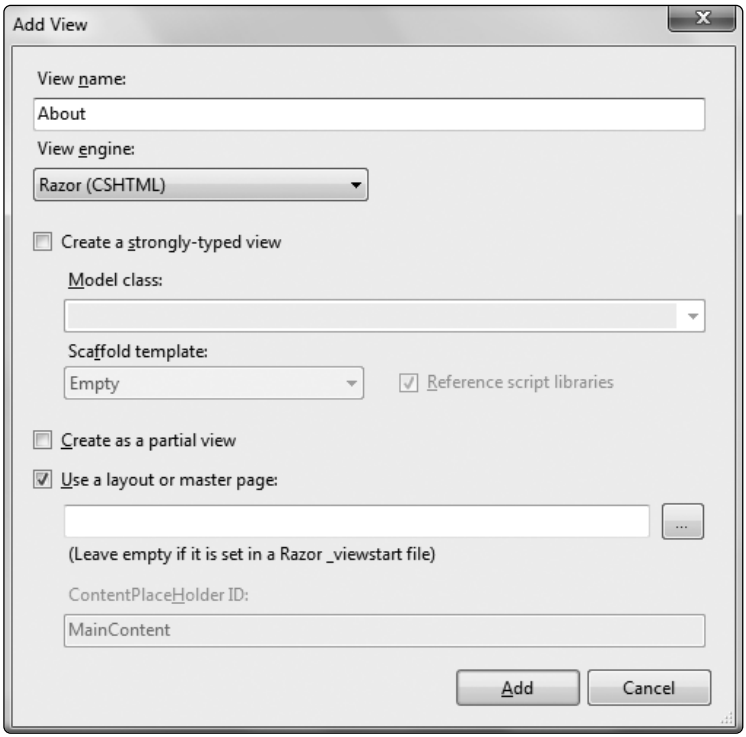


Eğer Layout sayfasının yolu sayfa içerisinde belirtilmezse, varsayılan olarak **_ViewStart.cshtml** dosyası içerisinde yapılan yol tanımlaması geçerli olur.

Index sayfasının içeriğine istediğimiz şekilde bir içerik ekliyoruz. Aynı şekilde blog sitemizdeki **About** sayfasını yapalım. Aynı şekilde örnek uygulamamız içerisinde About sayfasını oluşturalım.

HomeController isimli Controller içerisinde `About` isimli `ActionResult` metodumuzu oluşturalım. Daha sonra oluşturmuş olduğumuz metodumuza farenin sağ tuşu ile tıklayalım. Açılan menüden **Add View**'i seçelim.





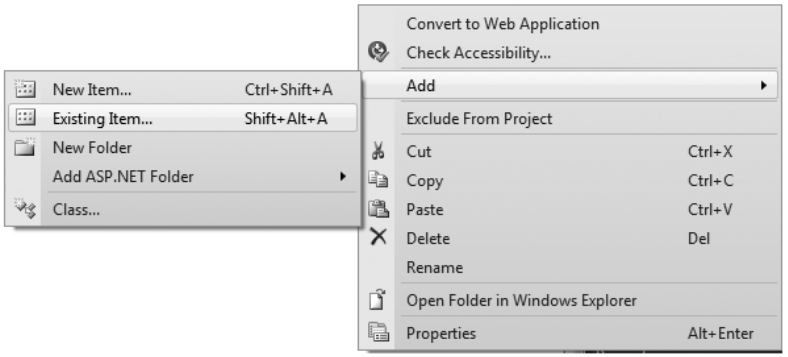
Gelen penceredeki **Add** butonuna tıklayalım. **Views** altında `About` isimli Controller ile aynı isimle **About.cshtml** isimli sayfamız Razor tabanlı olarak oluşturulur.

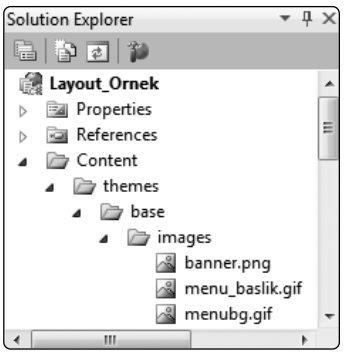


Örnek sitemiz içerisindeki sayfaların sol tarafında her zaman görüntülenecek bir menü tasarlayacağız. Ek olarak sayfaların üst tarafında görüntülenecek bir banner ekleyeceğiz.

Öncelikle banner'ı ekleyelim. Sayfa tasarımlarında sayfalarınızın menü, başlık vs. gibi elemanlarını yerleştirmek için `table` ya da `div` kullanabilirsiniz. Tasarımlarınızı div kullanarak çok daha kolay ve rahat bir şekilde yapabilirsiniz. Ayrıca standartlara uygun, esnek ve kullanışlıdır. Daha sonraki düzenlemelerinizi daha kolay yapabilecek bir şekilde kod yazmış olursunuz. CSS kullanarak div tasarımlarını çok daha etkili bir hale getirebilirsiniz. Biz de bu nedenle div kullanarak içeriklerimizi yerleştireceğiz.

Banner resmi olarak kullanacağımız dosyayı projemiz içerisine ekleyelim. **Solution Explorer** içerisinde **Proje** ismine farenin sağ tuşu ile tıklayarak **Add Existing Item** seçeneğine tıklıyoruz.





Ekleyeceğimiz resimleri **Content** klasörü içerisine ekliyoruz. Biz varsayılan olarak mevcut olan **Content/themes/base/images** klasörü içerisine ekledik.

Div içerisine banner resmi için HTML kodlarımızı aşağıda görüldüğü gibi ekliyoruz.

```
<div id =”header” style=”background-image:
       url(‘../../Content/themes/base/images/banner.png’);
       height: 138px;width: 1260px “>
</div>
```

Daha sonra ise sol tarafta görünecek menümüzü yine bir div içerisine ekleyelim. Div içerisine `ul` tag'ı ile bir liste oluşturacağız. Liste içerisindeki menü elemanlarına tıklandığında ilgili sayfaların çağrılması için ise `ActionLink HtmlHelper` metodunu kullanacağız. `ActionLink` metodu ile, belirtilen bir Controller'ın mevcut olup olmadığı kontrol edilmektedir. Mevcut ise Controller içerisinde belirtilen `ActionResult` metodu çalıştırılmaktadır.

```
ActionLink("Link Metni","ActionResult İsmi","Controller İsmi")
```

... şeklinde kullanılır.

Buna göre Layout.cshtml sayfasının son hali aşağıdaki gibi olacaktır:

```
<!DOCTYPE html>
<html>
<head>
    <title>@ViewBag.Title</title>
    <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet"
        type="text/css" />
    <script src="@Url.Content("~/Scripts/jquery-1.4.4.min.js")"
        type="text/javascript"></script>
</head>
<body>
    <div id ="header" style="background-image:
        url('../../Content/themes/base/images/banner.png');
        height: 138px;width: 1260px" > </div>
    <div style="float: left">
        <ul class="sidemenu" style="left: 0px; top: 0px; width: 198px" >
            <li style="background-image:
                url('../../Content/themes/base/images/menubg.gif');
                background-repeat: no-repeat;padding-left: 35px;
                padding-top: 13px; height: 41px;
                font-size: 15px;font-weight:bold">Sayfalar
            </li>
            <li>@Html.ActionLink("Ana Sayfa","Index","Home")</li>
            <li>@Html.ActionLink("Hakkımda","About","Home")</li>

            <li>@Html.ActionLink("Referanslar","Referans","Home")</li>
            <li>@Html.ActionLink("İletişim","iletisim","Home")</li>

            <li style="background-image:
                url('../../Content/themes/base/images/menubg.gif');
                background-repeat: no-repeat;padding-left: 35px;
                padding-top: 13px; height: 41px;font-size: 15px;
                font-weight:bold">Bağlantılarım</li>
            <li>
            <a href="http://www.vbnetnedir.com">VBNetNedir</a></li>
            <li><a href="http://www.kodlab.com">KodLab</a></li>
            <li><a href="http://www.volkanaktas.com">Volkan Aktaş</a></li>
            <li><a href="http://www.mustafakarakurt.com">Mustafa
                 Karakurt</a></li>
```

```
        </ul>
      </div>
      @RenderBody()
</body>
</html>
```

Yukarıda görüldüğü gibi örneğin...

```
Html.ActionLink("Ana Sayfa","Index","Home")
```

... satırında ActionLink üzerinde görüntülenecek metni `"Ana Sayfa"` olarak belirleyip, `Home` isimli Controller içerisindeki `Index` isimli ActionResult metodunun çalıştırılmasını sağladık.

`Url.Action` metodunu kullanarak da sayfalara yönlendirme yapabiliriz. Ancak `Url.Action` metodu ile sadece linke yönlendirme yapılmaktadır. `Url.Action` metodunu `<a href` ile birlikte kullanabilirsiniz.

Kullanımı şu şekildedir:

```
<a href=@Url.Action("Action Result İsmi","Controller İsmi")>Link Metni</a>
```

Örneğin; buna göre **Hakkımda** sayfasına link verecek olursak,

```
<a href=@Url.Action("About","Home")>Hakkımda</a>
```

... şeklinde sayfaya link verebiliriz.

Örneğimizdeki menümüz için CSS tanımlamalarımızın yer aldığı **Site.css** adlı dosyamıza aşağıdaki tanımlamaları ekledik:

```
ul.sidemenu {
      padding: 0;
      margin: 0;
      list-style: none;
      font-family: Verdana, Arial, Helvetica, sans-serif;
      font-size: 1.2em;
      border-bottom: 1px solid #6F3E04;
      width: 100px;
      background-color: #EEEEDC;
}
ul.sidemenu li {
      border-top: 1px solid #6F3E04;
      border-left: 1px solid #6F3E04;
```

```
      border-right: 1px solid #6F3E04;
}
ul.sidemenu li a {
      text-decoration: none;
      display: block;
      width: 176px;
      color: #333333;
      font-weight: bold;
      padding: 2px 10px;
}
ul.sidemenu li a:hover {
      background-color: #B9BB79;
      color: #EEEEDC;
}
```

Örnek site çalışmamızın ekran görüntüsü son durumda aşağıdaki gibi olacaktır:



RenderSection Metodu

`RenderSection` metodu, bir Layout sayfası içerisinde tanımlanmış olan section'lara ulaşmak amacı ile kullanılır. Bir Layout içerisinde bir adet `RenderBody` bulunabilirken, birden fazla `RenderSection` kullanılabilir.

Örnek web uygulamamızı `RenderSection` metodu ile yeniden düzenleyelim. Örnek olarak index sayfası üzerinde gerekli kod değişikliklerini yapacağız.

Index.cshtml sayfası içerisinde sol menümüz ve banner resmimiz için iki adet Section tanımlıyoruz.

Ekleyeceğimiz kod aşağıdaki gibi olacak:

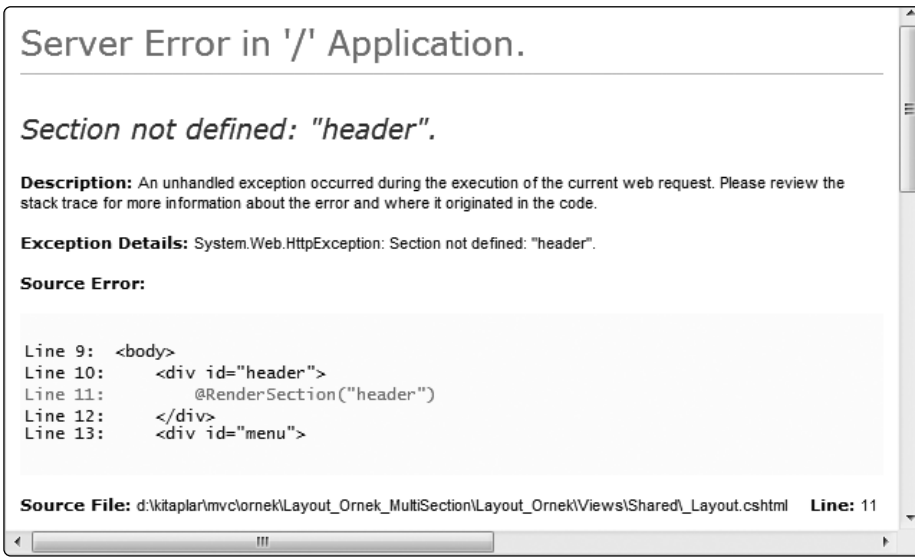
```
@section header {
<div id =”header” style=”background-image:
            url(‘../../Content/themes/base/images/banner.png’);
       height: 138px;width: 1260px” > </div>
}
@section menu {
<div style=”float: left”>
      <ul class=”sidemenu” style=”left: 0px; top: 0px; width: 198px” >
            <li style=”background-image:
                  url(‘../../Content/themes/base/images/menubg.gif’);
                  background-repeat: no-repeat;padding-left: 35px;
                  padding-top: 13px; height: 41px;
                  font-size: 15px;font-weight:bold”>Sayfalar
            </li>
            <li>@Html.ActionLink(“Ana Sayfa”,”Index”,”Home”)</li>
            <li><a href=@Url.Action(“About”,”Home”)>Hakkımda</a></li>
            <li>@Html.ActionLink(“Referanslar”,”Referans”,”Home”)</li>
            <li>@Html.ActionLink(“İletişim”,”iletisim”,”Home”)</li>
            <li style=”background-image:
                  url(‘../../Content/themes/base/images/menubg.gif’);
                  background-repeat: no-repeat;padding-left: 35px;
                  padding-top: 13px; height: 41px;
                  font-size: 15px;font-weight:bold”>Bağlantılarım</li>
            <li><a href=”http://www.vbnetnedir.com”>VBNetNedir</a></li>
            <li><a href=”http://www.kodlab.com”>KodLab</a></li>
            <li><a href=”http://www.volkanaktas.com”>Volkan Aktaş</a></li>
            <li><a href=”http://www.mustafakarakurt.com”>Mustafa
                  Karakurt</a></li>
      </ul>
</div>
}
```

_Layout.cshtml isimli Layout sayfamızın body tag’ları arası kodu ise aşağıdaki gibi değişecek:

```
<body>
    <div id=”header”>
```

```
        @RenderSection("header")
    </div>
    <div id="menu">
        @RenderSection("menu")
    </div>
    @RenderBody()
</body>
```

Uygulamamızı çalıştırdığımızda Layout içerisinde `header` ve `menu` isimli Section'lar çağırılarak çalıştırılacaktır. Ancak çağrılan Section bulunamadığında uygulamamız hata verecektir.



Bu hatanın oluşmasını engellemek için `RenderSection` metodunu...

```
RenderSection("Section İsmi", Bool)
```

... şeklinde kullanabiliriz. `Bool` parametresi; `true` ve `false` olmak üzere iki değer alır. `true` değerini aldığında, Section mevcut olmadığında programın çalışması kesilir. `false` değerini aldığında ise Section mevcut olmasa bile programın çalışması kesilmez.

Layout sayfamızın kodlarını buna göre aşağıdaki gibi düzenliyoruz:

```
<body>
    <div id="header">
```

```
            @RenderSection("header",false)
        </div>
        <div id="menu">
            @RenderSection("menu",false)
        </div>
        @RenderBody()
</body>
```

Uygulamamızda Section mevcut olmasa bile artık hata vermeden çalışacaktır.

Belirtilen Section'ın mevcut olup olmadığını `IsSectionDefined` metodu ile kontrol etmemiz mümkündür.

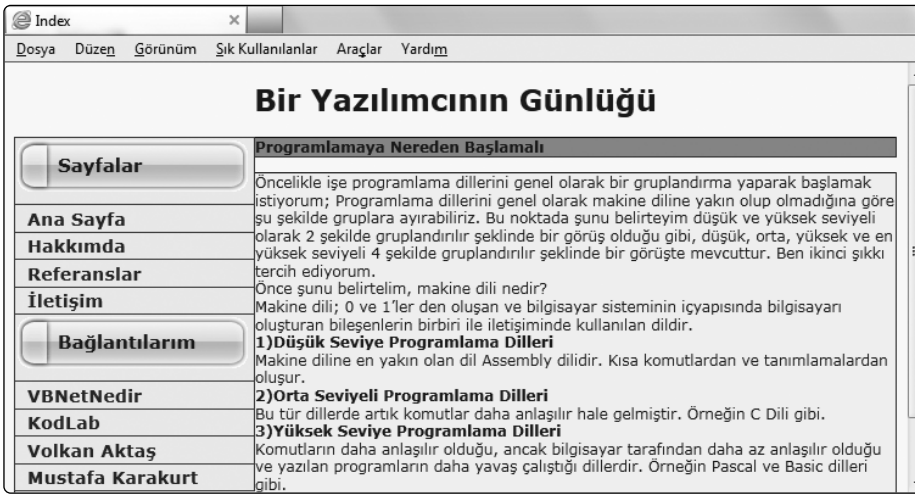
Örnek olarak **Index.cshtml** sayfamızda bulunan `menu` ve `header` isimli iki adet Section'dan `header` isimli section'ı silelim. Layout sayfamız içerisinde bulunan `body` tag'ı içerisindeki kodlarımızı ise aşağıdaki gibi düzenleyelim:

```
<body>
    <div id="header">
        @if (IsSectionDefined("header"))
        {
            @RenderSection("header")
        }
        else
        {
            <H1 align="center">Bir Yazılımcının Günlüğü</H1>
        }
    </div>
    <div id="menu">
        @RenderSection("menu",false)
    </div>
    @RenderBody()
</body>
```

Kodumuz içerisinde yer alan...

```
@if (IsSectionDefined("header"))
```

... satırında `header` isimli section'ın mevcut olup olmadığı kontrol edilmektedir. Eğer mevcut ise `header` isimli section çağrılmakta. Mevcut değilse *"Bir Yazılımcının Günlüğü"* şeklinde bir yazı, banner yerine görüntülenmektedir.



RenderPage Metodu

`RenderPage` metodu, herhangi bir sayfa içerisindeki bölümlerin belirtilen sayfalar ile doldurulması amacıyla kullanılır.

Örnek olarak **Index.cshtml** içerisinde görüntülenecek olan içeriği silerek **icerik.cshtml** adı ile yeni bir dosya oluşturarak içerisine yazalım. **Layout.cshtml** içerisindeki kodu ise aşağıdaki gibi düzenleyelim:

```
<body>
     <div id="header">
          @RenderSection("header",false)
     </div>
     <div id="menu">
          @RenderSection("menu",false)
     </div>
     @RenderPage("~/Views/Home/icerik.cshtml")
     @RenderBody()
</body>
```

Uygulamamızı çalıştırdığımızda **icerik.cshtml** isimli sayfa içerisindeki yazılarımız çağırılarak, **index.cshtml** dosyası içerisindeki body tag'ı içerisinde görüntülenecektir.



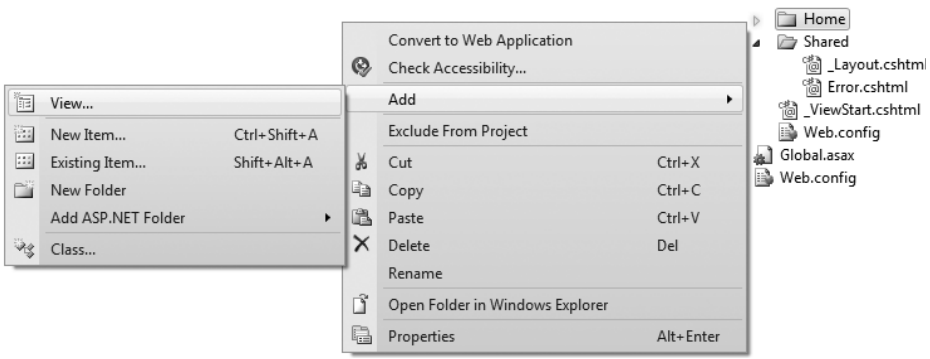
Partial View

MVC içerisinde sayfalara ait içeriklerin bir kısmının başka sayfalardan alınmasını sağlayabiliriz. Bu işlem için MVC içerisinde **Partial View** kullanılmaktadır. Kullanımı şu şekildedir:

```
@Html.Partial("Partial View İsmi")
```

Örnek olarak sayfalarımızın sağ tarafındaki menü içerisinde bulunan bağlantılar kısmını **Partial View** içerisinde oluşturarak görüntüleyelim.

Öncelikle menümüzü yeni bir **View** oluşturarak içerisinde tanımlayalım. **Solution Explorer** içerisinde **Home** klasörüne sağ tıklayıp **Add View**'ı seçiyoruz.



Gelen pencerede **Use a layout or master page** seçeneğini işaretleyerek **menu.cshtml** isimli yeni bir **View** oluşturuyoruz.



menu.cshtml dosyası içerisinde menü kodlarımızı aşağıdaki gibi yazıyoruz:

```
<div style="float: left">
 <ul class="sidemenu" style="left: 0px; top: 0px; width: 198px" >
  <li style="background-image:
       url('../../Content/themes/base/images/menubg.gif');
       background-repeat: no-repeat;padding-left: 35px;padding-top: 13px;
       height: 41px;font-size: 15px;font-weight:bold">Sayfalar
  </li>
  <li>@Html.ActionLink("Ana Sayfa","Index","Home")</li>
  <li><a href=@Url.Action("About","Home")>Hakkımda</a></li>
  <li>@Html.ActionLink("Referanslar","Referans","Home")</li>
  <li>@Html.ActionLink("İletişim","iletisim","Home")</li>
  <li style="background-image:
       url('../../Content/themes/base/images/menubg.gif');
```

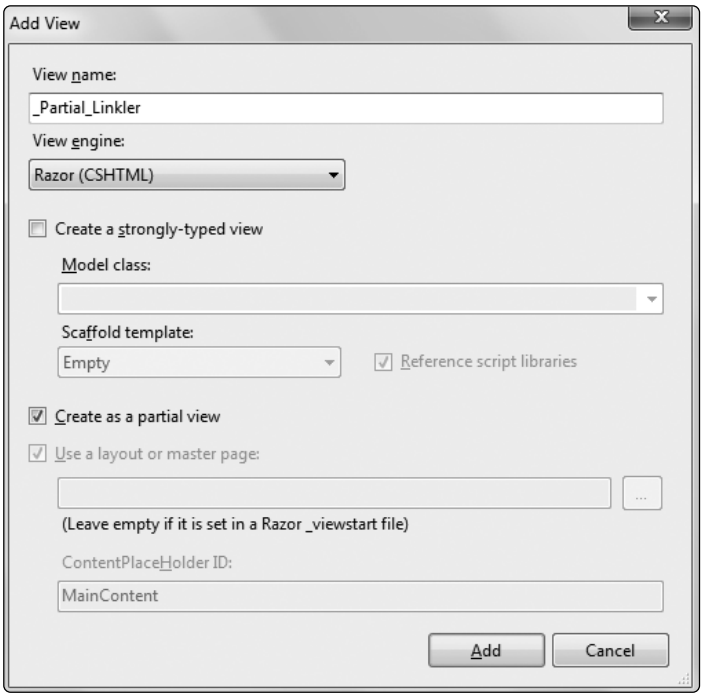
```
        background-repeat: no-repeat;padding-left: 35px;padding-top: 13px;
        height: 41px;font-size: 15px;font-weight:bold">Bağlantılarım</li>
    @Html.Partial("_Partial_Linkler")
  </ul>
</div>
```

Görüldüğü gibi **menu.cshtml** dosyası içerisinde **Bağlantılarım** başlığı altında listelenecek olan bağlantıları yazmayıp...

```
@Html.Partial("_Partial_Linkler")
```

... satırı ile **_Partial_Linkler** adlı **Partial View** içerisinden çağıracağımızı belirtiyoruz.

Partial View'ı oluşturmak için ise **Solution Explorer** içerisinde **Home** klasörüne sağ tıklayarak **Add View** seçeneğini tıklıyoruz. Gelen pencerede **Create as a partial view** seçeneğini işaretleyerek **_Partial_Linkler.cshtml** isimli yeni bir Partial View oluşturuyoruz.



Partial View içerisine aşağıdaki gibi **Bağlantılarım** başlığı altında listelenecek olan linkleri yazıyoruz.

```
<li><a href="http://www.vbnetnedir.com">VBNetNedir</a></li>
<li><a href="http://www.kodlab.com">KodLab</a></li>
<li><a href="http://www.volkanaktas.com">Volkan Aktaş</a></li>
<li><a href="http://www.mustafakarakurt.com">Mustafa Karakurt</a></li>
```

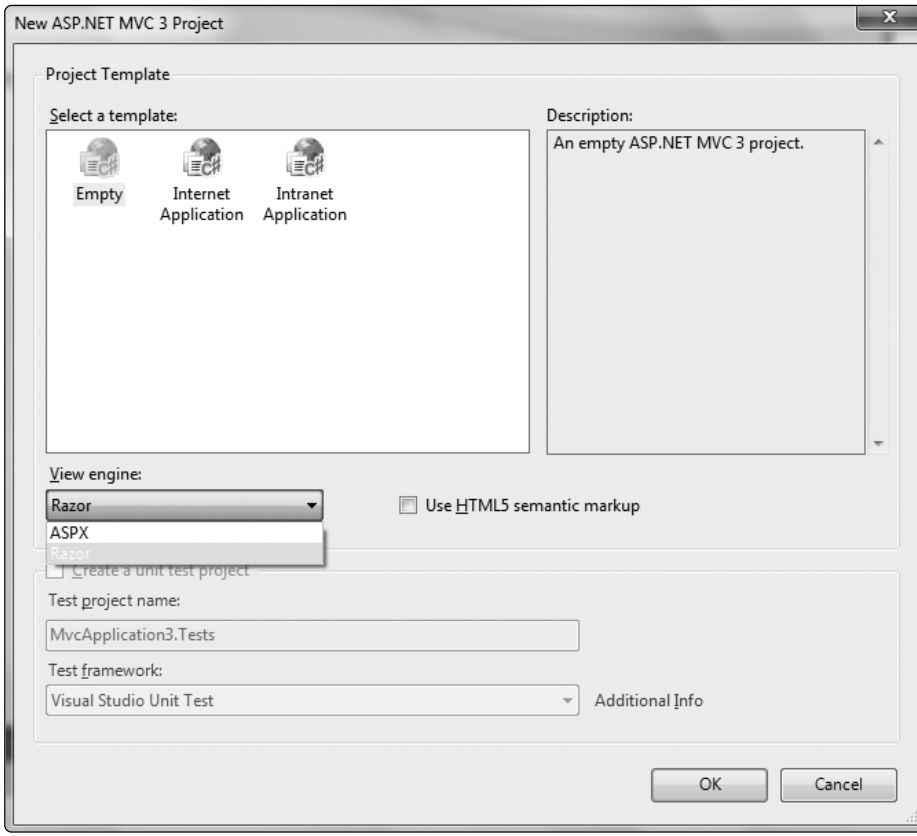
Uygulamamızı çalıştırdığımızda, sayfalarda bulunan menü içerikleri sorunsuzca görüntülenecektir.

RAZOR VIEW ENGINE

3

View Engine

View Engine, HTML ile XML görüntülerini veya herhangi bir içerik türünü görüntüleyip yayımlamak için geliştirilmiş bir sistemdir. Bu sistem, web uygulamalarımızın görsel kısmını daha kolay geliştirmek için tasarlanmıştır. **Server Side** dediğimiz sunucu taraflı kodlamalar ile **Client Side** dediğimiz kullanıcı tarafında bulunacak kodların beraber kullanılarak geliştirilmesini sağlamaktadır. **View Engine** sayesinde geliştirdiğimiz web uygulamalarında, sunum katmanı olan **View**, yani kullanıcılara görsel olarak hitap eden kısmın kod yazımı kolaylaşır. ASP.NET MVC1 ve ASP.Net MVC2 ile beraber kullandığımız **Web Form View Engine** (ASPX) yapısından sonra, Asp.NET MVC3 ile beraber **Razor View Engine** geliştirilmiştir. Visual Studio 2010 **ASP.NET MVC3** uygulaması açtığımızda varsayılan View Engine, **ASPX** ve **Razor** olarak karşımıza çıkmaktadır.



Bunların dışında da **View Engine** bulunmaktadır. Visual Studio eklentileri ile **Spark View Engine** ve **Nhaml View Engine** yüklenebilir. Proje oluştururken seçeceğiniz View Engine'ı, kodlama rahatlığına ve tercihinize göre belirleyebilirsiniz. Seçeceğiniz View Engine; kodlama rahatlığı, az ifade ile çok iş yapma, kodlamanın akışkan olması, daha rahat anlaşılabilir olması ve daha az klavye tuş kombinasyonu kullanımına (örneğin; özel karakterleri yazdırmak için kullandığımız **`Shift+tuş`** kombinasyonu: `<%`, `%>`, `!`) göre farklılık arz edebilir. Bu kitapta kullandığımız örneklerde ASP.NET MVC3 ile beraber gelen **Razor View Engine** kullandık. Dolayısıyla Razor View Engine söz dizimi (*syntax*) üzerinde duracağız.

Razor View Engine

Razor, Microsoft'un geliştirdiği yeni bir View Engine olup, ASP.NET MVC3 ile beraber gelmektedir. Razor, biçimlendirme kodları ile Server Side kodlarını karıştırmak için geliştirilmiştir. Diğer bir ifadeyle web sayfalarına sunucu tabanlı kod gömmek için geliştirilmiş bir sözdizimidir. Yapı olarak Server Side dediğimiz sunucu taraflı kodlama dilleri olan ASP ve PHP dillerine benzemektedir. Razor ile oluşturulmuş bir web sayfasında **Server Side** ve **Client Side** kodları bulunmaktadır. Kullanıcı sunucudan web sayfası isteğinde bulunduğunda sunucu, istemcinin tarayıcısına sayfayı göndermeden önce Server Side kodlarını çalıştırır ve daha sonra istemcinin tarayıcısına gönderir. **Server Side** kodları ile veritabanı işlemleri yapılabilir ve sayfamızı dinamik hale getirebiliriz. **Client Side** kodları, HTML etiketleri, JavaScript kodları ve stil kodlarından oluşur.

Razor; düz metinleri, biçimlendirme kodları olan HTML kodlarını ve Server Side kodlarını aynı kod bloğu içerisinde birleştirme özelliğine sahiptir. Razor, en az kod yazımı ve ifade ile daha çok işlem yapmayı amaçlayarak, proje geliştirmeyi daha pratik hale getirmeyi hedeflemektedir.

Razor, diğer View Engine'ların aksine kod bloklarını açıkça yazmamıza gerek kalmadan kendiliğinden algılar. Kodlamalar daha estetik ve ayırt edilebilirliği daha kolaydır. Razor için yeni bir dil kullanımına gerek kalmaksızın C# veya VB.Net ile kodlama yapılabilir.

Razor View Engine projelerindeki **Views**, **Partial Views** ve **Layouts** (*master pages*) dosyalarının uzantıları C# için **.cshtml**, VB.Net için **.vbhtml**'dir.

@ Operatörünün Kullanilmasi

Razor View Engine söz diziminde kodlamalar `@` karakteri ile başlar. Web Form View Engine'ın aksine kodlamalar `<%` karakterleri ile başlayıp `%>` karakteri ile bitmez. Razor, bir ifadenin başında bulunan `@` karakterini, kod ifadesinin başlangıcı olarak değerlendirir.

Bir örnek verecek olursak...

```
@DateTime.Now
```

... Görüldüğü gibi sayfamıza eklediğimiz tarih saat için `@` karakterini ve ilgili dildeki saat tarih kodunu yazmamız yeterli olacaktır. Ayrıca örneğimizde görüldüğü üzere diğer View Engine'lardan farklı olarak eklediğimiz kod bloğunun bitişini be-

lirtmemize gerek yoktur. Diğer View Engine'larda kod bloğunun başlangıcını ve bitişini aşağıdaki gibi belirtmemiz gerekmekteydi.

```
<%:DateTime.Now%>
```

Örneğimizde görüldüğü gibi Web Form View Engine (ASPX) söz diziminde kod bloğunun başlangıç ve bitişini belirtmemiz gerektiği gibi, özel karakterleri (<%, %>) yazmak için kullanacağımız tuş kombinasyonlarının (**Shift+tuş**) fazla olması ve daha çok etiket kullanmamız nedeniyle zaman kaybına sebebiyet vermektedir.

@ karakterinin tek satırlı kullanımı aşağıdaki gibidir:

```
@{var ad_soyad="Tahsin SEVİNÇ";}
@{var ad_soyad1="Volkan AKTAŞ";}
```

@ karakterinin çok satırlı kullanımı aşağıdaki gibidir:

```
@{
 var ad_soyad="Tahsin SEVİNÇ";
 var ad_soyad1="Volkan AKTAŞ";
 }
```

@ karakterinin satır içerisindeki kullanımı aşağıdaki gibidir:

```
<p>Bilişim Teknolojiler Öğretmeni:<b>@ad_soyad</b> ve <b>@ad_soyad1</b></p>
```

Çok Satırlı Kod Blokları Oluşturma

Razor View Engine söz diziminde tek satırlı veya birden çok satırlı kod blokları oluşturmak istediğimizde küme ({,}) parantezlerini kullanmamız gerekmektedir.

```
@{int sayi1 = 1;}
@{
 int sayi2 = 5;
 int toplam = sayi1 + sayi2;
 }
```

Burada belirtmemiz gereken önemli bir mevzu da noktalı virgül kullanımıdır. Satır sonlarını belirtmek için noktalı virgül ekliyoruz. Bir örnek vermemiz gerekirse:

```
@{
int sayi1 = 1;
int Sayi2 = 5;
int toplam = sayi1+ sayi2;
}
```

Görüldüğü gibi kod bloğu içerisindeki her satırın sonuna noktalı virgül koyduk. Satır içerisinde kod kullanımı gerektiği durumlarda küme parantezi ve sonuna noktalı virgül eklememize gerek yoktur. Satır içi ifade eklemek için kodun başına @ karakterini eklememiz yeterli olacaktır.

```
<p>İki sayının toplamı=@toplam</b>
```

Diğer View Engine'lerden farklı olarak Razor View Engine'da kod bloğu içerisinde html kodu veya metin yazmak istediğimizde ayrıca kodun başlangıç ve bitişini belirtmemize gerek yoktur. Razor, kod bloğu içerisindeki HTML kodlarını otomatik olarak algılayacaktır.

```
@{
int sayi1 = 1;
int Sayi2 = 5;
int toplam = sayi1+ sayi2;
<strong>1 ve 5 sayılarının toplamı:</strong>@toplam
}
```

Razor View Engine'ın avantajı yukarıdaki kod bloğunda daha iyi görülmektedir. Sunucu taraflı kodlarla, HTML biçimlendirme etiketleri bir arada kullanılmaktadır. Ayrıca başlangıç ve bitiş yerlerini belirtmemize gerek yoktur. **Web Form View Engine**'da aynı işlemi yapmak istediğimizde aşağıdaki kod bloğunu yazmamız gerekmektedir.

```
<%
int sayi1 = 1;
int Sayi2 = 5;
int toplam = sayi1+ sayi2;
%>
<strong>1 ve 5 sayılarının toplamı:</strong><%:toplam%>
```

Razor'da oluşturduğumuz kod bloğu içerisine eklediğimiz html etiketi içerisine kod yazdırmak veya değer göndermek istediğimizde, @ karakterini ekleyeceğimiz ifadenin başına koymamız yeterli olacaktır.

```
@{
int sayi1 = 1;
int Sayi2 = 5;
int toplam = sayi1+ sayi2;
<strong>1 ve 5 sayılarının toplamı: @toplam </strong>
}
```

Razor'da oluşturduğumuz kod bloğu içerisine html etiketleri kullanmadan metin eklemek istediğimizde; yazacağımız metni, `<text></text>` etiketleri arasına yazarak kod bloğu içerisine ekliyoruz.

```
@{
int sayi1 = 1;
int Sayi2 = 5;
int toplam = sayi1+ sayi2;
<text>1 ve 5 sayılarının toplamı: @toplam </text>
}
```

`<text>` etiketini birden fazla satırlı metin yazdırmak istediğimizde kullanabiliriz. Tek satırlık metin yazdırmak istediğimizde `@:` karakterlerini beraber kullanabiliriz.

```
@{
int sayi1 = 1;
int Sayi2 = 5;
int toplam = sayi1+ sayi2;
@:1 ve 5 sayılarının toplamı: @toplam
}
```

Kod bloğu içerisinde ürettiğimiz değeri yine kod bloğu içerisinde kullanma zorunluluğumuz yoktur. Küme parantezi ile oluşturduğumuz kod bloğunun içerisinde ürettiğimiz değeri, kod bloğunun dışında kullanmak için değişken isminin başına `@` karakteri koyarak satır içi ifade oluşturabiliriz.

```
@{
int sayi1 = 1;
int Sayi2 = 5;
int toplam = sayi1+ sayi2;
}
1 ve 5 sayılarının toplamı: @toplam
```

Peki, satır içi kod kullanımında birden fazla metin ve değişkenin yazılması istendiğinde ne yapmalıyız? Bu gibi durumlarda ifademizin başında `@{` karakterlerini koymalıyız ve ifademizin bitimini de yine `}` karakteri ile belirtmemiz gerekmektedir.

```
@{
int sayi1 = 1;
int Sayi2 = 5;
int toplam = sayi1+ sayi2;
}
@("1 ve 5 sayılarının toplamı:" + toplam + "'dır.")
```

@ karakteri ile beraber kullandığımız parantezlerle oluşturduğumuz ifadenin içerisine değer gönderebildiğimiz gibi sayısal işlemleri de yaptırabiliriz. Örnek verecek olursak:

```
@(1+5)
```

Yukarıdaki ifadede ekran çıktısı olarak 1 ve 5 sayılarının toplamı olan 6 sayısını yazdıracaktır.

Değişken Tanımlama ve Kullanımı

Razor söz diziminde değişken tanımlanıp kullanılabilir. Değişken tanımlamak için var anahtar kelimesi kullanılabilir. Oluşturacağınız değişkenin türünü bilmiyorsanız veya belirtmek istemiyorsanız var anahtar kelimesi kullanılır. Kullanacağınız dilin değişken tanımlama ve isimlendirme kuralları aynen Razor için de geçerlidir. Yani C# dilini kullanıyorsanız C# dilinde geçerli olan büyük küçük harf farklılıklarını algılama gibi özellikleri aynen geçerlidir. Değişken tanımlamaya ve tanımladığımız değişkende değer saklamaya örnek verecek olursak:

```
@{var tarih= DateTime.Now.DayOfWeek;}
```

Değişken tanımlayıp, değişkenin alacağı değer olarak haftanın gününü atadık. Şimdi değişkenimizin sayfanın içinde kullanımı için de değişken ismiyle çağırmamız yeterli olacaktır.

```
@tarih
```

Razor'da String türündeki değişkenleri tanımlarken ve kullanırken bazı özel karakterlerin yazımıyla ilgili dikkat etmemiz gereken kurallar vardır. var anahtar kelimesini kullanıp değişken tanımladığımızda ve tanımladığımız değişkenlerde metin saklamak istediğimizde "" kullanmalıyız.

```
@{var mesaj = "web sayfamıza hoş geldiniz!";}
```

String tanımladıktan sonra değişkenimizin içerisinde çift tırnak kullanmak istediğimizde çift tırnağı iki kere yazıp değişkenin önüne @ karakteri koymamız gerekir.

```
@{var mesaj = @"string türünde değer depolamak için çift tırnak
("") kullanmalıyız";}
```

Çift tırnak gibi Razor söz dizimi için özel olan karakterlerden bir tanesi de ters slash'tır. Ters slash (\) karakterini de string türünden depolamalarda kullanmak için metnimizi içine aldığımız çift tırnağın önüne eklemeliyiz.

```
@{var ImagePath = @"C:\HaberPortal\Content\image";}
```

Yine Razor söz dizimindeki özel karakterlerden biri olan @ karakterini, kod bloğunun başlangıcı olarak algıladığı için bu karakteri yalnız ekrana yazdırmak istediğimizde @@ şeklinde uygulamalıyız. Sayfamıza ekleyeceğimiz bir e-mail adresinin yazılışı aşağıdaki gibi olmalıdır.

```
mailadresi1@@msn.com, mailadresi2@@yahoo.com
```

Boşluk Kullanımı

Kod bloğu veya tek satırlık ifadelerde kullandığımız boşluklar, ifadenin çalışmasını etkilemez.

```
@{string kitap =                    "MVC";}
```

Ayrıca kod blokları içerisine eklediğimiz kodları farklı satırlara yazdığımızda da ifadeler çalışmaya devam edecektir.

```
@{
int
sayi1
=
1
;
}
```

Ancak String değerlerini tek satırda yazmadan bölerek farklı satırlarda yazdığımız takdirde hata verecektir. String ifadelerini farklı satırlara yazmak için + operatörünü veya @ karakterini kullanabiliriz.

```
@{ string metin= "sayı 1 ve
sayı 2 toplamı ";}
```

Yukarıdaki ifade çalışmayacaktır. Metni birden fazla satırda yazmak için @ karakterini kullanabiliriz.

```
@{
string metin =  @" sayı1
ve sayı2
toplamı";
}
```

Sunucu Taraflı Yorum ve Açıklama Eklemek

Oluşturduğumuz proje kodlarına açıklama ve yorum ekleyebiliriz. Yorum ve açıklama eklemek için yorum ve açıklamaları `@*` operatörü ve `*@` operatörü arasına yazmalıyız. Bu operatörler arasına bir satırlı veya çok satırlı yorum ekleyebiliriz.

```
@* eklenecek yorum *@
```

Ayrıca kullandığımız dilin yorum ekleme operatörlerini de kullanabiliriz. C# için tek satırlı yorum veya açıklama eklemek için `//` operatörünü, çok satırlı yorum veya açıklama eklemek için `/*` ve `*/` operatörlerini eklemeliyiz.

```
@{
int sayi1= 1; //birinci sayıya değer atıyoruz.
int sayi2= 5;
int toplam  =  sayi1 + sayi2;
/*iki
sayının toplamını veren kod */}
```

Kontrol Yapıları ve Döngüler

Razor View Engine, kullanacağımız kontrol yapıları (`if`, `else...`) ve döngüleri (`for`, `foreach...`) kod blokları içerisine yazmaksızın birbirinden ayırt edebilir. Bir örnek üzerinde inceleyecek olursak:

```
<form method="post">
<input type="text" name="sayi1" />
<input type="text" name="sayi2" />
<input type="submit" value="Çarp" />
</form>
@if (IsPost)
{
     string sayi1 = Request["sayi1"];
     string sayi2 = Request["sayi2"];
     Int32 toplam = Convert.ToInt32(sayi1) * Convert.ToInt32( sayi2);
     <text>
     Girdiğiniz birinci sayı:@sayi1<br />
     Girdiğiniz ikinci sayı:@sayi2<br />
     Girilen sayıların çarpımı:@toplam
     </text>
}
```

Örneğimizde HTML etiketleri ile bir form oluşturduk. Oluşturduğumuz form içerisine iki tane **TextBox** ve bir tane **Submit** ekledik. Sayfa ilk açıldığında gelen forma sayısal değerler girip Submit ile post işlemi gerçekleştirdiğimizde `if` kontrol yapısı sayfanın post olduğunu kontrol ediyor ve eğer post edilmişse çarpma işlemini gerçekleştirip ekrana sonucu yazdırıyor.

Yukarıdaki örnekte görüldüğü gibi `if` kontrol yapısı Razor View Engine içerisinde kullanırken ayrıca özel karakterler kullanmamıza gerek yoktur. `@` karakteri ile başlayan kontrol yapılarını Razor otomatik olarak algılayacaktır.

Tek satırlı kontrol yapılarının html içerisindeki kullanımı aşağıdaki gibi olmalıdır.

```
@if (IsPost)
{string sayi1=request[“sayi1”];}
```

Çok satırlı kontrol yapılarının HTML içerisindeki kullanımı, küme parantezi içerisinde kod blokları şeklinde olmalıdır.

```
@{ var onay=false;
   if (IsPost)
   {
   string sayi1=request[“sayi1”];
   }else{
   <b>Giriş yapınız!<b/>
   }
}
```

Foreach döngüsünün Razor ile kullanılması: Döngüler, kontrol yapılarında olduğu gibi tek satırlı ise; küme parantezine gerek kalmaksızın kullanılabilir. Çok satırlı kod bloklarında da küme parantezi ile kod bloğunu belirtmek gerekiyor. Aşağıdaki örneklerin birbirine alternatif döngü yapısı bulunmaktadır.

```
@{
     string[] gunler = new[] { “Pazartezi”, “salı”, “çarşamba” };
}
@foreach (var gun in gunler)
{
     <text>@gun<br /></text>
}
```

Yukarıdaki örnekte dizi tanımlarken ayrıca küme parantezi belirtmemiz gerekmektedir. Şimdi bu örneğe benzer şekilde diğer yazılışını inceleyelim.

```
@{
 string[] aylar = new[] {"eylül","ekim","kasım" };
 foreach(var ay in aylar)
       {
       <text>@ay<br /></text>
       }
 }
```

Bu örnekte de `foreach` döngüsünü kod bloğu içerisinde yazmak için küme parantezi kullanılmıştır. Burada dikkat etmemiz gereken bir diğer özellik de daha önce de belirttiğimiz gibi HTML etiketlerini kod bloklarının içerisinde kullanabiliyor olmamızdır.

Dosya ve Klasörlerin Yollarının Kodlarla Kullanımı

Diğer programlama dilleri ve kodlamalarda olduğu gibi Razor'da da dosyalarla çalışmamız gerektiği durumlarda dosyaların fiziksel yolunu belirtmemiz gerekir. Fiziksel yol, dosya veya klasörlerin sunucu bilgisayarlardaki veya yerel bilgisayarlardaki sabit disklerde bulundukları konumun tam adresidir (Örneğin; *C:/web/siteler/websayfam/index.html*).

Dosya ve klasörlerin yollarının kullanılması gerektiği durumlarda üç türlü yöntem kullanılır. Bunlar; mutlak yol, göreceli yol ve URL yoludur.

Mutlak yol, bir dosya veya klasörün sabit disk içerisindeki fiziksel yolunun tam olarak belirtilmesidir. Göreceli yol ise, hedef gösterilen dosya veya klasörün fiziksel yolunun tamamı değil de son bölümlerinin bir kısmı kullanılarak belirtilmesidir. Örneğin; bir web sitesinde kullandığımız bir resim için konum belirtirken mutlak yol olarak aşağıdaki gibi belirtebiliriz.

```
<img src="C:\calismalar\website\orneksitem\image\resim1.jpg"/>
```

Aynı web sitesi kod dosyasının içerisine resim kodunun konumunu belirtirken göreceli yol kullandığımızda kodumuz aşağıdaki gibi olacaktır.

```
<img src="image\resim1.jpg"/>
```

İkinci kod web sitesi dosyasının bulunduğu fiziksel konumun içerisindeki image klasörünü işaret etmektedir. Bir yönüyle kodun yazıldığı dosyanın konumuna göre resim dosyasının konumu şekillenmektedir. Şimdi bu iki kod arasındaki avantajlar ve dezavantajlar açısından bakacak olduğumuzda, birinci kodun barındırıldığı ser-

ver veya yerel bilgisayar değiştirildiğinde dosya konumu değiştiği için çalışmayacaktır. İkinci kodda ise dosya göreceli yol kullandığı için server değişikliğinde veya yerel bilgisayar değişikliğinde çalışmaya devam edecektir. Web sitemizi barındıracak serverdaki klasörlerin konum bilgilerini her zaman bilemeyebiliriz. Dolayısıyla göreceli yol kullanmak daha avantajlı olacaktır.

Bir diğer konum belirleme yöntemi de URL yoludur. URL yolu, bir web sitesinin domain isminden sonra klasör veya dosya isminin belirtilerek konum bilgisinin elde edilmesidir.

Örnek verecek olursak: `http://www.webdomain.com/image/resim1.jpg`

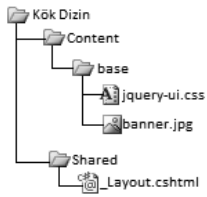
şeklinde belirtilmiş bir URL yolunun sunucu bilgisayardaki konumu aşağıdaki gibi olabilir.

`C:\inetpub\wwwroot\websitleri\web1\image\resim1.jpg`

Şimdi Razor söz diziminde bu yolların nasıl belirtildiğine bakalım. Razor'da yollar belirtilirken ~ operatörünü, `Server.MapPath` metodunu ve `Href` metodunu kullanır.

~ Operatörünün Kullanimi

Bilgisayardaki klasörlerin hiyerarşik yapılarını düşündüğümüz zaman, hiyerarşinin başında bulunan klasöre **kök dizin** denir.



Razor söz dizimin de server'da bulunan klasörlerin kök dizinini belirtmek için ~ operatörü kullanılır. ~ operatörünün kullanımıyla göreceli yol belirtmiş oluruz. Daha önce de belirttiğimiz gibi göreceli yol kullanımı sayesinde sistemimizi başka bir sunucu bilgisayara taşıdığımızda ve bilgisayarda bulunduğu konumu değiştirdiğimizde sistemimiz çalışmaya devam edecektir.

```
@{
 var layout= "~/Shared";
 var stylefile = "~/Content/base/jquery-ui.css";
 }
```

Yukarıdaki kodu ele aldığımızda göreceli yol belirtmek için kodun başına ~ operatörü kullanılmıştır. Bu kodun kullanıldığı dosya kök dizinden itibaren dosyaların hiyerarşisini kullanmakta olacaktır. Sistemimizi yapılandırmak istediğimizde sistemimiz çalışmasına devam edecektir.

Server.MapPath Metodu

`Server.MapPath` metodu kendisine gönderilen dosyanın konumu döndürür. Örneğin; `Server.MapPath("/Database1.mdf")` şeklinde bir parametre gönderildiğinde `Database1.mdf` dosyasının tam yolunu *C:\inetpub\wwwroot\siteler\yenisite\app_data\Database1.mdf* şeklinde döndürecektir. Bu yöntemle sistemimizde mevcut olan bir veritabanı dosyasının tam konumunun gerektiği durumlarda kullanabiliriz. Sitemizin barındırıldığı sunucu bilgisayarların klasör yapılarını bilmeyebiliriz. Bu durumlarda `Server.MapPath` yöntemi işimizi görecektir.

```
@Server.MapPath("Database1.mdf")
```

Href Metodu

`Href` metodu Razor içerisinde kullandığımız yolları tarayıcının anlayacağı şekilde dönüştürür. Örneğin; ~ operatörüyle oluşturduğumuz bir yolu web sayfamız içerisinde bir resmin adresi olarak kullanmak istediğimizde; tarayıcılar ~ operatörünün ne olduğunu anlamayacaklardır. Çünkü ~ operatörü Razor için kullanılan bir operatördür. Bunun web tarayıcısının anlayacağı biçimde dönüştürülmesi için `Href` metodu kullanılır.

Bir örnekle anlatacak olursak:

```
@{
 var layout= "~/Shared";
 var stylefile = "~/Content/base/jquery-ui.css";
 var image = "~/Content/base/image";
 }
```

Yukarıdaki kodumuzu incelediğimizde üç tane değişken tanımlanmış ve bu değişkenlere web sayfamızda kullanacağımız resim, stil ve layout dosyalarının yolları ~ operatörü kullanılarak atanmıştır. Bu değişkenleri direkt olarak HTML etiketleri içerisinde kullanabilmek için aşağıdaki gibi `Href` metodunu kullanmalıyız.

```
<img src=="@Href(myImagesFolder)/banner.jpg" />
<link rel="stylesheet" href="@Href(stylefile)" type="text/css" />
```

CONTROLLER 4

Controller'ın İşlevi

Controller yapısını incelemek için öncelikle MVC'nin hayat döngüsüne bakalım...

Kullanıcıdan bir istek geldiğinde **global.asax** içerisindeki **MapRoute** yapısı ile yönlenecek **Controller** ve **ActionResult**'a yönlenir. Action çalıştığında ilgili işlemler gerçekleştirilir. Eğer gerçekleştirilecek işlemler arasında veritabanı işlemi varsa, Controller Model ile iletişime geçerek isteğini bildirir. Modelden dönen datayı işler ve gösterime hazır hale getirir. Son olarak gösterime hazır hale gelen veri View'e gönderilir ve View üzerinde gösterimi yapılır.

Controller; **Model** ve **View** arasında verileri işleyerek iletişim görevi görür.

Action Kavramı

Varsayılan olarak bir url'i incelediğimizde *..\Home\Index* olarak göreceksiniz. Burada Home, Controller'ımızın adıdır. **Index** ise Controller içerisindeki Action'ınımızın adıdır. Actionlar'ın erişim belirleyicisi **public** olmalıdır. C#'ta varsayılan erişim belirleyicisi **private** olduğu için, Action'lar public olarak belirtilmelidir.

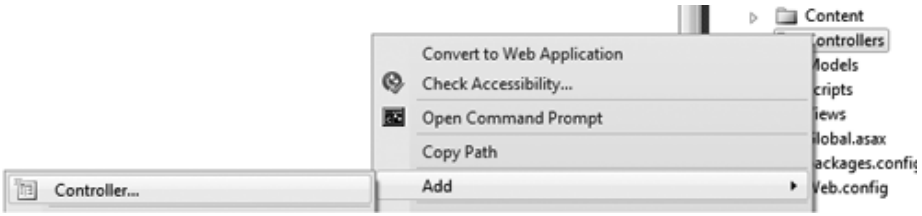
Action'lar aşağıdaki ActionResult ile geri dönerler:

- **`ViewResult:`** HTML markup döndürür.
- **`EmptyResult:`** Sunumu sonlandırır.
- **`RedirectResult:`** Farklı bir adrese yönlendirme yapar.
- **`JsonResult:`** JSON değeri Ajax için kullanılabilir duruma getirir.

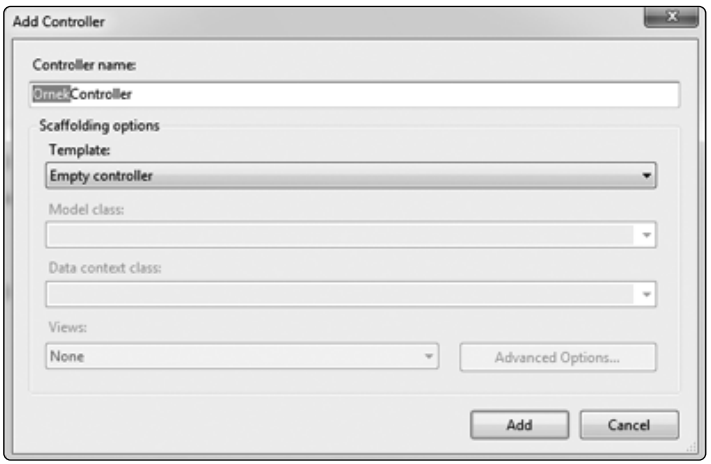
- **`JavaScriptResult:`** Kullanılabilir JavaScript değeri döndürür.
- **`ContentResult:`** Metin içeriği sunar.
- **`FileContentResult:`** İndirilebilir dosyaları temsil eder (*Binary*).
- **`FilePathResult:`** İndirilebilir dosyaları temsil eder (*Dosya yolu*).
- **`FileStreamResult:`** İndirilebilir dosyaları temsil eder (*Stream olarak*).

Controller Oluşturmak

Controller klasörümüzün üzerine sağ tıklayarak **Add>Controller**'ı seçiyoruz.



Controller'ımızı oluşturmak için ayarlarımızı yapıyoruz.



- **Controller name:** Controller adımızı yazıyoruz.
- **Template:** Controller oluşurken belirttiğimiz model yapısına uygun Action'ları oluşturmak istersek **Template** kısmından belirliyoruz.
- **Model Class:** Controller'ımızda işleyeceğimiz modelimizi seçiyoruz.

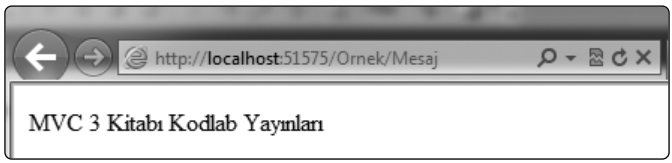
Action Oluşturmak

Index Action'ı Controller'ımız oluşturulduğunda hazır olarak gelir.

```
public ActionResult Index()
{
    return View();
}
```

`OrnekController` içerisine yeni bir Action oluşturup, projemizi çalıştıralım.

```
public string Mesaj()
{
    return "MVC 3 Kitabı Kodlab Yayınları";
}
```



Mesaj Action'ı çalıştığında mesajımız ekranda görünecektir.

Controller'dan View'e Veri Aktarımı

Controller üzerinde oluşturduğumuz yapıyı View'e taşımanın birçok yolu vardır. Bunlardan en sık kullanılanlar **Viewbag** ve **Model** olarak veriyi taşıma yoludur.

ViewBag kullanılırken Controller sayfamızda ViewBag yazıp nokta koyduktan sonra ViewBag adı ile eşittir yazıp eşitleyeceğimiz ifadeyi yazıyoruz. View sayfamızda da ViewBag'imizi aynı şekilde kullanıyoruz.

ViewBag'ın controllerda kullanımı:

```
public ActionResult Ornek()
{
    ViewBag.Yazi="Viewbag örneği";
    return View();
}
```

ViewBag'in View'de kullanımı:

```
<h2>Ornek</h2>
<p>@ViewBag.Yazi</p>
```

Kodumuzun sonucu aşağıdaki gibidir:



Viewbag örneği

Model nesnemizi kullanmak için öncelikle bir model oluşturmamız gerek. Modelimize Controller üzerinde erişip datamızı Controller üzerinde şekillendiriyoruz. Ardından View'imizi oluşturuyor ve View oluştururken kullanacağımız modeli belirliyoruz. Son olarak View üzerinde Model nesnesi yardımıyla verilerimize erişebiliriz.

Örneğimizde kullandığımız model:

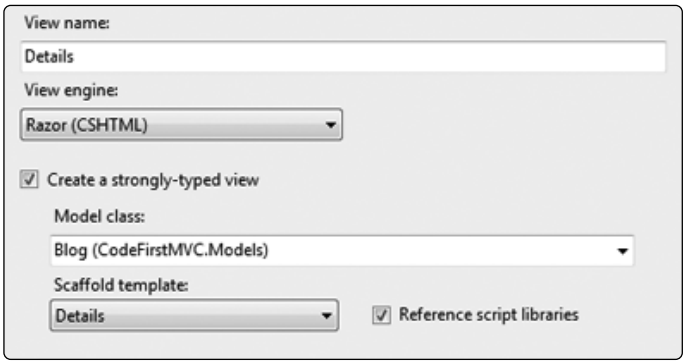
```
public class Blog
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string BloggerName { get; set; }
    public DateTime DateCreated { get; set; }
    public virtual ICollection<Post> Posts { get; set; }
}
```

Controller'ımıza yazdığımız Action aşağıdaki gibidir:

```
BlogContext db = new BlogContext();
public ActionResult Details(int id)
{
     return View(db.Blogs.ToList().
               Where(x=>x.Id==id).FirstOrDefault());
}
```

Action'ımızı oluştururken `id` parametresini gönderiyoruz. Bu parametre ile blog modelimiz üzerinden gönderilen ID'ye ait kayıtlarımızı alıyoruz. Action'ımıza bu parametreyi url kısmının sonuna ekleyerek gönderiyoruz. Örneğin; */Blog/Details/4* olarak girilen URL'de `id` değeri `4` olarak gelir.

View'imizi oluştururken modelimizi View'imizin template'ini seçiyoruz.



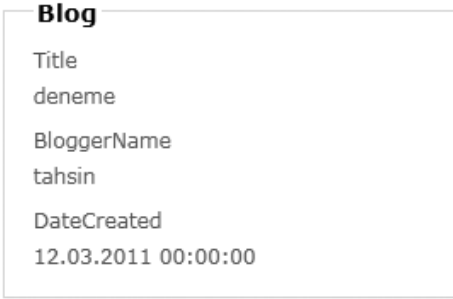
Son olarak View'imizi oluşturuyoruz. View yapımızı inceleyecek olursak, üst kısımda model nesnemiz tanımlanıyor.

```
@model CodeFirstMVC.Models.Blog
```

Seçtiğimiz template'den dolayı View üzerinde verilerimizi gösterecek yapı otomatik olarak oluşmaktadır. Bu yapılar, Model nesnemizi kullanarak verileri göstermemizi sağlıyor.

```
<legend>Blog</legend>
<div class="display-label">Title</div>
<div class="display-field">
    @Html.DisplayFor(model => model.Title)
</div>
<div class="display-label">BloggerName</div>
<div class="display-field">
    @Html.DisplayFor(model => model.BloggerName)
</div>
<div class="display-label">DateCreated</div>
<div class="display-field">
    @Html.DisplayFor(model => model.DateCreated)
</div>
```

Uygulamamızı çalıştırdığımızda aşağıdaki gibi bir sayfa elde edeceğiz.



MODEL VE VERİTABANI İŞLEMLERİ

5

Model Katmanı

Model katmanını en genel biçimde tanımlayacak olursak; veritabanı işlemlerinin yapıldığı yerdir diyebiliriz. Bu tanımı biraz genişletmemiz gerekirse, uygulamanın veriye erişimini ve iş mantığını gerçekleştiren nesneleri içeren katmandır. Yani veritabanına veri eklenirken veya veri çekilirken hangi kurallara göre yapılacağı ve bu işlemlerin yapıldığı kısımdır. Bu katmanda **Linq To SQL** dosyaları, **Entity Framework** dosyaları veya data taşıyan nesneler bulunur. Model katmanı, Controller katmanı ile iletişim halindedir. Model katmanında **ORM** (**O***bject* **R***elation* **M***apping*) araçları sayesinde veritabanı tablolarını nesne olarak kullanabiliriz.

Model Validation

Validation işlemi, veri girişlerinde girilecek verinin uygunluğunu kontrol etmemizi sağlar. Validation işlemi iki türlü gerçekleştirilir. Bunlar; client tarafında JavaScript ile ve server tarafında modelimizi oluştururken ekleyeceğimiz attribute'lar iledir. Konumuzda model validation anlatılacaktır. Modelimizi nesne olarak oluşturduktan sonra attribute'larımızı ilgili value'ların üzerine yazıyoruz. Attribute'lar `System.ComponentModel.DataAnnotations` isim alanı altında bulunur.

Aşağıda projemizi oluşturduğumuzda otomatik olarak oluşan `AccountModels`'den bir class var.

```
public class ChangePasswordModel
{
```

```
    [Required]
    [DataType(DataType.Password)]
    [Display(Name = "Current password")]
    public string OldPassword { get; set; }
    [Required]
    [StringLength(100, ErrorMessage =
         "The {0} must be at least {2}characters long.",
         MinimumLength = 6)]
    [DataType(DataType.Password)]
    [Display(Name = "New password")]
    public string NewPassword { get; set; }
    [DataType(DataType.Password)]
    [Display(Name = "Confirm new password")]
    [Compare("NewPassword", ErrorMessage =
         "The new password and confirmation password do not match.")]
    public string ConfirmPassword { get; set; }
}
```

`[ ]` arasına yazılan attribute'lar, altlarında bulunan veriyi nitelemektedir. Aşağıdaki örneği incelersek:

```
[Required]
[DataType(DataType.Password)]
[Display(Name = "Current password")]
public string OldPassword { get; set; }
```

`Required` ile bu alanın girilmesi zorunlu bir alan olduğunu, `DataType` değerinin `Password` olmasıyla, bu alanın şifre olduğu, `DisplayName` ile form alanlarında kullanmak üzere yazısını tanımlıyoruz.

Modelimiz üzerinde yaptığımız bu şekilde tanımlamalar ile girilecek verilerin istediğimiz şekilde girilmesini sağlıyoruz. Modeli validate ederken sık kullandığımız attribute'lar, açıklamalarıyla aşağıda verilmiştir.

- **`Required:`** Girilmesi zorunlu alan.

  Kullanımı: `[Required]`

- **`Display Name:`** Form alanında görünen Ad.

  Kullanımı: `[Display(Name = "Kullanıcı Adı")]`

- **`StringLength:`** Girilecek verinin uzunluğu.

Kullanımı: `[StringLength(100, ErrorMessage = "6 ile 100 harf arası değer alır", MinimumLength = 6)]`

- **Range:** Girilebilecek en büyük ve en küçük sayı tanımlanır.

  Kullanımı: `[Range(0,100,ErrorMessage="0 ile 100 arası bir sayı")]`

- **DataType:** Alan özel bir tanım ise (url, e-mail, telefon numarası...) veri tipi tanımlanır.

  Kullanımı: `[DataType(DataType.EmailAddress)]`

- **ScaffoldColumn:** Otomatik olarak oluşturulan formlarda, bu alanın oluşup oluşmamasını sağlar.

  Kullanımı: `[ScaffoldColumn(false)]`

- **Remote:** Bir Action ile veri kontrolü yapmamızı sağlar. En çok kullanılan `overload` metodunun 1. parametresi `action name`, 2. parametresi `controller`, 3. parametresi uyarı mesajını alır.

  Kullanımı: `[Remote("AdKontrol","KullaniciIslemleri",ErrorMessage="Bu kullanıcı adı kullanılmaktadır.")]`

- **Compare:** Karşılaştırma için kullanılır. 1. parametre karşılaştırma yapılacak alanı, ikinci parametre uyarıyı belirtir.

  Kullanımı: `[Compare("Sifre",ErrorMessage="Girdiğiniz şifreler birbiri ile uyuşmuyor")]`

- **RegularExpression:** Düzenli ifadeler için kullanılır.

  Kullanımı: `[RegularExpression(@"\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*", ErrorMessage = "Mail adresini istenilen formatta giriniz.")]`

- **AllowHTML:** Html kodları girileceği belirtilir.

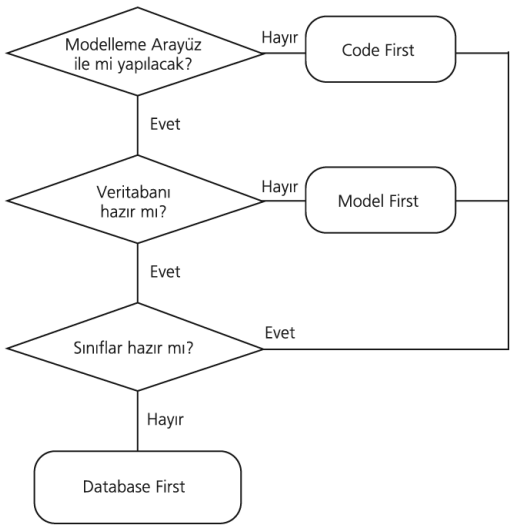
  Kullanımı: `[AllowHtml]`

Modelleme Yaklaşımları

Model katmanımızda **LinqToSQL** ve **Entity Framework** gibi yapılar aracılığı ile veritabanımızı kolaylıkla modelleyebiliriz. Modellenen veritabanının yapısı, tablolar sınıf olarak ve alanlar sınıfların özelliği olarak aktarılır. Ayrıca ekleme, silme, listeleme, güncelleme metotları da modelleme yapacağımız araç tarafından otomatik olarak oluşturulur.

Veritabanı modelleme işlemi için üç farklı yaklaşım vardır:

1. **Database First:** Veritabanınız önceden tasarlanmışsa veya farklı bir yerde hazır duruyorsa modelleme aracı sizin için tabloların sınıflara dönüşümü ile okuma, yazma, güncelleme, silme metotlarını oluşturur.
2. **Model First:** Kod veya SQL sorguları yazmak istemiyorsanız, model oluşturulur ve bu modeldeki yapıya göre scriptl'ler ve tablolar oluşturulur.
3. **Code First:** Modelleme aracının kodları kendisi üretmesi bazen sorun olabiliyor ve geliştiriciler bu durumdan rahatsız olabiliyor. Bu durumda code first yapısı çok uygun olabiliyor. Bu yöntem kullanılırken Entity Framework veritabanını kendisi oluşturuyor.



Veritabanı Kavramı

Veritabanı; verileri düzenli, sistematik ve ilişkili şekilde tutmamızı sağlayan yapılardır. Veritabanındaki veriler taşınabilir, güncellenebilir ve yönetilebilir yapıdadır. Bir veritabanı yapısı bir veya birden fazla tablodan oluşur. Tablolar, benzer tipte verilerin bulunduğu iki boyutlu yapıdır. Bir tablonun üstünde alanları tanımlayan başlıklar bulunur.

Alanlar tanımlanırken **Alan Adı** ve **Veri Tipi** ile tanımlanır. Alan adı, o alanı belirten tanımlayıcı isimdir. Veri tipi ise; alana eklenecek verinin en fazla olabilecek boyutunu ve türünü belirler. Veritabanı oluştururken kullanacağımız en sık veri tipleri aşağıda belirtilmiştir:

- **`Bit:`** Fiziksel olarak `1` bayt yer kaplar. Bit `1` ya da `0` değeri depolar.
- **`Smallint:`** Fiziksel olarak `2` baytlık yer kaplar `-32768` ile `32767` arasında değerleri alır.
- **`Int:`** Fiziksel olarak `4` baytlık yer kaplar `-2.147.483.648` ile `2.147.483.647` arasındaki değerleri depolar.
- **`Decimal:`** Ondalıklı değerler alır.
- **`Varchar:`** En fazla `8000` karaktere kadar olan yazısal değerleri depolar.
- **`Text:`** Çok fazla uzunluktaki karakterleri depolar. Boyutu değişkendir.
- **`DateTime:`** Tarih verisini tutar.

  `Varchar` ve `Text`'in başına `n` ifadesi gelirse `UNICODE` karakter tipini destekler.

  Bir alan veritabanı üzerinde `NULL` olarak tanımlanırsa o alana veri girilmeden bir satırın kaydedilebileceği anlamına gelir.
- **`Identity:`** Bir alanın ardışık değerler alması sağlanır. Verilen değere göre artan veriler satırın ilgili alanına set edilir. Tanımlayıcı alanlarımıza (*primary key*) identity özelliği verilir ve birer birer artırılacak şekilde ayarlanır. Bu alanlar NULL olamazlar.

En sık kullanılan veritabanları Microsoft Office uygulaması ile gelen Access, Microsoft SQL Server, Oracle ve MySQL'dir. Biz örneklerimizde **SQL Server** kullanacağız.

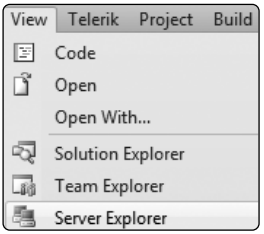
LINQ To SQL İLE VERİTABANI İŞLEMLERİ

LINQ kelime olarak **Language Integrated Query** yani dil ile bütünleştirilmiş sorgu anlamını taşımaktadır. Bildiğiniz gibi veritabanı işlemleri için SQL kullanıyorduk. LINQ ile artık bu sorgulama işlemleri için ayrı bir dil kullanmaya gerek kalmaz ve veritabanı üzerinde ekleme, silme, güncelleme gibi işlemler gerçekleştirebiliriz. LINQ To SQL ile .NET sınıflarını kullanarak ilişkisel veritabanı modeli oluşturulur. LINQ To SQL, veritabanı içerisindeki transaction, view ve stored prosedür'leri destekler. Veritabanı içerisindeki bu tür nesneler .NET nesnelerine dönüştürülerek üzerinde kolayca işlemler gerçekleştirilir.

LINQ to SQL yapısını anlatırken örneğimizde modelleme yaklaşımı olarak da **Database First** yaklaşımını kullanacağız. Konu anlatımı için örneğimizde basit bir ziyaretçi defteri uygulaması hazırlayacağız. Uygulamamızda mesajların yazıldığı, okun-

duğu ve site yöneticisi tarafından silme, düzenleme ve onaylama işlemlerinin yapılacağı sayfalar olacak. Veritabanımızda ise mesajların tutulacağı bir tablo olacak.

Öncelikle boş bir **MVC 3 Razor** projesi oluşturuyoruz. Ardından veritabanımızı oluşturacağız. **View>Server Explorer** adımlarını izleyerek **Server Explorer**'ı açıyoruz.



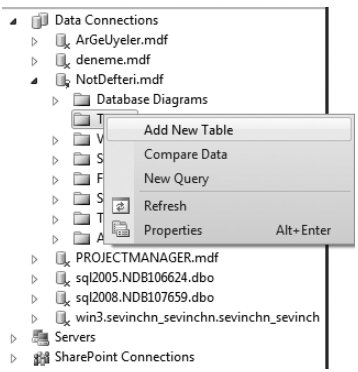
Connect to Database butonuna tıklıyoruz.



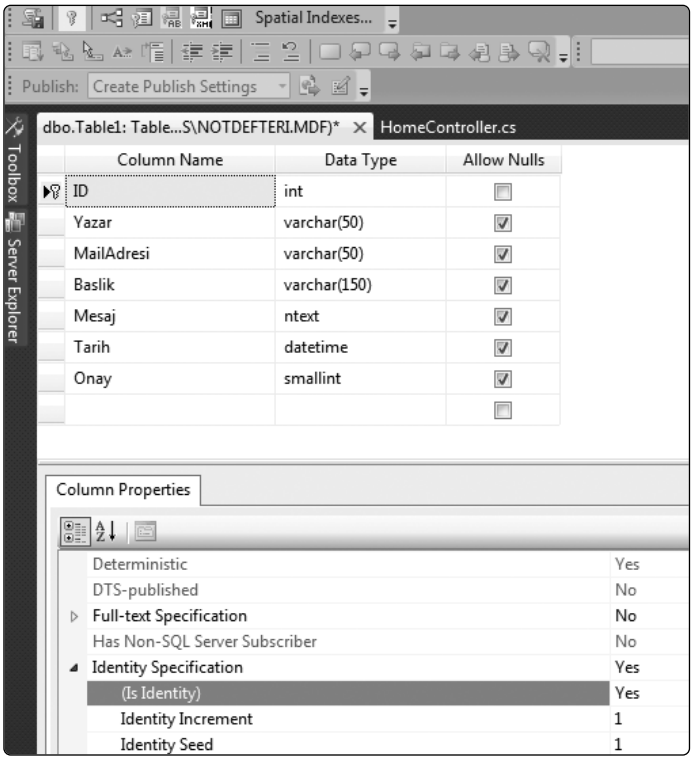
Veritabanı adımızı giriyoruz. **OK** butonuna basıyoruz. *Veritabanını oluşturmak istediğinizden emin misiniz?* şeklinde bir uyarı penceresi açılacaktır; onayladıktan sonra veritabanımız oluşacaktır. Veritabanı oluşturulması sırasında hata oluşursa **SQL Server Configuration Manager**'dan ayarları kontrol ediniz.



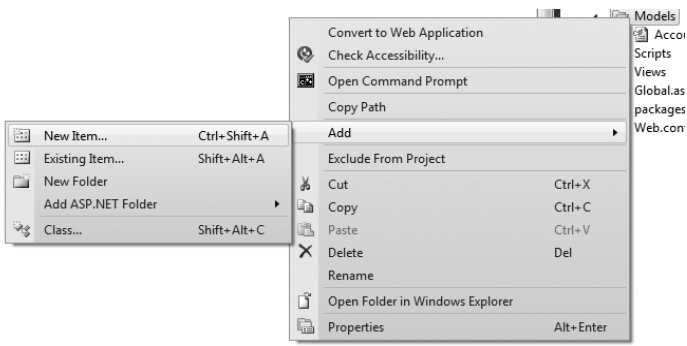
Data Connections altında veritabanımızı göreceksiniz. Şimdi tablolarımızı oluşturalım.

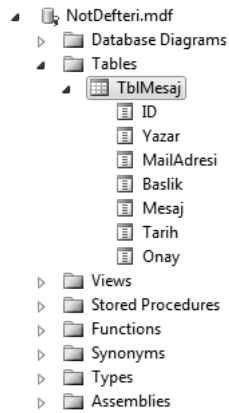


Tablomuzu aşağıdaki şekilde oluşturuyoruz. **ID** alanını **Primary Key** olarak atıyoruz ve **Identity** durumunu aktif hale getiriyoruz.

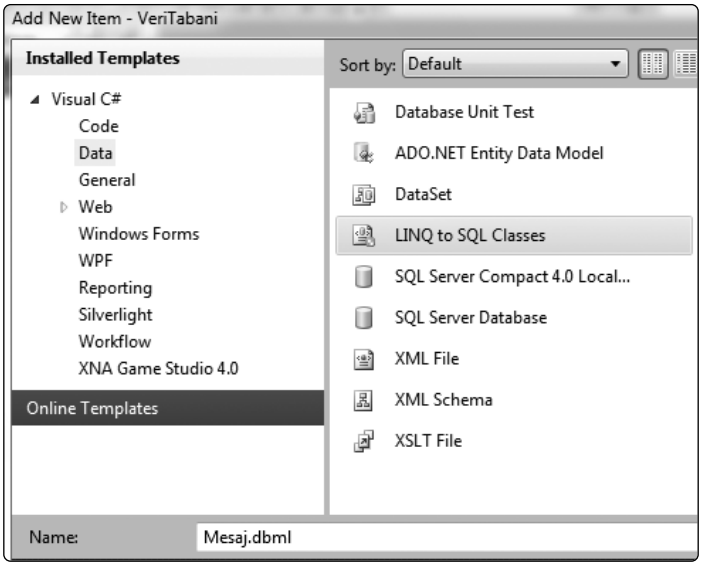


İşlemlerimizi kaydediyoruz ve tablo adımızı giriyoruz. Veritabanımızın son hali yandaki gibi oluyor. Veritabanımızı oluşturduktan sonra **LinqToSQL** ile modelleme yapıyoruz. **Model** klasörüne sağ tıklayıp **Add>New Item** yolunu izliyoruz.

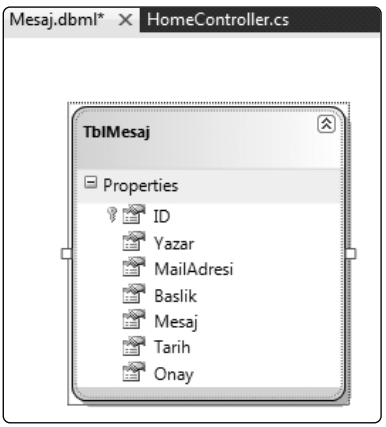




Açılan pencereden **Data** altında yer alan **LINQ to SQL Classes**'a tıklayıp modelimizin adını giriyoruz ve **Add** butonuna tıklayıp işlemimizi tanımlıyoruz.



Server Explorer penceresinden tablomuzu açılan ekranın üzerine sürükleyip bırakıyoruz. Karşınıza önce bir uyarı çıkacaktır. Bu işlem yapılırken **connection string**'in **Web.config** üzerinde otomatik olarak oluşturulmasını istiyorsanız, **Evet** demeniz yeterli.



Tüm bu işlemleri yaptıktan sonra kaydedip **Mesaj.dbml** penceresini kapatabilirsiniz.

Model klasörümüzün içerisine öncelikle veritabanı işlemlerini gerçekleştireceğimiz bir class oluşturuyoruz.

Sınıfımızda öncelikle `DataContext`'imizi **global** değişken olarak oluşturuyoruz. `DataContext` modelimize bağlanmamızı sağlıyor. Linq To Sql'de modelimizi oluştururken `DataContext`'imiz de otomatik olarak oluşmuştur. Modelimize verdiğiniz adın sonuna `DataContext` ekleyerek görebilirsiniz.

```
MesajDataContext dbContext=new MesajDataContext();
```

Şimdi class'ımız içerisinde ekleme, listeleme ve silme metotlarını oluşturacağız. Fakat bu metotlarımız, değişiklikleri model üzerinde yapmaktadır. Bu işlemlerin veritabanında kalıcı şekilde değişmesi için bir de `Kaydet` isminde bir metot oluşturacağız. Dikkat ederseniz güncelleme metodundan bahsetmedik. Güncelleme metodu controller class'ı içerisinde ayrı metot olarak tanımlıdır.

Ekleme işlemini yapan metodumuz:

```
public void EkleMesaj(TblMesaj ptMesaj)
{
    dbContext.TblMesajs.InsertOnSubmit(ptMesaj);
}
```

Listeleme işlemini yapan metodumuz:

```
public List<TblMesaj> ListeleMesaj(bool pbSadeceOnayli)
{
    if(pbSadeceOnayli)
        return dbContext.TblMesajs.Where(x => x.Onay==1).ToList();
    else
        return dbContext.TblMesajs.ToList();
}
```

Listeleme işleminde sadece onaylanmış mesajları listelemek için `true` bir parametre göndermeliyiz. Parametremizi `false` olarak gönderirsek tüm kayıtlar gelecektir.

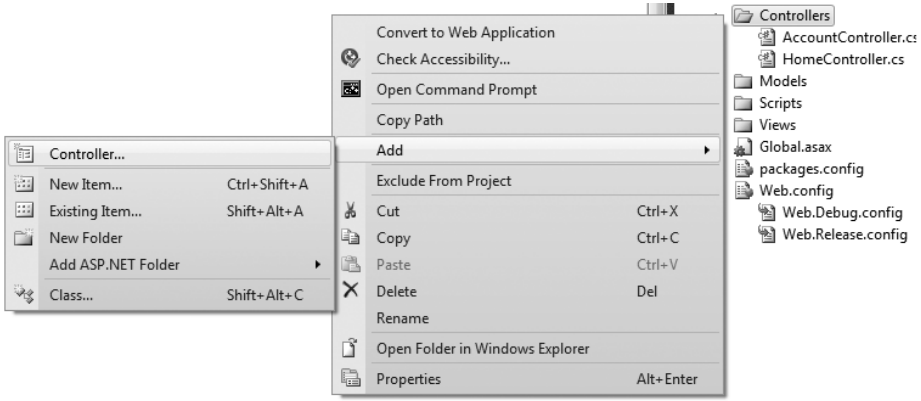
Silme işlemini yapan metodumuz:

```
public void SilMesaj(TblMesaj ptMesaj)
{
    dbContext.TblMesajs.DeleteOnSubmit(ptMesaj);
}
```

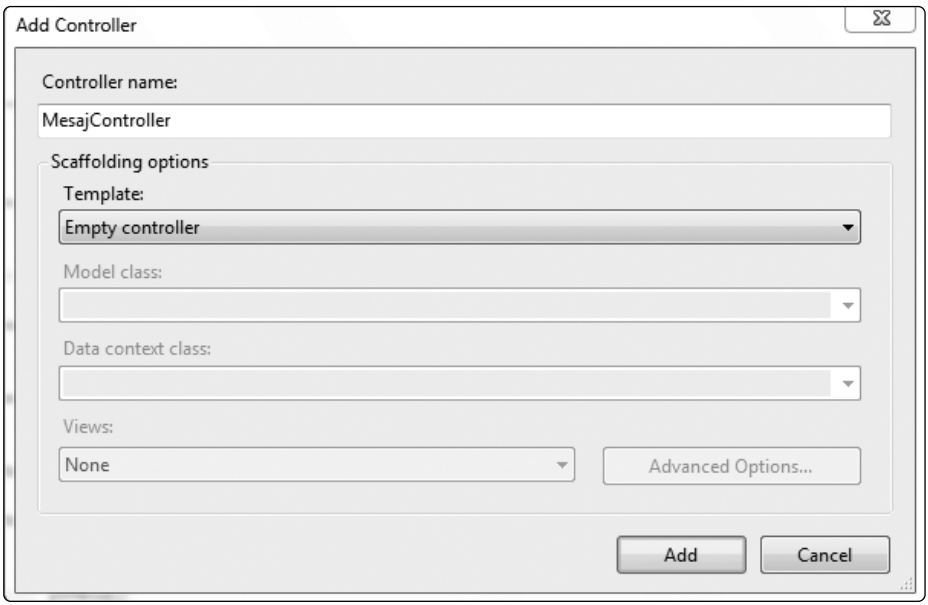
Kaydetme işlemini yapan metodumuz:

```
public void Kaydet()
{
    dbContext.SubmitChanges();
}
```

Şimdi sıra geldi **Controller** ve **View**'lerin oluşmasına. Öncelikle `MesajController` adında bir controller oluşturuyoruz.



Controller adına `MesajController` yazıyoruz, **Scaffolding Options** kısmında **Template** olarak **Empty Controller** diyoruz ve **Add** butonuna basıyoruz.



Controller'ımız sadece **Index** adlı ActionResult döndüren metot ile oluştu. Şimdi mesaj okuma ve ekleme sayfalarını yazalım. Fakat öncelikle veritabanı işlemlerini

yazdığımız sınıfı oluşturmamız gerek. Bu sınıf Controller içerisinde direkt gözükmeyecektir. Bunun sebebi; farklı bir namespace içerisinde olmasıdır. Bu yüzden önce namespace'imizi ekliyoruz. İlgili namespace'imiz **ProjeAdı.Models** şeklindedir.

```
using VeriTabani.Models;
```

Global bir değişken olarak sınıfımızı oluşturalım.

```
DBMesaj cMesaj=new DBMesaj();
```

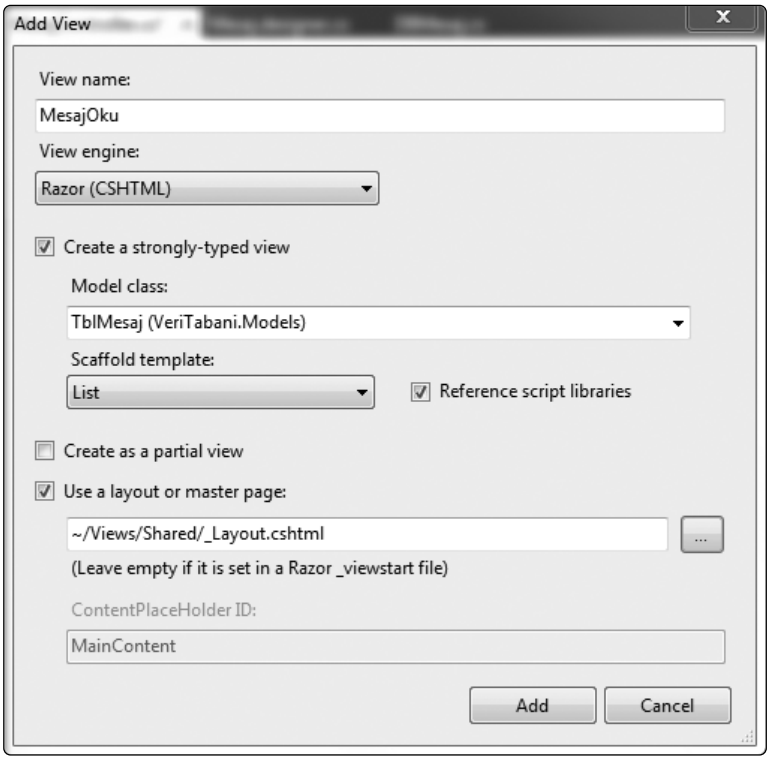
Öncelikle listeleme işlemini yapan ActionResult'ı oluşturalım.

```
public ActionResult MesajOku()
{
    return View(cMesaj.ListeleMesaj(true));
}
```

`MesajOku` isminde ActionResult'ımızı oluşturduk ve geriye mesajlarımızın listesi dönecek. Parametre olarak `true` dediğimiz için sadece onaylı mesajlar dönecek. Sıra geldi View'imizi oluşturmaya...

ActionResult üzerine sağ tıklayıp **AddView**'e tıklıyoruz. Açılan pencereden ayarlarımızı yapıyoruz.

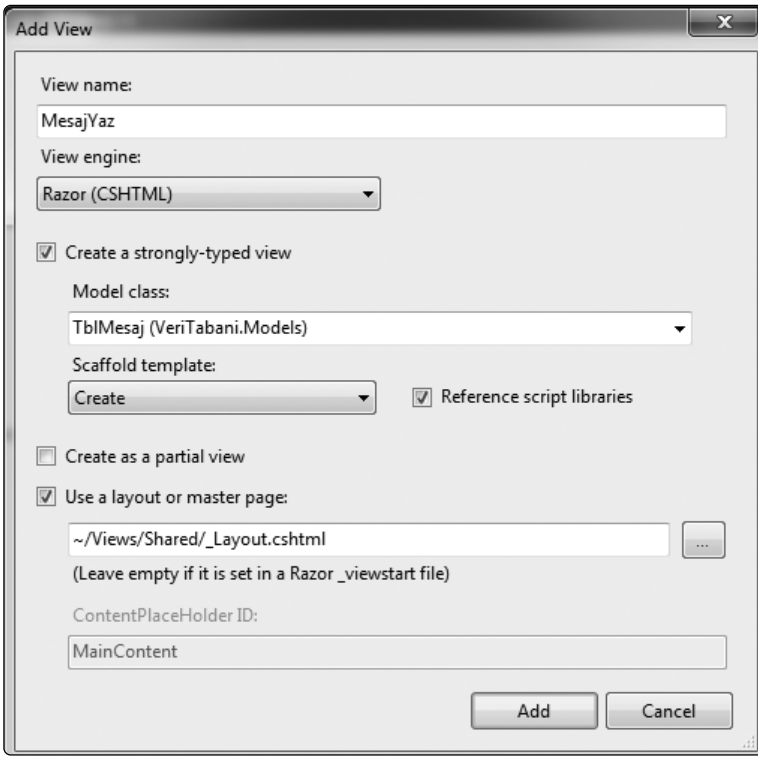
Create a strongly-typed view seçeneğini işaretliyoruz. **Model class** olarak `TblMesaj` üzerinde işlem yapacağımız için onu seçiyoruz. **Scaffold template** olarak listeleme işlemi yapılacağı için **List** seçeneğini seçiyoruz. Son olarak **masterpage**'imizi de seçip **Add** butonuna basıyoruz.



View sayfamız otomatik olarak oluşacaktır. Herhangi bir değişiklik yapmanıza gerek olmadan sayfa sorunsuzca çalışır, fakat ilerleyen kısımlarda birkaç değişiklik yapacağız.

Mesaj ekleme işlemini yazalım. Fakat mesaj ekleme kısmının iki aşaması olacak. Birinci aşamada sayfanın ilk açılışında form sadece gözükecek. İkinci aşamada ise girilen bilgilerin kaydedilmesi için butona basıldığında olacak işlemler yer alacak. Önce veri eklemek için View'imizi oluşturacağız, fakat bu iş bizim için otomatik olarak yapılacaktır. Yine önce `MesajYaz` ActionResult'ımızı yazdıktan sonra View'imizi ekleyeceğiz. Bu kez **View Template**'imiz **Create** olacak.

```
public ActionResult MesajYaz()
{
    return View();
}
```



Form sayfamız bizim için oluşturuldu. Buradan tarih ve onay girişini siliyoruz. Çünkü bu kısımları biz arka tarafta gireceğiz.

```
        @Html.ValidationMessageFor(model => model.MailAdresi)
    </div>

    <div class="editor-label">
        @Html.LabelFor(model => model.Baslik)
    </div>
    <div class="editor-field">
        @Html.EditorFor(model => model.Baslik)
        @Html.ValidationMessageFor(model => model.Baslik)
    </div>

    <div class="editor-label">
        @Html.LabelFor(model => model.Mesaj)
    </div>
    <div class="editor-field">
        @Html.EditorFor(model => model.Mesaj)
        @Html.ValidationMessageFor(model => model.Mesaj)
    </div>

    <div class="editor-label">
        @Html.LabelFor(model => model.Tarih)
    </div>
    <div class="editor-field">
        @Html.EditorFor(model => model.Tarih)
        @Html.ValidationMessageFor(model => model.Tarih)
    </div>

    <div class="editor-label">
        @Html.LabelFor(model => model.Onay)
    </div>
    <div class="editor-field">
        @Html.EditorFor(model => model.Onay)
        @Html.ValidationMessageFor(model => model.Onay)
    </div>
```

Mesaj kısmı ise çok satırlı bir giriş alanı olması daha iyi olacağı için buradaki **Editor**'ü **TextArea** ile değiştirmemiz gerekiyor.

```
@Html.TextAreaFor(model => model.Mesaj)
```



Butona bastığınızda herhangi bir işlem olmayacaktır. Sayfadaki veriler post edildiğinde çalışacak metodumuzu yazalım:

```
[HttpPost]
public ActionResult MesajYaz(FormCollection collect)
{
       TblMesaj mesaj=new TblMesaj();
       mesaj.Baslik=Request.Form[“Baslik”];
       mesaj.MailAdresi=Request.Form[“MailAdresi”];
       mesaj.Mesaj=Request.Form[“Mesaj”];
       mesaj.Onay=0;
       mesaj.Tarih=DateTime.Now;
       mesaj.Yazar=Request.Form[“Yazar”];
       cMesaj.EkleMesaj(mesaj);
       cMesaj.Kaydet();
       return View();
}
```

Özellik olarak baş kısma `HttpPost` ekledik ki post işleminde bu metot çalışsın. Ayrıca yazdığımız, metot sayfamızı çalıştıran ActionResult'ın overload edilmiş hali. Bu metodumuz `FormCollection` tipinde bir parametre alıyor. Metodumuz içerisinde öncelikle modelimiz `TblMesaj` sınıfından bir nesne türetiyoruz. Bu nesnenin elemanlarına ilgili form elemanlarındaki değeri atıyoruz. Ardından `EkleMesaj` metodumuzu kullanarak verimizi ekliyor ve `Kaydet` metodu ile yapılan değişiklikleri kaydediyoruz. Fakat mesaj kaydettiğimizde listeleme sayfasında gözükmeyecektir. Bunun sebebi; kaydedilen mesajların onay durumu `0` iken, listeleme sayfasında onay durumu sadece `1` olanlar listeleniyor.

Şimdi uygulamamızda mesajların onaylanacağı sayfayı yapalım. Sayfamızda mesajlar listelenecek ve her mesajın yanında link bulunacak. Bu linke tıklandığında mesaj onaylı ise onayı kaldırılacak, onaysız ise onaylanacak. Öncelikle daha önceki gibi listeleme işlemi yapıyoruz.

```
public ActionResult MesajOnayla()
{
     return View(cMesaj.ListeleMesaj(false));
}
```

Fakat View sayfamızda belli değişiklikler olacak. Öncelikle başlıklara **Onayla/Onay Kaldır** adlı bir başlık ekliyoruz.

```
<table>
  <tr>
    <th>
      Onayla/ Onay Kaldır
    </th>
    <th>
      Yazar
    </th>
    <th>
      MailAdresi
    </th>
    <th>
```

Şimdi satırlarımıza linkimizi ekleyelim.

```
@foreach(var item in Model)
{
  <tr>
  <td>
  @Html.ActionLink("İşlemi Yap", "OnayIslemi", new { piID=item.ID, psOnayDurum=(item.Onay==0)?1:0 })
  </td>
    <td>
      @Html.DisplayFor(modelItem => item.Yazar)
    </td>
    <td>
      @Html.DisplayFor(modelItem => item.MailAdresi)
```

Action linkimizi inceleyelim:

- **Text:** İşlemi Yap.
- **`ActionResult:OnayIslemi`**
- **Parametreler:** `piID` kaydın ID'si, `psOnayDurum` ise kaydın onay durumu, `0` ise `1`, `1` ise `0` değerini alıyor. Bunun sebebi onaylı bir kayıt ise onay kaldırma, onaysız bir kayıt ise onaylama işlemi yapılacaktır.

Son olarak onaylama işlemini yapan ActionResult'ımızı yazalım.

```
public ActionResult OnayIslemi(int piID, short psOnayDurum)
{
     TblMesaj mesaj=new TblMesaj();
     mesaj=cMesaj.ListeleMesaj(false).Where
        (x => x.ID == piID).SingleOrDefault();
     mesaj.Onay=psOnayDurum;
     UpdateModel(mesaj);
     return View("MesajOnayla",cMesaj.ListeleMesaj(false));
}
```

Kodumuz içerisinde, `TblMesaj` sınıfından mesaj adlı nesne türetiyoruz. Bu nesnemize düzenleme işlemi yapılacak kaydı atıyoruz. Kaydın onay durumuna değişecek onay durumunu atıyoruz ve `UpdateModel` ile modelimizi güncelleyip değişikliklerimizi kaydediyoruz. Son olarak ActionResult'ımız `MesajOnayla` View'i olarak değer döndürüyor. Bu View'e tekrar listeyi gönderiyoruz.



Son olarak **MesajOku** sayfasına geri dönüp listeleme işleminde ufak bir değişiklik yapıyoruz. **MesajOku** sayfamızın son şekli aşağıdaki gibi olacaktır.

```
@foreach(var item in Model)
{
  <tr>
    <td>
      @Html.DisplayFor(modelItem => item.Yazar)
    </td>
    <td>
      <b>@Html.DisplayFor(modelItem => item.Baslik)</
    </td>
  </tr>
  <tr>
    <td>
      @Html.DisplayFor(modelItem => item.Tarih)
    </td>
    <td>
      @Html.DisplayFor(modelItem => item.Mesaj)
    </td>
  </tr>
}
```



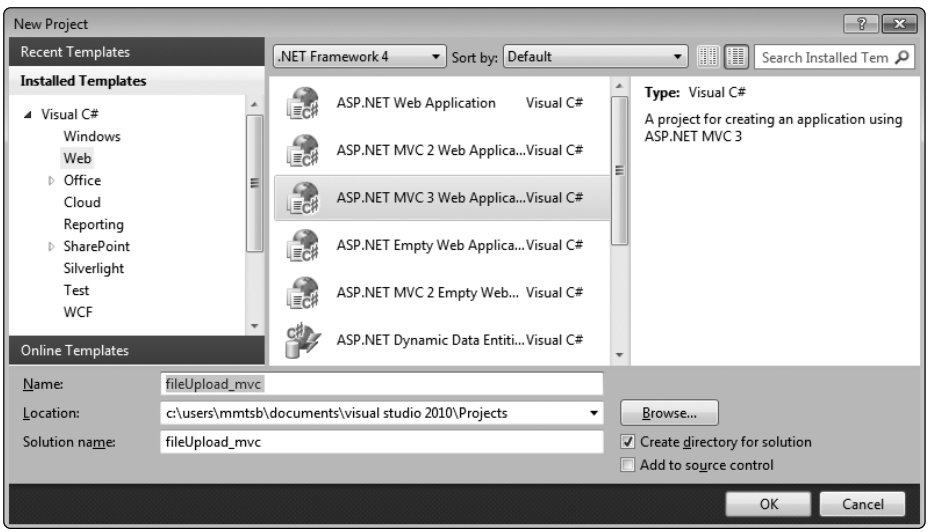
Entity Framework İle Veritabanı İşlemleri

Entity Framework, .NET'e entegre olarak çalışan ve veri tabanında oluşturmuş olduğunuz nesnelerin kod tarafında karşılığını oluşturan bir arayüzdür. Entity Framework'u daha iyi anlamak için örnek bir proje yapacağız.

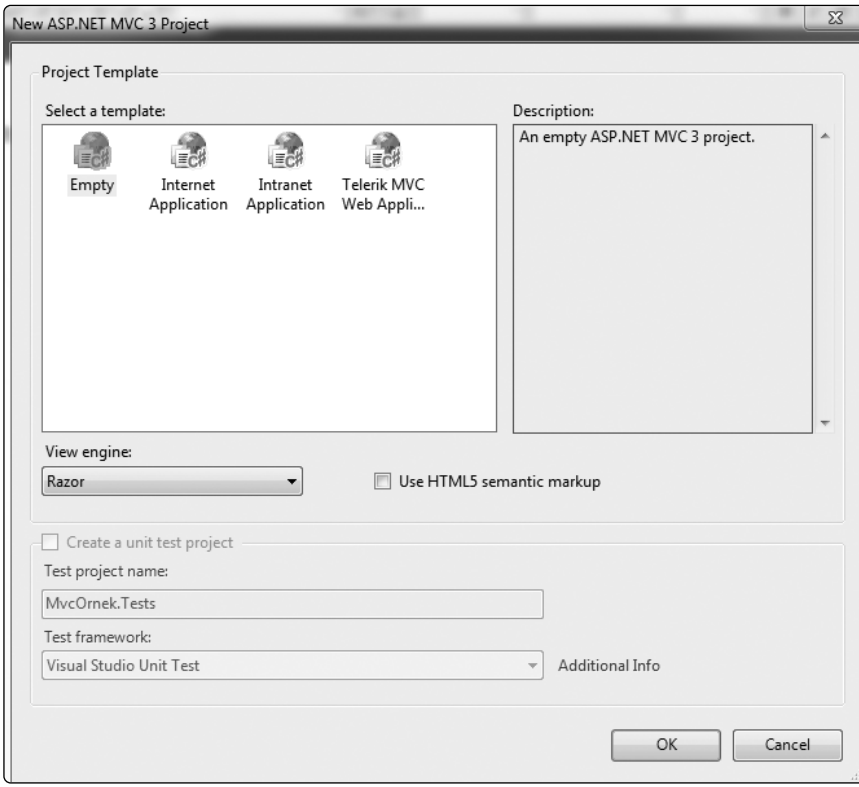
Bu projemizin amacı; web ortamında server'a yüklediğimiz dosyaların boyutu, ismi, türü ve uzantısı gibi bilgileri bir database'e kaydettikten sonra database'de tanımladığımız anahtar alan bilgisiyle, yani id numarasıyla dosyayı kaydetmek olacaktır. Peki, bu proje ne işimize yarayacak veya kullanım alanları nelerdir?

Bu dosya yükleme yöntemiyle admin sayfalarında yüklediğimiz dosyaları veritabanı üzerinden tekrar çağırabiliriz veya buna benzer olarak kullanıcı tanımlama sayfası oluşturup, her kullanıcının yükleyeceği dosyayı, kendi kişisel fotoğrafını, kullanıcının id numarası ve fotoğrafın uzantısıyla servera kaydedebiliriz.

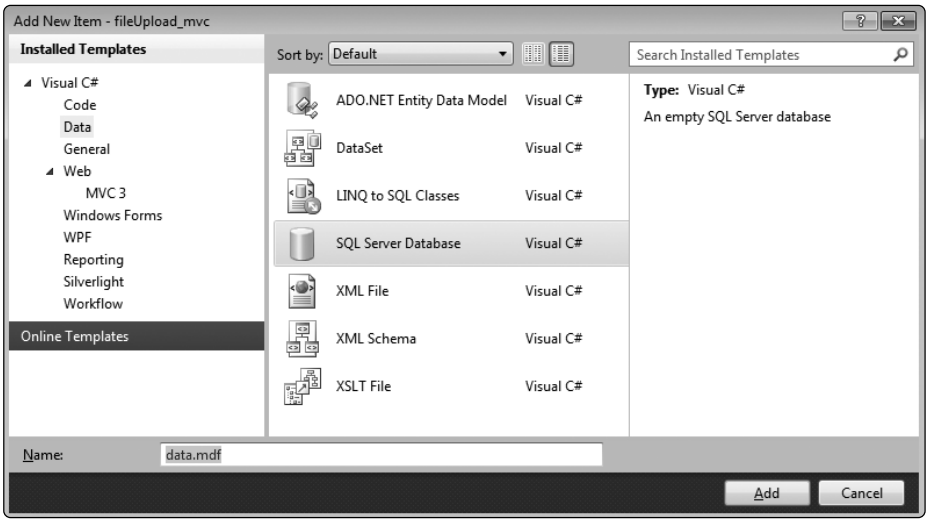
Örnek uygulamamız için `fileUpload_mvc` isminde **ASP.NET MVC3 Web Application** projesi oluşturalım.



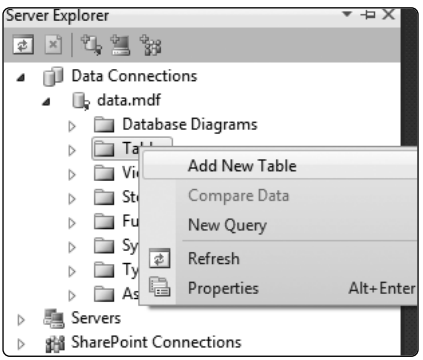
Projemizi isimlendirdikten sonra karşımıza çıkan ekranda projemizde kullanacağımız **View Engine** seçenekleri belirecektir. Çıkan ekranda **Razor View Engine**'ı seçiyoruz.



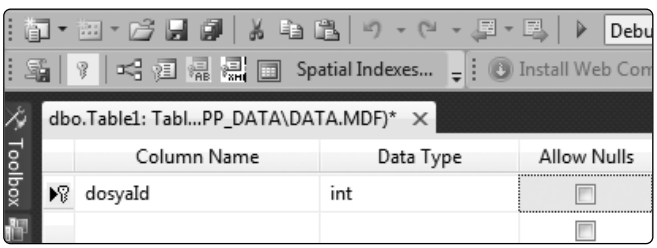
Upload ettiğimiz dosyaların bilgilerini saklamak için veritabanı dosyası oluşturuyoruz. Oluşturacağımız veritabanı dosyası içerisine upload edilecek dosyaların uzantılarını, türlerini, boyutlarını ve dosya açıklamaları ekleyebiliriz. Bunun için **Solution Explorer** penceresinde proje üzerinde sağ tıklayarak **Add>New Item** seçeneğini tıklıyoruz.



Açılan pencerede **SQL Server** database'i seçip **data.mdf** olarak adlandırdıktan sonra **Add** tuşuna basarak ekliyoruz. Daha sonra **Server Exlorer** penceresinden **data.mdf** dosyamızı seçiyoruz. **data.mdf**'nin altında bulunan **data.mdf>tables>Add New Table** seçeneğini seçiyoruz.



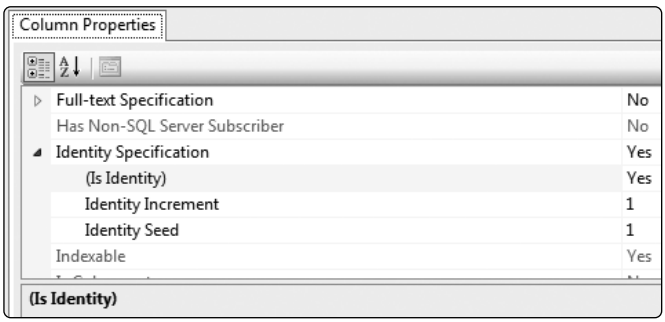
Aşağıdaki gibi bir ekran karşımıza çıkacaktır.



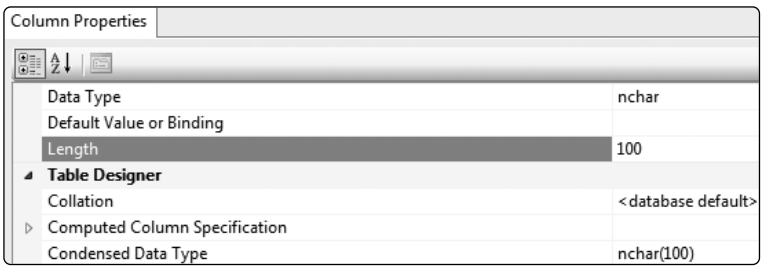
Bu ekranda **Column Name** alan adını, **DataType** veri türünü, **Allow Nulls** da oluşturacağımız alanın, veri kaydı sırasında bu alanın boş bırakılıp bırakılmayacağını

ayarlamak için kullanılmaktadır. Ayrıca her alanla ilgili ayrıntılı ayarların yapılabilmesi için alt bölümdeki **Column Properties** kullanılmaktadır. Öncelikle her kayıt için oluşturacağımız `dosyaId` isimli alanı benzersiz ve otomatik artan sayı biçiminde belirlemeliyiz. Bu sayede kayıtları indekslerken, arama yaparken ve kayıt çağırırken veritabanı içerisinde karışıklık olmasını engellemiş oluruz. Alanımızın birincil alan olduğunu belirtmek için anahtar simgesine tıklıyoruz.

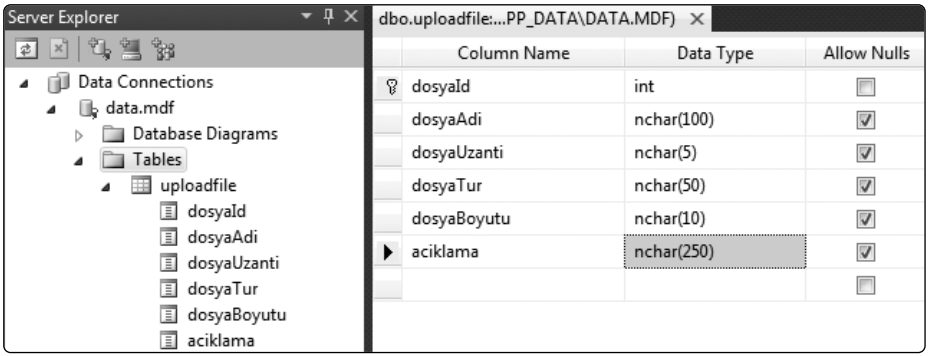
Alanımızın veri türünü int olarak belirttikten sonra, alt bölümde açılan **Column Options** alanından alanla ilgili ayarları yapıyoruz. Burada alanımızın otomatik sayı olarak artması için **Identity Specification** bölümünü açıp **Is Identity** özelliğini **Yes** yapıyoruz. Bundan sonra her kayıt için `dosyaId` alanının değeri otomatik olarak artacaktır.



Diğer alanların özelliklerini de belirtmek için aşağıdaki adımları izliyoruz. Öncelikle `dosyaAdi` alanı oluşturmamız gerekiyor. Bu alanı oluştururken veri türü olarak nchar seçiyoruz. Bu veri türünü seçtikten sonra veri türünün uzunluğunu belirtmek için **Column Option** bölümünde **Length** özelliğini `100` yaparak bu alana girilebilecek verinin uzunluğunun maksimum `100` karakter olabileceğini belirtiyoruz.

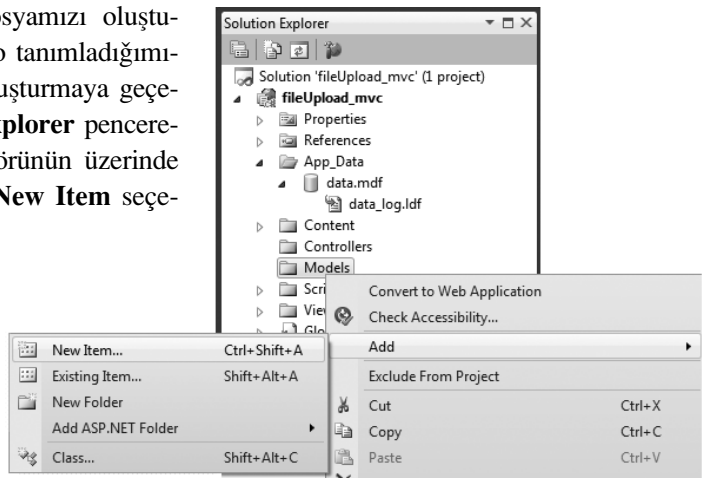


Diğer alanlar için de alan adlarını ve alanlara girilecek verilerin maksimum uzunluğunu belirttiğimizde veritabanı dosyamız içerisindeki alanlarımız aşağıdaki gibi olacaktır.

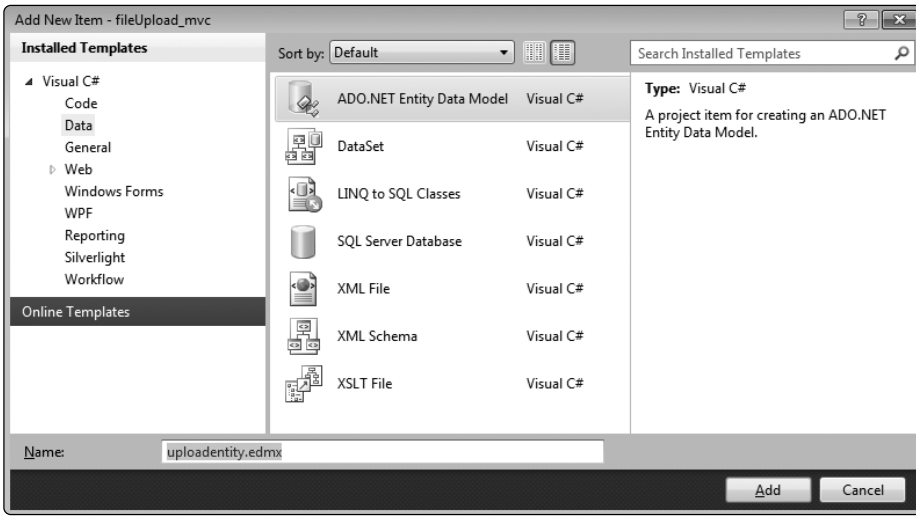


Tablomuzu **uploadfile** ismi ile kaydediyoruz. Oluşturduğumuz tabloda kullanıcı, dosya yüklerken dosyaya isim verebilecek ve açıklama ekleyebilecek. Dosyaya verdiği isim sadece veritabanında görünecek. Server tarafında `dosyaId` alanındaki otomatik sayı, dosyanın ismi olarak kaydedilecektir.

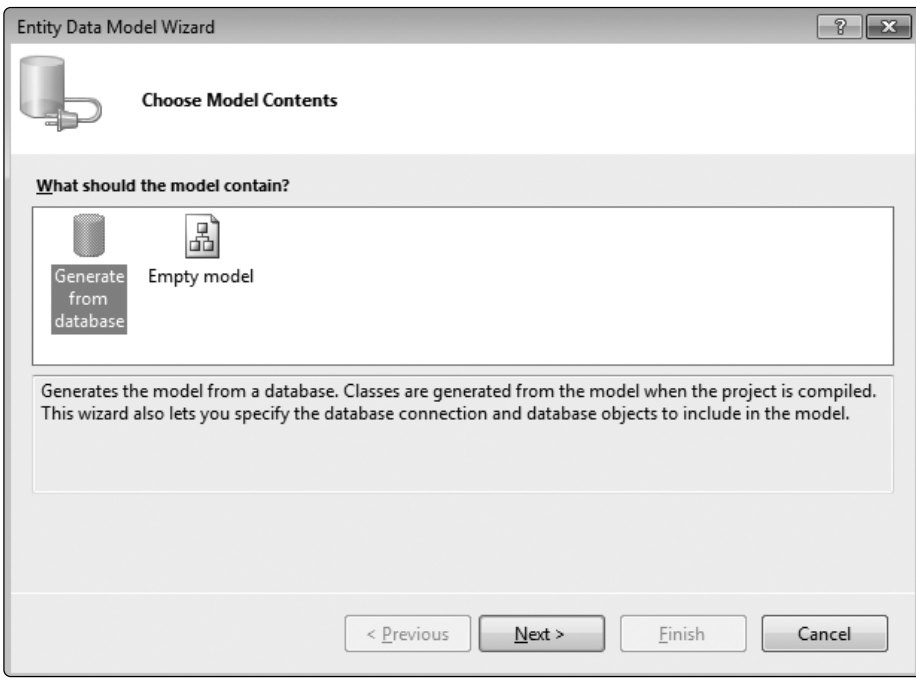
Şimdi database dosyamızı oluşturup, içerisinde tablo tanımladığımıza göre bağlantı oluşturmaya geçebiliriz. **Solution Explorer** penceresinde **Models** klasörünün üzerinde sağ tıklayıp **Add>New Item** seçeneğine tıklayalım.



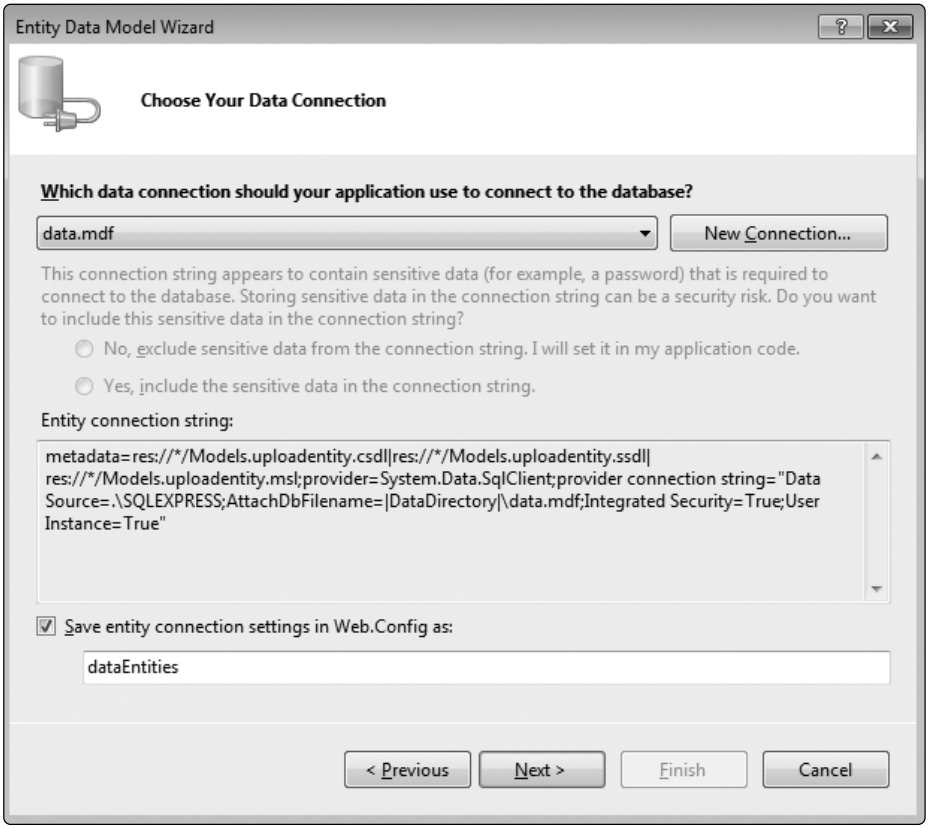
Daha sonra açılan pencereden **Data** kategorisinde bulunan **ADO.NET Entity Data Model**'i seçip **uploadentity.edmx** olarak isimlendiriyoruz.



Entitiy Data Model'imizi daha önce oluşturmuş olduğumuz veritabanından üreteceğimiz için **Generate from database** seçeneğini seçip **İleri** diyoruz.

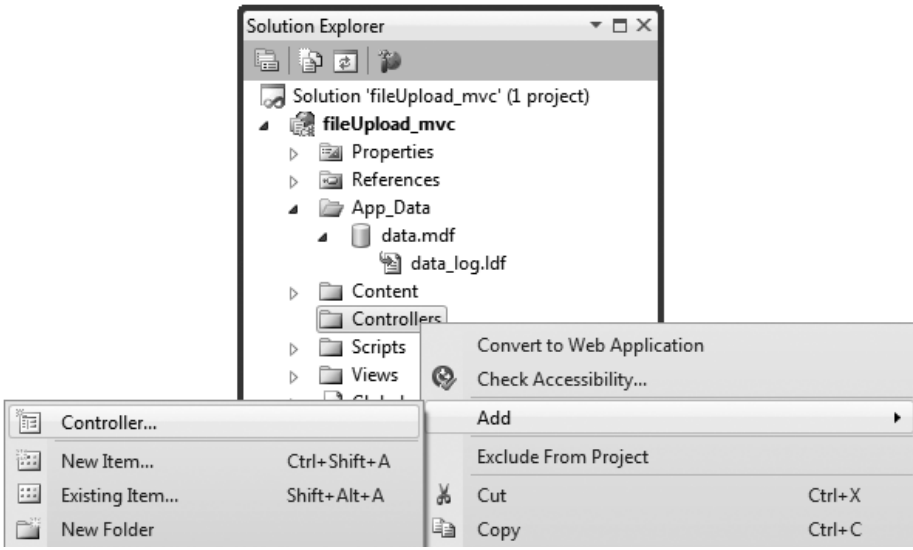


Bağlantı için oluşturduğumuz SQL veritabanı dosyasının yolunun bağlantı metnini aşağıdaki ekranda görüyoruz. Yine **Next** deyip devam ediyoruz.

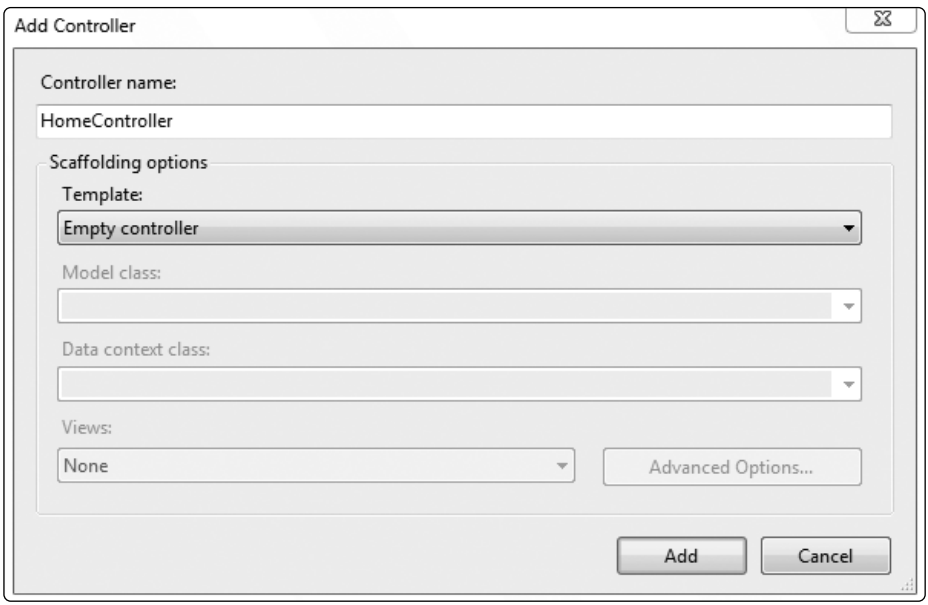


Bir sonraki ekranda veritabanı dosyamızla bağlantı kurulduktan sonra bu veritabanı dosyası içerisinde bulunan nesnelerin dizildiği ve seçmek için kutucukların bulunduğu ekran karşımıza çıkıyor. Bu ekranda veritabanı dosyası içerisindeki tablo nesnesini seçip **Finish** butonuna tıklıyoruz.

Bu işlemlerden sonra projemize Controller ekleyerek veritabanı işlemleri yapacağız. Bunun için **Solution Explorer** penceresi içerisinde bulunan klasörlerden **Controller** klasörünün üzerinde sağ tıklayarak **Add>Controller** komutunu tıklıyoruz.



Bu komuttan sonra karşımıza çıkan pencerede **Controller** için isim belirtiyoruz ve **Add action methods for create, update, delete, and details scenarios** seçeneğini seçili duruma getiriyoruz.



`HomeController.cs` dosyası içerisinde bulunan index metoduna aşağıdaki gibi kodlar ekliyoruz.

Burada tanımlama yaparken verdiğimiz **dataEntity**'i **Web.config** dosyasının içerisinde inceleyebiliriz.

Web.config | HomeController.cs | _ViewStart.cshtml

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.Web.Mvc;
6  using fileUpload_mvc.Models;
7  using System.Data.Objects.DataClasses;
8
9  namespace fileUpload_mvc.Controllers
10 {
11     public class HomeController : Controller
12     {
13         //
14         // GET: /Home/
15         private dataEntities upload_db=new dataEntities();
16         public ActionResult Index()
17         {
18             return View(upload_db.uploadfile.ToList());
19         }
20
21     }
22 }
```

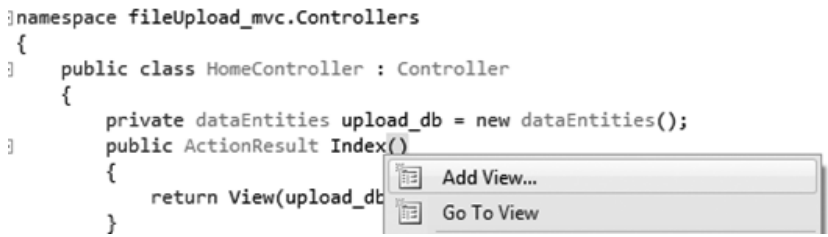
Eklediğimiz `HomeController.cs` dosyasının içerisine using olarak `fileupload_mvc.models`'i ekliyoruz. Daha sonra...

```
private dataEntities upload_db = new dataEntities();
```

... kodu ile `dataEntites` nesnesi ekliyoruz. Bu nesne ile veritabanımızdan veri işlemlerini yapabiliriz. **Index ActionResult** içerisine de veritabanındaki verileri listelemesi için aşağıdaki kodu yazıyoruz.

```
return View(upload_db.uploadfile.ToList());
```

Projemizin `Index` bölümünün **Controller** kodlarını tamamlamış olduk. Listeleme yapabilmek için şimdide bir **View** sayfası oluşturacağız. Bunun için **Index ActionResult** kodunun üzerinde sağ tıklayarak **Add>View** komutunu çalıştırıyoruz.



Karşımıza çıkan pencerede ayarları yapıyoruz.

`Index` isimli View otomatik olarak oluşacaktır. Bu noktadan sonra projemizi çalıştırdığımızda veritabanında bulunan `uploadfile` tablosundaki verileri aşağıdaki gibi sıralayacaktır.

Index

	dosyaId	dosyaAdi	dosyaUzanti	dosyaTur	dosyaBoyutu	aciklama
Edit | Details | Delete	1	tuana	a			a
Edit | Details | Delete	2	taha sami karakurt	jpg			yok

Create New

Bundan sonra ise yeni bir kayıt oluşturma sayfasını tasarlayıp, daha sonra bunun üzerinde değişiklikler yaparak dosya yükleme sayfası haline getireceğiz.

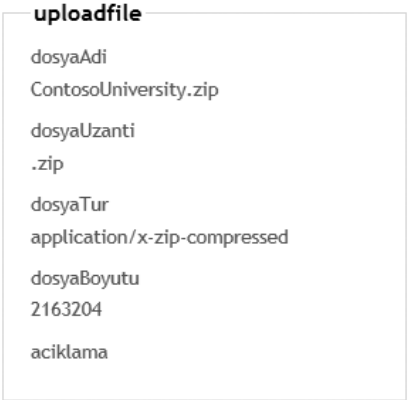
Ekleme işlemi için öncelikle dosyalarımızı upload edeceğimiz klasörü oluşturmamız gerekmektedir. Dosyalarımızı yüklemek için `App_Data` klasörü altında `UpLoads` adında klasör oluşturuyoruz. Öncelikle Controller içerisinde aşağıdaki Action metotlarımızı oluşturuyoruz.

```
public ActionResult Ekle()
{
    return View();
}
[HttpPost]
public ActionResult Ekle(HttpPostedFileBase file)
{
  if(file.ContentLength>0)
  {
    var fileName=Path.GetFileName(file.FileName);
    var path=Path.Combine(Server.MapPath("~/App_Data/UpLoads"),
```

```
                fileName);
        file.SaveAs(path);
        uploadfile Dosya=new uploadfile();
        Dosya.dosyaAdi=Path.GetFileName(file.FileName);
        Dosya.dosyaUzanti=Path.GetExtension(path);
        Dosya.dosyaBoyutu=file.ContentLength.ToString();
        Dosya.dosyaTur=file.ContentType;
        upload_db.uploadfile.AddObject(Dosya);
        upload_db.SaveChanges();
    }
    return View();
}
```

View sayfamız aşağıdaki gibi olmalıdır.

```
<h2>Ekle</h2>
<form action="" method="post" enctype="multipart/form-data">
    <label for="file">Filename:</label>
    <input type="file" name="file" id="file" />
    <input type="submit" />
</form>
```



Burada öncelikle dosyamızı upload ediyoruz ve ardından dosya bilgilerimizi veritabanına kaydediyoruz.

Silme işlemi için sayfamıza `ID` parametresini gönderiyoruz. İlk açılışta silinecek kaydın bilgilerini gösteriyoruz.

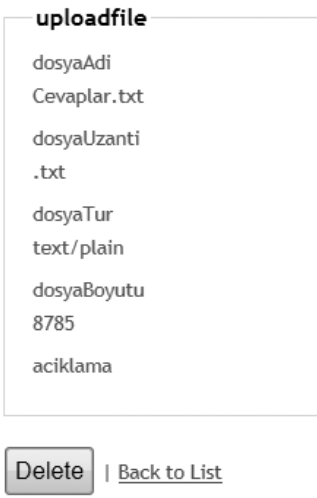
```
public ActionResult Delete(int id)
{
  return View(upload_db.uploadfile.ToList().
       Where(x=>x.dosyaId==id).SingleOrDefault());
}
```

Action üzerinde sağ tıklayarak **Add View** diyerek View'imizi oluşturuyoruz.

Sayfamızda butonumuza tıklandığında aşağıdaki **Action** çalışacak.

```
[HttpPost]
public ActionResult Delete(int id,FormCollection collect)
{
    uploadfile Dosya=new uploadfile();
    Dosya=upload_db.uploadfile.ToList().
            Where(x => x.dosyaId==id).SingleOrDefault();
    upload_db.uploadfile.DeleteObject(Dosya);
    upload_db.SaveChanges();
    return View("Index", upload_db.uploadfile.ToList());
}
```

Önce silinecek kaydımızı seçiyoruz, silme işlemini yapıp değişiklikleri kaydediyoruz. Son olarak yeni listeyi görmek için **Index View**'ini çağırıyoruz.



Kaydımızın detaylarını görmek için `Detay` adında bir Action oluşturuyoruz.

```
public ActionResult Detay(int id)
{
    return View(upload_db.uploadfile.ToList().
Where(x => x.dosyaId==id).SingleOrDefault());
       }
```

Bu Action'ımıza görüntülemek istediğimiz kaydın `ID`'sini parametre olarak gönderiyoruz. View'imiza bu kaydı gönderiyoruz. **Action** üzerinde sağ tıklayarak **Add View**'i seçip **Scaffolding Template** olarak **Detail** belirliyoruz.

uploadfile

dosyaAdi
ContosoUniversity.zip

dosyaUzanti
.zip

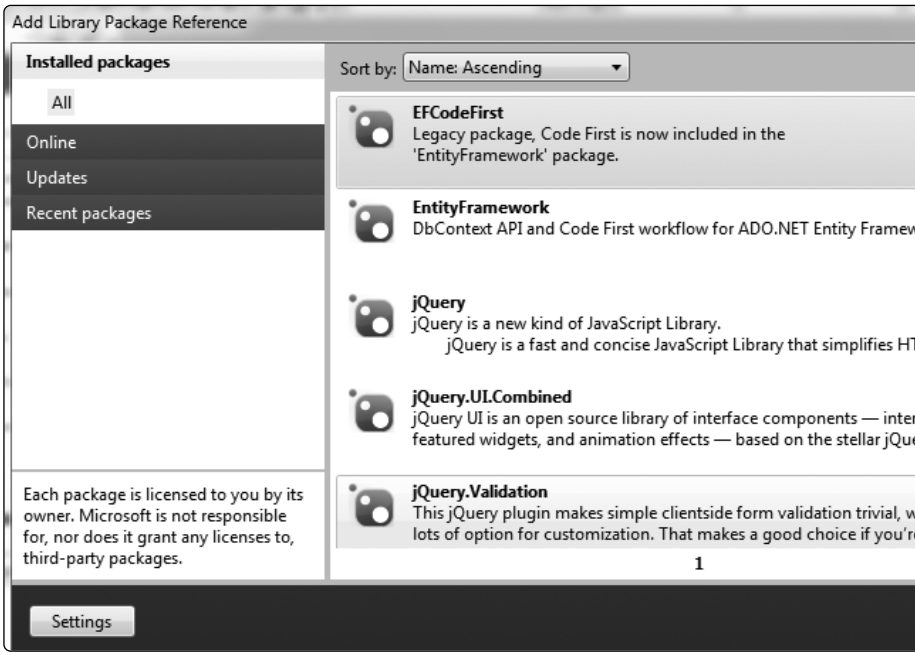
dosyaTur
application/x-zip-compressed

dosyaBoyutu
2163204

aciklama

Entity Framework ile Code First Modelleme

Daha önce de belirttiğimiz gibi `Code First` yapısı; `Database First` ve `Model First`'den farklı olarak veritabanı modelleme işlemini tasarım merkezli değil, kod merkezli yapmamızı sağlıyor. Projemizi oluşturduktan sonra `Nuget` ile projemize `EFCodeFirst`'i ekliyoruz.



`Code First` yapımızda modelimizi oluştururken entity sınıflarımız ile aradaki ilişkiyi, isimlendirme kuralları ile oluşturuyoruz. Örneğin; `ID` olarak alan yazdığımızda otomatik olarak o alan `Primary Key` olarak tanımlanır.

İnceleyeceğimiz örnekte `Kisi` ve `Yazi` adında tablolarımız olacak. `Kisi` tablomuz ile yazı tablomuz arasında bire-çoklu (*one to many*) ilişki olacak. Yani bir yazıyı yazan bir kişidir, ama bir kişinin birden fazla yazısı olabilir.

Şimdi model sınıfımızı oluşturalım. Bunun için öncelikle **Model** klasörümüz içerisine bir class oluşturuyoruz ve aşağıdaki namespace'lerimizi ekliyoruz:

```
using System.Data.Entity;
using System.Data.Entity.ModelConfiguration.Conventions;
using System.ComponentModel.DataAnnotations;
```

Modelimiz aşağıdaki gibi olacaktır:

```
public class Kisi
{
    public int KisiId { get; set; }
```

```
    public string Adi { get; set; }

    [DataType(DataType.EmailAddress)]
    public string EMail { get; set; }
    public virtual ICollection<Yazi> Yazis { get; set; }
}
public class Yazi
{
    public int YaziId { get; set; }
    [StringLength(150,ErrorMessage= 4-150 arası karakter girebilirsiniz",
       MinimumLength=4)]
    public string Baslik { get; set; }
    public string Content { get; set; }
    public int KisiId { get; set; }
    public DateTime Tarih { get; set; }
    public virtual Kisi Kisi { get; set; }
}
```

Belirttiğimiz tablolar arası ilişkiyi modelimizde `Kisi` class'ına yazdığımız:

```
public virtual ICollection<Yazi> Yazis { get; set; }
```

ve `Yazi` class'ına yazdığımız...

```
public virtual Kisi Kisi { get; set; }
```

... ifadeleri ile oluşturuyoruz.

`Code First` modelleme yönteminde veritabanımız yazdığımız model class'ına göre otomatik olarak oluşuyor. Bunun için `DbContext` sınıfından türeyen bir **Context Class** oluşturuyoruz.

```
public class OrnekContext:DbContext
{
  public DbSet<Kisi> Kisis { get; set; }
  public DbSet<Yazi> Yazis { get; set; }
  protected override void OnModelCreating(DbModelBuilder modelBuilder)
  {
      modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
  }
}
```

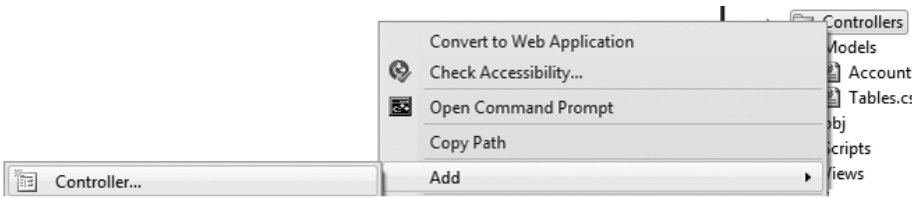
`OrnekContext` sınıfımızda veritabanımızda oluşacak tablolarımızı belirtiyoruz.

Veritabanımızın oluşması için herşey tamam. Fakat **Connection String** ifademiz yok. **Connection String** ifadesini **Web.config** içerisine yazarken name özelliğine **context** sınıfımızın adını veriyoruz. **Connection String** ifadesi aşağıdaki gibi olacaktır:

```
<connectionStrings>
    <add name="OrnekContext" connectionString = "Data Source =
|DataDirectory|Ornek.sdf" providerName="System.Data.SqlServerCe.4.0"/>
</connectionStrings>
```

Veritabanımız oluştuğunda **AppData** klasörü içerisinde `Ornek.sdf` şeklinde oluşacaktır.

Artık Controller'ımızı oluşturabiliriz. **Controller** klasörümüze sağ tıklayıp **Add Controller** seçeneğine tıklıyoruz.



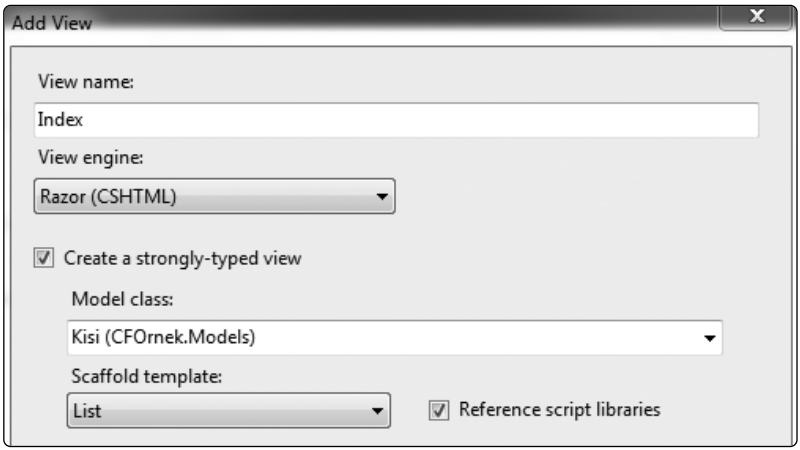
Controller'ımızın adını yazıp **Add** butonuna tıklıyoruz. Controller'ımızı oluşturduğumuzda öncelikle model için namespace'imizi ekleyip, ardından veritabanı işlemleri için oluşturduğumuz **Context** sınıfından nesne oluşturuyoruz.

```
using CFOrnek.Models;
OrnekContext db=new OrnekContext();
```

`Index View`'inde kişilerimizi listelemek için `Index ActionResult`'ını aşağıdaki gibi değiştiriyoruz:

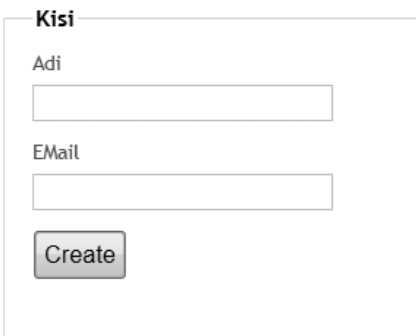
```
public ActionResult Index()
{
    return View(db.Kisis.ToList());
}
```

View'imiz `Index`, **Model Class**'ımız `Kisi`, **Scaffold Template** `List` olacak şekilde oluşturuyoruz.



Ardından kişi eklemek için sayfalarımızı oluşturalım ve ardından ilgili View'imizi oluşturalım.

```
public ActionResult Ekle()
{
    return View();
}
[HttpPost]
public ActionResult Ekle(FormCollection collect)
{
    Kisi kisi=new Kisi();
    kisi.Adi=Request.Form["Adi"];
    kisi.EMail=Request.Form["Email"];
    db.Kisis.Add(kisi);
    db.SaveChanges();
    return View();
}
```



Kişi işlemlerine burada ara verip yazı işlemlerine geçelim. Bunun için öncelikle bir Controller ekliyoruz. Öncelikle `Index View`'imiz de yazılarımızı listeleyip View'imizi oluşturalım.

```
public ActionResult Index()
{
    return View(db.Yazis.ToList());
}
```

View sayfamızda ufak bir değişiklik yapacağız. `Yazi` sınıfı içerisinde, `Kisi` sınıfının özelliklerini taşıyan bir değerimiz vardı. Burada kişinin adını yazdıracağımız için View'imizin ilgili kısmını aşağıdaki gibi düzenliyoruz:

```
<td>
    @Html.DisplayFor(modelItem => item.Baslik)
</td>
<td>
    @Html.DisplayFor(modelItem => item.Kisi.Adi)
</td>
<td>
    @Html.DisplayFor(modelItem => item.Tarih)
</td>
```

Baslik	Kisi	Tarih	
Deneme	Tahsin Sevinç	21.08.2011 23:43:41	Edit | Details | Delete
Ajax	Tahsin Sevinç	21.08.2011 23:44:02	Edit | Details | Delete
Code First	Said Yiğit	21.08.2011 23:44:18	Edit | Details | Delete

Yazı ekleme işlemi için öncelikle controller'ımıza **ActionResult**'ımızı yazıyoruz. Fakat ekleme işlemi içerisinde farklı olarak bazı değişiklikler yapacağız. Öncelikle tarih değerleri, controller sayfamızdan otomatik olarak yazılacak. Yani **View** kısmından girişi yapılmayacak. Ayrıca kişileri de **Combobox**'dan seçtirelim. Bunun için kişileri **viewbag** üzerinde bir yapıda tutacağız.

ActionResult'larımız aşağıdaki gibi olacak:

```
public ActionResult Ekle()
{
    IEnumerable<SelectListItem> items =
```

```
                db.Kisis.ToList().Select(c => new SelectListItem {
                Value=c.KisiId.ToString(),Text=c.Adi.ToString()});
      ViewBag.Kisiler = items;
      return View();
}
```

View'imizde kişilerin listelendiği **DropDownList**'i aşağıdaki gibi oluşturacağız:

```
<div class="editor-field">
     @Html.DropDownList("Kisiler",
                (IEnumerable<SelectListItem>)ViewBag.Kisiler)
     @Html.ValidationMessageFor(model => model.KisiId)
</div>
```

Ekleme işlemini yapan ActionResult'ımız:

```
[HttpPost]
public ActionResult Ekle(FormCollection collect)
{
     Yazi pYazi=new Yazi();
     pYazi.Baslik=Request.Form["Baslik"];
     pYazi.Content=Request.Form["Content"];
     pYazi.Tarih=DateTime.Now;
     pYazi.KisiId=Convert.ToInt32(Request.Form["Kisiler"]);
     db.Yazis.Add(pYazi);
     db.SaveChanges();
     IEnumerable<SelectListItem> items =
            db.Kisis.ToList().Select(c => new SelectListItem {
            Value=c.KisiId.ToString(), Text=c.Adi.ToString() });
     ViewBag.Kisiler = items;
     return View();
}
```

Son olarak kişilerin detay bilgilerinin yer alacağı View'imizi oluşturacağız. Fakat burada farklı olarak kişilerin yazdığı yazıları da sayfada listeleyelim. Bunun için kişi işlemlerini yaptığımız Controller'a aşağıdaki ActionResult'ı ekliyoruz.

```
public ActionResult Oku(string piID)
{
     Kisi kisi=db.Kisis.ToList().
            Where(x => x.KisiId==Convert.ToInt32(piID)).
```

```
            SingleOrDefault();
    return View(kisi);
}
```

View'imize yazıların listelenmesi için aşağıdaki gibi bir ekleme yapıyoruz:

```
<div class="display-label">
    Yazıları</div>
  <div class="display-field">
    <table>
      @foreach(var item in Model.Yazis)
      {
        <tr>
          <td>
            @Html.DisplayFor(modelItem => item.Baslik)
          </td>
          <td>
            @Html.DisplayFor(modelItem => item.Tarih)
          </td>
        </tr>
      }
    </table>
  </div>
```

Oku View'imizin çalışması için kişinin ID'sini parametre olarak gönderiyoruz. Örneğimizdeki 1 numaralı ID'ye sahip kullanıcının bilgileri aşağıdaki gibi görünecektir:

Kisi

Adi

Tahsin Sevinç

EMail

tsevinc@mail.com

Yazıları

Deneme	21.08.2011 23:43:41
Ajax	21.08.2011 23:44:02

MVC ROUTING 6

URL YÖNLENDİRME

URL, web sayfalarına ulaşmak ve web sayfaları içerisinde gezinti yapmak için kullanılan, protokolleri olan söz dizimidir. URL, aslında server'da bulunan fiziki bir dosyayı çağırmaktadır. Çağıracağı dosya, server içerisinde ana dizinde olabileceği gibi bazen daha alt dizinlerde de olabilir.

Örneğin; *http://www.domainname.com/dizinismi/dosya.aspx?parametre=değer*

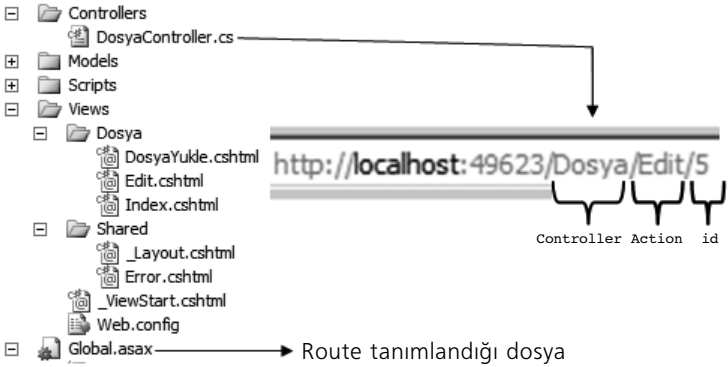
Bu şekildeki bir URL'yi incelediğimizde; server'da bulunan dosyanın fiziki yolunu, uzantısını ve aldığı parametreyi rahatlıkla görebiliriz. URL tanımlarken sayfanın içeriğine uygun, anlamlı ve kullanıcıların rahatlıkla hatırlayabileceği şekilde olmasına dikkat etmemiz gerekmektedir. Çünkü sayfa içeriğine uygun ve anlamlı kelimeler içeren bir URL, arama motorlarında kelimeler indekslenirken sayfanızı üst sıralara taşıyacaktır. URL'nin kısa ve anlamlı kelimelerden oluşması da kullanıcıların hatırında kalması açısından avantaj sağlayacaktır. Yukarıda vermiş olduğumuz yapıya sahip bir URL'yi, içeriğe uygun, anlaşılır ve kullanıcıların hatırında kalabilir şekilde yazmak biraz zordur. Büyük projelerde bu imkansız hale gelebilir. MVC projelerinde bu problem **URL Routing** sayesinde giderilmeye çalışılmıştır. **MVC Framework URL Routing**, web sitelerine istekte bulunan kullanıcıların isteklerini yönlendirmek için geliştirilmiş bir sistemdir. URL Routing'in yaptığı iş, kullanıcıdan gelen istekleri Controller içerisinde bulunan fonksiyonlara (*Action*) iletmektir. Her isteğe karşı gelecek eylemler için URL üretir. Bu yapıda gerçek bir URL yoktur. İstekler, **Route** tarafından ilgili yerlere bildirilir ve o isteklere karşı gelecek URL üretilir.

Diğer bir ifade ile sayfamızdan ne tür isteklerde bulunulabileceğini belirlemek için kullandığımız bir yapıdır. Bu yapıda gerçek bir URL olmadığı için sayfamıza bizim belirlediğimiz kurallar dışında ulaşmak mümkün değildir. Bu da web sitemizin güvenliği açısından önem arz eder.

Şimdi bir MVC uygulamasında Route yapısını inceleyelim. Route, MVC uygulamaları içerisinde **Global.asax.cs** dosyası içerisinde tanımlanmaktadır. Bir MVC uygulaması açtığımızda `global.asax` içerisinde varsayılan olarak aşağıdaki gibi bir kod yapısı bulunmaktadır.

```
public class MvcApplication : System.Web.HttpApplication
{
  public static void RegisterGlobalFilters(GlobalFilterCollection
         filters)
  {
       filters.Add(new HandleErrorAttribute());
  }
  public static void RegisterRoutes(RouteCollection routes)
  {
       routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
       routes.MapRoute(
            "Default", // Route ismi
            "{controller}/{action}/{id}", // URL'nin alacağı parametreler
            new { controller = "Personel", action = "Index",
            id = UrlParameter.Optional }, // Parametrelerin varsayılan değerleri
            new { controller = new NotEqual("Admin")} );
  }
  protected void Application_Start()
  {
       AreaRegistration.RegisterAllAreas();
       RegisterGlobalFilters(GlobalFilters.Filters);
       RegisterRoutes(RouteTable.Routes);
  }
}
```

`Global.asax.cs` dosyası içerisinde `default` olarak adlandırılmış yukarıdaki **MapRoute**'u incelediğimizde birinci bölümde controller adı, ikinci bölümde Controller içerisindeki Action ve üçüncü bölümde de `id` adında bir değişken bulunmaktadır. Yukarıdaki MapRoute'u kullanacak bir sitemizin olduğunu düşündüğümüzde internet tarayıcımızda görünecek URL ve sitemizin dizin yapısı aşağıdaki gibi olacaktır.



Kullanıcı bu URL ile istekte bulunduğunda aşağıdaki kod çalışacaktır.

```
DosyaYukle.Edit(5)
```

Global.asax.cs Dosyasının Varsayılan Kod Yapısı

Yeni bir MVC uygulaması açtığımızda `Global.asax.cs` dosyası içerisinde varsayılan değer olarak atamalar yapılmıştır. Bu varsayılan değerler **Home** ve **Index** olarak belirlenmiştir. Boş bir MVC uygulaması açtığımızda ve `Global.asax.cs` dosyası içerisindeki varsayılan `Home` ve `Index` değerleri dışında bir isimlendirmeyle Controller ve Action oluşturduğumuzda uygulamamızın düzgün çalışması ve sayfa bulunamadı hatası vermemesi için `Global.asax.cs` dosyasının içerisindeki varsayılan değerleri değiştirmemiz gerekmektedir.

Aşağıdaki şekilde `Global.asax.cs` dosyası ve `HomeController.cs` dosyalarının içerisinde bulunan değerler ve bu değerlerin neyi ifade ettiği görülmektedir.

Global.asax.cs Dosyasının içeriği

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute(
        "Default", // Route name
        "{controller}/{action}/{id}", // URL with parameters
        new { controller = "Home", action = "Index", id = UrlParameter.Optional }
    );
}
```

HomeController.cs Dosyasının içeriği

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        ViewBag.Message = "Welcome to ASP.NET MVC!";

        return View();
    }
}
```

Global.asax.cs	HomeController.cs
Home	`HomeController` adındaki Controller
Index	`HomeController` dosyası içerisindeki `Index()` Action

Şimdi bir MVC uygulaması açtığımızda varsayılan kodlarla gelen `Global.asax.cs` dosyasında tanımlanmış `MapRoute` metodunu ve varsayılan yapısı ile cevap verebileceği istekleri inceleyelim. `Routes`, `RouteCollection` sınıfının bir üyesidir. `MapRoute` metodu da web sitemize gelecek istekleri yönlendirecek URL modelinin parametrelerini ve parametrelerin hangi Controller ve Action'a işaret ettiği bilgisinin tanımlandığı metottur. Bu metodu daha sonra "Route Tanımlama" başlığı altında detaylıca inceleyeceğiz. Şimdi varsayılan yapısıyla bu metodun cevap verebileceği isteklere bakalım.

```
routes.MapRoute(
     "Default", // Route ismi
     "{controller}/{action}/{id}", // URL'nin alacağı parametreler
           new { controller = "Home", action = "Index",
```

```
            id = UrlParameter.Optional }
);
```

Yukarıdaki `MapRoute` yapısına sahip bir web sitemizin olduğunu ve bu web sitesinde bir personel sayfamızın olduğunu var sayarsak, gelen istekleri aşağıdaki gibi yönlendirecektir.

İstek	URL	Controller	Action Metodu	id
Sayfaya Giriş	/	`HomeController`	`Index()`	`""`
Personelimiz Sayfası	/Personel	`PersonelController`	`Index()`	`""`
İdari Personel Sayfası	/Personel/Yonetim	`PersonelController`	`Yonetim()`	`""`
Kurum Müdürü	/Personel/Ozgecmis/1	`PersonelController`	`Ozgecmis()`	`"1"`

Yukarıdaki tabloyu incelediğimizde bir URL modeli için oluşturulan parametreler `/` karakteri ile ayrılmıştır.

Route Tanimlama

Bir MVC uygulamasına URL ile istek geldiği zaman MVC Framework, `Global.asax.cs` içerisindeki yönlendirme kuralları ile karşılaştırma yaparak en uygun yönlendirme kuralını (`MapRoute`) seçer. Seçtiği yönlendirme kuralı (`MapRoute`) içerisinde tanımlanmış olan Controller ve Action'a yönlendirir. MVC uygulamasında varsayılan olarak tanımlanan yönlendirme kuralı (`MapRoute`) her zaman ihtiyaçlarımızı karşılamayabilir. Bu durumda kendi yönlendirme kurallarımızı tanımlamamız gerekir. Yönlendirme kurallarının (`MapRoute`) daha iyi anlaşılması için bu noktadan sonra basit bir haber portalı uygulaması üzerinden anlatmaya çalışacağız.

Haber portalımızın genel yapısı aşağıda görüldüğü gibi olacaktır.



Haber portalımızda `Kategori/id` şeklindeki url bağlantısına tıklandığında ilgili kategoride bulunan haberler listelenecek ve `/id` şeklindeki haberin devamı linkine tıklandığında ilgili haberin detayları gösterilecektir.

Haber portalımızda `HaberController` adı ile oluşturduğumuz Controller'ın kod yapısı da aşağıdaki gibi olacaktır.

HaberController.cs

```
HaberPortal.Controllers.HaberController                    db
namespace HaberPortal.Controllers
{
    public class HaberController : Controller
    {
        DataHaberEntities db = new DataHaberEntities();
        public ActionResult Index()...     //Sitemizin Anasayfası
        public ActionResult Detay(int id)... //Haberlerin detaylarının çağrıldığı action
        public ActionResult Kategori(int kid) ... //Haberleri kategorilerine göre listeleyen action
        public ActionResult Ara(string ara) ... //Haberler içerisinde arama yapan action
        public ActionResult Arsiv(string tarih) ... // Tarihlere göre haberleri listeleyen action
        public ActionResult Ekle()...  //yeni haber eklemek için kullanılacak form action
        [ValidateInput(false)]
        [AcceptVerbs(HttpVerbs.Post)]
        public ActionResult Ekle(HttpPostedFileBase haberform)...// yeni haberi kaydetmek için kullanılacak action
    }
}
100 %
```

Haber portalımızın `HaberController.cs` dosyası içerisindeki Action metotlarını incelediğimizde ihtiyacımız olan URL yapıları aşağıdaki gibi olacaktır.

Tetiklenecek Action	Yapacağı İş	URL İstekleri
Haber/index	Anasayfayı çağıracak.	/
Haber/Detay/id	Haberin detayını `id` parametresi ile çağırıyor.	/12
Haber/Kategori/kid	Haberleri kategori `kid` parametresine göre çağırıyor.	/Kategori/3
Haber/Ara/aranacak metin	Haberler içerisinde arama yapacak.	/Ara/aranacak metin
Haber/Arsiv/Tarih	Haberleri tarihine göre çağıracak.	/Arsiv/03.04.2011
Haber/Ekle	Haber Eklemek için kullanılacak.	/Haber/Ekle

Yukarıdaki gibi gelen URL isteklerini aşağıdaki gibi tanımlanmış olan **default mapRoute** yanlış yönlendirecektir.

```
routes.MapRoute(
     "Default", // Route ismi
     "{controller}/{action}/{id}", // URL'nin alacağı parametreler
     new { controller = "Home", action = "Index",
     id = UrlParameter.Optional }
);
```

Örneğin; haberleri kategorilerine göre çağırmak için...

```
Kategori/3
```

... şeklinde bir url isteğinde bulunduğumuzda **default MapRoute**, **Kategori**'yi Controller olarak, üç parametresini de Action olarak algılayacaktır. Bu durumda sayfada hata oluşacaktır. Sitemize gelen istekleri aşağıdaki gibi sırayla `MapRoute` tanımlayarak oluşturabiliriz. Örneğin; sitemize giren kullanıcı adres çubuğuna `/Arsiv/03.04.2011` şeklinde bir istekte bulunduğunda...

```
Action: Arsiv
Parametre: 03.04.2011
```

... şeklinde olacak.

Controller'ın ise haber olarak algılanması için tanımlayacağımız `MapRoute` aşağıdaki gibi olacaktır.

```
routes.MapRoute(
     "ArsivRoute", // Route ismi
     "Arsiv/{tarih}", // URL'nin alacağı parametreler
     new { controller = "Haber", action = "Arsiv",
     tarih = "" } // Varsayılan parametreler
);
```

Yukarıda tanımlanan `MapRoute`, `Arsiv/03.04.2011` URL isteği geldiği zaman Controller olarak `HaberController`, Action olarak `Arsiv` ve `03.04.2011` tarihini de `Arsiv` Action'a gönderilen parametre olarak algılayacaktır.

Route Tanımlarken Dikkat Edilmesi Gereken Kurallar

Route tanımlarken kullanacağımız metod `MapRoute` metodudur. `MapRoute()` metodunun alabileceği parametreler şöyledir;

1. Tanımlayacağımız Route ismi.
2. URL kalıbı.
3. Route parametrelerinin varsayılan değerleri.
4. Route ile eşleşecek URL isteklerine eklenecek kısıtlamalar.
5. `IRouteConstraint` isim uzayından türetilen sınıf ile oluşturulmuş nesnenin `Match` metoduna gelen parametrelerin eşleşmesi sonucu, geriye `true` değerini döndürür ve Route eşleşir.

Route tanımlarken, öncelikle tanımladığımız route, `Global.asax.cs` içerisindeki **Default** olarak isimlendirilmiş Route'tan önce gelmelidir. Haber portalı örneğimiz üzerinden inceleyecek olursak; `ArsivRoute` olarak tanımladığımız yönlendirme kuralı, `Global.asax.cs` dosyasındaki Default olarak tanımlanmış yönlendirme kuralından önce gelmelidir.

```
routes.MapRoute(
     "Default", // Route ismi
     "{controller}/{action}/{id}", // URL'nin alacağı parametreler
     new { controller = "Home", action = "Index",
        id = UrlParameter.Optional }
```

```
);
routes.MapRoute(
     "ArsivRoute", // Route ismi
     "Arsiv/{tarih}", // URL'nin alacağı parametreler
     new { controller = "Haber", action = "Arsiv",
     tarih = "" } // Varsayılan parametreler
);
```

Sıralama yukarıdaki gibi olursa yani **Default** olarak tanımlanan route, `ArsivRoute` olarak tanımlanmış yönlendirme kuralından önce gelirse **Default Route** çalışır, `ArsivRoute` çalışmaz ve sayfa hata verir.

Route tanımlarken dikkat etmemiz gereken diğer bir kural da şudur: Küme parantezi ile belirlediğimiz parametreler arka arkaya tanımlanamaz.

```
{Kategori}{id}
```

Yukarıdaki gibi tanımlarsak hatalı tanımlamış oluruz. İki parametre arasında başka karakter veya karakterler olmalıdır.

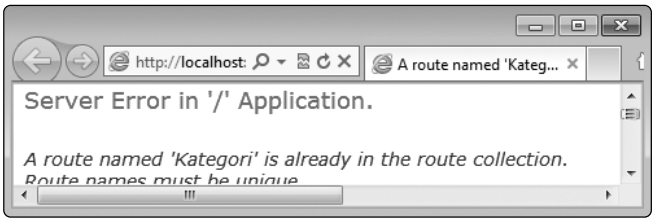
```
{Kategori}/{id}
{Kategori}.{id}
{Kategori}haber{id}
{Kategori}-{Guncel}/{id}
```

Yukarıda tanımladığımız Route URL yapıları sorunsuz çalışacaktır.

Route tanımlamasında dikkat edilmesi gereken diğer kural da, tanımlanan Route ismi benzersiz olmalıdır. Aşağıdaki gibi tanımlanan Route ismi tekrarlanırsa...

```
routes.MapRoute(
     "Kategori", // Route ismi
     "Detay/{id}",
     new { controller = "Haber", action = "Detay", id = "" }
);
routes.MapRoute(
     "Kategori", // Route ismi
     "kategori/{kid}",
     new { controller = "Haber", action = "Kategori", kid = "" }
);
```

... *"Route ismi benzersiz olmalıdır"* hata mesajı verecektir.



Route Tanımlama Örnekleri

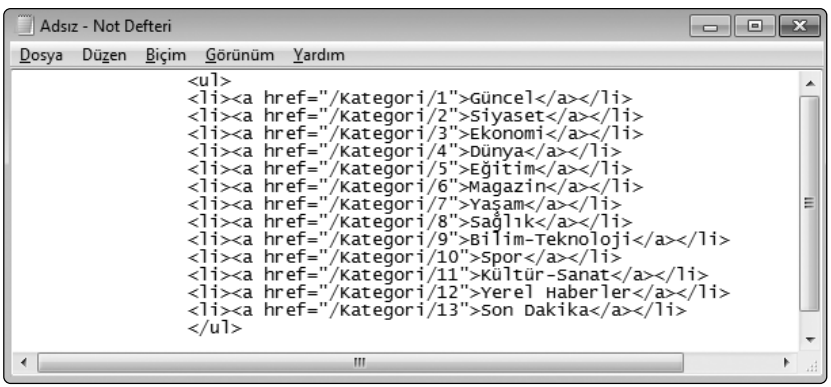
Route tanımlamayı daha iyi anlayabilmek için birkaç tane Route tanımlayalım. Haber portalımızda bulunan haberleri kategorilerine göre çağırmak için aşağıdaki tabloda olduğu gibi bir URL isteği oluşturabiliriz.

Tetiklenecek Action	Yapacağı İş	URL İsteği
Haber/Kategori/kid	Haberleri kategori `kid` parametresine göre çağırıyor.	/Kategori/3

Tablodaki URL isteğini incelediğimizde, `HaberController` içerisinde bulunan `Kategori` Action metoduna gönderilen parametre değerine göre, ilgili kategorideki haberler sıralanacaktır. Bunun için tanımlayacağımız Route aşağıdaki gibi olmalıdır.

```
routes.MapRoute(
    “Kategori”,
    “kategori/{kid}”,
    new { controller = “Haber”, action = “Kategori”, kid = “” }
);
```

Görüldüğü gibi tanımlamış olduğumuz Route, Controller olarak `HaberController`, Action olarak `Kategori` ve gelen parametreyi de `Kategori` Action metoduna `kid` değişkeniyle aktarmaktadır. Kategori linkleri de aşağıdaki gibi olacaktır.



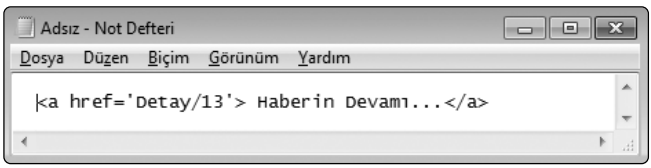
Haberlerin detayları için aşağıdaki tabloda olduğu gibi bir URL isteği oluşturabiliriz.

Tetiklenecek Action	Yapacağı İş	URL İsteği
Haber/Detay/id	Haberin detayını `id` parametresi ile çağırıyor.	/12

Tablodaki URL isteğini incelediğimizde, `HaberController` içerisinde bulunan `Detay` Action metoduna gönderilen parametre değerine göre ilgili haberlerin detayı gösterilecektir. Bunun için tanımlayacağımız Route aşağıdaki gibi olmalıdır.

```
routes.MapRoute(
     “Detay”,
     “Detay/{id}”,
     new { controller = “Haber”, action = “Detay”, id = “” }
);
```

Oluşturduğumuz `Detay` isimli route'da görüldüğü gibi Controller olarak `HaberController`, Action olarak `Detay` ve gelen parametreyi de `Detay` Action metoduna `id` değişkeniyle aktarmaktadır. Detay linki de aşağıdaki gibi olacaktır.



Route İçin Kısıtlamalar Oluşturma

Route tanımlarken gelen URL isteklerindeki parametrelere belli kısıtlamalar ekleyebilirsiniz. Kısıtlama eklediğiniz zaman, kısıtlama dışında kalan istekler karşılanmayacak ve sayfada hata oluşmayacaktır. Örneğin; haber portalımızda haberlerin detaylarını çağırırken ilgili haberin `id` numarasını kullanıyoruz. Dolayısıyla `id` parametresinin değeri sayısal olmalıdır. `Detay/ekonomi` şeklinde bir URL isteği geldiğinde sayfada hata oluşacaktır. Detay sayfasını açarken URL isteği `/Detay` şeklinde parametresiz de gelmiş olabilir. Bu durumda kısıtlama eklememişsek Route eşleşecektir ve `Detay` Action metoduna parametresiz gittiği için sayfamızda hata oluşacaktır. Sayfamızda hatalar oluşmaması için URL isteklerinin parametresiz gelmemesi ve gelen parametrelerin istediğimiz gibi gelmesi için parametrelere kısıtlamalar eklemeliyiz. Kısıtlamaları üç temel yöntemle ekleyebiliriz.

Bunlar;

- Düzenli ifadelerle kısıtlama ekleme,
- Http metoduyla kısıtlama ekleme,
- Özel kısıtlamalar eklemedir.

Düzenli İfadeler Kullanarak Kısıtlama Oluşturma

Regular Expressions (*Düzenli İfadeler*), bir string ya da string dizisi içerisinden çeşitli ifade kalıplarını kullanarak bu kalıplara uyan yeni string ya da string dizileri elde etmek amacı ile kullanılır.

Düzenli ifadeler ile birlikte kullanılan ve arama, karşılaştırma ve değiştirme işlemlerinde kullanılan karakterler mevcuttur. Bunlara **metacharacter** adı verilir. Bu karakterlerin başlıca olanlarını aşağıdaki gibi sıralayabiliriz:

$: Bir metnin bitiş pozisyonunu belirtir. Multiline özelliği aktif edildiğinde bir satırın bitiş pozisyonunu ifade eder.

(): Bir alt ifadenin başlangıç ve bitişine işaret eder.

***:** Kendisinden önce belirtilen karakterin belirtilen metin içerisinde ya hiç olmadığını, ya da bir veya birden fazla mevcut olduğunu belirtir.

+: Kendisinden önce belirtilen karakterin veya alt ifadenin bir veya daha fazla defa tekrarlanabileceğini belirtir.

[]: Bir aralığın başlangıcını belirtir.

?: Kendisinden önce belirtilen karakterin belirtilen metin içerisinde ya hiç olmadığını, ya da mevcut olduğunu belirtir.

\: Kendisinden sonra belirtilen karakterin arama kriterini oluşturan ifade içerisine dahil edilmesini sağlar.

/: Bir düzenli ifadenin başlangıcını veya bitişini belirtir.

^: Karşılaştırma işleminin başlayacağı ilk karakteri belirtir.

{}: Kendisinden önce gelen karakterin ne kadar tekrar edileceğini belirtir.

"D" veya **"d":** Decimal

Gelen URL istekleri parametresiz geldiği zaman, tanımladığımız Route ile eşleşmemesini istiyorsak, düzenli ifadeleri kullanabiliriz.

Örneğin; aşağıdaki gibi haber portalımızda tanımladığımız bir Route olsun.

```
routes.MapRoute(
     "ArsivRoute", //Route ismi
     "Arsiv/{tarih}", //URL parametresi
     new { controller = "Haber", action = "Arsiv",
     tarih = "" }      // Varsayılan parametreler
);
```

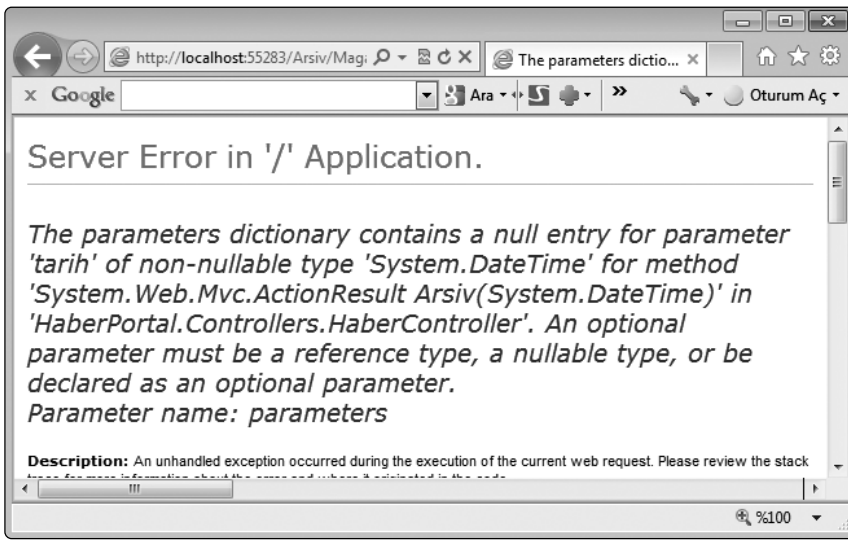
Tanımladığımız `ArsivRoute` aşağıdaki URL isteği ile eşleşecek ve sayfada hata oluşmayacaktır.

Tetiklenecek Action	URL İsteği	Yapacağı İş
Haber/Arsiv(tarih)	Arsiv/03.04.2011	03.04.2011 tarihli haberleri getirir.

Ancak `ArsivRoute` aşağıdaki URL istekleri ile de eşleşecektir.

Tetiklenecek Action	URL İsteği	Yapacağı İş
Haber/Arsiv(tarih)	Arsiv/magazin	Hata oluşacaktır.
Haber/Arsiv(tarih)	Arsiv/	Hata oluşacaktır.

Yukarıdaki gibi gelen URL isteklerinde parametresi tarih olmayan veya boş olan değerleri kısıtlayacak hiçbir şey yoktur. Sayfada aşağıdaki geçersiz tarih hata mesajı oluşacaktır.



Bu durumda sayfamızın tarih hata mesajı vermemesi için gelen parametrenin tarih türünden olması gerekir. Dolayısıyla bizim geçersiz tarih parametrelerini ve boş değerle gelmiş parametreleri kısıtlamamız gerekir. Bu kısıtlamayı **Regular Expressions** kullanarak yapabiliriz. `ArsivRoute` olarak tanımladığımız yönlendirme kuralına aşağıdaki kodu ekleyeceğiz.

```
new { tarih = @"\d{2}.\d{2}.\d{4}" }
```

Eklediğimiz bu kodu incelediğimizde tarih parametresi `01.01.0001` formatında olmalıdır. Son durumda `ArsivRoute` yönlendirme kuralımız aşağıdaki gibi olacaktır.

```
routes.MapRoute(
     "ArsivRoute", //Route ismi
     "Arsiv/{tarih}", //URL parametresi
     new { controller = "Haber", action = "Arsiv",
     tarih = "" }, // Varsayılan parametreler
     new { tarih = @"\d{2}.\d{2}.\d{4}" }
);
```

`ArsivRoute` yönlendirme kuralının yeni halini incelediğimizde gelen `Arsiv/URL` isteğinden sonra gelecek tarih parametresi iki tane rakam bir tane nokta sonra tekrar iki tane rakam, bir tane nokta ve dört tane rakam (`01.01.0001`) olduğu zaman `ArsivRoute` ile eşleşecektir.

Düzenli ifadeleri kullandığımıza göre haber portalımızı biraz daha geliştirebiliriz. Haber portalına giren kullanıcının adres çubuğuna `Arsiv/` yazmadan `/03.04.2011` şeklinde tarih girerek ilgili tarihteki haberleri sıralamasını istiyorsak `ArsivRoute` yönlendirme kuralımız aşağıdaki gibi olmalıdır.

```
routes.MapRoute(
     "ArsivRoute", //Route ismi
     "{tarih}", //URL parametresi
     new { controller = "Haber", action = "Arsiv",
     tarih = "" }, // Varsayılan parametreler
     new { tarih = @"\d{2}.\d{2}.\d{4}" }
);
```

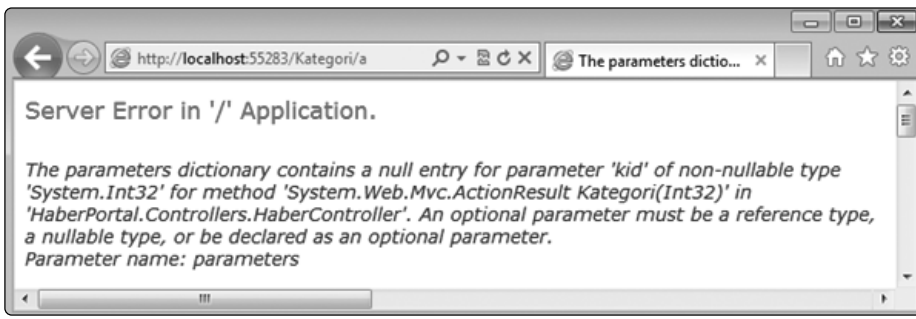
`ArsivRoute` yönlendirme kuralımızın yeni yapısını incelediğimizde ikinci parametre `"Arsiv/{tarih}"` yerine `"{tarih}"` olarak değiştirilmiştir. Bu durumda kullanıcı tarayıcının adres çubuğunda aşağıdaki gibi istekte bulunabilir.

```
siteadı/03.04.2011
```

Şimdi haber portalı projemizin `global.asax.cs` dosyası içerisinde `"Kategori"` ve `"Detay"` olarak isimlendirdiğimiz yönlendirme kurallarını inceleyelim.

```
routes.MapRoute(
     "Detay", // Route ismi
     "Detay/{id}",
     new { controller = "Haber", action = "Detay", id = "" }
);
routes.MapRoute(
     "Kategori", // Route ismi
     "kategori/{kid}",
     new { controller = "Haber", action = "Kategori", kid = "" }
);
```

Tanımladığımız bu iki route, `Kategori/` ve `Detay/` URL istekleri ve bunlardan sonra gelebilecek tüm parametrelerle eşleşecektir. Dolayısıyla `id` ve `kid` parametrelerinin rakam olmadığı durumlarda aşağıdaki gibi tür uyumsuz hata mesajı vereceklerdir.



`Kategori Route` ve `Detay Route` ile eşleşen, ama rakamdan oluşmayan URL isteklerini engellememiz gerekiyor. Yine bunu engellemek için Route tanımlarken düzeli ifadelerle kısıtlamamız gerekiyor. Aşağıdaki gibi değiştirdiğimiz `Kategori Route` ve `Detay Route` işimizi görecektir.

```
routes.MapRoute(
     "Detay", // Route ismi
     "{id}",
     new { controller = "Haber", action = "Detay", id = "" },
     new { id = @"\d+" }
);
```

```
routes.MapRoute(
     “Kategori”, // Route ismi
     “kategori/{kid}”,
     new { controller = “Haber”, action = “Kategori”, kid = “” },
     new { kid = @”\d+” }
);
```

Kategori Route ve Detay Route tanımlarken kullandığımız dördüncü parametre düzenli ifade oluyor. Bu ifadelere göre Kategori/ ve / URL isteklerinden sonra gelecek id ve kid parametreleri rakamlardan oluşacak, bir ve birden fazla basamaklı olabilecektir. Detay olarak tanımladığımız yönlendirme kuralının yeni haline göre haberlerin detaylarını çağırırken Detay/12 şeklinde değil de /12 şeklinde çağırmamız yeterli olacaktır.

Http Metodu Kullanarak Kısıtlama Oluşturma

HttpMethodConstraint sınıfını kullanarak gelen URL isteklerini kısıtlayabiliriz. Örneğin; gelen URL isteklerinin sadece **http** metodunun **post** olduğu durumlarda karşılaştırma yapmasını istiyorsak, bu yöntemi kullanabiliriz. **Http Metod Kısıtlayıcıları** kullanmak için URL isteğinin eşleşeceği route'un http method özelliğini belirlememiz gerekir.

Haber portalımıza yeni haber eklemek için kullanacağımız route'un sadece http method post olduğu zaman eşleşeceği Route aşağıdaki gibi olmalıdır.

```
routes.MapRoute(
     “HaberEkle”,
     “Haber/Ekle”,
     new { controller = “Haber”, action = “Ekle” },
     new { method = new HttpMethodConstraint(“POST”) }
);
```

HaberEkle Route'u incelediğimizde dördüncü parametre olarak...

```
new { method=new HttpMethodConstraint(“POST”) }
```

... eklenmiştir. Bu eklediğimiz kod, gelen URL isteğinin http metodu post olduğu durumlarda eşleşeceği anlamına gelmektedir.

Bu kısıtlama yöntemiyle ilgili değinmemiz gereken bir husus da bu yöntemi kullanmaksızın oluşturduğumuz action metodunun başına [AcceptVerbs(HttpVerbs.Post)]

ekleyerek kullanılacak `http` metodunu belirleyebiliriz. Bu sayede tanımladığımız Route `httpmethod` kısıtlaması kullanmamıza gerek kalmaz.

Özel Kısıtlamalar Oluşturma

Gelişmiş bir Route kısıtlama oluşturmak için veya kullanacağımız özel Route kısıtlamaları için en ideal kısıtlama yöntemidir. Bu yöntemle `IRouteConstraint` isimli `Interface`'den sınıf türetiyoruz. Oluşturduğumuz sınıfa ekleyeceğimiz özel kısıtlama kodlarını yazıyoruz. Ve Route tanımlarken, Route içerisine bu sınıftan türemiş nesneyi ekliyoruz. Eklediğimiz Route içerisindeki nesne, gelen url isteklerine göre `true` veya `false` değer döndürecektir. Gelen değer `false` olduğu taktirde Route eşleşmeyecektir. `true` olduğu zaman eşleşecektir. `IRouteConstraint` interface'den türettiğimiz sınıf içerisinde nasıl kısıtlamalar oluşturabiliriz? Bunu incelediğimizde `IRouteConstraint` interface'inde bulunan `Match` metodundan yararlanabiliriz. Bu metodun aldığı değerler aşağıda sıralanmıştır.

```
bool Match(
       HttpContextBase httpContext,
       Route route,
       string parameterName,
       RouteValueDictionary values,
       RouteDirection routeDirection
)
```

`IRouteConstraint` interface'den türeteceğimiz sınıfın içerisinde bulunan `Match` metodunun döndüreceği değer `bool` türündendir. Bu sınıftan türetilecek nesne, Route içerisinde eşleşeceği URL isteklerinin özelliklerini Match metoduyla taşır. Bu özellikleri kısaca açıklayacak olursak;

- **`HttpContextBase httpContext:`** URL isteğinin Http bilgilerini barındırır.
- **`Route route:`** URL isteğinin eşleşip eşleşmediğini kontrol eden yönlendirme kurallının özelliklerini barındırıyor.
- **`string parameterName:`** Parametre isimlerini kontrol ediyor.
- **`RouteValueDictionary values:`** Route'un içerdiği parametrelerin alacağı değerleri barındırıyor.
- **`RouteDirection routeDirection:`** Bir URL isteği oluştuğunda veya bir istek yakalandığı zaman kısıtlamaların kontrolünün olup olmadığını gösteriyor.

Şimdi haber portalı projemiz üzerinden bir örnek gösterecek olursak, haber portalımızın `Arsiv` isimli Action'a ulaşması için daha önceden oluşturduğumuz route'u değiştirelim ve yeni oluşturduğumuz route'a gelen URL isteklerini daha etkin bir şekilde yakalasın. Aşağıda oluşturduğumuz Route tarih formatı `gg.aa.yyyy` şeklinde olan istekleri karşılamaktadır.

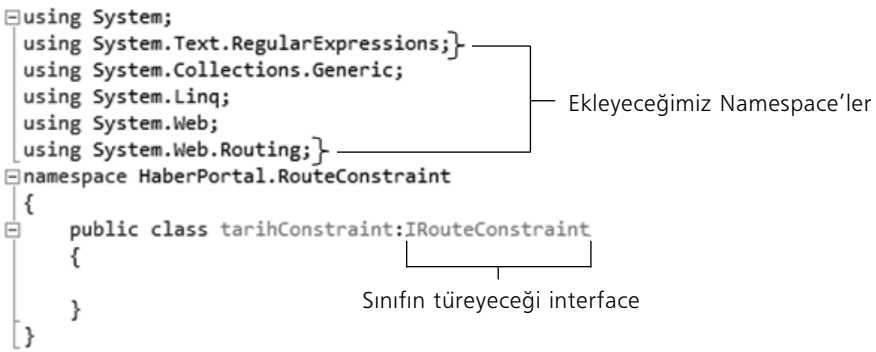
```
routes.MapRoute(
     "ArsivRoute", // Route ismi
     "{tarih}",
     new { controller = "Haber", action = "Arsiv",
     tarih = "" }, // Varsayılan parametreler
     new { tarih = @"\d{2}.\d{2}.\d{4}" }
);
```

Yeni oluşturacağımız Route sayesinde tarih formatı `gg.aa.yyyy` ve `gg-aa-yyyy` şeklinde olan isteklerle de eşleşecektir.



Öncelikle uygulamamıza yeni bir klasör ekleyip ismini `RouteConstraint` olarak belirliyoruz ve bu klasöre `tarihConstraint.cs` isminde bir sınıf ekliyoruz.

Eklediğimiz sınıfı açıp aşağıdaki değişiklikleri yapıyoruz.



Düzenlemeleri yaptıktan sonra aşağıdaki `Match` metodunu ekliyoruz ve sınıfımızın içerisinde kullanacağımız `Match` metodunun parametrelerini tanımlıyoruz. Tanımladığımız parametrelerden...

`System.Web.Routing.RouteValueDictionary`

... özelliğini kullanarak gelen tarih parametresinin değerini elde edeceğiz. Elde ettiğimiz tarih parametresinin değerini dizi içerisinde düzenli ifadelerle tanımladığımız tarih formatı desenleri ile karşılaştıracağız. Gelen tarih parametresinin formatı düzenli ifadelerle eşleştiği zaman `true` değeri döndürecektir ve Route eşleşecektir. Aksi takdirde `false` değeri dönecek ve Route eşleşmeyecektir.

```
namespace HaberPortal.RouteConstraint
{
  public class tarihConstraint:IRouteConstraint
  {
    public bool Match(HttpContextBase h, Route r,string p,
        RouteValueDictionary rv, RouteDirection rd)
    {
        bool  sonuc= false;
        string a = rv[p] as string;
        if (!string.IsNullOrWhiteSpace(a))
        {
            string[] kalip = { @"\d{2}-\d{2}-\d{4}", @"\d{2}.\d{2}.\d{4}"};
            foreach (string i in kalip)
            {
                if (Regex.Match(a, i).Success)
                {
                    sonuc = true;
                    break;
                }
            }
        }
    return sonuc;
    }
  }
}
```

Kodu incelediğimizde, gelen istekte bulunan parametrenin değeri `a` değişkenine aktarılıyor. Düzenli ifadelerin bulunduğu bir dizi içerisindeki tarih desenleri ile karşılaştırılıyor ve karşılaştırma doğru olduğu takdirde sonuç `true` olarak dönüyor.

Şimdi bu sınıfı kullanarak oluşturacağımız Route yapısını inceleyelim. Route içerisine, oluşturduğumuz `tarihConstraint` sınıfından türemiş nesne ekleyeceğimiz için `Global.asax.cs` dosyasının başına aşağıdaki namespace'i ekliyoruz.

```
using HaberPortal.RouteConstraint;
```

Oluşturacağımız Route aşağıdaki gibi olmalıdır.

```
routes.MapRoute(
     "ArsivRoute", // Route ismi
     "{tarih}",
     new { controller = "Haber", action = "Arsiv",
     tarih = "" }, // Varsayılan parametreler
     new { tarih = new tarihConstraint() }
);
```

Tanımladığımız Route yapısında dördüncü parametre olarak...

```
new {tarih= new tarihConstraint()}
```

... şeklinde kullandığımız kod, `tarihConstraint` sınıfından nesne oluşturuyor ve nesnenin değeri, gelen URL isteğine göre `true` veya `false` olarak dönüyor. Nesne true değeri döndürdüğü zaman Route eşleşiyor.

Konuyu daha iyi anlamak için iki tane örnek daha yapalım. Tanımlayacağımız yeni route, admin sayfanın login olması ile ilgili olsun. Eğer kullanıcı login olmamışsa Route eşleşmesin ve sayfaya ulaşılamasın. Öncelikle `RouteConstraint` klasörüne yeni bir sınıf ekliyoruz. Eklediğimiz sınıfa daha önceki örnekte yaptığımız gibi...

```
using System.Web.Routing;
```

... namespace'i sınıfımızın başına ekliyoruz ve sınıfımızı `IRouteConstraint` interface'den türetiyoruz. Bu işlemleri yaptıktan sonra `Match` metodunu sınıfımıza ekliyoruz. Son durumda sınıfımızın yapısı aşağıdaki gibi olacaktır.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Routing;
namespace HaberPortal.RouteConstraint
{
  public class Admin:IRouteConstraint
  {
    public bool Match(HttpContextBase a, Route b,string c,
         RouteValueDictionary d, RouteDirection e)
    {
```

```
    }
  }
}
```

`HttpContextBase.Request.IsAuthenticated` özelliğini kullanarak sınıfımıza aşağıdaki kodları ekliyoruz.

```
public bool Match(HttpContextBase a, Route b, string c,
RouteValueDictionary d, RouteDirection e)
{
    bool girisislemi = false;
    if (a.Request.IsAuthenticated)
    {
        girisislemi=true;
    }else{
        girisislemi=false;
    }
    return girisislemi;
}
```

Şimdi `Global.asax.cs` dosyası içerisine Route tanımlayalım. Tanımlayacağımız `routes.MapRoute` metodunun dördüncü parametresi aşağıdaki olmalıdır.

```
routes.MapRoute(
    "Admin", // Route ismi
    "{controller}/{action}", // URL parametresi
    new { controller = "Haber", action = "Admin"}, // Varsayılan parametreler
    new { admin = new Admin() }
);
```

`Admin` sınıfından türettiğimiz nesne, gelen URL isteğine göre admin girişi yapılmışsa, yani `authenticated` olmuşsa, `true` değeri döndürecektir ve Route eşleşecektir.

Bir diğer örneğimizde de oluşturacağımız sınıf, `http` method kontrolü yaparak Route karşılaştırması yapacaktır.

Diğer örneklerimizde olduğu gibi `RouteConstraint` klasöründe yeni bir sınıf oluşturuyoruz. Oluşturduğumuz sınıfı `IRouteConstraint` isimli interface'den türetiyoruz. Sınıfımıza `using` komutunu kullanarak aşağıdaki namespace'i dahil ediyoruz.

```
using System.Web.Routing;
```

Oluşturduğumuz sınıfın kod yapısı aşağıda görüldüğü gibi olacaktır.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Routing;
namespace HaberPortal.RouteConstraint
{
  public class HttpMethod:IRouteConstraint
  {
    public bool Match(HttpContextBase a, Route b,string c,
         RouteValueDictionary d, RouteDirection e)
    {
         if (a.Request.RequestType==”Post”)
         {
             return true;
         }else{
             return false;
         }
    }
  }
}
```

Sınıfımız `HttpContextBase.Request.RequestType` özelliğini kullanarak eşleştirme yapacaktır. Eşleştirmenin sonucu gelen URL isteğinin `HttpMethod` yöntemi `Post` olduğu takdirde eşleşecektir.

`Global.asax.cs` içerisine ekleyeceğimiz `HaberEkle` isimli Route aşağıdaki gibi olacaktır.

```
routes.MapRoute(
    “HaberEkle”,
    “Haber/Ekle”,
    new { controller = “Haber”, action = “Ekle”},
    new { method = new HttpMethod() }
);
```

Gelen URL İsteklerinin Tamamının Eşleşmesi

Oluşturduğumuz bir route'un gelen url istekleriyle eşleşebilmesi için gelen istekteki bölüm sayısının eşit olması gerekir. Örneğin; aşağıdaki gibi iki bölümden oluşan bir route, üç bölümden oluşan bir Route isteği ile eşleşmeyecek ve sayfa çağrılamayacaktır.

```
routes.MapRoute(
     "Arama", // Route ismi
     "Ara/{detay}", // URL parametresi
     new { controller = "Haber"} // Varsayılan parametreler
);
```

Bu Route yapısına gelen `ara/aranacak` kelime şeklindeki URL isteği ile eşleşecektir. Çünkü varsayılan değer olarak belirtilen `Haber Controller, Ara` ifadesini de Action olarak alacaktır. Ancak Route aşağıdaki gibi üç bölümden oluşursa eşleşme olmayacaktır.

```
routes.MapRoute(
     "Arama", // Route ismi
     "Ara/{detay}/{id}", // URL parametresi
     new { controller = "Haber"} // Varsayılan parametreler
);
```

Gelen URL isteğinin her halükarda eşleşmesini istiyorsak aşağıdaki gibi `catch all` yöntemini kullanmalıyız. Bu yönteme göre `Haber Controller` içerisinde bulunan `Ara` Action metoduna gelen parametreler haberlerin detayında arama yapmak için kullanacaktır. Gelen URL isteğindeki `"Ara/"` ifadesinden sonra gelen tüm metin `Ara` Action metoduna gelen parametre olarak algılanacaktır.

```
routes.MapRoute(
     "Arama", // Route ismi
     "Ara/{*ara}", // URL parametresi
     new { controller = "Haber",action="Ara"} // Varsayılan parametreler
);
```

Bu Route yapısını incelediğimizde (`*`) yıldız karakterinin kullanıldığını görüyoruz. Kullandığımız `catch all` parametresi sayesinde gelen `Ara/` şeklindeki URL isteklerinin hepsi `Arama` rotasıyla eşleşecektir.

Catch-All Parametresi İle 404-Sayfa Bulunamadı Oluşturmak

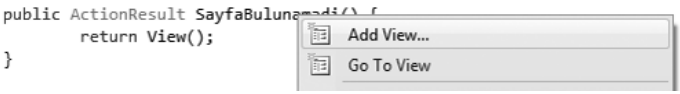
Tüm gelen istekleri `Catch-All` parametresi olan (`*`) yıldız karakteri ile yakalayabildiğimize göre gelen isteklerin karşılanmadığı durumlarda kendi sayfa bulunamadı mesajımızı oluşturabiliriz. Bunun için öncelikle şöyle bir mantık geliştirmemiz gerekiyor. Gelen URL isteği tanımladığımız hiçbir Route ile eşleşmediği durumlarda sistem tarafından sayfa bulunamadı hatası verecektir. Bu tür istekler geldiğinde bunu yakalayıp hata mesajı sayfamızın gösterilmesini sağlamalıyız. Oluşturacağımız Route aşağıdaki gibi olmalıdır.

```
routes.MapRoute(
     "Error",
     "{*.}",
     new { controller = "Haber",action="SayfaBulunamadi"}
);
```

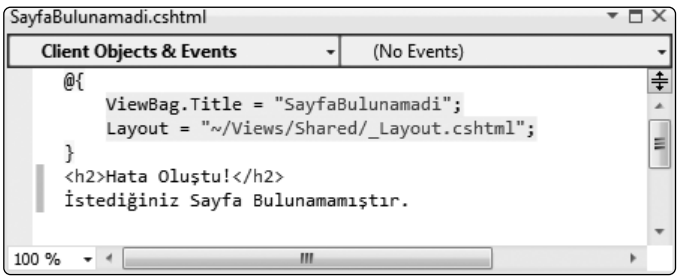
`Error` olarak adlandırdığımız Route yapısını incelediğimizde URL kalıbı veya deseni olarak aldığımız `{*.}` parametresi, gelen tüm URL desenleriyle eşleşecektir. Sayfa bulunamadı hata mesajı için `Haber` Controller içerisinde `SayfaBulunamadi` adında Action metodumuzu tanımlıyoruz.

```
public ActionResult SayfaBulunamadi(){
       return View();
}
```

`SayfaBulunamadi` Action metodunu ekledikten sonra bu metot çağrıldığında açılacak sayfa için metot ismi üzerinde sağ tıklayıp **Add View** komutunu çalıştırıyoruz.



Açılan pencerede **Layout** sayfamızı seçip **Add** butonuna basıyoruz. Aşağıdaki gibi `SayfaBulunamadi.cshtml` sayfası oluşacaktır. Oluşan sayfaya aşağıdaki hata mesajını yazıyoruz.



Yaptığımız bu işlemlerden sonra `Global.asax.cs` dosyası içerisinde tanımladığımız `Error Route`, diğer tanımladığımız route'lardan sonra ve default olarak isimlendirdiğimiz route'dan önce gelmelidir. Bu şekliyle projemizi çalıştırdığımız zaman karşımıza aşağıdaki gibi bir hata mesajı verecektir.



Henüz hatalı bir istekte bulunmadığımıza göre ilk açılışta ana sayfanın gelmesi gerekiyordu. Ancak tanımladığımız `Error Route`, `default Route`'dan önce tanımlandığı için ve `Error Route` tüm URL isteklerini karşıladığı için `default Route` ile eşleşmesine izin vermemiştir. Route tanımlama kurallarına göre `default` olarak isimlendirilen route, `Global.asax.cs` dosyası içerisinde en alt sırada olması gerekmektedir. Bu durumda öyle bir şey yapmalıyız ki sayfaya ilk girildiğinde ana sayfa çağrılsın, yani `Default Route` ile eşleşsin. Bunun için kısıtlama kurallarından yararlanabiliriz. `IRouteConstraint` interface'den bir sınıf türetip gelen isteğin ana sayfa olup olmadığını kontrol ettirebiliriz. Daha önce haber portalı projemiz içerisinde oluşturduğumuz `RouteConstraint` klasörüne bir tane `hata.cs` isminde sınıf oluşturuyoruz. Oluşturduğumuz sınıfı `IRouteConstraint` isimli interfaceden türeteceğiz. Sınıfımıza `Match` metodunu ekleyeceğiz. `Match` metodu ile gelen URL isteğini kontrol ettirecek kodu eklediğimizde sınıfımızın kodları aşağıdaki gibi olacaktır.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
```

```
using System.Web.Routing;
namespace HaberPortal.RouteConstraint
{
  public class Hata:IRouteConstraint
  {
    public bool Match(HttpContextBase a, Route b,string c,
         RouteValueDictionary d, RouteDirection e)
    {
         if (a.Request.RawUrl==”/”)
         {
             return false;
         }else{
             return true;
         }
    }
  }
}
```

Şimdi yapmamız gereken tek şey; `Global.asax.cs` dosyası içerisinde tanımladığımız `Error Route` içerisine, `Hata.cs` sınıfından türemiş nesne ekleyerek kısıtlama oluşturmak.

```
routes.MapRoute(
          “Error”,
          “{*.}”,
          new { controller = “Haber”,action=”SayfaBulunamadi”},
          new{a=new Hata()}
);
```

Artık sayfa bulunamadı hata mesajı sayfamız sorunsuz çalışacaktır.

Route Ignore Etmek

MVC uygulamalarında gelen URL istekleri `Global.asax.cs` dosyası içerisinde tanımladığımız route'larda bulunan URL desenleri ile eşleşerek ilgili Controller içerisindeki Action metodunu çağırmaktadır. Peki, her seferinde gelen URL isteklerinin eşleşmesini istemediğimiz durumlarda ne olacak? Soruyu değiştirecek olursak, neden bazı Route'ların görmezden gelinmesi istenebilir? Böyle bir şeye sitemizde bulunan statik dosyaların çağrılması durumlarında ihtiyaç duyabiliriz. Konuyu biraz açacak olursak bazı arama motorları ve tarayıcılar, sitemizi indekslemek, kategorize etmek için `Robots.txt` dosyasını kullanırlar. Veya sitemizde bulunan ima-

ge dosyaları veya js dosyalarının tarayıcılar tarafında istekte bulunulduğu durumlarda Route Ignore edilerek gelen isteklere cevap verebiliriz.

Örneğin; `http://www.domainismimiz.com/Robots.txt` şeklinde bir URL isteği arama motorları tarafından yapıldığı zaman Mvc Route, bunu kendisinde bulunan mevcut route'lar ile eşleştirerek, Controller dosyaları içerisinde bulunan Action metodu olarak `Robots.txt` ifadesini algılayacaktır. Bu tür bir yanlışlığı önlemek için `routes.IgnoreRoute` metodunu kullanabiliriz. Aşağıda `Global.asax.cs` dosyası içerisine eklediğimiz kod ile `http://www.domainismimiz.com/Robots.txt` URL isteğinin karşılanmasını sağlayacağız.

```
public static void RegisterRoutes(RouteCollection routes)
{
       routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
       routes.IgnoreRoute("robots.txt");
```

NotEqual Kisitlamasi Oluşturmak

Projemizde kullandığımız Controller Action metoduna belirli parametrelerin gitmesini istemiyorsak, bu kısıtlama yöntemini kullanabiliriz. Bu yöntem sayesinde sitemizde güvenlik problemlerini çözebiliriz. Bu tür bir kısıtlamayı `IRouteConstraint` isimli interface'den bir sınıf türeterek oluşturuyoruz. Oluşturduğumuz sınıfın `Match` metodunu kullanarak gelen parametreleri kontrol ediyoruz. Gelen parametre ile bizim Route içerisinde oluşturduğumuz nesnede belirttiğimiz parametre aynı olduğu takdirde nesne `false` değerini döndürecektir ve eşleşme olmayacaktır. Haber portalı projemizde oluşturduğumuz `RouteConstraint` klasörünün içerisine `NotEqual` isminde bir sınıf oluşturuyoruz. Oluşturduğumuz sınıfı `IRouteConstraint` isimli interface'den türetiyoruz. Sınıfımıza `IRouteConstraint` interface `Match` metodunu ekliyoruz. Son durumda sınıfımızın içeriği aşağıdaki gibi olacaktır.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Routing;
namespace HaberPortal.RouteConstraint
{
  public class NotEqual:IRouteConstraint
  {
    private string parametre;
```

```
    public NotEqual(string nesnedengelenparametre)
    {
      parametre = nesnedengelenparametre;
    }
    public bool Match(HttpContextBase a, Route b, string c,
    RouteValueDictionary d, RouteDirection e)
  {
    var gelenparametre = d[c].ToString();
    return string.Compare(gelenparametre, parametre, true) != 0;
  }
 }
}
```

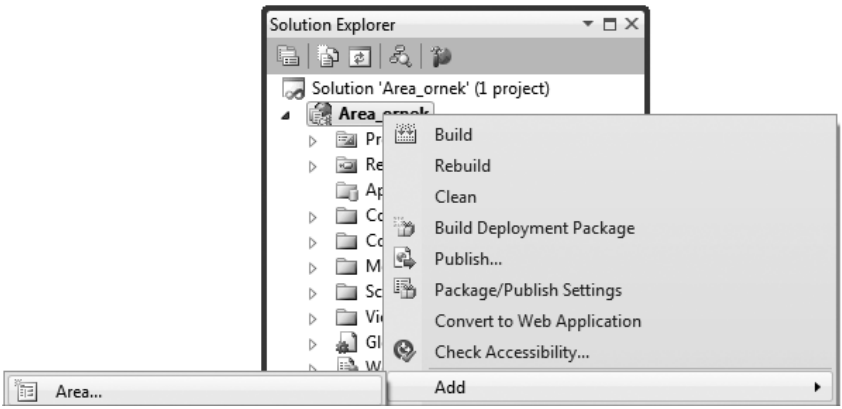
Kodumuzu inceleyelim... URL isteği ile gelen parametreyi string veri türüne çevirerek Route kalıbı içerisinde tanımladığımız parametrelerle karşılaştırıyor, karşılaştırma için `string.Compare` metodunu kullanıyoruz. `Compare` metodunda üçüncü parametre olarak `true` ekledik. Bu parametre sayesinde küçük büyük harf ayrımı yapmaksızın gelen URL isteğindeki parametre ile Route içerisinde tanımladığımız parametreler karşılaştırılacaktır. Sınıfımızı tanımladıktan sonra `Global.asax.cs` dosyası içerisinde tanımlayacağımız Route aşağıdaki gibi olacaktır.

```
routes.MapRoute(
    "Default", // Route ismi
    "{controller}/{action}/{id}", // URL parametresi
    new { controller = "Haber",action="Index",
    id=UrlParameter.Optional },
    new { controller = new notEqual("Admin")}
);
```

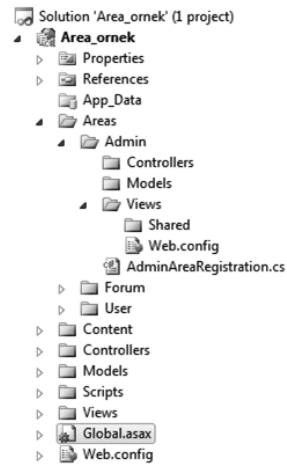
MVC Areas Routing

Büyük projelerde projenin geliştirilebilir olası ve karmaşayı önlemek için uygulamamızı bölümlere ayırabiliriz. Bu ayırma işlemi için **area** yapısını kullanacağız. Örneğin; bir haber portalımız olduğunu düşünürsek, bu haber portalında politika, dünya, yerel, teknoloji, eğitim vs. şeklinde kategoriler oluşturabiliriz. Her kategorinin sayfa tasarımı ve içeriğini de farklı yapabiliriz. Bu şekilde oluşturacağımız bir projede area kullanmadığımızı düşünürsek her **View** klasörünün altında birden çok layout olacak ve aynı durum Controller klasörünün altında controller'lar şeklinde karşımıza çıkacaktır. Area sayesinde mvc'nin **Model View Controller** klasör ya-

pısının dışına çıkarak kendi yapımızı oluşturabiliriz. Bu şekilde karmaşık bir yapıdan daha anlaşılır bir yapıya sahip proje oluşturabiliriz. Bir örnek üzerinden konumuzu anlatacak olursak senaryo olarak bir site içerisindeki **forum**, **user** ve **admin** bölümlerinin olduğunu düşünelim. Tüm kullanıcılara açık olan sitenin ana dizini içerisine area ekleyip bunların altında da diğer bölümleri oluşturabiliriz. Örneğimiz için yeni bir **ASP.NET MVC3 Web Application** projesini seçtikten sonra karşımıza çıkacak ekranda **Internet Application Template**'i seçiyoruz. Projemiz oluştuktan sonra **Solution Explorer** penceresinde...



... uygulamanın üzerinde sağ tıklayıp, yukarıdaki işlemi yaptıktan sonra area eklenecektir. Bu işlemi üç kere tekrarlayıp `forum`, `admin` ve `user` şeklinde adlandıralım.



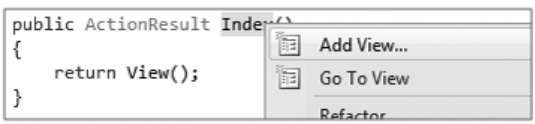
Şekilde görüldüğü gibi `Area` klasörünün altında `Admin`, `Forum` ve `User` alanlarını oluşturduk. Bu alanların her birinin altında `Controllers`, `Models` ve `Views` klasörleri oluştu. Bunun yanı sıra her alanın ismiyle isimlendirilmiş `.cs` uzantılı dosyada default olarak oluştu. `Admin` alanının içinde `AdminAreaRegistration.cs`, oluşturduğumuz `User` alanının altında `UserAreaRegistration.cs` ve `Forum` alanının altında da `ForumAreaRegistration.cs` oluşmuş oldu.

Bu dosyalar ne anlama geliyor? Daha önce `global.asax.cs` dosyasında ele aldığımız Route yapıları bu dosyalarda da tanımlanmaktadır. Her alanın altında bulunan Controller, ilgili alana gelen istekleri yönlendirmek için, View ilgili bölümün sayfa

tasarımları için ve model de ilgili alanın sorgu ve veritabanları için kullanılmak üzere eklenmiştir. Bu yapı sayesinde projemiz daha anlaşılabilir ve geliştirilmeye yatkın hale geldi. Ana dizinde bulunan `global.asax` dosyası içerisinde bulunan `AreaRegistration.RegisterAllAreas()` kod parçası her bölümün içerisinde tanımlanmış olan `AreaRegistration.cs` içerisindeki URL yapılarının tüm proje için tanımlanmasını sağlar. Dolayısıyla oluşturduğumuz bölümler için URL kalıpları tanımlamak ve değiştirmek istiyorsak ilgili bölümün `AreaRegistration.cs` dosyası içerisinde Route tanımlamalıyız. Aşağıda oluşturduğumuz `admin` bölümündeki `AdminAreaRegistration.cs` dosyasının kod yapısı görünmektedir.

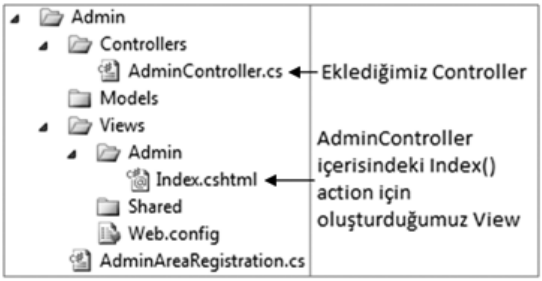
```
namespace Area_ornek.Areas.Admin
{
    public class AdminAreaRegistration : AreaRegistration
    {
        public override string AreaName
        {
            get
            {
                return "Admin";
            }
        }
        public override void RegisterArea(AreaRegistrationContext context)
        {
            context.MapRoute(
                "Admin_default",
                "Admin/{controller}/{action}/{id}",
                new { action = "Index", id = UrlParameter.Optional }
            );
        }
    }
}
```

Örnek uygulamamıza tekrar geri dönecek olursak: `admin` bölümüne Controller eklemek için ilgili bölümde bulunan Controller klasörüne sağ tıklayıp **Add>Add Controller** komutunu çalıştırdıktan sonra, karşımıza gelen pencerede `AdminController` ismi vererek **OK** butonuna tıklıyoruz. Burada dikkat etmemiz gereken önemli bir kural var; Controller isimleri benzersiz olmalıdır.



Oluşturduğumuz `AdminController` içerisinde `Index()` Action'a View ekliyoruz.

Son durumda `admin area` bölümünün klasör yapısı ve klasörlerin içerisinde bulunan dosyalar aşağıdaki resimde görüldüğü gibi olacaktır.



Şimdi uygulamamızın ana dizininde bulunan **View>Home** klasörünün içindeki `index.cshtml` dosyasına bir `ActionLink` ekleyerek, admin bölümünün ana sayfasına ulaşmaya çalışalım. Ekleyeceğimiz `ActionLink` aşağıdaki şekilde belirtildiği gibi olmalıdır.

```
<ul>
<li>@Html.ActionLink("Admin Anasayfa", "Index", "Admin", new { area = "Admin" }, null)</li>
<li>@Html.ActionLink("User Anasayfa", "Index", "User", new { area = "User" }, null)</li>
<li>@Html.ActionLink("Forum Anasayfa", "Index", "Forum", new { area = "Forum" }, null)</li>
</ul>
```

Url Metni Action Controller Bölüm

Web tarayıcının adres çubuğunda bölümlerin adresleri `admin` bölümü için `Admin/Admin/Index` şeklinde olacaktır.

HELPER

7

Helper'lar, View'ler içerisinde kullanılan ve uygulamalarımızın görünüm kısmında kullanılan metotlardır. HTML olarak çıktı üretirler. MVC içerisinde varsayılan olarak gelen **Html Helper** metotlarıdır. Bunun dışında daha sonra da yükleyerek kullanabileceğiniz **Web Helper** metotları da bulunmaktadır.

Helper başlığı altında aşağıdaki Helper çeşitlerini işleyeceğiz:

- Html Helper
- Web Helper
- MicrosoftWeb Helper
- PayPal Helper

Html Helper Metotları

Html Helper bir View içerisindeki HTML kontrolleri oluşturmak için kullanılan metotlardır. Bütün HTML Helper metotları HTML çıktı üretir ve geriye string değer döndürür. HTML Helper kullanmadan da MVC ile web sayfaları oluşturabilirsiniz. Ancak HTML Helper kullanarak uygulamalarınızı daha kolay bir biçimde, daha az bir çaba ile geliştirebilirsiniz. Html Helper sınıfında .NET Framework 3.0 ile kullanılmaya başlanan **extension methods** (*genişletme metotları*) kullanmaktayız. Genişletme metotları, kullanmakta olduğumuz metotlara yeni özellikler eklemek amacı ile kullanılırlar. HTML Helper metotlarına...

```
Html.HtmlHelper Metodu ismi
```

... şeklinde erişebiliriz.

Html Helper metotlarından sık kullanılanlarını inceleyelim.

ActionLink

`ActionLink` metodu ile, belirtilen bir Controller'ın mevcut olup olmadığı kontrol edilmekte, mevcut ise Controller içerisinde belirtilen `ActionResult` metodu çalıştırılmaktadır.

```
ActionLink("Link Metni","ActionResult İsmi","Controller İsmi")
```

Örneğin...

```
@Html.ActionLink("Ana Sayfa","Index","Home")
```

... şeklinde yazdığımız metot, `Ana Sayfa` şeklinde bir link oluşturacak. Linke tıklandığında `Home` isimli Controller içerisindeki `Index` isimli `ActionResult` çalışacaktır.



`ActionLink` metodunu resimlere link vermek için kullanamazsınız. Bunun için `Url.Action` metodunu kullanabilirsiniz. Örneğin; yukarıda tanımladığımız `ActionLink` metodunu resim linki şeklinde aşağıda görüldüğü gibi tanımlayabiliriz.

```
<a href=@Url.Action("Ana Sayfa")><img src="../../Content/AnaSayfa.png"
alt="AnaSayfa"/></a>
```

BeginForm

`BeginForm` metodu, bir html form elemanının başlangıcını belirtir ve form elemanını oluşturur. Birkaç farklı kullanım biçimi vardır.

```
@using (Html.BeginForm())
{
    <!-- Form içeriği -->
}
```

şeklinde hiçbir parametre kullanmadan kullanabileceğimiz gibi...

```
@using (Html.BeginForm("Action ismi", "Controller ismi"))
{
    <!-- Form içeriği -->
}
```

... şeklinde Action ismini ve Controller ismini belirterek de kullanabiliriz. Ya da...

```
@using (Html.BeginForm("Action ismi", "Controller ismi",
FormMethod.Post, new { id = "FormAdı" }))
```

```
{
    <!-- Form içeriği -->
}
```

... şeklinde de kullanabiliriz. `FormMethod` özelliği, `Post` ve `Get` olmak üzere iki değer alır. Form ile, `Post` işlemi mi `Get` işlemi mi yapılacağını belirlenir. Son parametre ile ise oluşturulacak olan form nesnesi ile ilgili çeşitli html niteliklerini (*Html Attributes*) belirtilir. Yukarıdaki örnekte oluşturmuş olduğumuz formun `id` özelliğini `FormAdı` olarak belirledik.

Label

Bilgi görüntülemek amacıyla kullanılır. İçeriği daha sonra değiştirilemez.

```
@Html.Label("İsminiz")
```

TextBox

Bilgi girişi amacıyla kullanılır.

```
@Html.TextBox("isim")
```

biçiminde kullanıldığında ekranda boş bir `TextBox` oluşturur.

```
@Html.TextBox("isim","Ahmet")
```

biçiminde kullanıldığında ise ekranda içerisinde *"Ahmet"* yazan bir `TextBox` oluşturulacaktır. *"Ahmet"* değeri, varsayılan değer olarak kabul edilmektedir.

Varsayılan değer olarak çeşitli metotları da kullanabiliriz. Örneğin...

```
@Html.TextBox("KayıtTarihi",@DateTime.Now.ToShortDateString())
```

... biçimindeki kullanımda `TextBox` içerisinde varsayılan değer olarak günün tarihi görüntülenmektedir.

```
@Html.TextBox("isminiz",null,new { @maxlength = "10" })
```

Bu kullanımda öncelikle `TextBox`'ın varsayılan değerini boş bırakmak için `null` değerini kullandık. Son parametre ile ise `maxlength` isimli html niteliğini kullanarak `TextBox` içerisine girilecek karakter sayısını `10` karakter ile sınırladık.

TextArea

Birden fazla satırdan oluşan değerleri görüntülemek amacıyla kullanılır.

```
@Html.TextArea("icerik")
```

Bu şekilde kullandığımız `TextArea` metodu, varsayılan görünümü ile görüntülenecektir.



İstenildiği takdirde html niteliklerini kullanarak `TextArea` nesnesinin satır ve sütun sayısını belirleyebiliriz.

```
@Html.TextArea("icerik", new { cols = 50, rows = 8 })
```

Bu şekilde sütun sayısını `50`, satır sayısını `8` olarak belirleyebiliriz.



CheckBox

Onay ve seçim işlemleri için kullanılır. İşaretli ve işaretsiz olmak üzere iki konum alır.

```
@Html.CheckBox("Mail Adresim Görünsün")
```

Bu kullanımda varsayılan olarak işaretsiz biçimde bir `CheckBox` görüntülenecektir.

```
@Html.CheckBox("Mail Adresim Görünsün",true)
```

Bu kullanımda ise işaretli biçimde bir `CheckBox` görüntülenecektir. `True` değerini `false` olarak değiştirdiğimizde ise `CheckBox` işaretsiz olarak görüntülenecektir.

RadioButton

Birden fazla seçenekten yalnızca birini seçme durumlarında kullanılır.

```
@Html.Label("Cinsiyetiniz")
@Html.Label("Bay")
@Html.RadioButton("Cinsiyet","Bay",true)
@Html.Label("Bayan")
@Html.RadioButton("Cinsiyet","Bayan",false)
```

Örneğimizde `Bay` ve `Bayan` değerlerine sahip iki adet `RadioButton` oluşturuyoruz. Burada dikkat edilmesi gereken nokta şudur: `RadioButton` nesnelerini gruplandırarak içlerinden yalnızca birinin seçilip diğerlerinin seçilememesi için bütün `RadioButton` nesneleri aynı isme sahip olmalıdır. Yani...

```
@Html.RadioButton("Cinsiyetiniz","Bay",true)
@Html.RadioButton("Cinsiyet","Bayan",false)
```

... şeklinde tanımlanan `RadioButton` nesnelerinin her ikisi de aynı anda seçilebilir. Çünkü birincisinin ismi `Cinsiyetiniz` diğerinin ise `Cinsiyet` şeklinde farklı isimlerdedir. Gruplandırabilmek için ilk örneğimizde olduğu gibi...

```
@Html.RadioButton("Cinsiyet","Bay",true)
@Html.RadioButton("Cinsiyet","Bayan",false)
```

... her ikisi de aynı isme sahip olmalıdır.

DropDownList

İçerisindeki değerleri açılabilen liste şeklinde görüntüleyen nesnedir. Hem `DropDownList` hem de `ListBox` `Html Helper` metotlarının her ikisi de `SelectList` veya `MultiSelectList` nesnelerini kullanabilir. Bu nesneler ile `DropDownList` ve `ListBox` içerisinde değerleri listeletebiliriz.

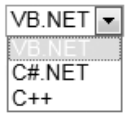
Örnek olarak `SelectList` kullanarak bir `DropDownList` içerisine değerleri aktaralım.

```
@Html.DropDownList("diller",
    new []
    {
        new SelectListItem
        {
            Text = "VB.NET",
            Value = "lst1"
        },
        new SelectListItem
        {
            Text = "C#.NET",
            Value = "lst2"
        },
        new SelectListItem
        {
            Text = "C++",
            Value = "lst3"
        }
    },
    new { id = "LstDiller" }
)
```

Tıklanmadan önceki durumu.



Tıklandıktan sonraki durumu.



ListBox

İçerisindeki değerleri liste şeklinde görüntüleyen nesnedir. Örnek olarak yukarıda `DropDownList` için yaptığımız örneği `ListBox` için yapalım.

Örnek olarak `SelectList` kullanarak bir `DropDownList` içerisine değerleri aktaralım.

```
@Html.ListBox("diller",
     new []
     {
          new SelectListItem
          {
               Text = "VB.NET",
               Value = "lst1"
          },
          new SelectListItem
          {
               Text = "C#.NET",
               Value = "lst2"
          },
          new SelectListItem
          {
               Text = "C++",
               Value = "lst3"
          }
     },
     new { id = "LstDiller" }
)
```



Kodumuz sonucunda oluşan **ListBox**'ın görünümü.

Password

TextBox görünümünde olup yazılan değerleri şifre karakterleri ile gizler.

```
@Html.Label("Şifre: ")
@Html.Password("Şifre")
```

Yukarıdaki kod satırlarının sonucunda oluşan Password nesnesine değer girdiğimizde, girilen değerler kullanıcıdan gizlenir.



Şimdiye kadar anlattığımız Helper metotlarının birçoğunu kullanarak örnek bir iletişim formu yapalım. İletişim formumuza girilen veriler mail hesabı üzerinden bize iletilecek.

Öncelikle HomeController.cs dosyası içerisine aşağıdaki kod satırlarını ekliyoruz:

Mail göndermek için **WebMail** nesnesini kullanacağız. WebMail nesnesini kullanarak herhangi bir SMTP sunucusu üzerinden mail gönderebilirsiniz. Bu nesneye System.Web.Helpers sınıfı altından erişebiliriz. Bu nedenle...

```
using System.Web.Helpers;
```

... satırı ile bu sınıfı projemize dahil ediyoruz. Daha sonra aşağıdaki kod satırlarını ekleyebiliriz.

```
[HttpGet]
public ActionResult form_bilgisi()
{
    return View();
}
[HttpPost]
public ActionResult form_bilgisi(string Mail,
string konu, string icerik)
{
    WebMail.SmtpServer = "smtp.gmail.com";
    WebMail.UserName = "mail adresiniz";
    WebMail.Password = "mail şifreniz";
    WebMail.EnableSsl = true;
    WebMail.SmtpPort = 587;
    WebMail.Send("Gönderilecek mail adresi",
```

```
            konu, icerik, Mail);
        return RedirectToAction("formbilgisi_gonder");
    }
    public ActionResult formbilgisi_gonder()
    {
        return View();
    }
```

Örneğimizde Gmail sunucusunu kullanarak mail göndereceğimiz için mail ayarları, Gmail SMTP ayarlarına göre yapılmıştır. Siz de kullanacağınız SMTP sunucusuna bağlı olarak bu ayarları özelleştirebilirsiniz.

Örneğimizde `form_bilgisi` adlı `ActionResult` içerisinde mail gönderme işlemi gerçekleştirilmekte. Mail gönderme işlemi tamamlandıktan sonra...

```
return RedirectToAction("formbilgisi_gonder");
```

... satırı ile `formbilgisi_gonder` isimli `ActionResult`'a yönlendirme yapılmaktadır.

`HomeController.cs` isimli dosyamız içerisindeki kod yazma işlemini tamamladıktan sonra...

```
public ActionResult form_bilgisi(string Mail,
string konu, string icerik)
```

... biçiminde yazdığımız `ActionResult` içerisinde farenin sağ tuşu ile tıklayarak **Add View** seçeneği ile `form_bilgisi.cshtml` isimli View'i oluşturuyoruz.

`form_bilgisi.cshtml` isimli View içerisine aşağıdaki kod satırlarını ekliyoruz:

```
@{ MvcForm mvcfrm = Html.BeginForm();}
<fieldset>
    <legend>ÖNERİLERİNİZ-YORUMLARINIZ</legend>
    <p>
        @Html.Label("İsminiz")
        @Html.TextBox("isminiz")
    </p>
    <p>
        @Html.Label("Mail Adresiniz")
        @Html.TextBox("Mail")
    </p>
    <p>
        @Html.Label("Konu")
```

```
            @Html.TextBox("Konu")
        </p>
        <p>
            @Html.Label("İçerik")
            @Html.TextArea("icerik",
                new { cols = 50, rows = 8 })
        </p>
        <p>
            <input type="submit" value="Gönder" />
        </p>
</fieldset>
@{ mvcfrm.Dispose();}
```

Kodumuzda `Html.BeginForm()` metodu ile form oluşturma işlemini başlatıp, içerisine form elemanlarını ekledikten sonra `mvcfrm.Dispose()` metodu ile form'un görüntülenmesi bittikten sonra form elemanlarının hafızadan temizlenmesini sağlıyoruz.

`form_bilgisi.cshtml` isimli dosyamız içerisindeki kod yazma işlemini tamamladıktan sonra **HomeController.cs** dosyası içerisindeki `formbilgisi_gonder` isimli ActionResult içerisinde farenin sağ tuşu ile tıklayarak **Add View** seçeneği ile `formbilgisi_gonder.cshtml` isimli View'ı oluşturuyoruz.

`formbilgisi_gonder.cshtml` isimli View içerisine yalnızca sonuç hakkında bilgi vermek amaçlı olarak aşağıdaki kod satırını eklememiz yeterlidir.

```
<h2>Bilgiler Gönderildi</h2>
```

Uygulamamızı çalıştıralım. Form içerisine örnek olarak aşağıdaki değerleri girelim.





Gönder butonuna tıkladığımızda mail gönderme işlemi doğru olarak tamamlandığında sonuç sayfası görüntülenecektir.

ValidationSummary

Bir ya da birden fazla kontrolün kullanımı esnasında meydana gelen hata mesajlarının görüntülenip, görüntülenmeyeceğini belirler. Ayrıca hata meydana gelmesi durumunda görüntülenecek olan genel hata mesajını belirler.

Örnek olarak mail ile iletişim bilgilerini gönderen form uygulaması üzerinde gerekli değişiklikleri yapalım.

Öncelikle **HomeController.cs** dosyası içerisindeki `form_bilgisi` adlı `ActionResult` metodu üzerinde biraz değişiklik yapacağız. Önceki halinde herhangi bir hata kontrolü kullanmamıştık. `Try catch` yapısını kullanarak hataları yakalayacağız.

`form_bilgisi` isimli `ActionResult`'un yeni hali aşağıdaki gibi olacak:

```
[HttpPost]
public ActionResult form_bilgisi(string Mail,
string konu, string icerik)
{
    try
    {
        WebMail.SmtpServer = "smtp.gmail.com";
        WebMail.UserName = "mail adresiniz";
        WebMail.Password = "mail şifreniz";
        WebMail.EnableSsl = true;
        WebMail.SmtpPort = 587;
        WebMail.Send("Gönderilecek mail adresi", konu, icerik, Mail);
        return RedirectToAction("formbilgisi_gonder");
     }
     catch (Exception ex)
     {
        ViewData.ModelState.AddModelError
            ("iletisim", ex.ToString());
     }
     return View();
}
```

Kodumuz içerisinde...

```
ViewData.ModelState.AddModelError("iletisim", ex.ToString());
```

...satırı ile herhangi bir hata yakalandığında belirtilen key için hata mesajlarının geriye döndürülmesi sağlanır. Örneğimizde formumuzun adı `iletisim` olduğu için key adı olarak `iletisim` şeklinde belirttik.

Bu işlemin ardından `form_bilgisi.cshtml` isimli View içerisine...

```
Html.ValidationSummary()
```

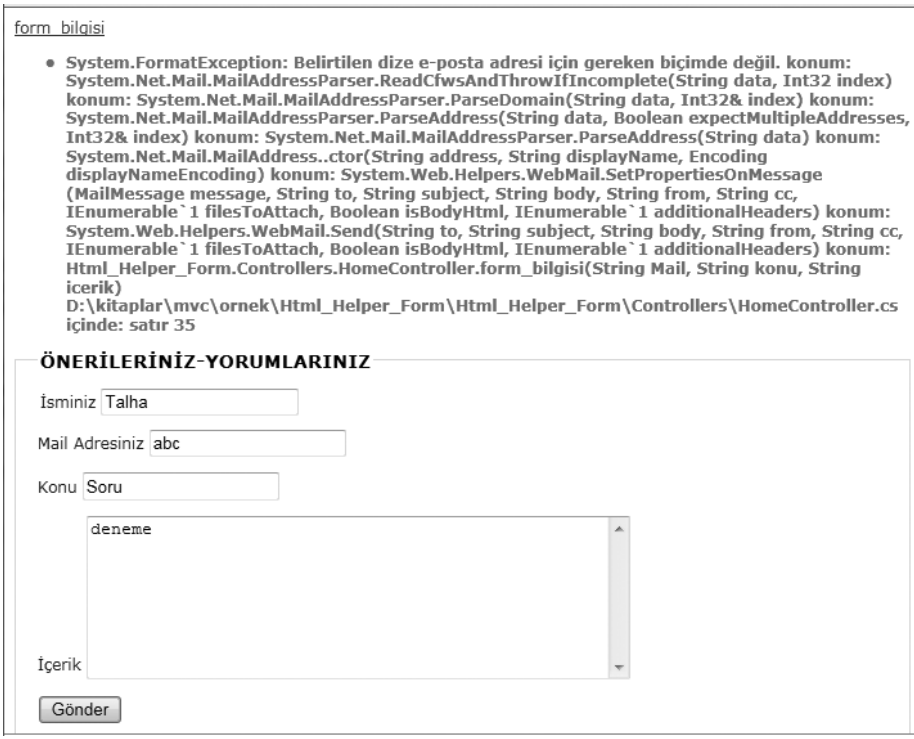
... satırını eklememiz gerekiyor. `ValidationSummary` metodu...

```
Html.ValidationSummary(true)
```

... şeklinde `true` parametresi ile kullanıldığında model düzeyindeki hataları görüntüler.

```
Html.ValidationSummary(false)
```

Böylece `false` parametresi ile kullanıldığında meydana gelen tüm hataları görüntüler.

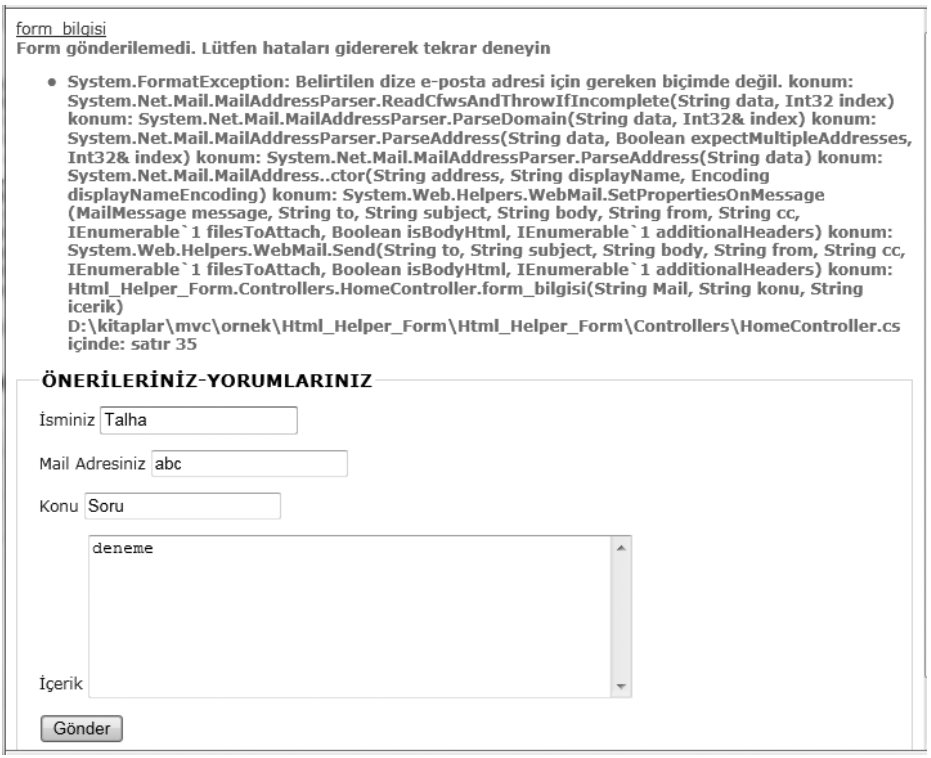


```
Html.ValidationSummary()
```

Bu şekilde kullanıldığında ise varsayılan olarak false parametresini almış gibi işlem görür.

```
Html.ValidationSummary ("Form gönderilemedi. Lütfen hataları gidererek
tekrar deneyin")
```

Bu şekilde ise öncelikle belirtilen hata mesajı, ardından da meydana gelen tüm hatalar görüntülenir.



ValidationMessage

Bir kontrolün kullanımı esnasında meydana gelen hata mesajlarının görüntülenip, görüntülenmeyeceğini belirler. Ayrıca hata meydana gelmesi durumunda görüntülenecek olan hata mesajını belirler.

Örnek olarak mail ile iletişim bilgilerini gönderen form uygulaması üzerinde gerekli değişiklikleri yapalım.

Öncelikle **HomeController.cs** dosyası içerisindeki `form_bilgisi` adlı `ActionResult` metodu üzerinde biraz değişiklik yapacağız.

`form_bilgisi` isimli `ActionResult`'un yeni hali aşağıdaki gibi olacak:

```
[HttpPost]
public ActionResult form_bilgisi(string mail,
string konu, string icerik)
{
    try
    {
        WebMail.SmtpServer = "smtp.gmail.com";
        WebMail.UserName = "Kaynak mail adresi";
        WebMail.Password = "Mail Şifresi";
        WebMail.EnableSsl = true;
        WebMail.SmtpPort = 587;
        WebMail.Send("Mail adresi", konu, icerik, mail);
        if (!Regex.IsMatch(mail.Trim(),
              @"^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$"))
        {
          ViewData.ModelState.AddModelError("mail",
               "Mail adresi geçersiz!");
        }
        else if (konu.Trim().Length == 0)
        {
          ViewData.ModelState.AddModelError("konu",
               "Konu alanı boş olmamalı!");
        }
        else if (icerik.Trim().Length == 0)
        {
          ViewData.ModelState.AddModelError("icerik",
               "İçerik alanı boş olmamalı!");
        }
        else return RedirectToAction("formbilgisi_gonder");
     }
     catch (Exception ex)
     {
        ViewData.ModelState.AddModelError
            ("iletisim", ex.ToString());
     }
     return View();
}
```

Kodumuz içerisinde tanımladığımız `key` için belirtilen hata mesajlarının geriye döndürülmesi sağlanır. Örneğimizde formumuzun adı `iletisim` olduğu için key adı olarak `iletisim` şeklinde belirttik. Örneğin...

```
if (icerik.Trim().Length == 0)
{
     ViewData.ModelState.AddModelError("icerik",
          "İçerik alanı boş olmamalı!");
}
```

... satırlarında `icerik` adlı değişken kontrol edilir. Eğer `icerik` değişkeninin değeri boş ise `icerik` adlı key için *"İçerik alanı boş olmamalı!"* şeklinde hata mesajı atanmasını sağladık.

```
if (!Regex.IsMatch(mail.Trim(),
  @"^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$"))
{
       ViewData.ModelState.AddModelError("mail", "Mail adresi geçersiz!");
}
```

satırlarında ise `Mail` adlı değişken kontrol edilerek eğer `Mail` değişkeninin değeri mail adresi biçimi ile uyuşmuyor ise, `Mail` adlı key için *"Mail adresi geçersiz!"* şeklinde hata mesajı atanmasını sağladık. Ancak mail adresi üzerindeki kontrolü **Regular Expressions** (*Düzenli İfadeler*) ile gerçekleştirdiğimiz için kodumuzun başına...

```
using System.Text.RegularExpressions;
```

... satırını ekleyerek `System.Text.RegularExpressions` sınıfını projemize dahil etmemiz gerekmektedir.

`form_bilgisi.cshtml` isimli View içerisindeki kodları ise aşağıdaki gibi düzenliyoruz.

```
@Html.ValidationSummary("iletisim")
@{ MvcForm mvcfrm = Html.BeginForm();}
<fieldset>
     <legend>ÖNERİLERİNİZ-YORUMLARINIZ</legend>
     <p>
          @Html.Label("İsminiz")
          @Html.TextBox("isminiz")
     </p>
     <p>
```

```
            @Html.Label(“Mail Adresiniz”)
            @Html.TextBox(“mail”)
            @Html.ValidationMessage(“mail”, “*”)
        </p>
        <p>
            @Html.Label(“Konu”)
            @Html.TextBox(“konu”)
            @Html.ValidationMessage(“konu”, “*”)
        </p>
        <p>
            @Html.Label(“İçerik”)
            @Html.TextArea(“icerik”, new { cols = 50, rows = 8 })
            @Html.ValidationMessage(“icerik”, “*”)
        </p>
        <p>
            <input type=”submit” value=”Gönder” />
        </p>
</fieldset>
@{ mvcfrm.Dispose();}
```

Kodumuz içerisinde örneğin...

```
@Html.ValidationMessage(“mail”, “*”)
```

... satırında mail adresi ile ilgili bir hata oluştuğunda `Mail` adlı `key`'e atanmış olan hata kontrolü çalıştırılarak, mail adresi girişi yapılan kontrolün yan tarafında belirttiğimiz string (örneğimizde `*` karakteri) ve formun başlığı altında, atanan hata mesajı görüntülenecektir.

Uygulamamızı çalıştırıp içerik bölümü boş bırakalım.



Görüldüğü gibi İçerik girişi yapılan kontrolün yan tarafında belirttiğimiz string (örneğimizde * karakteri) ve formun başlığı altında, atanan hata mesajı görüntülenecektir.

NOT

HTML Helper metotlarının ayrıca extension metotları da mevcuttur. Bu sayede bu metotları farklı biçimlerde kullanabilmemiz mümkündür. Şimdiye kadar kullandığımız `Label`, `TextBox`, `CheckBox` gibi tüm metotların `LabelFor`, `TextBoxFor`, `CheckBoxFor` şeklinde extension metotları mevcuttur. Extension metotlar sayesinde model içerisinde bulunan özellik ve değerlere erişilerek bu özellik ve değerlerin `Label` ve `TextBox` gibi Html nesneleri ile görüntülenmesini sağlayabiliriz.

Model içerisindeki değerlere erişebilmek için Html Helper metodunu...

```
@Html.ExtensionMetotAdı.(Model, Özellik ya da Değer)
```

... biçiminde kullanabiliriz.

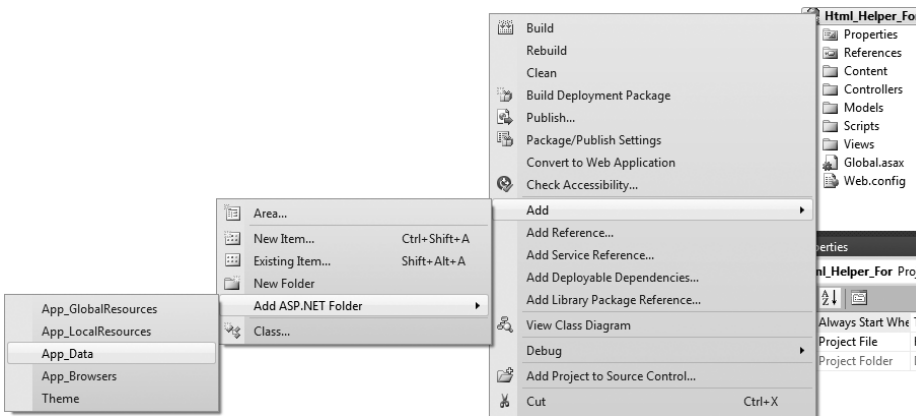
Örneğin; `LabelFor` nesnesini

```
@Html.LabelFor(model => nesne.telefon)
```

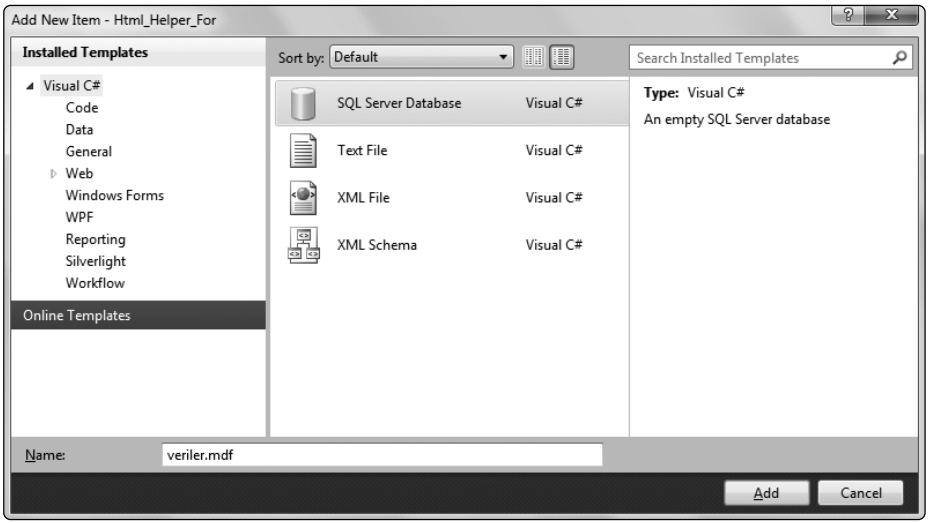
... şeklinde kullandığımızda `nesne` adlı model içerisinde bulunan `telefon` isimli veritabanı alanındaki değeri gösterir.

Örnek olarak bir veritabanı oluşturarak, veritabanındaki kayıtları listeleyen bir uygulama yapalım. Örneğimiz için boş bir **ASP.NET MVC 3** uygulaması oluşturuyoruz.

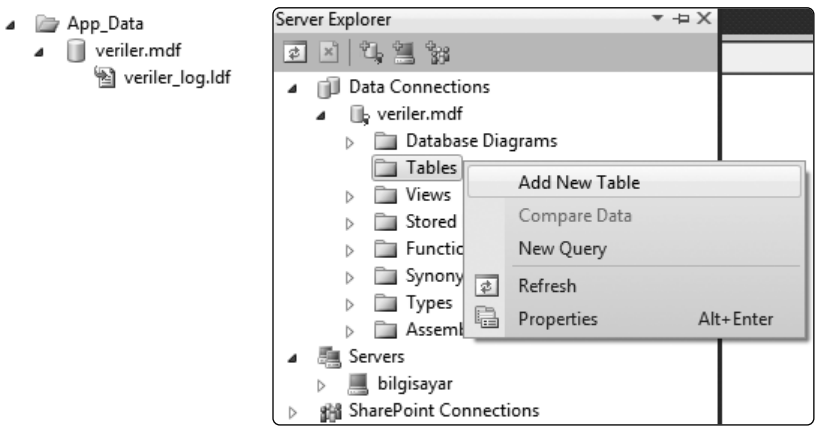
İşe veritabanımızı oluşturmakla başlayalım. Veritabanımızı `App_Data` klasörü altında oluşturacağız. `App_Data` klasörü mevcut değilse **Project/Add ASP .NET Folder** menüsü ile ya da **Solution Explorer**'a farenin sağ tuşu ile tıklayarak **Add/ASP.NET Folder** menüsü ile ekleyebiliriz.



Klasörümüzü ekledikten **Project/Add New Item** menüsü ile ya da **Solution Explorer** içerisinde klasörümüze farenin sağ tuşu ile tıklayarak **Add/New Item** menüsü ile yeni bir **SQL Server** veritabanı ekleyeceğiz.

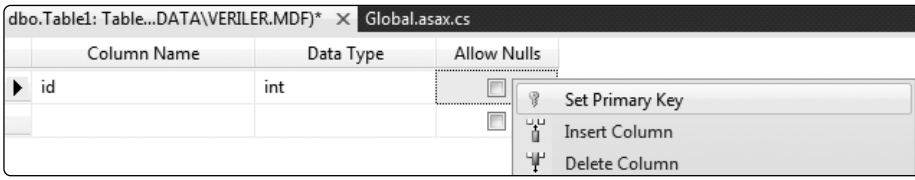


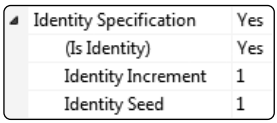
Veritabanımızın ismini `veriler` şeklinde belirledim.



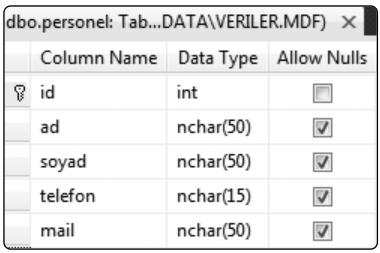
Veritabanını ekledikten sonra tablo ve alanları oluşturma aşamasına geldik. **Server Explorer** içerisinde veritabanı ismine tıklayıp, açılan listeden **Tables** seçeneğine farenin sağ tuşu ile tıklayarak **Add New Table** menüsünü seçiyoruz.

Alanlarımızı oluşturmaya başlıyoruz. İlk olarak `id` adlı alanımızı oluşturarak veri türünü `int` olarak belirliyoruz.



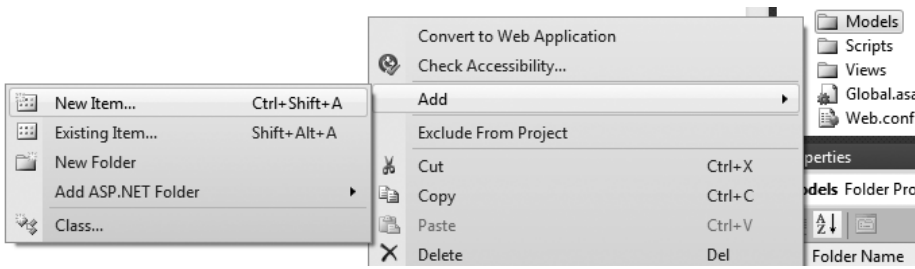


Alanımızı `Primary Key` olarak belirleyip, değerlerin otomatik olarak artması için **Identity Specification** başlığı altındaki özellikleri resimde görüldüğü gibi belirliyoruz.

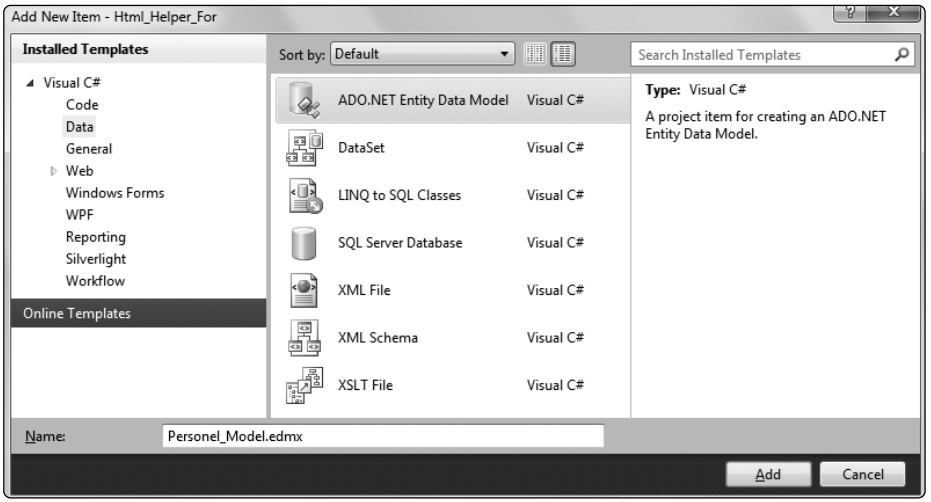


Alanlarımızın tümünü resimde görüldüğü gibi tanımlıyoruz. Tablomuzun ismini `personel` olarak belirledim. Alanları oluşturduktan sonra uygulamamızda kullanmak için örnek kayıtlar girebiliriz.

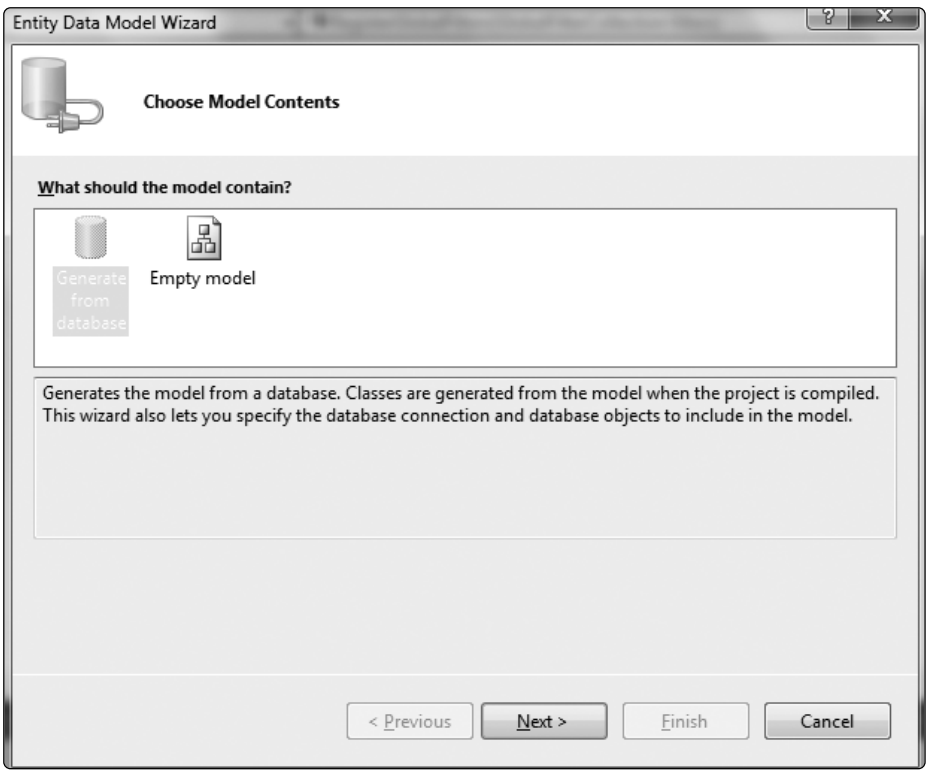
Veritabanını oluşturma işleminin bitmesinin ardından veritabanı üzerindeki işlemleri yürütebilmemiz için yeni bir Model oluşturacağız. Bu işlem için, **Project/Add New Item** menüsü ile ya da **Solution Explorer**'da **Models** klasörüne farenin sağ tuşu ile tıklıyoruz.



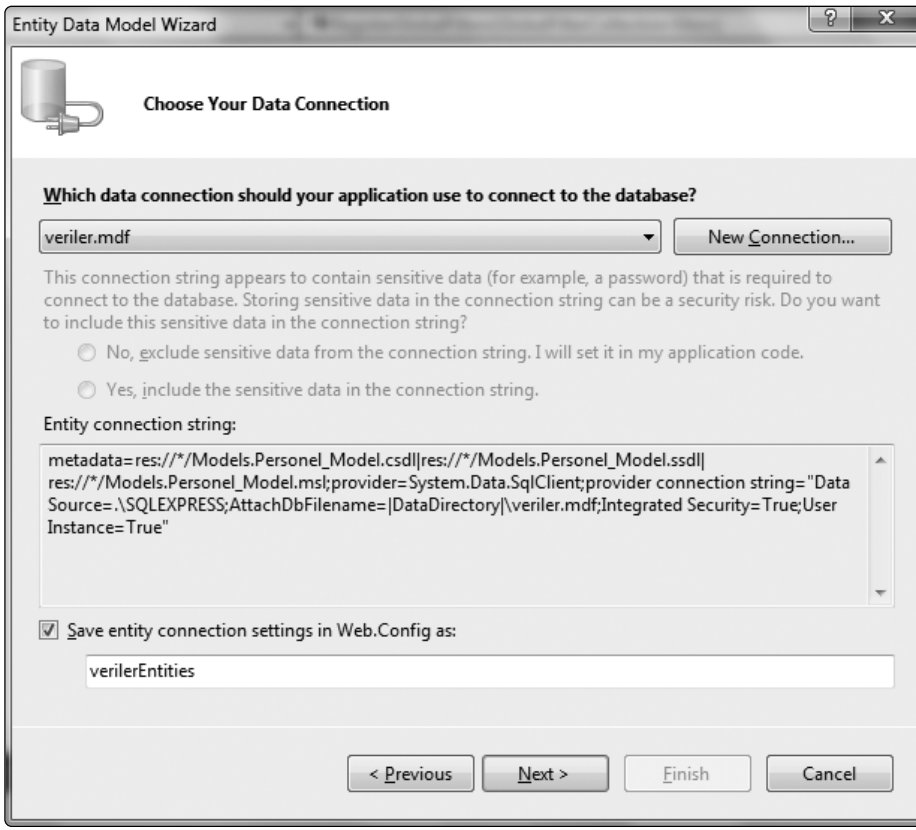
Gelen pencereden **ADO.NET Entity Data Model** seçeneğini seçiyoruz. Modelimizin ismini `Personel_Model` olarak belirledim.



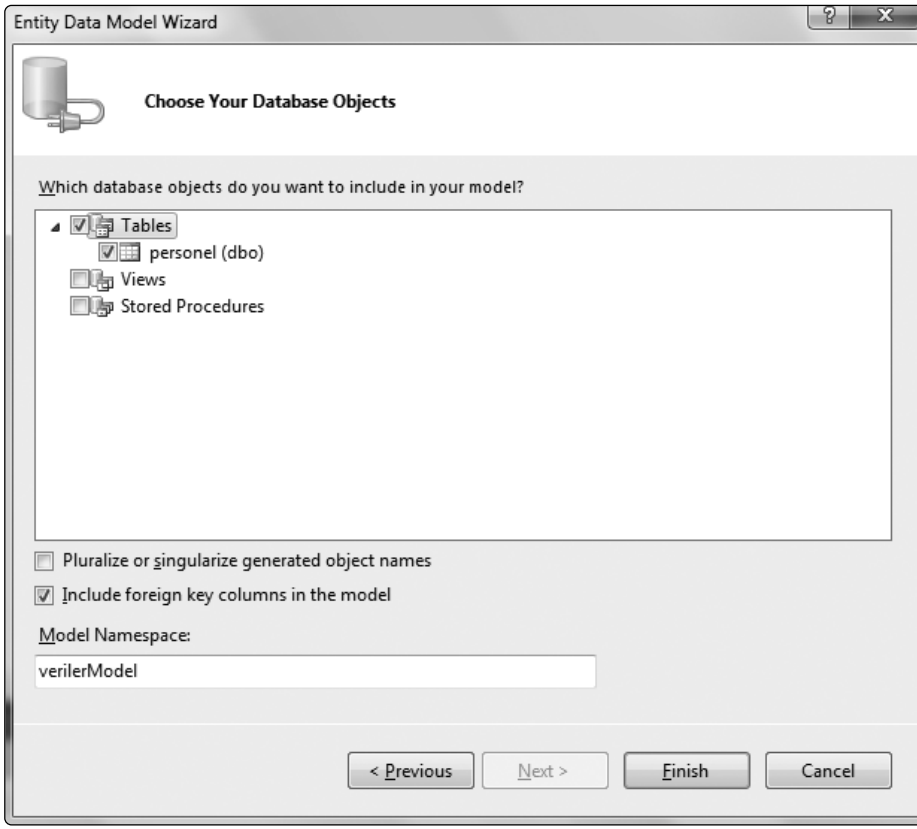
Add butonuna tıklamamızın ardından karşımıza oluşturacağımız **Entity Data Model** ile ilgili ayarların bulunduğu pencere gelecektir.



Oluşturacağımız **Data Model**'i daha önce oluşturmuş olduğumuz veritabanımızdan oluşturacağımız için gelen pencerede **Generate from database** seçeneğini seçiyoruz.

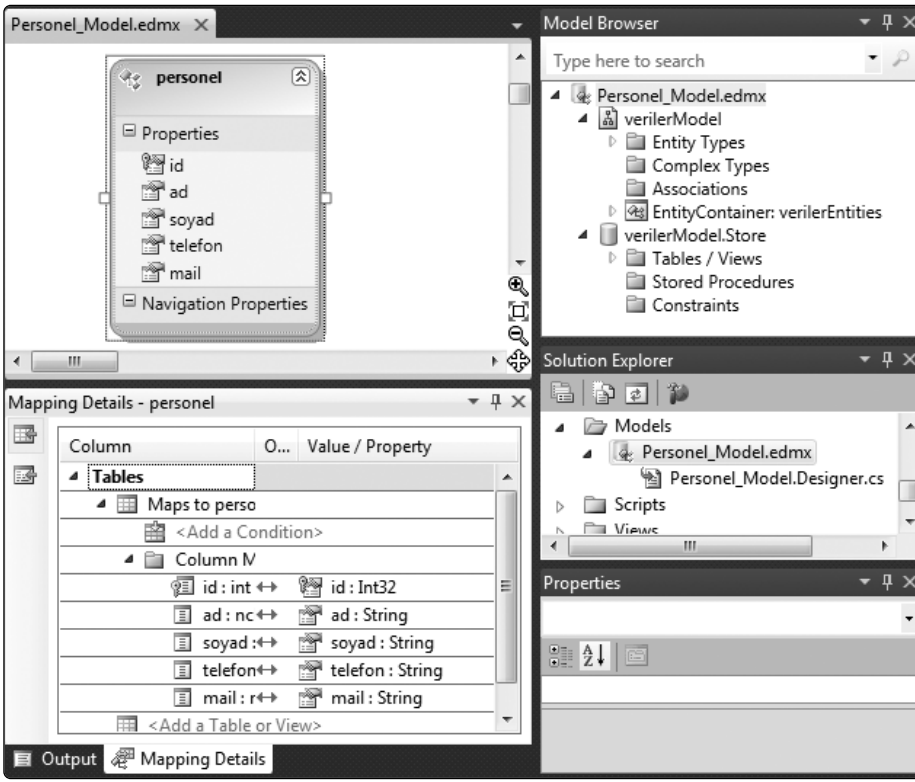


Gelen pencerede veritabanımızı seçerek, **Web.config** içerisine kaydedilecek olan **connection string** (*bağlantı cümlesi*) ismini `verilerEntities` olarak belirliyoruz.

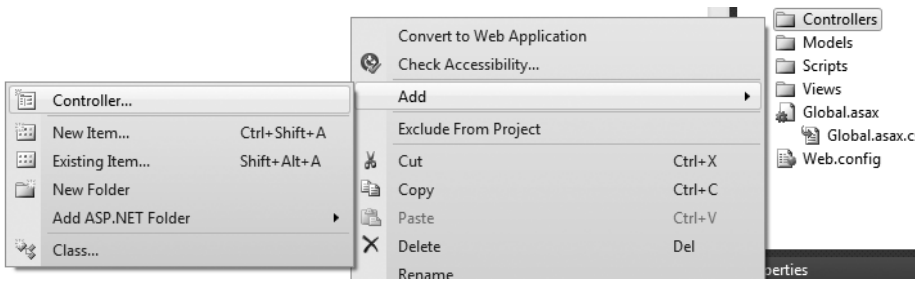


Son penceremizde **Data Model** içerisinde kullanacağımız veritabanı nesnelerini seçiyoruz. Biz örneğimiz için yalnızca `personel` isimli tablomuzu seçiyoruz. Uygulamamız içerisinde model yapısına erişebilmek için model **namespace** ismini `verilerModel` olarak belirliyoruz.

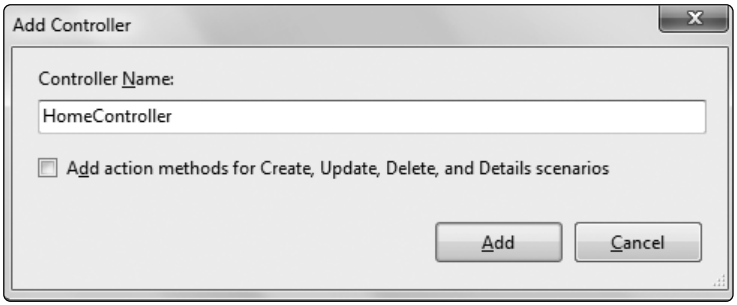
Bu işlemin ardından **Finish** butonuna tıkladığımızda `Personel_Model` isimli **Entity Data Model** oluşturulacaktır.



Bu işlemin ardından Controller'ımızı oluşturacağız. **Project** menüsünden **Add New Item/MVC 3 Controller Class** seçeneği ile ya da **Controllers** klasörüne farenin sağ tuşu ile tıklayarak **Add/Controller** menüsü ile Controller ekleme işlemini başlatıyoruz.



Gelen pencerede Controller ismini `HomeController` olarak belirliyoruz.



Veritabanındaki değerleri sadece listeleyip, kayıt, düzenleme, ekleme gibi işlemler yapmayacağımız için **Add action methods for Create**, **Update**, **Delete and Details scenarios** seçeneğini işaretlemiyoruz.

HomeController.cs dosyası içerisindeki `Index` isimli `ActionResult` metodunun kodunu aşağıdaki gibi değiştirelim:

```
private Models.verilerEntities _entities = new Models.verilerEntities();
public ActionResult Index()
{
    return View(_entities.personel.ToList());
}
```

Kodumuz içerisinde...

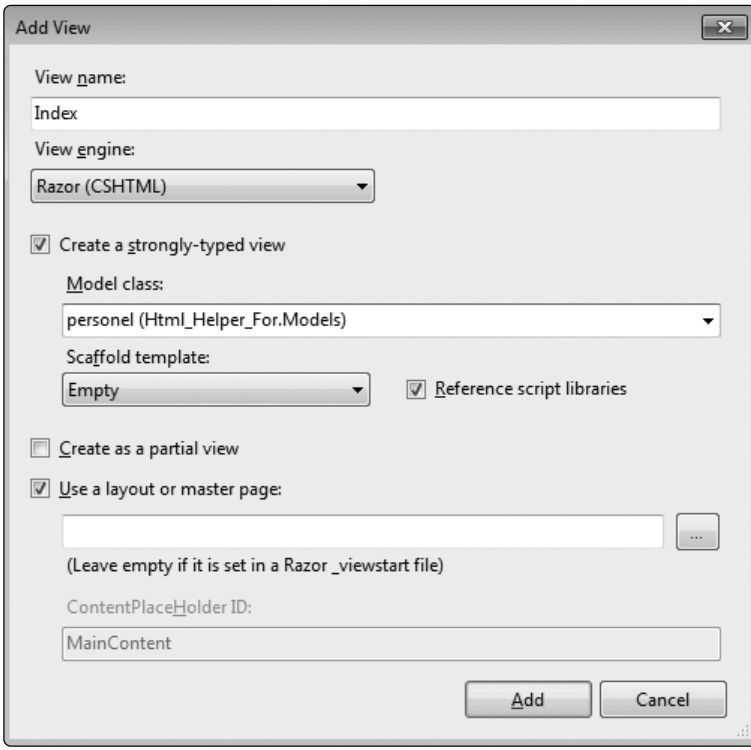
```
private Models.verilerEntities _entities = new Models.verilerEntities();
```

... satırında `_entities` Microsoft Entity Framework tarafında oluşturmuş olduğumuz veritabanı nesnelerini temsil eder. `_entities` değişkenini **Entity Designer** tarafından oluşturulan `verilerEntities` sınıfından türetiyoruz.

```
return View(_entities.personel.ToList())
```

satırında veritabanı içerisindeki personel tablosundan verileri çekiyoruz. `ToList()` metodu tablodan dönen değerleri generic collection türüne dönüştürmek amacıyla kullanılır.

`Index` isimli `ActionResult` metoduna farenin sağ tuşu ile tıklayarak **Add View** seçeneğini tıklıyoruz.



Gelen pencerede **Create a strongly-typed View** seçeneğini işaretleyerek, oluşturmuş olduğumuz personel isimli **Entity Data Model**'i seçiyoruz. **Scaffold template** seçeneğini ise **Empty** olarak belirliyoruz. Çünkü gerekli kodları biz yazacağız.

Oluşturulan `Index.cshtml` isimli View'in kodu, varsayılan olarak aşağıdaki gibidir:

```
@model Html_Helper_For.Models.personel
@{
    ViewBag.Title = "Index";
}
<h2>Index</h2>
```

Bu kodu aşağıdaki kodla değiştiriyoruz. Sonuç olarak `Index.cshtml` isimli View kodumuz aşağıdaki gibi olacak:

```
@model IEnumerable<Html_Helper_For.Models.personel>
@{
```

```
    ViewBag.Title = "Index";
}
<h2>Kayıtlar</h2>
@{var type = typeof(Html_Helper_For.Models.personel);}
@{var properties = type.GetProperties();}
<table>
    <tr>
        @foreach (System.Reflection.PropertyInfo property in properties)
        {
            <th>
                @if (property.Name != "EntityState" &&
                        property.Name != "EntityKey")
                {
                    @Html.Label(property.Name)
                }
            </th>
        }
    </tr>
@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(model => item.id)
        </td>
        <td>
            @Html.DisplayFor(model => item.ad)
        </td>
        <td>
           @Html.DisplayFor(model => item.soyad)
        </td>
        <td>
            @Html.DisplayFor(model => item.telefon)
        </td>
        <td>
            @Html.DisplayFor(model => item.mail)
        </td>
    </tr>
}
</table>
```

Kodumuz içerisinde...

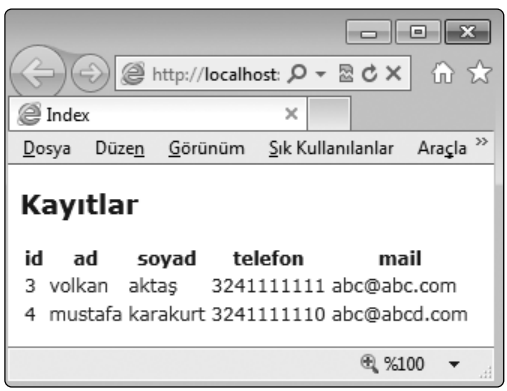
```
@{var type = typeof(Html_Helper_For.Models.personel);}
@{var properties = type.GetProperties();}
@foreach (System.Reflection.PropertyInfo property in properties)
{
    <th>
        @if (property.Name != “EntityState” &&
                property.Name != “EntityKey”)
        {
             @Html.Label(property.Name)
        }
    </th>
}
```

... satırlarında `PropertyInfo` sınıfını kullanarak **Entity Data Model** içerisinde yer alan personel tablosunun özelliklerine ve **metadata** bilgilerine erişim sağlıyoruz. Bu şekilde index sayfamız içerisinde personel tablosundaki alan isimlerini elle yazmak yerine, bu bilgilerin otomatik olarak çekilmesini sağlayacağız. Ancak bu bilgiler çekilirken `EntityState` ve `EntityKey` isimli özelliklerin gelmesini istemediğim için bu özelliklerin listelenmesini engelliyoruz ve `Name` özelliği ile yalnızca alan isimlerini `Html.Label` nesneleri içerisinde listeliyoruz.

```
@foreach (var item in Model) {
```

... satırında **Model** içerisindeki nesnelerin adedi kadar bir döngü kuruyoruz. `Html.DisplayFor` nesnesini kullanarak Model içerisindeki alan isimlerini belirterek veritabanındaki kayıtları görüntülüyoruz.

Uygulamamızı çalıştırdığımızda veritabanındaki kayıtlar listelenmektedir.



Web Helper Metotları

Web Helper metotları, `System.Web.Helpers` sınıfı altında yer alır. Web sayfalarında kullanabileceğiniz `Chart`, `WebGrid`, `WebMail` gibi nesneleri içerir. Web Helper metotlarının başlıca olanlarını inceleyelim...

Chart

Veri grafiği eklemek amacıyla kullanılır.

Kullanımı şu şekildedir:

```
Chart(width, height [, template] [, templatePath )
```

Chart Helper metodu, grafik üzerinde işlemler gerçekleştirmek ve çeşitli özellikleri değiştirmek amacıyla bazı ek metotlara sahiptir.

- **`AddTitle:`** Grafik üzerine eklenecek başlığı belirler.
- **`AddLegend:`** Grafik üzerinde görüntülenen veriler hakkında bilgi görüntülemek amacıyla kullanılır.
- **`AddSeries:`** Grafik üzerinde x ve y eksenlerinde görüntülenecek olan değerleri belirler.

  Kullanımı şu şekildedir:

  ```
  Chart.AddSeries([isim] [, chartType] [, chartArea] [, axisLabel]
  [, legend] [, markerStep] [, xValue] [, xField] [, yValues] [, yFields]
  [, options])
  ```

  `ChartType:` Grafik türünü belirtir ve şu değerleri alır: `Area`, `Bar`, `Pie`, `Column`, `Line`, `Stock`.

  Eğer grafik türü belirtilmezse, varsayılan olarak Column grafik türü görüntülenir.

  `ChartArea:` Grafiğin ekleneceği x ve y koordinatlarını belirtir.

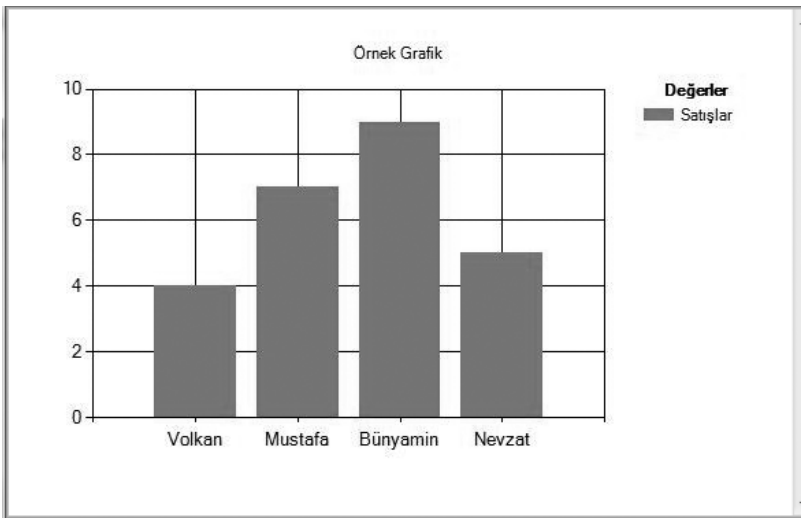
  `Options:` Grafikle ilgili çeşitli ayarların yapılabilmesini sağlar.
- **`Write:`** Grafiğin belirtilen ayarlarla sayfa içerisinde oluşturulmasını sağlar.

Örnek olarak boş bir **MVC 3 Razor** uygulaması oluşturup, `index.cshtml` isimli View içerisine aşağıdaki kodları ekleyelim:

```
@using System.Web.Helpers
@{
    ViewBag.Title = "Index";
```

```
}
@{
    var grafik = new Chart(width: 550, height: 320)
       .AddLegend("Değerler")
       .AddTitle("Örnek Grafik")
       .AddSeries(
       name: "Satışlar",
       xValue: new[] {  "Volkan", "Mustafa", "Bünyamin", "Nevzat"},
       yValues: new[] { "4", "7", "9", "5"})
       .Write();
}
```

Grafiğimizin türünü belirtmediğimiz için grafik varsayılan olarak **Column** grafik türü ile oluşturuldu.



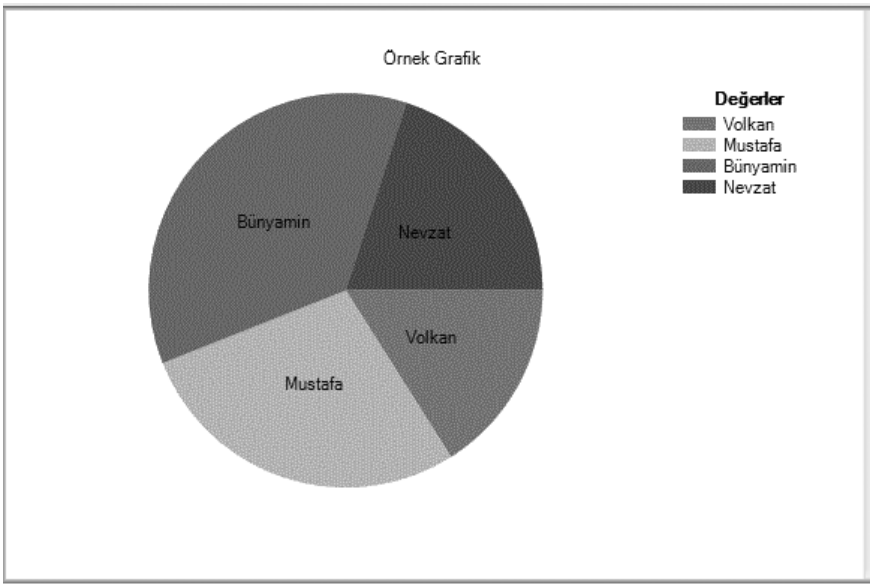
Örneğimizde herhangi bir seçenek belirtmediğimiz için grafiğimizin resim türü de varsayılan olarak `jpeg` olarak görüntülenmektedir.

Örneğimizi hem grafik türünü hem de resim kayıt türünü değiştirecek şekilde yeniden düzenleyelim.

```
@using System.Web.Helpers
@{
    ViewBag.Title = "Index";
```

```
}
@{
  var grafik = new Chart(width: 550, height: 320)
    .AddLegend("Değerler")
    .AddTitle("Örnek Grafik")
    .AddSeries(
    name: "Satışlar",chartType:"Pie",
    xValue: new[] {  "Volkan", "Mustafa", "Bünyamin", "Nevzat"},
    yValues: new[] { "4", "7", "9", "5"})
    .Write(format: "gif");
}
```

Uygulamamızın son hali aşağıdaki gibidir:



Grafik içerisindeki değerlerin otomatik olarak çekilmesi ile ilgili bir uygulama yapalım. Örneğimizde Türkçe ay isimleri otomatik olarak çekilerek karakter uzunluklarına göre grafik çizdirilmektedir.

Kodumuz aşağıdaki gibi olacak:

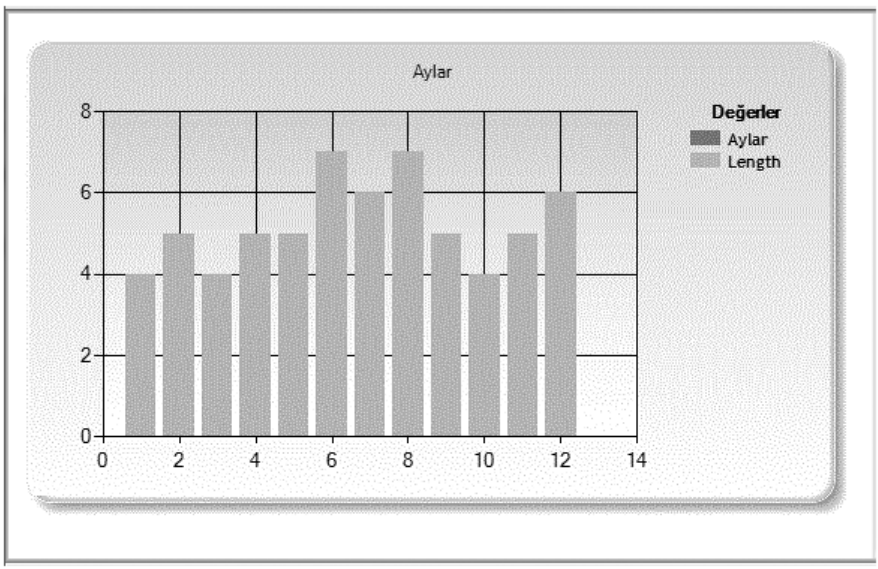
```
@using System.Web.Helpers
@{
```

```
    ViewBag.Title = "Index";
}
@{
    System.Globalization.CultureInfo culture = new
              System.Globalization.CultureInfo("tr-TR");
    System.Globalization.DateTimeFormatInfo t = culture.DateTimeFormat;
    var grafik = new Chart(width: 550, height: 320,
              theme: ChartTheme.Blue)
      .AddLegend("Değerler")
      .AddTitle("Aylar")
      .AddSeries(chartType: "column",name:"Aylar")
      .DataBindTable(t.MonthNames)
      .Write(format: "gif");
 }
}
```

Örneğimizde `CultureInfo` sınıfını kullanarak tarih zaman formatını Türkçe olarak değiştirdik. `DataBindTable` metodu ile de ay isimlerini çekerek grafik içerisinde görüntüledik.

```
theme: ChartTheme.Blue
```

... özelliği ile ise grafik şablonunu değiştirerek aşağıda görüldüğü gibi grafiğimizin görüntülendiği alana gölgeli ve 3 boyutlu bir görünüm verdik. `ChartTheme` özelliğinin alabileceği değerlerin tümü şunlardır: `Blue`, `Green`, `Yellow`, `Vanilla`, `Vanilla3D`.



WebMail

SMTP mail göndermek amacı ile kullanılır. Mail gönderirken kullanılabilecek bazı yardımcı metotlar ve özellikler aşağıdaki gibidir:

- **`SmtpServer:`** Mail gönderme işleminde kullanılacak olan SMTP adresini belirtir.
- **`UserName:`** Mail gönderme işleminde kullanılacak olan mail adresinin kullanıcı adı.
- **`Password:`** Mail gönderme işleminde kullanılacak olan mail adresinin şifresi.
- **`EnableSsl:`** Mail gönderme işleminde kullanılacak olan mail adresi SSL protokolünü kullanıyorsa, bu özelliğe `true` değeri aktarılmalıdır.
- **`SmtpPort:`** Mail gönderme işleminde kullanılacak olan mail adresinin kullandığı port numarası.
- **`Send:`** Belirtilen ayarlara göre mail gönderme işlemini gerçekleştirir. Kullanımı aşağıdaki gibidir:

  ```
  Send("Gönderilecek mail adresi", konu, icerik, "Göndericinin mail adresi");
  ```

Örnek olarak Gmail üzerinden mail gönderme işlemini gerçekleştirecek bir kod yazalım.

```
@{
    WebMail.SmtpServer = "smtp.gmail.com";
    WebMail.UserName = "kullanıcı adı";
    WebMail.Password = "şifre";
    WebMail.EnableSsl = true;
    WebMail.SmtpPort = 587;
    WebMail.Send("Gönderilecek mail adresi", "deneme", "örnek içerik",
"Göndericinin mail adresi");
     }
```

Crypto

Şifreleme ve şifre çözmede kullanılan metotlardan birisi de **Hash** metotlarıdır. **Hash** metotlarında bir anahtar değere göre özel bir algoritma ile kaynak metin üzerinde şifreleme ve şifre çözme işlemi gerçekleştirilir. MVC içerisinde de **Crypto Helper**'ını kullanarak **Hash** işlemleri gerçekleştirilmektedir. **Crypto Helper** metodu ile birlikte kullanılabilecek yardımcı metotlar aşağıdaki gibidir:

- **`GenerateSalt:`** Rastgele byte değerlerinden oluşan diziler oluşturmak amacıyla kullanılır.

- **Hash(Byte(), String):** Belirtilen bir byte dizisi için hash değeri geriye döndürür.
- **Hash(String, String):** Belirtilen bir string değer için hash değeri geriye döndürür.
- **HashPassword:** Belirtilen bir şifre değeri için RFC 2898 hash değerini döndürür.
- **SHA1:** Belirtilen bir string değer için SHA-1 hash değerini döndürür.
- **SHA256:** Belirtilen bir string değer için SHA-256 hash değerini döndürür.
- **VerifyHashedPassword:** Belirtilen RFC 2898 hash ve password değerlerini karşılaştırarak aynı olup olmadığını belirtir.

Bir metin kutusuna girilen içeriği şifreleyen örnek bir uygulama yapalım. Boş bir MVC 3 Razor uygulaması oluşturuyoruz. Öncelikle HomeController'ı oluşturarak içerisine aşağıdaki kodları ekleyelim:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Helpers;
namespace Helper_crypto.Controllers
{
    public class HomeController : Controller
    {
        // GET: /Home/
        public ActionResult Index()
        {
            return View();
        }
        [HttpPost]
        public ActionResult Sifre()
        {
          string Password = Request[“GirilenSifre”];
          if (Password.Length>0)
          {
            string salt = Crypto.GenerateSalt();
            string hashedPass = Crypto.HashPassword(Password);
            bool esit = Crypto.VerifyHashedPassword(hashedPass, Password);
```

```
            System.Text.StringBuilder s = new
                        System.Text.StringBuilder();
            s.Append("<fieldset>");
            s.Append("<legend style='font-size: large; font-weight: bold'>");
            s.Append("Dönen Değer </legend>");
            s.Append("Orjinal Şifre: ");
            s.Append(Password + "<br />");
            s.Append("Geçerli Salt Değeri: ");
            s.Append(salt + "<br />");
            s.Append("Hash Değeri: ");
            s.Append(hashedPass + "<br />");
            s.Append("Hash değeri ve Şifre Uyumlu mu? ");
            s.Append(esit );
            return Content(s.ToString());
          }
          else
            return Content("Lütfen Şifre Alanını Boş Bırakmayın!");
          }
    }
}
```

Örneğimizde `Crypto` Helper'ını kullanabilmek için `System.Web.Helpers` sınıfını projemize ekledik.

```
string salt = Crypto.GenerateSalt()
```

... satırında `GenerateSalt` metodunu kullanarak rasgele byte değerlerinden oluşan `salt` isimli bir değişken tanımladık.

```
string hashedPass = Crypto.HashPassword(Password)
```

... satırında girilecek olan şifre değerini Hash işlemine tabi tutarak `hashedPass` isimli bir `RFC 2898` hash değeri elde ettik.

```
bool esit = Crypto.VerifyHashedPassword(hashedPass, Password)
```

... satırında ise girilen şifre değeri ile `hashedPass` değişkeni içerisinde tutulan hash değeri karşılaştırılarak eğer aynı ise değişkenimiz `true` değerini, farklı ise `false` değerini alır.

```
System.Text.StringBuilder s = new System.Text.StringBuilder();
```

... satırında `s` isimli bir `StringBuilder` nesnesi oluşturuyoruz.

`s.Append`

... metoduyla da oluşturmuş olduğumuz `s` isimli `StringBuilder` nesnesi içerisine `index` sayfası içerisinde görüntüleyeceğimiz değerleri ekliyoruz.

Verilerimizi index sayfası içerisine Ajax yardımıyla görüntüleyeceğimiz için öncelikle `_Layout.cshtml` dosyası içerisinde `<head> </head>` tagları arasına;

```
script src="@Url.Content("~/Scripts/jquery.unobtrusive-ajax.js")"
type="text/javascript"></script>
```

... satırını ekliyoruz.

Son olarak **Index.cshtml** sayfasının kodlarını yazıyoruz. Kodumuz aşağıdaki gibi olacak:

```
@{
    ViewBag.Title = "Index";
}
<h2>Crypto Helper Örneği</h2>

@{
    HtmlHelper.UnobtrusiveJavaScriptEnabled = true;
}
<fieldset>
<legend>Crypto Helper</legend>
 @using (Ajax.BeginForm("Sifre",
            new AjaxOptions { HttpMethod = "POST",
            UpdateTargetId = "sonuc",
            InsertionMode = InsertionMode.Replace
            }))
   {
        @Html.Label("Şifreyi giriniz:")
        <br />
        @Html.TextBox("GirilenSifre")
        <br />
        <input type="submit" value="İşlem Yap" />
    }
</fieldset>
<div id="sonuc"/>
```

Örneğimizde...

```
HtmlHelper.UnobtrusiveJavaScriptEnabled = true;
```

... satırında unobtrusive JavaScript özelliğini aktifleştiririz. UnobtrusiveJavaScript yapısı HTML ve JavaScript kodlarını birbirinden ayırmak amacıyla kullanılır.

```
@using (Ajax.BeginForm("Sifre",
            new AjaxOptions { HttpMethod = "POST",
            UpdateTargetId = "sonuc",
            InsertionMode = InsertionMode.Replace
            }))
```

... satırlarında HomeController içerisinde bulunan Sifre isimli ActionResult metodunu çağırıyoruz. Dönen değerleri kodumuzun en altında...

```
<div id="sonuc"/>
```

... satırı ile oluşturmuş olduğumuz

sonuc isimli div tag'ı içerisinde görüntülüyoruz.

```
InsertionMode = InsertionMode.Replace
```

... satırında ise dönen değerlerin her defasında ne şekilde ekleneceğini belirliyoruz. Replace şeklinde belirttiğimizde her görüntüleme işlemi öncesinde var olan değerler silinerek, yerine yeni değerler yazılır.

InsertionMode özelliği ayrıca InsertBefore ve InsertAfter şeklinde iki değer daha alır. InsertBefore seçildiğinde yeni değer, görüntülenen değerden daha önce yazılır. InsertAfter seçildiğinde ise daha sonra yazılır.

Örneğimizi çalıştırıp, TextBox içerisine örnek bir değer girelim. Formun alt tarafında sonucu görebiliriz.



WEBIMAGE

Resim görüntüleme, yükleme ve resimler üzerinde çeşitli efektler ve işlemler gerçekleştirmek için çeşitli özelliklere ve yardımcı metotlara sahiptir. Bu özellik ve yardımcı metotların önemlilerine değineceğiz.

- **`Resize:`** Resmi yeniden boyutlandırmak amacıyla kullanılır. Kullanımı aşağıdaki gibidir.

  ```
  WebImage.Resize(int width, int height, bool preserveAspectRatio,
  bool preventEnlarge)
  ```

  `preserveAspectRatio:` true değerini aldığında boyutlandırma işlemi yapılırken genişlik ve yükseklik değerleri birbiri ile orantılanarak değişir. false değerini aldığında ise genişlik ve yükseklik birbirinden bağımsız olarak değişir.

  `preventEnlarge:` true değerini aldığında yeniden boyutlandırma işlemi sonucunda elde edilen resim, orijinal resimden daha büyükse resim yeniden boyutlandırılmaz. `false` değerini aldığında ise herhangi bir kontrol yapılmadan resim yeniden boyutlandırılır.

- **`Save:`** Resmi kaydetmek amacıyla kullanılır. Kullanımı aşağıdaki gibidir.

  ```
  WebImage.Save([Optional, VarsayılanParametreDeğeri (null)]
  string filePath, [Optional, VarsayılanParametreDeğeri (null)]
  string imageFormat)
  ```

  `filePath:` Dosyanın kaydedileceği klasör yolunu belirtir.

imageFormat: Dosyanın kaydedileceği dosya formatını belirtir.

- **AddTextWatermark:** Resim üzerine yazı şeklinde filigran eklemek amacıyla kullanılır. Kullanımı aşağıdaki gibidir:

```
WebImage.AddTextWatermark(string text,
    [Optional, VarsayılanParametreDeğeri (“Black”)] string fontColor,
    [Optional, VarsayılanParametreDeğeri (12)] int fontSize,
    [Optional, VarsayılanParametreDeğeri (“Regular”)] string fontStyle,
    [Optional, VarsayılanParametreDeğeri (“Microsoft Sans Serif”)]
           string fontFamily,
    [Optional, VarsayılanParametreDeğeri (“Right”)] string
           horizontalAlign,
    [Optional, VarsayılanParametreDeğeri (“Bottom”)] string
           verticalAlign,
    [Optional, VarsayılanParametreDeğeri (100)] int opacity,
    [Optional, VarsayılanParametreDeğeri (5)] int padding)
```

Text: Filigran yazısı olarak kullanılacak metni belirtir.

horizontalAlign: Yatay hizalama biçimini belirtir.

verticalAlign: Dikey hizalama biçimini belirtir.

opacity: Saydamlık değerini belirtir.

padding: Eklenen filigran yazı ile resmin kenarlığı arasındaki mesafeyi belirler.

- **AddImageWatermark:** Resim üzerine başka bir resmi filigran olarak eklemek amacıyla kullanılır. Kullanımı aşağıdaki gibidir:

```
WebImage.AddImageWatermark( string watermarkImageFilePath,
    [Optional, VarsayılanParametreDeğeri (0)] int width,
    [Optional, VarsayılanParametreDeğeri (0)] int heigth,
    [Optional, VarsayılanParametreDeğeri (“Right”)] string horizontalAlign,
    [Optional, VarsayılanParametreDeğeri (“Bottom”)] string verticalAlign,
    [Optional, VarsayılanParametreDeğeri (100)] int opacity,
    [Optional, VarsayılanParametreDeğeri (5)] int padding)
```

watermarkImageFilePath: Filigran resmi olarak kullanılacak olan resmin dosya yolunu belirtir.

- **Crop:** Resmin belirtilen kısmını keserek yeni bir resim elde etmek amacıyla kullanılır.

Kullanımı aşağıdaki gibidir.

```
WebImage.Crop ([Optional, VarsayılanParametreDeğeri (0)] int top,
    [Optional, VarsayılanParametreDeğeri (0)] int left,
    [Optional, VarsayılanParametreDeğeri (0)] int bottom,
    [Optional, VarsayılanParametreDeğeri (0)] int right)
```

`top`, `left`, `bottom` ve `right` değerleri sırasıyla üst, sol, alt ve sağ koordinat değerlerini belirtir.

- **`FlipHorizontal:`** Resmi yatay yönde tersine çevirmek amacıyla kullanılır. Kullanımı aşağıdaki gibidir.

```
WebImage.FlipHorizontal()
```

- **`FlipVertical:`** Resmi dikey yönde tersine çevirmek amacıyla kullanılır. Kullanımı aşağıdaki gibidir.

```
WebImage.FlipVertical()
```

- **`RotateLeft:`** Resmi sola döndürmek amacıyla kullanılır. Kullanımı aşağıdaki gibidir.

```
WebImage.RotateLeft()
```

- **`RotateRight:`** Resmi sağa döndürmek amacıyla kullanılır. Kullanımı aşağıdaki gibidir.

```
WebImage.RotateRight()
```

`WebImage` Helper metodunu kullanarak resim yükleyen ve resim üzerinde yukarıda saydığımız işlemleri gerçekleştiren örnek bir uygulama yapalım. Boş bir **MVC 3 Razor** uygulaması oluşturuyoruz. Öncelikle `HomeController`'ı oluşturarak içerisine aşağıdaki kodları ekleyelim:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Helpers;
using System.IO;
namespace Helper_WebImage.Controllers
{
```

```
public class HomeController : Controller
{
  // GET: /Home/
  public ActionResult Index()
  {
      return View();
  }
  public ActionResult FiligranYaziEkle(string dosyaUrl)
  {
    var resim = new WebImage(Server.MapPath(Path.Combine("~.",
              dosyaUrl)));
    resim.AddTextWatermark("www.kodlab.com", fontSize: 30,
              fontColor: "", horizontalAlign: "Right",
              verticalAlign: "Bottom", opacity: 70);
    resim.Save(Server.MapPath(dosyaUrl));
    return Redirect(dosyaUrl);
  }
  public ActionResult FiligranResimEkle(string dosyaUrl)
  {
    var resim = new WebImage(Server.MapPath(Path.Combine("~.",
              dosyaUrl)));
    resim.AddImageWatermark(Server.MapPath(Path.Combine("~.",
              @"\Content\images\filigran.gif")), width: 300,
              height: 50, horizontalAlign: "Center",
              verticalAlign: "Bottom", opacity: 100, padding: 10);
    resim.Save(Server.MapPath(dosyaUrl));
    return Redirect(dosyaUrl);
  }
  public ActionResult Crop(string dosyaUrl)
  {
    var resim = new WebImage(Server.MapPath(Path.Combine("~.",
              dosyaUrl)));
    var yukseklik = resim.Height;
    var genislik = resim.Width;
    if (genislik > yukseklik)
    {
        var yatayCrop = (genislik - yukseklik) / 2;
        resim.Crop(0, yatayCrop, 0, yatayCrop);
    }
    else if (yukseklik > genislik)
    {
```

```
            var dikeyCrop = (yukseklik - genislik) / 2;
            resim.Crop(dikeyCrop, 0, dikeyCrop, 0);
        }
        resim.Save(Server.MapPath(dosyaUrl));
        return Redirect(dosyaUrl);
      }
      public ActionResult yatayCevir(string dosyaUrl)
      {
        var resim = new WebImage(Server.MapPath(Path.Combine("~.",
                  dosyaUrl)));
        resim.FlipHorizontal();
        resim.Save(Server.MapPath(dosyaUrl));
        return Redirect(dosyaUrl);
      }
      public ActionResult dikeyCevir(string dosyaUrl)
      {
        var resim = new WebImage(Server.MapPath(Path.Combine("~.",
                  dosyaUrl)));
        resim.FlipVertical();
        resim.Save(Server.MapPath(dosyaUrl));
        return Redirect(dosyaUrl);
      }
      public ActionResult solaDondur(string dosyaUrl)
      {
        var resim = new WebImage(Server.MapPath(Path.Combine("~.",
                  dosyaUrl)));
        resim.RotateLeft();
        resim.Save(Server.MapPath(dosyaUrl));
        return Redirect(dosyaUrl);
      }
      public ActionResult sagaDondur(string dosyaUrl)
      {
        var resim = new WebImage(Server.MapPath(Path.Combine("~.",
                  dosyaUrl)));
        resim.RotateRight();
        resim.Save(Server.MapPath(dosyaUrl));
        return Redirect(dosyaUrl);
      }
  }
}
```

Örneğimizde `WebImage` Helper'ını kullanabilmek için `System.Web.Helpers` sınıfını projemize ekledik. Kodumuz içerisinde resim üzerinde gerçekleştireceğimiz her bir işlem için bir `ActionResult` metodu tanımladık. `ActionResult` metotlarını tanımlarken `dosyaUrl` şeklinde tanımladığımız değişken ile index sayfasından gelen resim dosyası url'sini aldık.

```
var resim = new WebImage(Server.MapPath(Path.Combine("~.", dosyaUrl)))
```

... satırında `resim` adında bir `WebImage` nesnesi oluşturuyoruz. `Server.MapPath` metodu ile tam dosya adını alıyoruz. `Path.Combine` metodu ile ise dosya yolunu oluşturuyoruz. Bu metotları kullanabilmek için `System.IO` sınıfını projemize dahil etmemiz gerekmekte.

Resim üzerinde gerçekleştirdiğimiz işlemlerin bitmesinin ardından...

```
resim.Save(Server.MapPath(dosyaUrl))
```

... satırında Save metodunu kullanarak resmimizi kaydedip...

```
return Redirect(dosyaUrl)
```

... satırında web sayfasının resim dosyasının url'sine yönlenmesi sağlanmaktadır.

Controller kodumuzu yazmamızın ardından `Index` isimli yeni bir View ekleyerek oluşturmuş olduğumuz **Index.cshtml** dosyasının kodlarını yazalım. Kodumuz aşağıdaki gibi olacak:

```
@{
    ViewBag.Title = "Index";
}
<h2>WebImage Örneği</h2>
@{
  WebImage resim = null;
  var dosya = "";
  var dosyaPath = "";
  var dosyaThumbnailPath = "";
  var dosyaboyut = 0;
  if (IsPost)
  {
    resim = WebImage.GetImageFromRequest();
    if (resim != null)
    {
        dosya = Guid.NewGuid().ToString() + "_" +
```

```
            Path.GetFileName(resim.FileName);
        dosyaPath = @"\Content\images\" + dosya;
        resim.Save(@"~" + dosyaPath);
        dosyaThumbnailPath = @"\Content\images\thumbnail\" + dosya;
        resim.Resize(width: 100, height: 100,
             preserveAspectRatio: true,preventEnlarge: true);
        resim.Save(@"~" + dosyaThumbnailPath);
        dosyaboyut = resim.GetBytes().Length / 1024;
    }
  }
}
<form action="" method="post" enctype="multipart/form-data">
<fieldset>
  <legend>Resim Yükleme</legend>
  <label for="Image">Resim</label>
  <input type="file" name="Image" />
  <br/>
  <input type="submit" value="Yükle" />
</fieldset>
</form>
<fieldset>
  <legend>Thumbnail</legend>
  @if(dosyaPath != ""){
    <div>
        <img src="@Href(dosyaThumbnailPath)" alt="Thumbnail resmi" />
        <br/>
        <a href="@Href(dosyaPath)" target="_self">
            Tam boyut için lütfen tıklayınız
        </a>
        <p>
        Resim boyutu  @dosyaboyut (kb)
        </p>
        <fieldset>
        <legend>Resim İşlemleri</legend>
        <p>
          <a href="@Url.Action("FiligranYaziEkle", "Home",
                  new { dosyaUrl=dosyaPath}) "target="_Self">
             Filigran yazısı eklemek için lütfen tıklayınız
          </a>
        </p>
```

```
        <p>
          <a href="@Url.Action("FiligranResimEkle", "Home",
                new { dosyaUrl=dosyaPath}) "target="_Self">
             Filigran resmi eklemek için lütfen tıklayınız
          </a>
        </p>
        <p>
          <a href="@Url.Action("Crop", "Home",
                new { dosyaUrl=dosyaPath}) "target="_Self">
             Resmi kırpmak için lütfen tıklayınız
          </a>
        </p>
        <p>
          <a href="@Url.Action("yatayCevir", "Home",
                new { dosyaUrl=dosyaPath}) "target="_Self">
             Resmi yatay yönde çevirmek için lütfen tıklayınız
          </a>
        </p>
        <p>
          <a href="@Url.Action("dikeyCevir", "Home",
                new { dosyaUrl=dosyaPath}) "target="_Self">
             Resmi dikey yönde çevirmek için lütfen tıklayınız
          </a>
        </p>
        <p>
          <a href="@Url.Action("solaDondur", "Home",
                new { dosyaUrl=dosyaPath}) "target="_Self">
             Resmi sola döndürmek için lütfen tıklayınız
          </a>
        </p>
        <p>
          <a href="@Url.Action("sagaDondur", "Home",
                new { dosyaUrl=dosyaPath}) "target="_Self">
             Resmi sağa döndürmek için lütfen tıklayınız
          </a>
        </p>
        </fieldset>
    </div>
  }
</fieldset>
```

Örneğimizde...

```
resim = WebImage.GetImageFromRequest()
```

... satırında resim değişkenine form içerisinde upload edilen resmi aktardık.

```
dosya = Guid.NewGuid().ToString() + "_" + Path.GetFileName(resim.FileName)
```

satırında ise resim değişkeninden, dosya adlı, benzersiz bir geçici isme sahip bir değişken oluşturuyoruz.

```
dosyaPath = @"\Content\images\" + dosya;
resim.Save(@"~" + dosyaPath);
```

... satırlarında ise resmimizi `content\images\` klasörü altına kaydediyoruz.

```
dosyaThumbnailPath = @"\Content\images\thumbnail\" + dosya;
resim.Resize(width: 100, height: 100, preserveAspectRatio: true,
        preventEnlarge: true);
resim.Save(@"~" + dosyaThumbnailPath);
```

... satırlarında ise resmimizi yeniden boyutlandırarak thumbnail görünümü için `Content\images\thumbnail` klasörü altına kaydediyoruz.

```
dosyaboyut = resim.GetBytes().Length / 1024
```

satırında ise resmimizin kilobyte cinsinden boyutunu alıyoruz.

Oluşturmuş olduğumuz thumbnail görünümünü...

```
<img src="@Href(dosyaThumbnailPath)" alt="Thumbnail resmi" />
```

... satırında img tag'ı içerisinde görüntülüyoruz.

```
<a href="@Href(dosyaPath)" target="_self">
   Tam boyut için lütfen tıklayınız
</a>
```

Bu satırlarda oluşturmuş olduğumuz bir link yardımıyla, linke tıklandığında resmin orijinal boyutunun görüntülenmesini sağlıyoruz.

Resim üzerinde geçekleştireceğimiz işlemler için ise...

```
<a href="@Url.Action("FiligranYaziEkle", "Home",
            new { dosyaUrl=dosyaPath}) "target="_Self">
   Filigran yazısı eklemek için lütfen tıklayınız
</a>
```

... örneğinde olduğu gibi bir link oluşturup, `Url.Action` metodu yardımıyla `HomeController` içerisindeki ilgili ActionResult metodunu çağırıyoruz. `ActionResult` metodunu çağırırken...

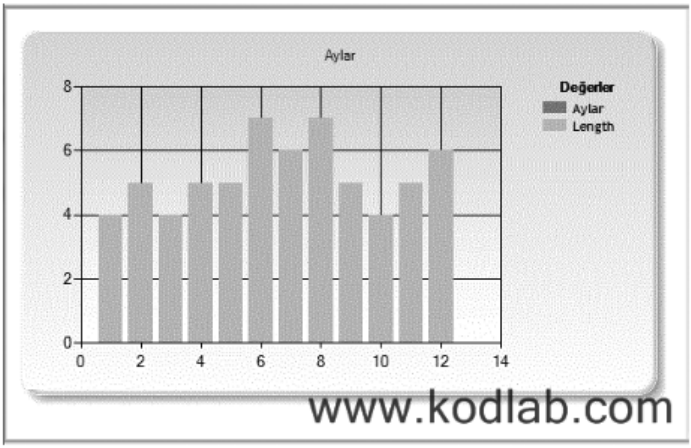
```
new { dosyaUrl=dosyaPath}
```

... şeklinde ilgili `ActionResult` içerisindeki dosyaURL parametresine `dosyaPath` değişkeninin değerini gönderiyoruz.

Uygulamamızı çalıştıralım ve örnek bir resim yükleyelim.



Resim üzerinde yapılan işlemlerden örnek olarak yazı ile bir filigran ekleyelim.



WebGrid

Html table nesnesini kullanarak bir web sayfası içerisinde verileri görüntüler.

```
var değişkenismi = new WebGrid(Model)
```

... şeklinde tanımlanır.

`WebGrid` Helper metodu ile birlikte kullanılan özellik ve yardımcı metotları inceleyelim.

- **`GetHtml:`** Belirtilen özellik ve niteliklere göre bir WebGrid'i HTML çıktısı olarak üretir.

dbo.personel: Tab...DATA\VERILER.MDF)

Column Name	Data Type	Allow Nulls
id	int	☐
ad	nchar(50)	☑
soyad	nchar(50)	☑
telefon	nchar(15)	☑
mail	nchar(50)	☑

Örnek olarak en basit şekliyle bir WebGrid oluşturalım. Öncelikle yeni bir SQL veritabanı oluşturuyoruz. Daha önceki örneklerimizde kullandığımız `veriler.mdf` isimli veritabanını kullanıyoruz. Alan yapısı yandaki gibiydi.

Yeni bir Entity Data Model oluşturarak ismini `verilerEntities` olarak belirledik.

`HomeController` içerisine aşağıdaki kodları yazıyoruz:

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

```
using System.Web;
using System.Web.Mvc;
namespace Web_Helper_WebGrid.Controllers
{
  public class HomeController : Controller
  {
       // GET: /Home/
       private Models.verilerEntities _entities = new
             Models.verilerEntities();
       public ActionResult Index()
       {
             return View(_entities.personel.ToList());
       }
  }
}
```

`Index.cshtml` isimli View kodumuz ise aşağıdaki gibi olacak:

```
@{
     ViewBag.Title = "Index";
}
<!DOCTYPE html>
<html>
<body>
@{
     var grid = new WebGrid(Model);
     @grid.GetHtml();
}
</body>
</html>
```

Uygulamamızı çalıştırdığımızda veritabanındaki kayıtların `WebGrid` içerisine listelendiğini görürüz.

ad	id	mail	soyad	telefon
volkan	3	abc@abc.com	aktaş	3241111111
mustafa	4	abc@abcd.com	karakurt	3241111110
bünyamin	5	abc@abcde.com	karahan	3241111112
irfan	6	ab@abcde.com	mergan	3241111113
hale	7	abc@abcd.com	katmer	3241111114
ismail	8	abc@bcde.com	güven	3241111115
enver	9	abc@abcd.com	arslan	3241111116
ibrahim	10	a@abcde.com	aşı	3241111117

- **`Column:`** Yukarıdaki ekran görüntüsünde görüldüğü gibi sütun isimleri sırasız olarak listelendi. Bunu düzenlemek için `Column` özelliğini kullanabiliriz. Column özelliği sütun isimlerini manuel olarak eklemek amacıyla kullanılır. **Index.cshtml** dosyası içerisindeki...

```
@grid.GetHtml();
```

... satırını...

```
@grid.GetHtml(columns: grid.Columns(grid.Column("id"),
                       grid.Column("ad"),grid.Column("soyad"),
                       grid.Column("telefon"),grid.Column("mail")));
```

... şeklinde yeniden düzenleyelim. Görüldüğü gibi veritabanı içerisindeki alan isimlerini istediğimiz sırada elle yazdık. Uygulamamızı yeniden çalıştırdığımızda artık sütunların belirttiğimiz sırada listelendiğini görürüz.

id	ad	soyad	telefon	mail
3	volkan	aktaş	3241111111	abc@abc.com
4	mustafa	karakurt	3241111110	abc@abcd.com
5	bünyamin	karahan	3241111112	abc@abcde.com
6	irfan	mergan	3241111113	ab@abcde.com
7	hale	katmer	3241111114	abc@abcd.com
8	ismail	güven	3241111115	abc@bcde.com
9	enver	arslan	3241111116	abc@abcd.com
10	ibrahim	aşı	3241111117	a@abcde.com

- **`canPage:`** `true` değerini aldığında veritabanındaki kayıtlar sayfalar halinde listelenir. `false` değerini aldığında ise tüm kayıtlar tek bir sayfada listelenir.
- **`rowsPerPage:`** Her bir sayfada kaç satır veri görüntüleneceğini belirtir.

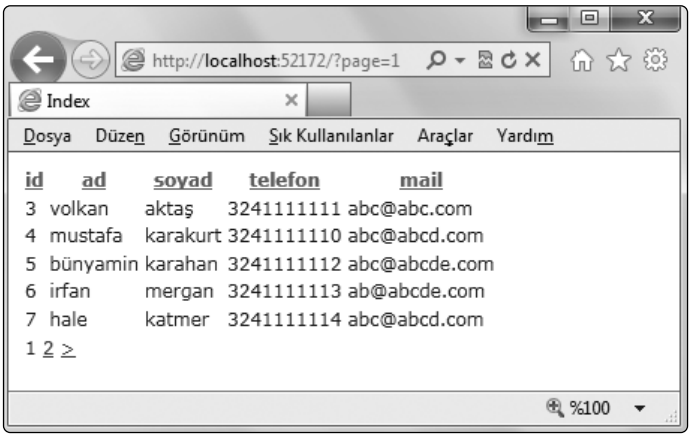
Index.cshtml dosyası içerisindeki...

```
var grid = new WebGrid(Model);
```

... satırını...

```
var grid = new WebGrid(Model,canPage: true, rowsPerPage: 5);
```

... şeklinde yeniden düzenleyelim. Uygulamamızı yeniden çalıştırdığımızda artık veritabanındaki kayıtların sayfalar halinde ve her bir sayfada 5'er kayıt olacak şekilde listelendiğini görürüz.



- **Pager:** Sayfalama işlemi esnasında sayfalar arasında hareket etmek için kullanılacak linklerle ilgili özellikleri ve ayarları belirler.

  Kullanımı aşağıdaki gibidir:

```
WebGrid.Pager([Optional,VarsayılanParametreDeğeri
                  ("WebGridPagerModes.Numeric |
                  WebGridPagerModes.NextPrevious ")]
                  WebGridPagerModes mode,
    [Optional, VarsayılanParametreDeğeri (null)] string firstText,
    [Optional, VarsayılanParametreDeğeri (null)] string previousText,
    [Optional, VarsayılanParametreDeğeri (null)] string nextText,
    [Optional, VarsayılanParametreDeğeri (null)] string lastText,
    [Optional, VarsayılanParametreDeğeri (5)] int numericLinksCount)
```

WebGridPagerModes: Sayfalama işleminin nasıl gerçekleştireleceğini belirler. Şu değerleri alabilir:

- **FirstLast:** Yalnızca ilk sayfa ve son sayfa linkleri görüntülenir.

- **NextPrevious:** Yalnızca önceki sayfa ve sonraki sayfa linkleri görüntülenir.

- **Numeric:** Yalnızca sayfa numaralarını gösteren linkler görüntülenir.

- **All:** Yukarıdaki seçeneklerin hepsi birden görüntülenir.

firstText: İlksayfa yazısı.

previousText: Önceki sayfa yazısı.

nextText: Sonraki sayfa yazısı.

lastText: Son sayfa yazısı.

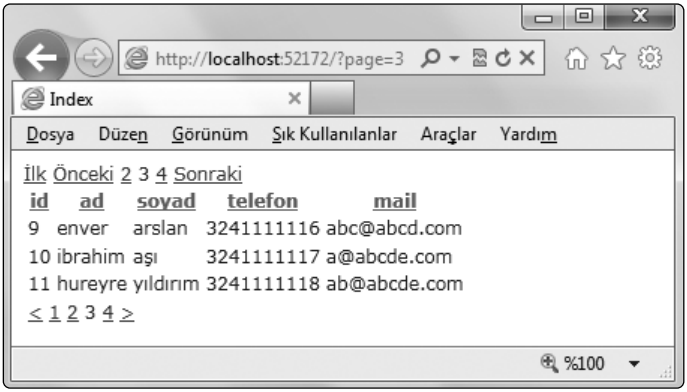
numericLinksCount: Veriler listelenirken görüntülenecek olan sayfa numarası linklerinin sayısını belirtir.

Index.cshtml dosyası içerisindeki WebGrid oluşturma kodunu aşağıdaki gibi düzenleyelim:

```
var grid = new WebGrid(Model,canPage: true, rowsPerPage: 3);
@grid.Pager(WebGridPagerModes.All,previousText:"Önceki",
      nextText:"Sonraki",firstText:"İlk",
      lastText:"Son", numericLinksCount:3);
@grid.GetHtml(columns: grid.Columns(grid.Column("id"),
      grid.Column("ad"),grid.Column("soyad"),
      grid.Column("telefon"),grid.Column("mail")));
```

`rowsPerPage` özelliğini `3` olarak belirleyerek her sayfada `3` kayıt görüntülenmesini sağladık. `WebGridPagerModes` özelliğini `All` olarak belirleyerek sayfada hem sayfa numarası linklerinin hem de *"Önceki, Sonraki, İlk, Son"* şeklinde kendi belirlediğimiz sayfa yönlendirme linklerinin görüntülenmesini sağladık. `numericLinksCount` özelliğini `3` olarak belirleyerek sayfa numarası linklerinden `3` adedinin görüntülenmesini sağladık.

WebGrid'in son hali aşağıdaki gibidir:



- **`DefaultSort:`** Varsayılan sıralama işleminin hangi sütuna göre yapılacağını belirler.

  WebGrid kodumuzu aşağıdaki gibi düzenleyelim:

```
var grid = new WebGrid(Model,canSort:true,defaultSort:"ad");
@grid.GetHtml(columns: grid.Columns(grid.Column("id"),
      grid.Column("ad"),grid.Column("soyad"),
      grid.Column("telefon"),grid.Column("mail")));
```

  Kodumuzda sıralama işlemini ad sütununa göre gerçekleştirdik.

id	ad	soyad	telefon	mail
5	bünyamin	karahan	3241111112	abc@abcde.com
12	enes	yıldırım	3241111119	abc@abcde.com
9	enver	arslan	3241111116	abc@abcd.com
7	hale	katmer	3241111114	abc@abcd.com
11	hureyre	yıldırım	3241111118	ab@abcde.com
10	ibrahim	aşı	3241111117	a@abcde.com
6	irfan	mergan	3241111113	ab@abcde.com
8	ismail	güven	3241111115	abc@bcde.com
4	mustafa	karakurt	3241111110	abc@abcd.com
13	suat	üre	3241111121	aaa@abcde.com

1 2 ≥

- **`SortColumn:`** WebGrid içerisinde sıralama işleminin hangi sütuna göre yapıldığı bilgisini verir.

- **SortDirection:** Sıralama işleminin yönünü belirtir. Aşağıdaki değerleri alır:

  Ascending: Sıralama işlemi küçükten büyüğe doğru yapılır.

  Descending: Sıralama işlemi büyükten küçüğe doğru yapılır.

Örnek olarak sıralama işleminin yönünü, JavaScript kullanarak ok işaretleri yardımı ile artan ya da azalan şekilde değiştirelim.

WebGrid kodumuzu aşağıdaki gibi düzenliyoruz:

```
@{
     var grid = new WebGrid(Model,canSort:true,defaultSort:"ad");
     @grid.GetHtml(columns: grid.Columns(grid.Column("id"),
                     grid.Column("ad"),grid.Column("soyad"),
                     grid.Column("telefon"),grid.Column("mail")));
}
     <script type="text/javascript">
          $('a[href*="sort=@grid.SortColumn"]').parent().
             append('@(grid.SortDirection==SortDirection.
                   Ascending ? "?" : "?")');
     </script>
```

Uygulamamızı çalıştırdığımızda kodumuz içerisinde `defaultSort:"ad"` şeklinde sıralama işlemi varsayılan olarak `ad` alanına göre gerçekleştirildiği için `grid.SortColumn` özelliği ile `ad` alanı alınıp sıralama işleminde kullanacağımız ok işaretleri ad alanının yanında görüntülendi. Ok işaretlerine tıklanarak ilgili JavaScript kodunun çalıştırılması ile sıralama işleminin yönü `grid.SortDirection` özelliği ile belirlenmektedir.

id	ad▼	soyad	telefon	mail
3	volkan	aktaş	3241111111	abc@abc.com
13	suat	üre	3241111121	aaa@abcde.com
4	mustafa	karakurt	3241111110	abc@abcd.com
8	ismail	güven	3241111115	abc@bcde.com
6	irfan	mergan	3241111113	ab@abcde.com
10	ibrahim	aşı	3241111117	a@abcde.com
11	hureyre	yıldırım	3241111118	ab@abcde.com
7	hale	katmer	3241111114	abc@abcd.com
9	enver	arslan	3241111116	abc@abcd.com
12	enes	yıldırım	3241111119	abc@abcde.com

1 2 > ▼

- **Row.Count:** Toplam kayıt sayısını verir.

Örnek olarak WebGrid'in alt tarafında bir `Html.Label` içerisinde toplam kayıt sayısını görüntüleyelim:

```
var grid = new WebGrid(Model,canSort:true,defaultSort:"ad");
@grid.GetHtml(columns: grid.Columns(grid.Column("id"),
                grid.Column("ad"),grid.Column("soyad"),
                grid.Column("telefon"),grid.Column("mail")));
@Html.Label("Toplam Kayıt Sayısı= " + @grid.Rows.Count.ToString());
```

id	ad	soyad	telefon	mail
5	bünyamin	karahan	3241111112	abc@abcde.com
12	enes	yıldırım	3241111119	abc@abcde.com
9	enver	arslan	3241111116	abc@abcd.com
7	hale	katmer	3241111114	abc@abcd.com
11	hureyre	yıldırım	3241111118	ab@abcde.com
10	ibrahim	aşı	3241111117	a@abcde.com
6	irfan	mergan	3241111113	ab@abcde.com
8	ismail	güven	3241111115	abc@bcde.com
4	mustafa	karakurt	3241111110	abc@abcd.com
13	suat	üre	3241111121	aaa@abcde.com

1 2 >

Toplam Kayıt Sayısı= 10

`WebGrid` içerisinde kayıtları listelerken CSS özelliklerini kullanarak biçimlendirme yapabiliriz. Örnek bir biçimlendirme yapalım.

```
@{
    ViewBag.Title = "Index";
}
<!DOCTYPE html<head>
  <title>Index</title>
  <style type="text/css">
    .grid { margin: 4px; border-collapse: collapse;
            width: 350px; }
    .header { background-color: #CBC4C4; font-weight: bold;
            color:White }
    .grid th, .webGrid td { border: 1px solid #E8DEDE;
            padding: 5px; }
    .row { background-color: #E8DEDE; color: #000; }
    .id { background-color:#CBC4C4; width: 100px;
            font-weight:bold;}
  </style>
```

```
</head>
<html>
<body>
@{
    var grid = new WebGrid(Model,canSort:true, defaultSort:"id");
    @grid.GetHtml(tableStyle: "grid",
             htmlAttributes: new { id = "webgrid" },
             headerStyle: "header",
             alternatingRowStyle: "row",
             columns: grid.Columns(grid.Column("id",style:"id"),
             grid.Column("ad"),grid.Column("soyad"),
             grid.Column("telefon"),grid.Column("mail")));
}
</body>
</html>
```

CSS kodlarımızı...

```
<style type="text/css">
</style>
```

... tag'leri arasına yazdık.

.grid isimli özellik ile webgrid'in genel sitil özelliklerini,

.header isimli özellik ile webgrid'in başlık kısmının sitil özelliklerini,

.grid th isimli özellik ile webgrid'i oluşturan tablonun özelliklerini,

.row isimli özellik ile webgrid'in satırlarının sitil özelliklerini,

.id isimli özellik ile ise webgrid içerisinde listelenen alan isimlerinden id alanının stil özellikleri belirliyoruz.

`Grid.GetHtml` metodu ile webgrid'i oluştururken...

```
headerStyle: "header"
```

... örneğinde olduğu gibi her bir html niteliği için kullanılacak olan `style` özelliği ismini belirtiyoruz.

`WebGrid` içerisinde sütunları oluştururken diğer sütun isimlerinden farklı olarak `id` özelliğine `style` tanımlaması yaptığımız için `id` sütununu...

```
grid.Column("id",style:"id")
```

şeklinde `style` özelliği ismini belirterek ekliyoruz.

Uygulamızı çalıştırarak sitil özelliklerinin `WebGrid` üzerindeki etkisini görelim.

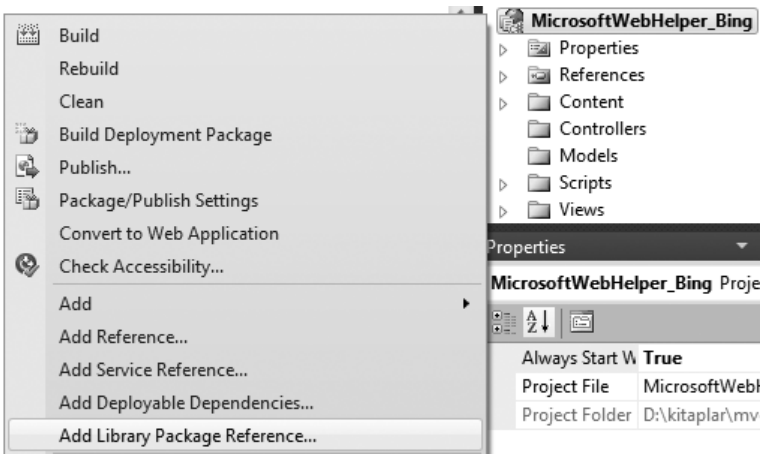
id ▲	ad	soyad	telefon	mail
3	volkan	aktaş	3241111111	abc@abc.com
4	mustafa	karakurt	3241111110	abc@abcd.com
5	bünyamin	karahan	3241111112	abc@abcde.com
6	irfan	mergan	3241111113	ab@abcde.com
7	hale	katmer	3241111114	abc@abcd.com
8	ismail	güven	3241111115	abc@bcde.com
9	enver	arslan	3241111116	abc@abcd.com
10	ibrahim	aşı	3241111117	a@abcde.com
11	hureyre	yıldırım	3241111118	ab@abcde.com
12	enes	yıldırım	3241111119	abc@abcde.com

1 2 >

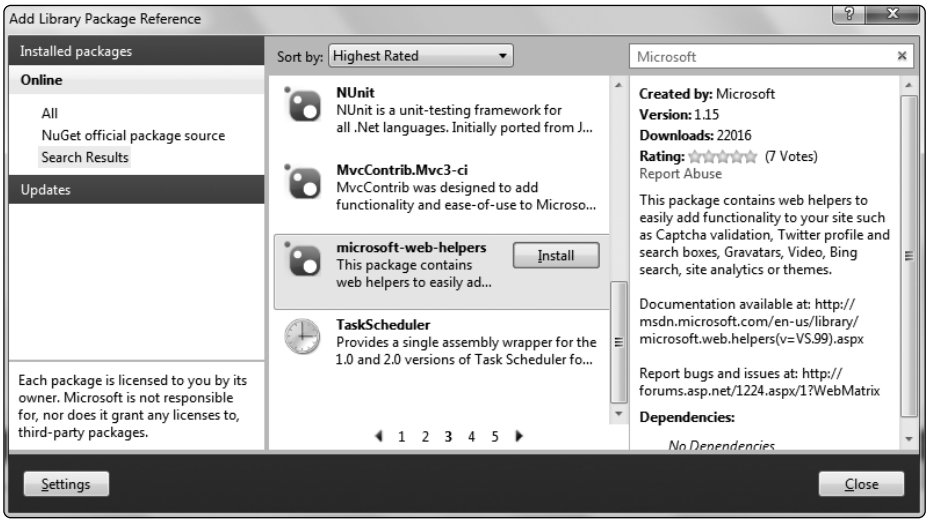
Microsoft Web Helper Metotları

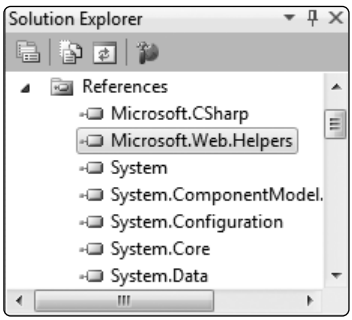
Microsoft Web Helper metotları MVC ile birlikte varsayılan olarak gelmemektedir. Daha sonra yüklenebilen bir ek paket şeklindedir. `Microsoft Web Helper` metotları Microsoft.Web.Helpers sınıfı altında yer alır.

Microsoft Web Helper metotlarını kullanabilmek için öncelikle **Solution Explorer** içerisinde proje ismine farenin sağ tuşu ile tıklayarak ya da **Project** menüsünü kullanarak **Add Library Package Reference** seçeneğini tıklıyoruz.

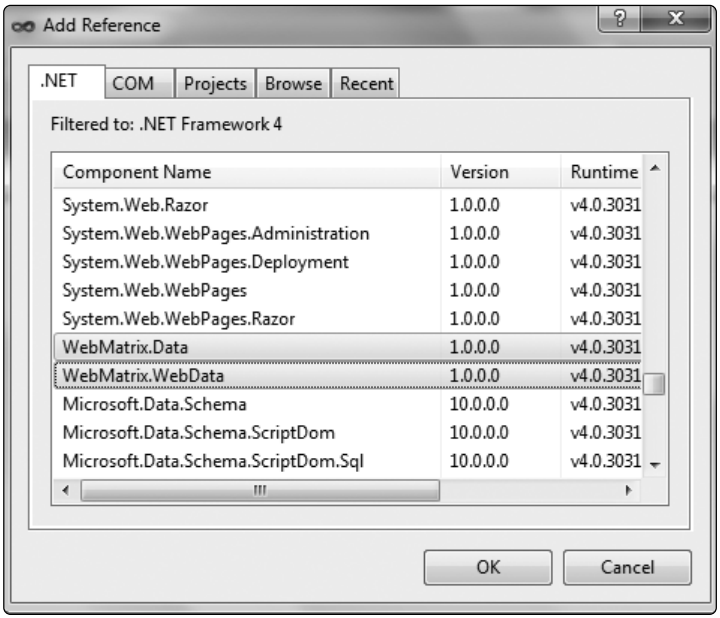


Daha sonra karşımıza gelen pencerede **Online** sekmesi altında yer alan `microsoft-web-helpers` isimli paketi seçerek **Install** butonunu tıklıyor ve paketimizi Visual Studio içerisine kuruyoruz.





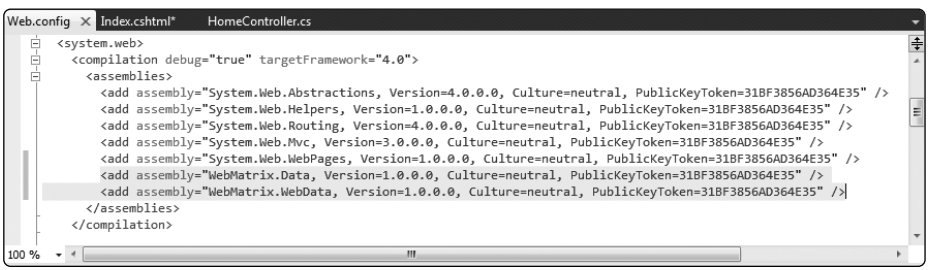
Bu işlemin ardından herhangi bir **MVC** projesi oluşturup **Solution Explorer** içerisini incelediğimizde `Microsoft.Web.Helpers` sınıfının eklendiğini görebiliriz. Ancak bu işlem maalesef `Microsoft.Web.Helpers` metotlarını kullanabilmemiz için yeterli olmuyor. `WebMatrix.Data` ve `WebMatrix.WebData` adlı dll dosyalarını referans olarak projemize eklememiz gerekiyor.



Bu işlemin ardından **Web.config** dosyasını açarak `system.web\compilation\assemblies` bölümüne...

```
<add assembly=”WebMatrix.Data, Version=1.0.0.0, Culture=neutral,
       PublicKeyToken=31BF3856AD364E35” />
<add assembly=”WebMatrix.WebData, Version=1.0.0.0, Culture=neutral,
       PublicKeyToken=31BF3856AD364E35” />
```

... satırlarını eklemeliyiz.



Son olarak...

```
@using Microsoft.Web.Helpers;
```

... şeklinde `Microsoft.Web.Helpers` sınıfını projemize ekleyerek kullanabiliriz.

Bu işlemleri bundan sonra yapacağımız her uygulama için tekrarlamalıyız.

Microsoft Web Helpers metotlarından sık kullanılanları sırasıyla inceleyelim:

Bing

Microsoft Bing arama motorunu kullanarak arama yapmak amacıyla kullanılır. Arama işlemi ile ilgili aşağıdaki özellikler kullanılabilir.

- **`Site.Title:`** Arama işleminin gerçekleştirileceği seçenekler arasına belirtilen site başlığını ekler.
- **`Site.URL:`** Arama işleminin gerçekleştirileceği seçenekler arasına belirtilen site URL'sini ekler.
- **`SearchBox:`** Arama işleminin gerçekleştirildiği bir arama penceresi görüntüler. Aşağıdaki gibi kullanılabilir.

  ```
  Bing.SearchBox([Optional, VarsayılanParametreDeğeri(322px)] int boxWidth,
  [Optional, VarsayılanParametreDeğeri (null)] string siteUrl,
  [Optional, VarsayılanParametreDeğeri (null)] string siteTitle)
  ```

  Optional parametrelerin kullanımı isteğe bağlıdır.

  `boxWidth:` Arama kutusunun genişliğini belirtir.

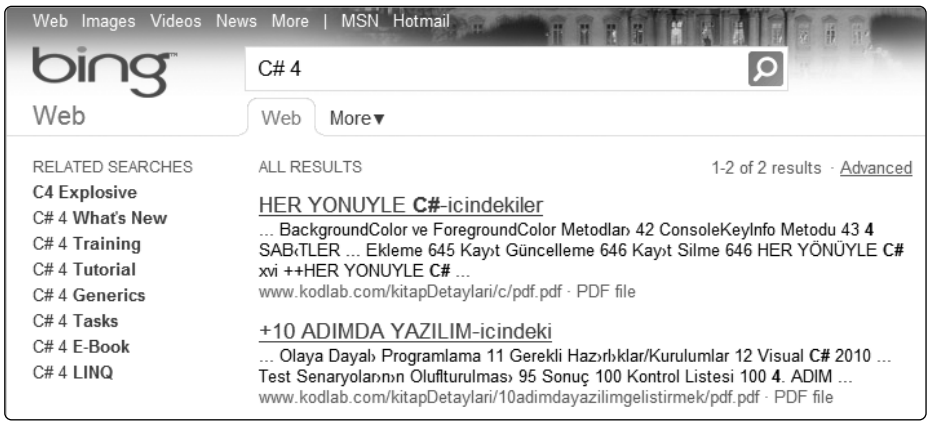
Örnek olarak boş bir **MVC 3 Razor** uygulaması oluşturalım. Microsoft Web Helpers sınıfını projemize eklemek için gerekli işlemleri yapalım. `HomeController'ı` oluşturduktan sonra, `Index` isimli View'i oluşturup aşağıdaki kodları yazalım:

```
@using Microsoft.Web.Helpers;
@{
     ViewBag.Title = "Index";
}
<h2>Index</h2>
@{
     Bing.SiteTitle = "KodLab";
     Bing.SiteUrl = "http://www.kodlab.com";
 }
@Bing.SearchBox()
```

Uygulamamızı çalıştırıp, örnek bir arama işlemi gerçekleştirelim. Görüldüğü gibi `Site.Title` ve `Site.Url` özelliği ile **KodLab** site adresini Arama seçenekleri arasına ekledik.



Aranacak metni yazıp sonucu görelim:



Facebook

Facebook uygulaması ile ilgili işlemler gerçekleştirmek ve internet sayfalarından Facebook sayfalarına erişim amacıyla kullanılır. Facebook ile ilgili aşağıdaki özellikler ve yardımcı metotlar kullanılabilir:

- **`LikeButton:`** Belirtilen web adreslerini Facebook ortamında beğenmek amacıyla kullanılır.

  Kullanımı şu şekildedir:

```
LikeButton( [Optional, VarsayılanParametreDeğeri (“”)] string href,
     [Optional, VarsayılanParametreDeğeri (“standard”)] string
           buttonLayout,
     [Optional, VarsayılanParametreDeğeri (true)] bool showFaces,
     [Optional, VarsayılanParametreDeğeri (450)] int width,
```

```
[Optional, VarsayılanParametreDeğeri (80)] int height,
[Optional, VarsayılanParametreDeğeri (“like”)] string action,
[Optional, VarsayılanParametreDeğeri (“”)] string font,
[Optional, VarsayılanParametreDeğeri (“light”)] string colorScheme,
[Optional, VarsayılanParametreDeğeri (“”)] string refLabel)
```



Optional parametrelerin kullanımı isteğe bağlıdır.

- **buttonLayout:** Butonun yerleşimi ile ilgili özellikleri belirler. Üç değer alabilir.

  standard: Yazılar butonun sağında ve arkadaşlarınızın profil fotoğrafları ise butonun altında yer alır. Minimum genişlik `225` pikseldir. Varsayılan genişlik `450` piksel, yükseklik ise `35` piksel (fotoğrafsız) veya `80` piksel (fotoğraflı olarak) değerindedir.

  button_count: Toplam beğenen sayısı butonun sağında yer alır. Minimum genişlik `90` pikseldir. Varsayılan genişlik `90` piksel, yükseklik ise `20` piksel değerindedir.

  box_count: Toplam beğenen sayısı butonun üstünde yer alır. Minimum genişlik `55` pikseldir. Varsayılan genişlik `90` piksel, yükseklik ise `65` piksel değerindedir.

- **showFaces:** `true` değerini aldığında, sizin ve arkadaşlarınızın profil fotoğraflarının butonun alt tarafında görüntülenmesini sağlar (`buttonLayout` özelliği standard olarak seçili olmalıdır) .
- **Action:** Butona tıklandığında gerçekleştirilecek olan olayı belirler. `like` (*beğen*) ve `recommend` (*paylaş*) isimli iki değer alabilir.
- **colorScheme:** Görüntülenecek olan butonun renk düzenini belirler. `light` ve `dark` şeklinde iki değer alabilir.

Örnek olarak `http://www.kodlab.com` adresi için beğen butonu görüntüleyelim. **Index.cshtml** isimli View kodu aşağıdaki gibi olacak:

```
@using Microsoft.Web.Helpers;
@{
    ViewBag.Title = “Index”;
}
@Facebook.LikeButton(“http://www.kodlab.com”)
```

Uygulamamızı çalıştırdığımızda belirtilen web adresi için beğen butonu görüntülenerek, web adresini kaç kişinin beğendiği de görüntülenmektedir.

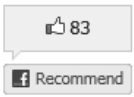


Facebook profilimize girip yaptığımız işlemin sonucunu görelim.



`LikeButton` metodunun `action` parametresini kullanarak belirtilen bir web sitesi için paylaş butonu görüntüleyelim.

```
@Facebook.LikeButton("http://www.kodlab.com",action:"recommend",
buttonLayout:"box_count",showFaces:true,colorScheme:"light")
```



Uygulamamızı çalıştırdığımızda belirtilen web adresi için paylaş butonu görüntülenerek, web adresini kaç kişinin beğendiği butonun üst tarafında görüntülenmektedir.

- **`ActivityFeed:`** Belirtilen web adresi için yapılan facebook etkinliklerini listeler. Kullanımı şu şekildedir:

```
ActivityFeed( [Optional, VarsayılanParametreDeğeri ("")] string site,
     [Optional, VarsayılanParametreDeğeri (300)] int width,
     [Optional, VarsayılanParametreDeğeri (300)] int height,
     [Optional, VarsayılanParametreDeğeri (true)] bool showHeader,
     [Optional, VarsayılanParametreDeğeri ("light")] string
          colorScheme,
     [Optional, VarsayılanParametreDeğeri ("")] string font,
     [Optional, VarsayılanParametreDeğeri ("")] string borderColor,
     [Optional, VarsayılanParametreDeğeri (false)] bool
          showRecommendations)
```

  `showRecommendations:` Etkinlikler için yapılan yorumların sayısını görüntülemeyi sağlar.

  `borderColor:` Etkinliklerin listeleneceği panel'in kenarlık rengini belirler.

```
@Facebook.ActivityFeed("http://www.microsoft.com",
showRecommendations:true)
```

Kodumuzu çalıştırdığımızda belirtilen web adresi için etkinlikler ve bu etkinlikler için yapılan yorum sayıları görüntülenmektedir.

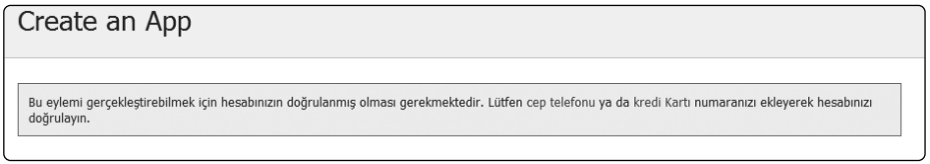


- **`Initialize:`** MVC içerisinde yorum gönderme, kullanıcı bilgilerini görüntüleme vs. gibi gelişmiş Facebook işlemleri için Facebook uygulama geliştiricisi olarak kayıtlı olmak gereklidir. Kayıt işlemi sonucunda alınan `appId` ve `appSecret` değerleri yapacağımız uygulamalarda kullanılmaktadır. `Initialize` metodu alınan `appId` ve `appSecret` değerlerinin uygulama içerisinde geçerli olmasını sağlar.

  ```
  Facebook.Initialize(string appId, string appSecret)
  ```

  ... şeklinde kullanılır.

Facebook uygulama geliştiricisi olarak uygulama geliştirmek için öncelikle yapmamız gereken ilk adım, *http://developers.facebook.com/setup/* adresine girerek bir **API key** ve **secret kod** almak. Bu adrese girip Facebook kullanıcı adı ve şifresi ile giriş yapmamızın ardından karşımıza aşağıda göründüğü gibi doğrulama yapmamıza dair bir uyarı gelecektir.



Doğrulama için cep telefonu ya da kredi kartı numaranızı kullanabilirsiniz. Ben cep telefonu ile doğrulamayı tercih ediyorum.





Cep telefonu numarasını girip **Onayla** butonuna tıklamamızın ardından cep telefonumuza gelen onay kodunu karşımıza gelen ekrana girerek **Onayla** butonuna tıklıyoruz.

Eğer girilen şifre doğruysa karşımıza aşağıdaki gibi bir pencere gelecek.





Bu işlemin ardından...

http://developers.facebook.com/setup/

... adresinde açmış olduğumuz giriş sayfamızı yeniliyoruz. Karşımıza gelen sayfada oluşturacağımız facebook uygulaması için gerekli değerleri üstteki gibi giriyoruz.

Site ismine...

```
volkanaktas
```

... site adresine uygulamamız için kendi sitemde oluşturduğum klasör adresi olan...

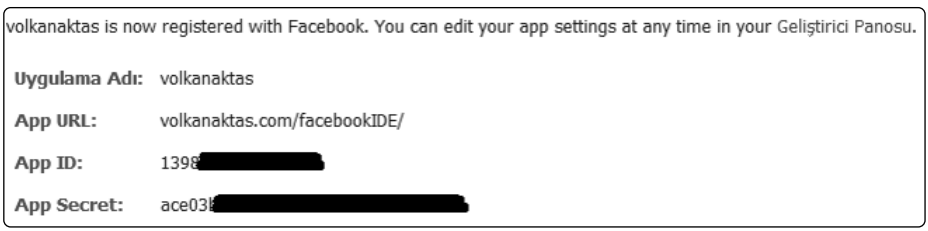
```
volkanaktas.com/facebookIDE/
```

... adresini giriyorum.

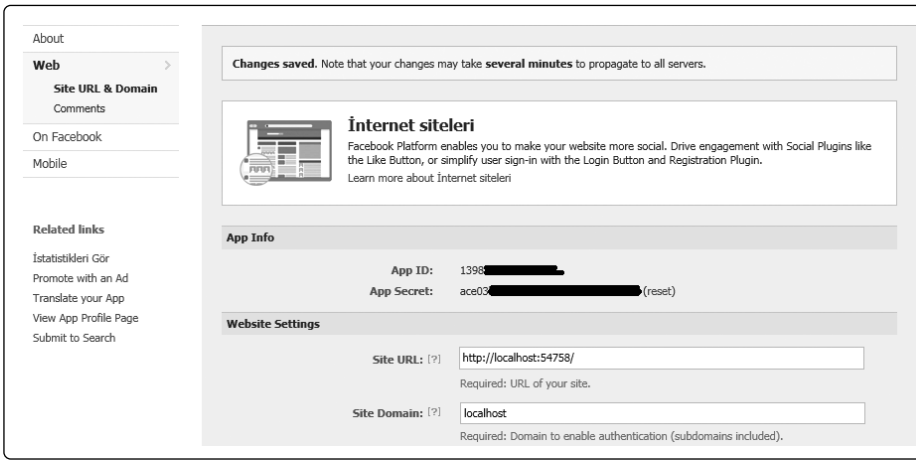
Create app butonuna tıkladığımızda karşımıza güvenlik kontrolü için resim yazılarını girmemizi isteyen bir ekran gelir.



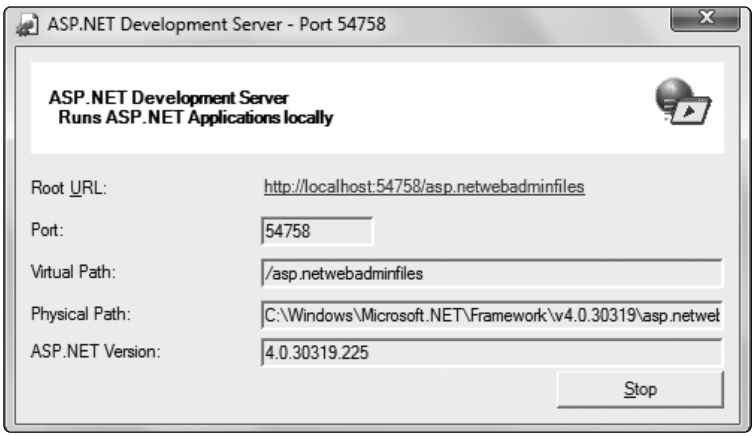
Güvenlik kodunu yazıp **Gönder** butonuna bastığımızda karşımıza uygulamamız için oluşturulan **APP ID** ve **App Secret** değerlerinin görüntülendiği sonuç sayfası gelecektir.



Ancak yukarıdaki seçenekler genel Facebook uygulama geliştirme seçenekleridir. MVC için iki küçük değişiklik yapacağız. Uygulama geliştirme penceresinde **Site URL & Domain** sekmesinde **Site URL** ve **Site Domain** değerlerini değiştiriyoruz.



Site URL değerini ben `http://localhost:54758/` şeklinde girdim. Siz de kendi local server ayarınıza göre bu değeri girebilirsiniz.



- **`GetInitializationScripts:`** Facebook JavaScript desteğinin etkinleştirilmesini sağlar.
- **`Comments:`** Bir web sayfası, resim, yazı gibi içerikler için yorum gönderme amaçlı olarak kullanılır.

  Kullanımı şu şekildedir:

```
Comments ([Optional, VarsayılanParametreDeğeri (“”)] string xid,
          [Optional, VarsayılanParametreDeğeri (550)] int width,
```

```
            [Optional, VarsayılanParametreDeğeri (10)] int numPosts,
            [Optional, VarsayılanParametreDeğeri (false)] bool
                reverseOrder,
            [Optional, VarsayılanParametreDeğeri (false)] bool
                removeRoundedBox)
```

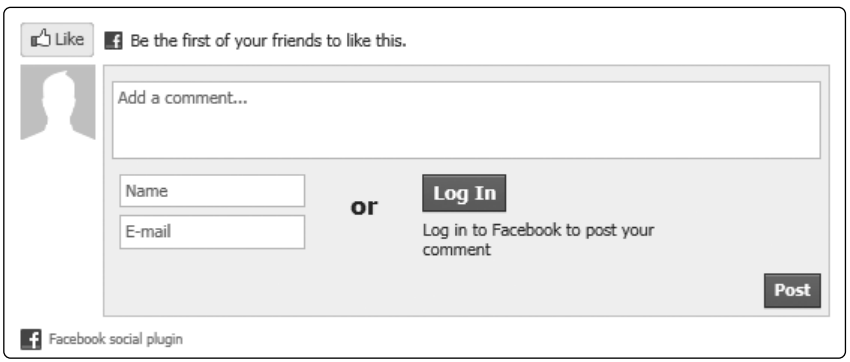
numPosts: Görüntülenecek olan yorum sayısını belirler.

reverseOrder: Yorumların görüntülenme sırasını tersine çevirir.

Aşağıdaki kodumuzu **Index.cshtml** isimli View içerisine yazıyoruz.

```
@using Microsoft.Web.Helpers;
@{
    ViewBag.Title = "Index";
}
@{
     Facebook.Initialize("appId değeri","appSecret değeri");
}
@Facebook.GetInitializationScripts()
@Facebook.Comments()
```

Aşağıda görüldüğü gibi yorum gönderme sayfası görüntülendi.



Bu sayfada kullanıcı adı ve e-mail adresini girerek yorum gönderebiliriz. Ya da login olarak direkt olarak yorum gönderebiliriz.



- **`LoginButtonTagOnly:`** Facebook hesabına giriş yapmak amacı ile kullanılır. Kullanımı şu şekildedir:

```
LoginButtonTagOnly(([Optional, VarsayılanParametreDeğeri (“”)]
    string buttonText,
          [Optional, VarsayılanParametreDeğeri (false)] bool
             autoLogoutLink,
          [Optional, VarsayılanParametreDeğeri (“long”)] string
             size,
          [Optional, VarsayılanParametreDeğeri (“short”)] string
             length,
          [Optional, VarsayılanParametreDeğeri (“”)] string onLogin,
          [Optional, VarsayılanParametreDeğeri (false)] bool
             showFaces,
          [Optional, VarsayılanParametreDeğeri (“”)] string
             extendedPermissions)
```

`autoLogoutLink:` Profil hesabına giriş işlemi gerçekleştirildikten sonra **Logout** butonunun görüntülenip görüntülenmeyeceğini belirler.

`showFaces:` `true` değerini aldığında, sizin ve arkadaşlarınızın profil fotoğraflarının görüntülenmesini sağlar.

`onLogin:` Profil hesabına giriş işlemi gerçekleştirildikten sonra yapılacak işlemi belirlemek amacıyla kullanılır. Bir url adresi kullanılarak istenilen yönlendirme gerçekleştirilebilir.

`extendedPermissions:` Facebook kullanıcılarının ve arkadaşlarımızın facebook profilimiz ile ilgili hangi bilgilere erişebileceğini belirler. **Read Permissions**, **Write Permissions** ve **Page Permissions** olmak üzere başlıca üç çeşit erişim izni grubu vardır. Bu erişim izinlerini aşağıdaki gibi sıralayabiliriz.

Read Permissions

User permission	Friends permission	Açıklama
user_about_me	friends_about_me	Profildeki **Bilgiler** bölümüne erişimi sağlar.
user_activities	friends_activities	Profildeki **Yakınlardaki Hareketler** bölümüne erişimi sağlar.
user_birthday	friends_birthday	Profildeki doğum tarihine erişimi sağlar.
user_checkins	friends_checkins	Yetkili kullanıcının ya da arkadaşlarımızın belirlediği kişilere erişimi sağlar.
user_education_history	friends_education_history	Profildeki eğitim durumu bilgilerine erişimi sağlar.
user_events	friends_events	Profildeki **Etkinlikler** bölümüne erişimi sağlar.
user_groups	friends_groups	Profildeki grupların listesine erişimi sağlar.
user_hometown	friends_hometown	Profildeki **Memleket** bilgisine erişimi sağlar.
user_interests	friends_interests	Profildeki **İlgilendikleri** bölümündeki bilgilere erişimi sağlar.
user_likes	friends_likes	Beğendiğimiz sayfa, etkinlik vb. listesine erişimi sağlar.
user_location	friends_location	Geçerli konum bilgisine erişimi sağlar.
user_notes	friends_notes	Profildeki **Notlar** bölümündeki bilgilere erişimi sağlar.
user_online_presence	friends_online_presence	Kullanıcının online/offline durumu bilgisine erişimi sağlar.
user_photo_video_tags	friends_photo_video_tags	Kullanıcı tarafından etiketlenmiş olan fotoğraf ve video bilgilerine erişimi sağlar.
user_photos	friends_photos	Kullanıcı tarafından yüklenmiş olan fotoğraflara erişimi sağlar.

user_relationships	friends_relationships	Profildeki **İlişki durumu** bilgisine erişimi sağlar.
user_relationship_details	friends_relationship_details	Profildeki **İlişki durumu** bölümündeki bilgilerin detaylarına erişimi sağlar.
user_religion_politics	friends_religion_politics	Profildeki **Siyasi görüş** bölümündeki bilgilere erişimi sağlar.
user_status	friends_status	Kullanıcının durumu ile ilgili son mesaja erişimi sağlar.
user_videos	friends_videos	Kullanıcı tarafından yüklenmiş olan videolara erişimi sağlar.
user_website	friends_website	Profilde belirtilen web adresine erişimi sağlar.
user_work_history	user_work_history	Kullanıcının çalışma geçmişine erişimi sağlar.
email	Ulaşılamaz	Kullanıcının e-posta adresine erişimi sağlar.
read_friendlists	Ulaşılamaz	Kullanıcının arkadaş listesine erişimi sağlar.
read_insights	Ulaşılamaz	Facebook istatistik servisinin sağladığı verilere erişimi sağlar.
read_mailbox	Ulaşılamaz	Facebook gelen mail kutusuna erişimi sağlar.
read_requests	Ulaşılamaz	Kullanıcının arkadaşlık istekleri bilgisine erişimi sağlar.
read_stream	Ulaşılamaz	Facebook haber kaynağı içerisindeki verilere erişimi sağlar.
xmpp_login	Ulaşılamaz	Kullanıcıların Facebook chat uygulamaları için oturum açabilmesini sağlar.
ads_management	Ulaşılamaz	Facebook reklamları ile ilgili api'leri yönetebilmeyi sağlar.

Write Permissions

Permission	Açıklama
publish_stream	Uygulamanızın arkadaşlarınızın gönderdiği yorumlar, beğendikleri ya da paylaştıkları içeriklere erişimini sağlar.
create_event	Uygulamanızın kullanıcılar adına etkinlikler oluşturmasını ve etkinlikler üzerinde düzenleme yapmasını sağlar.
rsvp_event	Uygulamanızın kullanıcılar adına etkinlikler için katılım durumu bağlantısı ekleyebilmesini sağlar.
sms	Uygulamanızın kullanıcıya sms mesajları gönderebilmesini ve kullanıcıdan gelen mesajları yanıtlayabilmesini sağlar.
offline_access	Uygulamanızın kullanıcı adına herhangi bir zamanda, yetki verilen istekleri gerçekleştirmesine olanak sağlar.
publish_checkins	Uygulamanızın kullanıcı adına, erişim sağlanabilecek kişileri belirlemesini sağlar.
manage_friendlists	Uygulamanızın kullanıcının arkadaş listelerini oluşturabilmesini ve bu listeler üzerinde düzenlemeler yapabilmesini sağlar.

Page Permissions

Permission	Açıklama
manage_pages	Uygulamanızın sayfalar üzerinde yönetici seviyesinde işlem yapabilmek için `access_tokens` almasını sağlar. Bu izin **Graph API** ile uyumludur.

Örnek olarak `LoginButtonTagOnly` yardımcı metodunu kullanarak Facebook hesabımıza giriş yapalım. Aşağıdaki kodumuzu `Index.cshtml` isimli View içerisine yazıyoruz.

```
@using Microsoft.Web.Helpers;
@{
    ViewBag.Title = "Index";
}
@{
    string Url = Context.Request.Url.GetLeftPart(UriPartial.Authority);
    Facebook.AppId = "appId değeri";
    Facebook.AppSecret = "appSecret değeri";
 }
```

```
@Facebook.GetInitializationScripts()
@Facebook.LoginButtonTagOnly("Giriş Yap", true, "long", "short",
            "loginRedirect('" + @Url + "')", true, "email")
```

Kodumuz içerisinde yer alan...

```
string Url = Context.Request.Url.
GetLeftPart(UriPartial.Authority)
```

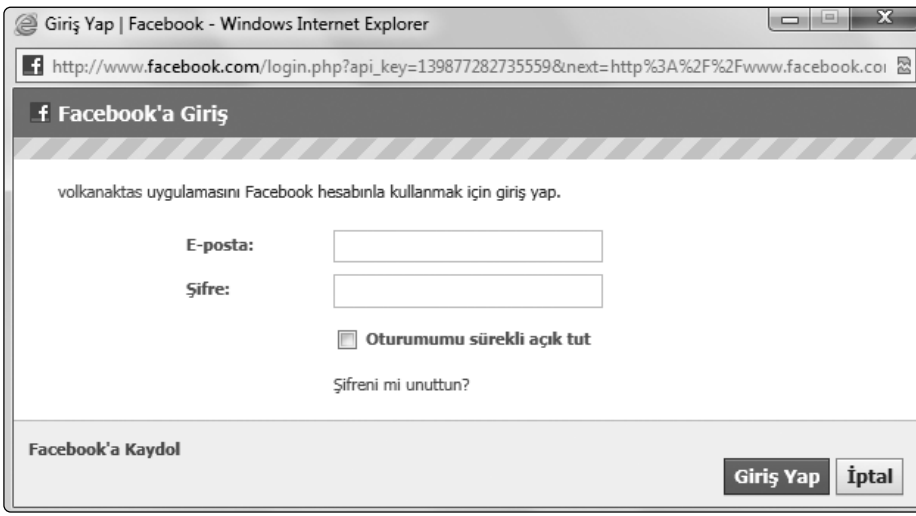
... satırı...

```
@Facebook.LoginButtonTagOnly("Giriş Yap", true, "long", "short",
"loginRedirect('" + @Url + "')", true, "email")
```

... satırından çağırılmakta ve bulunulan sayfanın url bilgisine erişilmekte, login işleminin tamamlanmasının ardından url adresine yönlendirme yapılmaktadır. `extendedPermissions` özelliğine email değerini atadığımızdan dolayı kullanıcılar e-posta adresimizi görebilmektedir.



Uygulamamızı çalıştırdığımızda öncelikle **Giriş Yap** butonu görüntülenecek, butona tıklandığında eğer daha önceden facebook'a giriş yaptığınızda **Oturumumu sürekli açık tut** seçeneğini işaretlememişseniz karşımıza **Facebook** giriş ekranı gelecektir.





Bu ekranda e-posta adresinizi ve şifrenizi girdikten sonra giriş yapmış oluyoruz.

- **LiveStream:** LiveStream eklentisini kullanarak sitenizi ziyaret eden kullanıcılar gerçek zamanlı olarak yorum gönderebilirler ve etkinliklerini paylaşabilirler.

  Kullanımı şu şekildedir:

```
LiveStream(([Optional, VarsayılanParametreDeğeri (400)] int width,
          [Optional, VarsayılanParametreDeğeri (500)] int height,
          [Optional, VarsayılanParametreDeğeri (“”)] string xid,
          [Optional, VarsayılanParametreDeğeri (“”)] string viaUrl,
          [Optional, VarsayılanParametreDeğeri (false)] bool
          alwaysPostToFriends)
```

  xid: Eğer bir sayfada birden fazla LiveStream eklentisi çalışıyorsa herbiri diğerinden benzersiz bir `xid` değeri ile ayrılır.

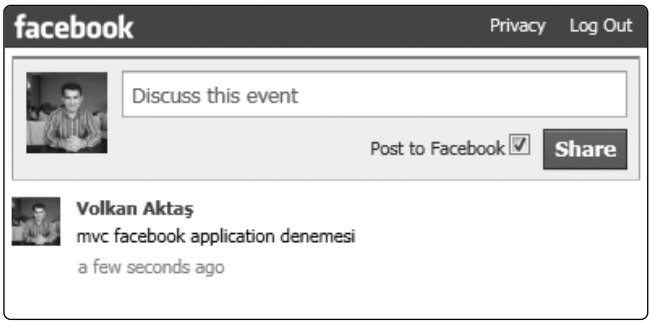
  viaURL: Uygulamanın üzerine tıklandığında yönlenecek olan url adresini belirtir. Eğer belirtilmezse varsayılan olarak bağlanılan url adresi kullanılır.

  alwaysPostToFriends: Kullanıcıların mesajlarının profillerinde görüntülenmesini sağlar.

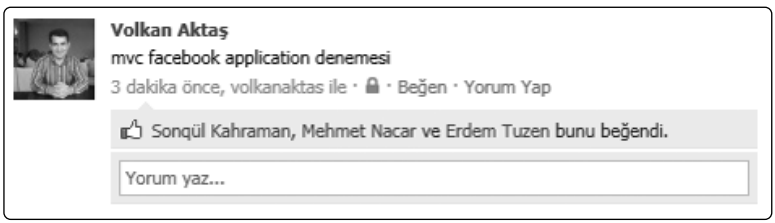
Örnek kodumuzu `Index.cshtml` isimli View içerisine yazıyoruz.

```
@using Microsoft.Web.Helpers;
@{
    ViewBag.Title = “Index”;
}
 @{
     Facebook.Initialize(“appId değeri”,”appSecret değeri”);
}
@Facebook.GetInitializationScripts()
@Facebook.LiveStream(400,200)
```

Uygulamamızı çalıştırıp örnek bir mesaj gönderiyoruz.



Gönderdiğimiz mesajın facebook profilimizde yayınlandığını görebiliriz.



TWITTER

İnternet sayfalarında sizin ya da başka twitter kullanıcılarının mesajlarını görüntülemek ve mesaj göndermek amacıyla kullanılır. Twitter ile ilgili aşağıdaki özellikler ve yardımcı metotlar kullanılabilir:

- **`Profile:`** Belirtilen Twitter profilinin mesajlarını görüntülemek amacıyla kullanılır.

  Kullanımı şu şekildedir:

```
Profile(string userName,
    [Optional, VarsayılanParametreDeğeri (250)] int width,
    [Optional, VarsayılanParametreDeğeri (300)] int height,
    [Optional, VarsayılanParametreDeğeri (“#333333”)] string
       backgroundShellColor,
    [Optional, VarsayılanParametreDeğeri (“#ffffff”)] string
       shellColor,
    [Optional, VarsayılanParametreDeğeri (“#000000”)] string
       tweetsBackgroundColor,
    [Optional, VarsayılanParametreDeğeri (“#4aed05”)] string
       tweetsLinksColor,
    [Optional, VarsayılanParametreDeğeri (4)] int numberOfTweets,
```

```
    [Optional, VarsayılanParametreDeğeri (false)] bool scrollBar,
    [Optional, VarsayılanParametreDeğeri (false)] bool loop,
    [Optional, VarsayılanParametreDeğeri (false)] bool live,
    [Optional, VarsayılanParametreDeğeri (true)] bool hashTags,
    [Optional, VarsayılanParametreDeğeri (true)] bool timestamp,
    [Optional, VarsayılanParametreDeğeri (false)] bool avatar,
    [Optional, VarsayılanParametreDeğeri (“default”)] string behavior,
    [Optional, VarsayılanParametreDeğeri (6)] int interval)
```

backgroundShellColor: Twitter uygulama penceresinin zemin rengini belirtir.

shellColor: Twitter uygulama penceresindeki yazı rengini belirtir.

tweetsBackgroundColor: Mesajların zemin rengini belirtir.

tweetsLinksColor: Mesajların alt tarafında görüntülenen linklerin rengini belirtir.

numberOfTweets: Görüntülenecek Twitter mesajı sayısını belirtir.

loop: Belirtilen mesaj sayısı kadar mesaj, görüntülendikten sonra tekrar başa dönülmesini sağlar.

live: Yeni mesajların anlık olarak güncellenip, güncellenmeyeceğini belirler.

hashTags: Etiketlerin görüntülenip görüntülenmeyeceğini belirler.

timestamp: Zaman bilgisinin görüntülenip, görüntülenmeyeceğini belirler.

avatar: Profil resminin görüntülenip görüntülenmeyeceğini belirler.

behavior: `default` ve `all` olmak üzere iki değer alır. `default` değerini aldığında `interval` özelliğinde belirtilen zaman aralığı içerisindeki mesajlar görüntülenir. `all` değerini aldığında tüm mesajlar görüntülenir.

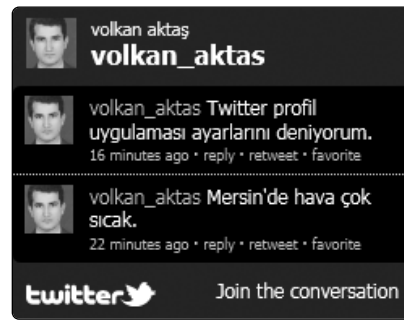
interval: behavior özelliği default değerini aldığında mesajların görüntülenme zaman aralığını belirler. Girilen değer `1000` ile çarpılır. Örneğin; `2` değeri girildiğinde `2000` değeri ifade edilmektedir.

Örnek olarak varsayılan özelliklerle twitter profilimizdeki mesajları görüntüleyelim. Örnek kodumuzu `Index.cshtml` isimli View içerisine yazıyoruz.

```
@using Microsoft.Web.Helpers
@{
    ViewBag.Title = “Index”;
}
@Twitter.Profile(“volkan_aktas”)
```



Varsayılan özellikler ile Twitter profil uygulamasının görünümü yandaki gibi olmaktadır.



Her mesajın yanında profil resminin görüntülenmesi ve iki adet mesaj görüntülenmesi için kodumuzu aşağıdaki gibi değiştiriyorum:

```
@Twitter.Profile("volkan_aktas", numberOfTweets:2,avatars:true)
```

- **Search:** Twitter kullanıcılarının mesajları içerisinde arama yapmak için kullanılır. Kullanımı şu şekildedir:

```
Search(string searchQuery,
      [Optional, VarsayılanParametreDeğeri (250)] int width,
      [Optional, VarsayılanParametreDeğeri (300)] int height,
      [Optional, VarsayılanParametreDeğeri (null)] string title,
      [Optional, VarsayılanParametreDeğeri (null)] string caption,
      [Optional, VarsayılanParametreDeğeri ("#8ec1da")] string
           backgroundShellColor,
      [Optional, VarsayılanParametreDeğeri ("#ffffff")] string
           shellColor,
      [Optional, VarsayılanParametreDeğeri ("#ffffff")] string
           tweetsBackgroundColor,
      [Optional, VarsayılanParametreDeğeri ("#444444")] string
           tweetsColor,
      [Optional, VarsayılanParametreDeğeri ("#1985b5")] string
           tweetsLinksColor,
      [Optional, VarsayılanParametreDeğeri (false)] bool scrollBar,
      [Optional, VarsayılanParametreDeğeri (true)] bool loop,
```

```
[Optional, VarsayılanParametreDeğeri (true)] bool live,
[Optional, VarsayılanParametreDeğeri (true)] bool hashTags,
[Optional, VarsayılanParametreDeğeri (true)] bool timestamp,
[Optional, VarsayılanParametreDeğeri (true)] bool avatars,
[Optional, VarsayılanParametreDeğeri (true)] bool topTweets,
[Optional, VarsayılanParametreDeğeri (“default”)] string behavior,
[Optional, VarsayılanParametreDeğeri (6)] int interval)
```

`title:` Arama penceresinin başlığını belirler.

`caption:` Arama penceresinin başlığı altında görüntülenecek yazıyı belirler.

`topTweets:` Popüler twitter mesajlarının görüntülenip görüntülenmeyeceğini belirtir.

Örnek olarak varsayılan özelliklerle Twitter ortamında arama yapalım. Örnek kodumuzu `Index.cshtml` isimli View içerisine yazıyoruz.

```
@using Microsoft.Web.Helpers
@{
    ViewBag.Title = “Index”;
}
@Twitter.Search(“visual studio”)
```

Varsayılan özellikler ile Twitter arama uygulamasının görünümü aşağıdaki gibi olmaktadır.





Arama penceresinin başlığında ve başlığının altında yazı görüntülenmesi için kodumuzu aşağıdaki gibi değiştiriyorum:

```
@Twitter.Search("visual studio",
      title:"Twitter Arama",
      caption:"Örnek Arama Sorgusu")
```

Uygulamamızın son görünümü yandaki gibidir.

- **`FollowButton:`** Kullanıcıların Twitter profilimize abone olmaları amacıyla kullanılır. Kullanımı şu şekildedir:

```
FollowButton(string userName,
[Optional, VarsayılanParametreDeğeri ("follow_me")] string followStyle,
[Optional, VarsayılanParametreDeğeri ("a")] string followColor)
```

- **`followStyle:`** Oluşturulan butonun biçim özelliklerini belirler. Aşağıdaki değerleri alabilir:
  - **`follow_me`** 
  - **`follow_bird`** 
  - **`twitter`** 
  - **`t_logo`** 
  - **`t_small`** 
  - **`t_mini`** 
- **`followColor:`** Oluşturulan butonun rengini belirler. Aşağıdaki değerleri alabilir:
  - **`a:`** Mavi • **`b:`** Gümüş • **`c:`** Siyah

Örnek olarak varsayılan özelliklerle butonumuzu oluşturalım. Örnek kodumuzu `Index.cshtml` isimli View içerisine yazıyoruz.

```
@using Microsoft.Web.Helpers
@{
    ViewBag.Title = "Index";
}
@Twitter.FollowButton("volkan_aktas")
```

Varsayılan özellikler ile butonumuzun görünümü aşağıdaki gibi olmaktadır.



- **`TweetButton:`** Kullanıcıların beğendikleri içerikleri paylaşmaları amacıyla kullanılır. Kullanımı şu şekildedir:

```
TweetButton([Optional, VarsayılanParametreDeğeri ("vertical")]
DataCount dataCount,
[Optional, VarsayılanParametreDeğeri ("")] string tweetText,
[Optional, VarsayılanParametreDeğeri ("")] string url,
[Optional, VarsayılanParametreDeğeri ("")] string language,
[Optional, VarsayılanParametreDeğeri ("")] string username,
[Optional, VarsayılanParametreDeğeri ("")] string relatedUsername,
[Optional, VarsayılanParametreDeğeri ("")] string
relatedUserDescription)
```

`dataCount:` Belirtilen url adresinin kaç defa twitlendiği **Tweet** butonu ile birlikte görüntülenebilir. Görüntülenme yeri ile ilgili olarak aşağıdaki değerleri alabilir:

• **`none:`** 

• **`horizontal:`** 

• **`vertical:`** 

`tweetText:` Gönderilecek olan mesaj metnini belirtir.

`url:` Paylaşılacak olan url adresini belirtir. Eğer belirtilmezse butonun yer aldığı sayfanın url değerini alır.

`language:` Kullanılacak olan yazım dilini belirtir. İngilizce için `en`, Türkçe için `tr` değerini alır.

`username:` Gönderilecek olan mesaj içerisinde bahsedilecek kullanıcı adını belirtir.

`relatedUsername:` Takip etmek için önerilecek olan diğer kullanıcı adı.

relatedUsername: Takip etmek için önerilecek olan diğer kullanıcı ile ilgili açıklamalar.

Örnek bir Tweet butonu oluşturalım. Örnek kodumuzu `Index.cshtml` isimli View içerisine yazıyoruz.

```
@using Microsoft.Web.Helpers
@{
    ViewBag.Title = "Index";
}
@Twitter.TweetButton(userName:"volkan_aktas",
       url:"http://www.kodlab.com",
       tweetText:"Bilişim yayıncılığının yeni yüzü ")
```

Kodumuzda `volkan_aktas` kullanıcısı aracılığıyla `http://www.kodlab.com` adresi ve `"Bilişim yayıncılığının yeni yüzü"` yazısı paylaşılmıştır.



Uygulamamızı çalıştırdığımızda karşımıza ilk olarak **tweet** butonu gelmektedir. Belirttiğimiz url adresinin daha önce 7 defa paylaşıldığını görüyoruz.

Daha sonra karşımıza **Twitter login** ekranı gelmektedir. Gerekli kullanıcı adı ve şifre bilgilerini giriyoruz.



Karşımıza **Tweet** paylaşım ekranı gelecektir. Burada kodumuzda yazdığımız içerikler karşımıza gelmektedir.



Tweet butonuna tıklayarak içeriğimizi paylaşıyoruz.



Paylaştığımız içeriğin sonucunu profilimizde görebiliriz.

- **`Faves:`** Profilimizde favori olarak işaretlediğimiz tweet mesajlarını görüntülemek amacıyla kullanılır.

  Kullanımı şu şekildedir:

```
TweetButton(string username,
      [Optional, VarsayılanParametreDeğeri (250)] int width,
      [Optional, VarsayılanParametreDeğeri (300)] int height,
      [Optional, VarsayılanParametreDeğeri (null)] string title,
      [Optional, VarsayılanParametreDeğeri (null)] string caption,
      [Optional, VarsayılanParametreDeğeri (“#43c43f”)] string
           backgroundShellColor,
      [Optional, VarsayılanParametreDeğeri (“#ffffff”)] string
           shellColor,
```

```
[Optional, VarsayılanParametreDeğeri (“#ffffff”)] string
     tweetsBackgroundColor,
[Optional, VarsayılanParametreDeğeri (“#444444”)] string
     tweetsColor,
[Optional, VarsayılanParametreDeğeri (“#43c43f”)] string
     tweetsLinksColor,
[Optional, VarsayılanParametreDeğeri (10)] int numberOfTweets,
[Optional, VarsayılanParametreDeğeri (true)] bool scrollBar,
[Optional, VarsayılanParametreDeğeri (false)] bool loop,
[Optional, VarsayılanParametreDeğeri (true)] bool live,
[Optional, VarsayılanParametreDeğeri (true)] bool hashTags,
[Optional, VarsayılanParametreDeğeri (true)] bool timestamp,
[Optional, VarsayılanParametreDeğeri (true)] bool avatars,
[Optional, VarsayılanParametreDeğeri (true)] bool topTweets,
[Optional, VarsayılanParametreDeğeri (“default”)] string behavior,
[Optional, VarsayılanParametreDeğeri (6)] int interval)
```



Bu özellikleri daha önce açıkladığımız için burada tekrar detaya girmiyoruz.

Örnek bir favoriler penceresi oluşturalım. Örnek kodumuzu `Index.cshtml` isimli View içerisine yazıyoruz.

```
@{
     ViewBag.Title = “Index”;
}
@Twitter.Faves(userName:”volkan_aktas”)
```



Varsayılan özellikler ile Twitter faves uygulamasının görünümü yandaki gibi olmaktadır.

Favoriler penceresinin başlığında ve başlığının altında yazı görüntülenmesi için, ayrıca renkler üzerinde değişiklik yapmak için kodumuzu aşağıdaki gibi değiştiriyoruz:

```
@Twitter.Faves(userName:"volkan_aktas",title:"Favori Konular",
       caption:"Volkan Aktaşın favorileri",
       backgroundShellColor:"#80997f",tweetsBackgroundColor:"#e8e3e8",
       tweetsLinksColor:"#51b04d")
```

Uygulamamızın son görünümü aşağıdaki gibidir.



- **`List:`** Profilimizde oluşturduğumuz listeler içerisinde mevcut olan kişilere ait Tweet mesajlarını görüntülemek amacıyla kullanılır.

  Kullanımı şu şekildedir:

```
TweetButton(string username, string list,
    [Optional, VarsayılanParametreDeğeri (250)] int width,
    [Optional, VarsayılanParametreDeğeri (300)] int height,
    [Optional, VarsayılanParametreDeğeri (null)] string title,
    [Optional, VarsayılanParametreDeğeri (null)] string caption,
    [Optional, VarsayılanParametreDeğeri ("#43c43f")] string
           backgroundShellColor,
    [Optional, VarsayılanParametreDeğeri ("#ffffff")] string shellColor,
```

```
    [Optional, VarsayılanParametreDeğeri (“#ffffff”)] string
           tweetsBackgroundColor,
    [Optional, VarsayılanParametreDeğeri (“#444444”)] string
           tweetsColor,
    [Optional, VarsayılanParametreDeğeri (“#43c43f”)] string
           tweetsLinksColor,
    [Optional, VarsayılanParametreDeğeri (10)] int numberOfTweets,
    [Optional, VarsayılanParametreDeğeri (true)] bool scrollBar,
    [Optional, VarsayılanParametreDeğeri (false)] bool loop,
    [Optional, VarsayılanParametreDeğeri (true)] bool live,
    [Optional, VarsayılanParametreDeğeri (true)] bool hashTags,
    [Optional, VarsayılanParametreDeğeri (true)] bool timestamp,
    [Optional, VarsayılanParametreDeğeri (true)] bool avatars,
    [Optional, VarsayılanParametreDeğeri (true)] bool topTweets,
    [Optional, VarsayılanParametreDeğeri (“default”)] string behavior,
    [Optional, VarsayılanParametreDeğeri (6)] int interval)
```

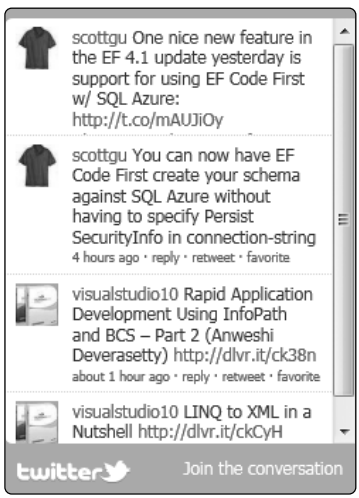
list: Profilimizde oluşturmuş olduğumuz liste adını belirtir.



Diğer özellikleri daha önce açıkladığımız için burada tekrar detaya değinmiyoruz.

Varsayılan özellikleri ile bir liste penceresi oluşturalım. Örnek kodumuzu `Index.cshtml` isimli View içerisine yazıyoruz.

```
@{
     ViewBag.Title = “Index”;
}
@Twitter.List(“volkan_aktas”,”dot-net”)
```



Uygulamamızın görünümü yandaki gibi olmaktadır. Profilimizde mevcut olan `dot-net` adlı listede mevcut olan kullanıcılara ait mesajlar anlık olarak listelenmektedir.

LinkShare

Belirtilen başlık ve url adresi ile sosyal ağ linklerini oluşturmak amacıyla kullanılır. Kullanımı şu şekildedir:

```
LinkShare.GetHtml(string pageTitle,
       [Optional, VarsayılanParametreDeğeri (null)] string
              pageLinkBack,
       [Optional, VarsayılanParametreDeğeri (null)] string
              twitterUserName,
       [Optional, VarsayılanParametreDeğeri (null)] string
              additionalTweetText,
Params LinkShareSite[] linkSites)
```

- **pageTitle:** Sayfa başlığını belirtir.
- **pageLinkBack:** Linklerde kullanılacak url adresini belirtir.
- **twitterUserName:** Twitter kullanıcı adını belirtir.
- **additionalTweetText:** Gönderilecek olan twitter mesajını belirler.
- **linkSites:** Görüntülenecek olan sosyal ağ linklerini belirtir. Eğer bir değer girilmezse varsayılan olarak `All` değeri geçerli olur ve tüm sosyal ağ linkleri görüntülenir. Aşağıdaki değerleri alabilir:
  - `Delicious`
  - `Digg`
  - `GoogleBuzz`
  - `Facebook`
  - `Reddit`
  - `StumbleUpon`
  - `Twitter`
  - `All`

Örnek olarak varsayılan özelliklerle `LinkShare` metodunu yazalım. Örnek kodumuzu `Index.cshtml` isimli View içerisine yazıyoruz.

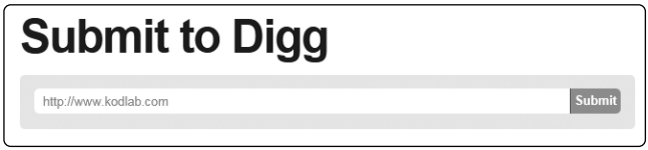
```
@using Microsoft.Web.Helpers
@{
    ViewBag.Title = "Index";
}
@LinkShare.GetHtml(pageTitle:"Kodlab Yayınevi",
          pageLinkBack:"http://www.kodlab.com")
```

Kodumuzda `http://www.kodlab.com` adresi için sosyal ağ linkleri oluşturuyoruz.

Varsayılan özellikler ile **LinkShare** uygulamasının görünümü yandaki gibi olmaktadır.



Buradaki sosyal linklerinden örneğin; **Digg**'e tıklayalım.



Görüldüğü gibi belirttiğimiz url adresi buraya yazdırıldı.

`linkSites` özelliğini kullanarak listelenen sosyal ağ linklerini biz seçelim. Kodumuzun yeni hali aşağıdaki gibi olacak:

```
@LinkShare.GetHtml(pageTitle:"Kodlab Yayınevi",
       pageLinkBack:"http://www.kodlab.com",
       linkSites: new LinkShareSite[] { LinkShareSite.Digg,
       LinkShareSite.StumbleUpon,LinkShareSite.Delicious })
```

Uygulamamızın görünümü yandaki gibi olmaktadır. Görüldüğü gibi sadece belirttiğimiz sosyal ağ linkleri listelendi.



Video

Multimedya dosyalarını oynatmak amacıyla kullanılır. **Flash**, **MediaPlayer** ve **Silverlight** olmak üzere üç çeşit player kullanılabilir.

- **`Flash:` swf** uzantılı Flash dosyalarını görüntülemek amacıyla kullanılır.

  Kullanımı şu şekildedir:

```
Video.Flash(string path,
            [Optional, VarsayılanParametreDeğeri (null)] string width,
            [Optional, VarsayılanParametreDeğeri (null)] string height,
            [Optional, VarsayılanParametreDeğeri (true)] bool play,
            [Optional, VarsayılanParametreDeğeri (true)] bool loop,
            [Optional, VarsayılanParametreDeğeri (null)] bool menu,
            [Optional, VarsayılanParametreDeğeri (null)] string bgColor,
            [Optional, VarsayılanParametreDeğeri (null)] string quality,
            [Optional, VarsayılanParametreDeğeri (null)] string scale,
            [Optional, VarsayılanParametreDeğeri (null)] string
                  windowMode
```

```
[Optional, VarsayılanParametreDeğeri (null)] string baseUrl,
[Optional, VarsayılanParametreDeğeri (null)] string version,
[Optional, VarsayılanParametreDeğeri (null)] object options,
[Optional, VarsayılanParametreDeğeri (null)] object
      htmlAtributes,
[Optional, VarsayılanParametreDeğeri (null)] string
      embedName)
```

path: Video dosyasının yolunu belirtir.

width: Video oynatıcısının pencere genişliğini belirtir.

height: Video oynatıcısının pencere yüksekliğini belirtir.

play: Videonun otomatik olarak başlatılıp başlatılmayacağını belirtir.

loop: Videonun sürekli olarak oynatılıp oynatılmayacağını belirtir.

menu: Video oynatma ayarlarının bulunduğu sağ fare menüsünün görüntülenip görüntülenmeyeceğini belirtir.

bgColor: Video oynatıcı uygulamasının zemin rengini belirtir.

quality: Video kalitesini belirtir. `low`, `autolow`, `autohigh`, `medium`, `high` ve `best` değerlerini alır.

scale: Video boyutunu ayarlar. Aşağıdaki değerleri alır:

- **exactfit:** Tüm video içeriği orijinal en boy oranları korunmadan görünür hale getirilir. Ancak video içeriği bozuk görüntülenebilir.

- **showall:** Orijinal en boy oranı korunurken, videonun tamamı görüntülenir. Ancak video oynatıcı video içeriğinin her bir kenarında video sınırlarını gösterir.

- **noorder:** Orijinal en boy oranı korunurken, video kenarlarında kırpılmalar olabilir.

windowMode: Video oynatıcısının görünümünü belirtir: `transparent`, `window` ve `opaque` değerlerini alır.

baseUrl: Flash medya dosyasının yolunu belirlemede kullanılabilecek url adresi.

version: Flash sürümü.

options: Flash html nesnesine parametre aktarmak için kullanılır.

htmlAtributes: Oluşturulacak olan Flash html nesnesi ile ilgili html niteliklerini belirtir.

embedName: Oluşturulacak olan Flash html nesnesi, video html referansı içerdiğinde kullanılan embed html öğesi için bir isim belirtir.

Örnek olarak **Solution Explorer** içerisinde **Content** klasörüne yüklediğimiz bir flash dosyasını görüntüleyecek uygulama yapalım. Örnek kodumuzu `Index.cshtml` isimli View içerisine yazıyoruz.

```
@using Microsoft.Web.Helpers;
@{
     ViewBag.Title = "Index";
}
@Video.Flash(path: "~/Content/CanakkaleGecilmez.swf",
              width: "400",height: "400",play: true,loop: true,
              menu: true,bgColor: "blue",quality: "medium",
              scale: "exactfit",windowMode: "transparent")
```



Uygulamamızın sonucu yukarıdaki gibi video oynatıcımız içerisinde belirttiğimiz Flash dosyası görüntülenmektedir. `menu` özelliğine `true` değerini verdiğimiz için video içerisinde sağ tıklanırsa, video ile ilgili seçeneklerin yer aldığı sağ menü görüntülenecektir.

Örneğimizde yaptığımız gibi lokal video dosyası görüntületmek yerine, herhangi bir web adresindeki dosyanın adresini yazarak da ilgili web adresindeki Flash dosyalarını oynatabilirsiniz.

- **MediaPlayer: wmv** uzantılı ses ve video dosyalarını oynatmak amacıyla kullanılır.

  Kullanımı şu şekildedir:

```
Video.MediaPlayer(string path,
            [Optional, VarsayılanParametreDeğeri (null)] string width,
            [Optional, VarsayılanParametreDeğeri (null)] string height,
            [Optional, VarsayılanParametreDeğeri (true)] bool autoStart,
            [Optional, VarsayılanParametreDeğeri (true)] bool loop,
            [Optional, VarsayılanParametreDeğeri (1)] int playCount,
            [Optional, VarsayılanParametreDeğeri (null)] string uiMode,
            [Optional, VarsayılanParametreDeğeri (false)] bool
                   stretchToFit,
            [Optional, VarsayılanParametreDeğeri (true)] bool
                   enableContextMenu,
            [Optional, VarsayılanParametreDeğeri (false)] bool mute,
            [Optional, VarsayılanParametreDeğeri (-1)] int volume,
            [Optional, VarsayılanParametreDeğeri (null)] string baseUrl,
            [Optional, VarsayılanParametreDeğeri (null)] object options,
            [Optional, VarsayılanParametreDeğeri (null)] object
                   htmlAtributes,
            [Optional, VarsayılanParametreDeğeri (null)] string
                   embedName)
```

autoStart: Videonun otomatik olarak başlatılıp başlatılmayacağını belirtir.

playCount: Videonun kaç defa tekrarlanacağını belirtir.

uiMode: Video oynatıcı arabiriminde hangi kontrollerin görüntüleneceğini belirtir. invisible, none, mini ve full değerlerini alır.

- **invisible:** Video oynatma kontrolleri ve video görüntülenmeyip, yalnızca ses çalınır.

- **none:** Video kontrolleri olmadan video oynatılır.

- **mini:** İleri ve geri sarma ve sonraki önceki butonları  olmadan, diğer video kontrolleri görüntülenir.

- **full:** Tüm video kontrolleri görüntülenir. 

stretchToFit: Videonun belirtilen genişlik ve yükseklik değerlerine sığdırılıp sığdırılmayacağını belirtir.

enableContextMenu: Video oynatma ayarlarının bulunduğu sağ fare menüsünün görüntülenip görüntülenmeyeceğini belirtir.

mute: `true` değerini aldığında video sessiz olarak oynatılmaya başlatılır. false değerini aldığında belirtilen ses değerleri ile video oynatılmaya başlatılır.

volume: Aldığı `0` ile `100` arasındaki değerler ile ses seviyesini belirler.

Örnek olarak **Solution Explorer** içerisinde **Content** klasörüne yüklediğimiz wmv uzantılı bir video dosyasını görüntüleyecek uygulama yapalım. Örnek kodumuzu `Index.cshtml` isimli View içerisine yazıyoruz.

```
@using Microsoft.Web.Helpers;
@{
    ViewBag.Title = "Index";
}
@Video.MediaPlayer(path: "~/Content/ikinci_bahar.wmv",
            width: "300",height: "300",autoStart: true,
            playCount: 2,uiMode:  "full",
            stretchToFit: true,enableContextMenu: true,
            mute: false,volume: 50)
```



Uygulamamızın sonucu yandaki gibi video oynatıcımız içerisinde belirttiğimiz wmv uzantılı video dosyası görüntülenmektedir. `uiMode` özelliğine `full` değerini verdiğimiz için video oynatıcının tüm kontrolleri görüntülenmektedir. `stretchToFit` özelliğine `true` değerini verdiğimiz için video `300x300` pencere boyutlarına otomatik olarak ayarlanmaktadır. `menu` özelliğine `true` değerini verdiğimiz için video içerisinde sağ tıklanırsa, video ile ilgili seçeneklerin yer aldığı sağ menü görüntülenecektir.

- **SilverLight:** Medya dosyalarını Silverlight tabanlı video oynatıcısı kullanarak oynatmak amacıyla kullanılır.

  Kullanımı şu şekildedir:

```
Video.MediaPlayer(string path,
            [Optional, VarsayılanParametreDeğeri (null)] string width,
            [Optional, VarsayılanParametreDeğeri (null)] string
                  height,
            [Optional, VarsayılanParametreDeğeri (null)] string
                  bgColor,
            [Optional, VarsayılanParametreDeğeri (null)] string
                  initParameters,
            [Optional, VarsayılanParametreDeğeri (null)] string
                  minimumVersion,
            [Optional, VarsayılanParametreDeğeri (true)] bool
                  autoUpgrade,
            [Optional, VarsayılanParametreDeğeri (null)] object
                  options,
            [Optional, VarsayılanParametreDeğeri (null)] object
                  htmlAtributes)
```

- **initParameters:** Silverlight .xap dosyasına parametre aktarmak amacıyla kullanılır
- **minimumVersion:** Belirtilen medya dosyasını oynatmak için gerekli olan en düşük Silverlight plugin versiyonunu belirtir.
- **autoUpgrade:** Silverlight plugin'inin yeni bir versiyonu çıktığında otomatik olarak güncellenip güncellenmeyeceğini belirtir.

Örnek olarak **Solution Explorer** içerisinde **Content** klasörüne yüklediğimiz xap uzantılı bir Silverlight uygulamasını görüntüleyecek uygulama yapalım. Örnek kodumuzu `Index.cshtml` isimli View içerisine yazıyoruz.

```
@using Microsoft.Web.Helpers;
@{
    ViewBag.Title = "Index";
}
@Video.Silverlight(path: "~/Content/CalendarDemo.xap",
             width: "400",height: "400",autoUpgrade: true)
```

Uygulamamızın sonucu aşağıdaki gibi **xap** uzantılı silverlight dosyası görüntülenmektedir.



ANALYTICS

Belirtilen bir web adresi için **Google Analytics** ve **StatCounter** gibi web tabanlı analitik araçları tarafından üretilen analiz sonuçları içeren script kodları oluşturmak amacıyla kullanılır.

- **`GetGoogleAsyncHtml:`** Bir web sayfası için **Google Analytics** tarafından üretilen ve eş zamansız olarak çağırılan script kodlarını oluşturur.

  Kullanımı şu şekildedir:

  ```
  Analytics.GetGoogleAsyncHtml(string webPropertyId)
  ```

  `webPropertyId:` Bir web sayfasını izlemek amacıyla oluşturulan benzersiz bir tanımlamadır. `UA-XXXXX-X` formatında oluşturulur. Kullanıcı hesap numarası ve profil numarası olmak üzere iki bölümden oluşur.



Daha önceden **Google Analytics** servisinde oluşturmuş olduğunuz hesabınızda, eklemiş olduğunuz web sitelerinize ait olan `webPropertyId` değerini *https://www.google.com/analytics/settings/* adresinden öğrenebilirsiniz.

Örnek olarak `GetGoogleAsyncHtml` metodunu kullanarak **Google Analytics** servisi tarafından site analiz kodu üretecek bir uygulama yapalım. Örnek kodumuzu `Index.cshtml` isimli View içerisine yazıyoruz.

```
@using Microsoft.Web.Helpers;
@{
     ViewBag.Title = “Index”;
}
@Analytics.GetGoogleAsyncHtml(“webPropertyId değeri “)
```

Uygulamamızı çalıştırıp sayfamızın kaynak kodunu incelediğimizde `<body>` `</body>` tag'leri arasında JavaScript kodumuzun üretildiğini görebiliriz.



Üretilen bu JavaScript kodunu analiz edilmesini istediğimiz web sitemizin sayfalarına ekleyerek, Google Analytic servisi tarafından analiz sonuçları üretilmesini sağlıyoruz.

- **`GetGoogleHtml:`** Bir web sayfası için **Google Analytics** tarafından üretilen script kodlarını oluşturur.

  Kullanımı şu şekildedir:

  `Analytics.GetGoogleHtml(string webPropertyId)`

`GetGoogleHtml` metodunu kullanarak **Google Analytics** servisi tarafından site analiz kodu üretecek bir uygulama yapalım.

```
@using Microsoft.Web.Helpers;
@{
     ViewBag.Title = “Index”;
}
@Analytics.GetGoogleHtml(“webPropertyId değeri”)
```

Uygulamamızı çalıştırıp sayfamızın kaynak kodunu incelediğimizde <body> </body> tag’leri arasında JavaScript kodumuzun üretildiğini görebiliriz.



- **`GetStatCounterHtml:`** Bir web sayfası için `StatCounter` tarafından üretilen script kodlarını oluşturur.

  Kullanımı şu şekildedir:

  `Analytics.GetStatCounterHtml(int project, string security)`

  `project:` `StatCounter` tarafından web siteniz için oluşturulan benzersiz bir proje numarasıdır.

  `security:` `StatCounter` tarafından web sitenizi doğrulamak için oluşturulan bir anahtar değerdir.

Daha önceden `StatCounter` sitesinde oluşturmuş olduğunuz hesabınızda, eklemiş olduğunuz web sitelerinize ait olan `security` değerini *http://beta.statcounter.com/ProjeNumarası/security/* adresinden öğrenebilirsiniz.



Örnek olarak `GetStatCounterHtml` metodunu kullanarak **StatCounter** servisi tarafından site analiz kodu üretecek bir uygulama yapalım. Örnek kodumuzu `Index.cshtml` isimli View içerisine yazıyoruz.

```
@using Microsoft.Web.Helpers;
@{
     ViewBag.Title = "Index";
}
@Analytics.GetStatCounterHtml(projectID,"Security Kodu")
```

Uygulamamızı çalıştırıp sayfamızın kaynak kodunu incelediğimizde `<body>` `</body>` tag'leri arasında JavaScript kodumuzun üretildiğini görebiliriz.



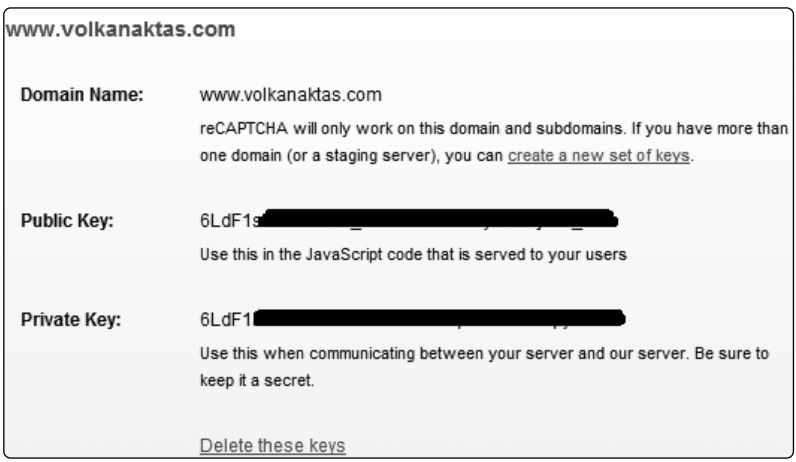
Üretilen bu JavaScript kodunu analiz edilmesini istediğimiz web sitemizin sayfalarına ekleyerek, `StatCounter` sitesi tarafından analiz sonuçları üretilmesini sağlıyoruz.

Recaptcha

Captcha karakterleri oluşturmak ve oluşturulan captcha karakterlerini girilen değerlerle karşılaştırmak amacıyla kullanılır. Öncelikle *http://www.google.com/recaptcha* adresinden kayıt olmamız gerekmekte. Kayıt olup giriş yaptıktan sonra **My Account** menüsünden bir domain adresi belirtip **Create Key** butonuna tıklıyoruz. Eğer belirtilen domain adresinin alt domainlerinde de **Captcha** kullanacaksak **Enable this key on all domains (global key)** seçeneğini işaretliyoruz.



Create Key butonuna tıkladığımız anda yönlendirildiğimiz sayfada MVC uygulamamız içerisinde kullanacağımız `Public` ve `Private Key` kodları bulunmaktadır.



Bu işlemin ardından uygulamamızı kodlamaya geçebiliriz.

Boş bir **MVC 3 Razor** projesi oluşturup, `HomeController` ve `Index` isimli View dosyalarımızı oluşturuyoruz. `HomeController` içerisine `Index` isimli ActionResult'a ek olarak aşağıdaki kod satırlarını ekliyoruz.

```
public ActionResult Kontrol()
{
     if (ReCaptcha.Validate(privateKey: "privateKey Kodu"))
     {
          return Content("Doğru");
     }
     else return Content("Yanlış");
}
```

`Index` isimli View içerisine ise aşağıdaki kod satırlarını ekliyoruz.

```
@using Microsoft.Web.Helpers;
@{
     ViewBag.Title = "Index";
}
<fieldset>
     <form method="post" action="/Home/Kontrol">
          <input type="submit" value="Tamam" />
          @ReCaptcha.GetHtml(publicKey:"publicKey Kodu",theme: "red")
     </form>
</fieldset>
```



Uygulamamız içerisinde **Tamam** butonuna tıklandığında `HomeController` içerisinde bulunan `Kontrol` isimli ActionResult metodu çağırılmaktadır. `ReCaptcha.GetHtml` metodu ile captcha karakterleri oluşturulmakta ve oluşturulan captcha karakterleri `Kontrol` isimli ActionResult içerisindeki `ReCaptcha.Validate` metodu yardımıyla

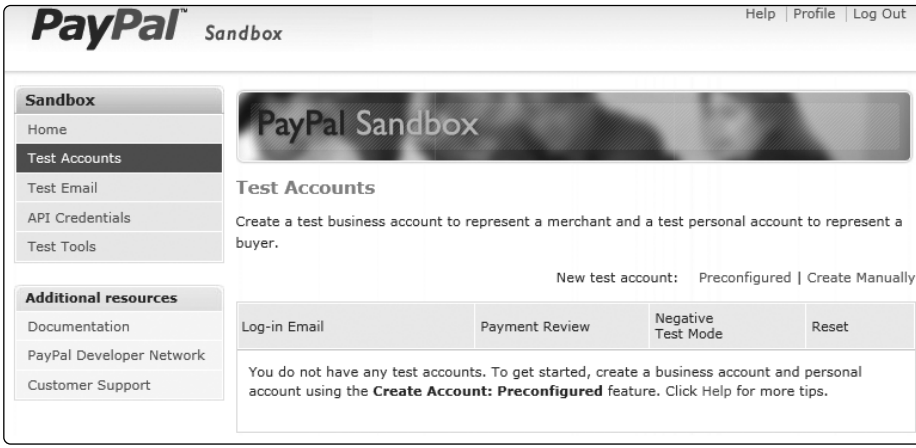
girilen değerlerle karşılaştırılmaktadır. Eğer her ikisi de uyumlu ise geriye `Doğru` değeri, uyumlu değil ise `Yanlış` değeri döndürülmektedir.

PayPal Helper

MVC içerisinde **PayPal** ödeme sistemini kullanmak için *http://paypalhelper.codeplex.com/* adresinden indirebileceğiniz **PayPal Helper**'ı kullanabilirsiniz.

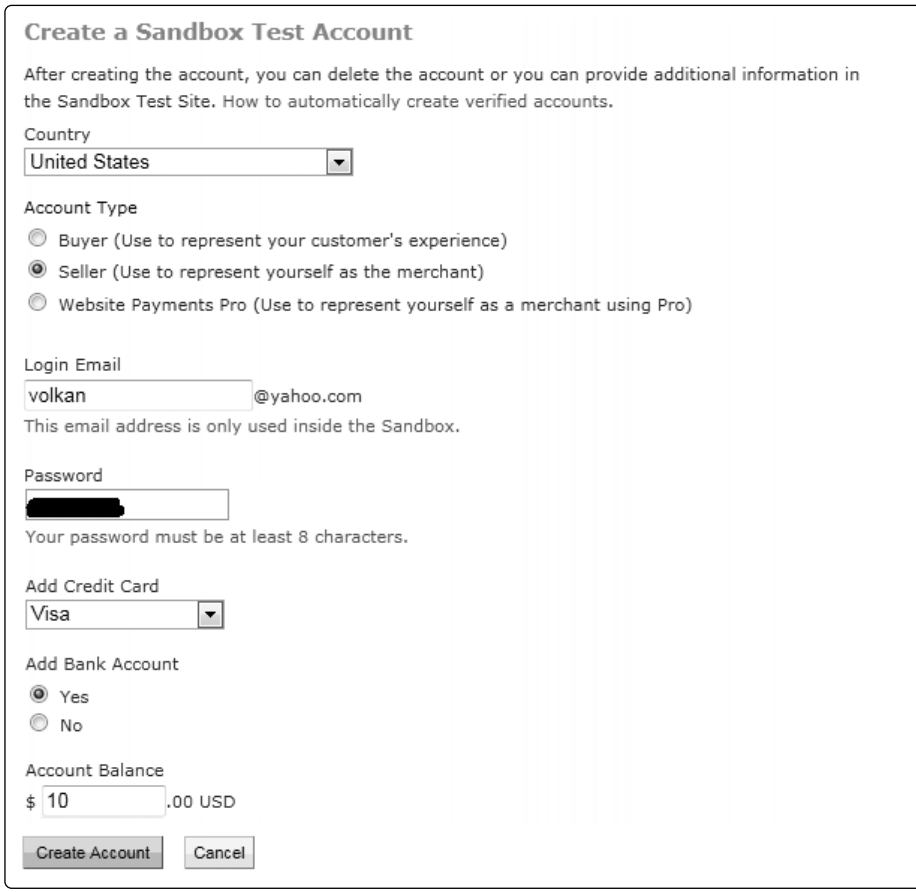
Ancak öncelikle PayPal developer api bilgilerini almalıyız.

Paypal sitesine developer olarak kaydolmak için öncelikle *https://developer.paypal.com* adresine girip kaydoluyoruz. Daha sonra ilk yapmamız gereken şey, **Test Accounts** menüsünden bir test hesabı oluşturmak.

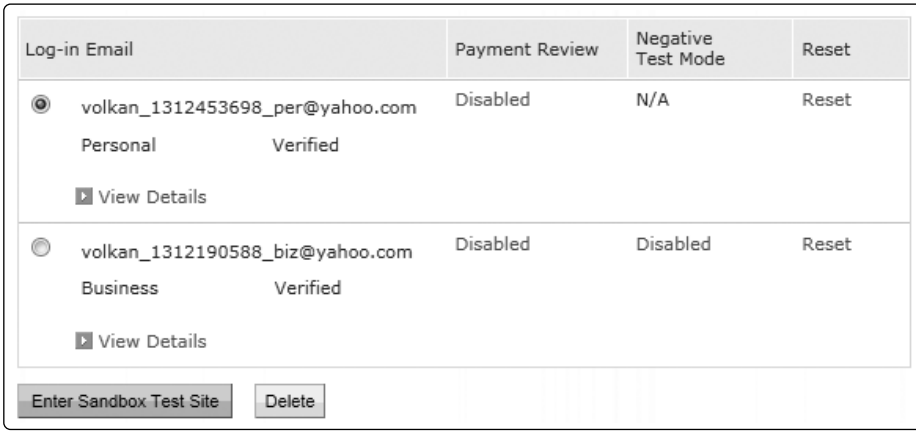


Gelen sayfada **Create Manually** ve **Preconfigured** şeklinde iki seçenek mevcuttur. **Create Manually** seçeneğinde test sitesi ile ilgili ayarları biz yapmaktayız. **Preconfigured** seçeneğinde ise ayarların bir kısmı önceden tanımlanmıştır. Biz **Preconfigured** seçeneğini tıklıyoruz.

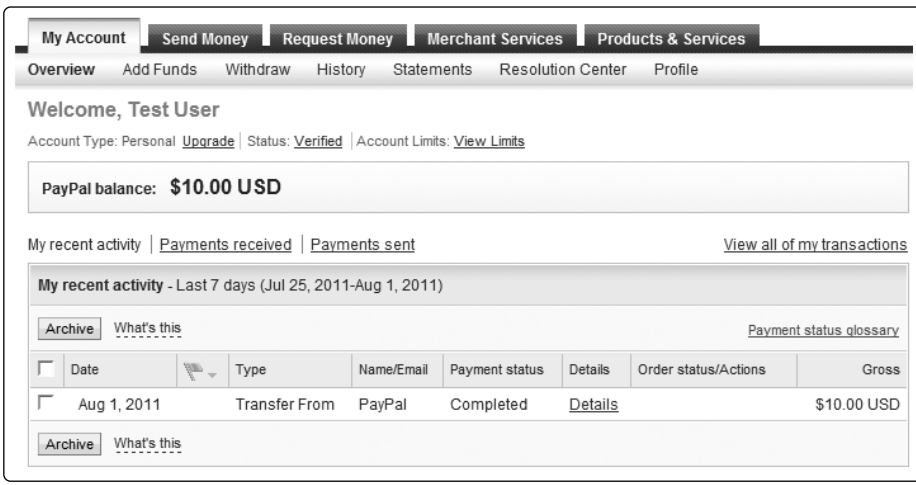
Ayarları aşağıdaki gibi yapıyoruz.



Buyer, **Seller** ve **Website Payments Pro** olmak üzere üç çeşit account tipi vardır. Biz hem site üzerinden satış yapma hem de satın alma işlemi yapacağımız için **Seller** ve **Buyer** seçeneklerini seçeceğiz. Öncelikle **Seller** seçeneğini seçip, gerekli bilgileri doldurduktan sonra **Create Account** butonunu tıklıyoruz. Aynı şekilde **Buyer** seçeneğini seçip aynı işlemleri gerçekleştiriyoruz. **Buyer Account** içerisinde **Account Balance** değerini `100$` olarak tanımladık. Bu miktar bize örnek satın alma işlemlerimiz için yeterli gelecektir. Bu işlemin ardından test account'larımız oluşturulup örnek mail adresleri verilir.

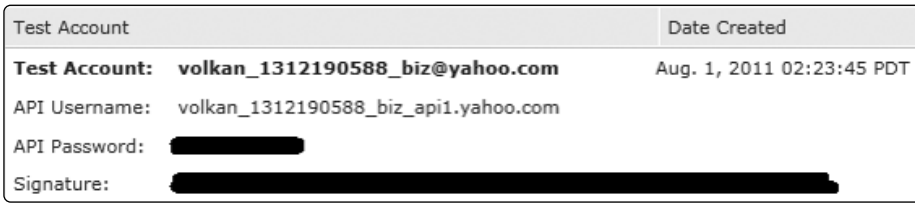


Enter Sandbox Test Site butonuna tıklayarak test siteniz ile ilgili işlemler yapabileceğiniz ayar ekranına ulaşabilirsiniz.



Test Account'umuz ile ilgili api bilgilerini öğrenmek için **API Credentials** menüsüne tıklıyoruz.

Örneğin; **Seller Account**'umuz ile bilgiler aşağıdaki gibidir.



Artık kodlama aşamasına geçebiliriz. Öncelikle *http://paypalhelper.codeplex.com/* adresinden **PayPal Helper**'ı indiriyoruz. Daha sonra projemizin içerisinde **Solution Explorer** penceresinde bin klasörüne, indirdiğimiz **PayPal.dll** dosyasını kopyalıyoruz.

Yine **Solution Explorer** içerisinde `_ViewStart.cshtml` dosyası içerisine aşağıdaki kod satırlarını yazıyoruz.

```
@{
     Layout = "~/Views/Shared/_Layout.cshtml";
     PayPal.Profile.Initialize
       ("apiUsername değeri", "apiPassword değeri",
       "apiSignature değeri","sandbox");
     PayPal.Profile.Language = "en_US";
     PayPal.Profile.CurrencyCode = "USD";
}
```

`Paypal.Profile.Initialize` metodu PayPal api'sini etkinleştirmek amacıyla kullanılır.

```
Paypal.Profile.Initialize(string apiUsername, string apiPassword,
string apiSignature, string environment)
```

... şeklinde kullanılır. `apiuserName`, `apiPassword` ve `apiSignature` değerlerini PayPal içerisinde daha önceden oluşturmuş olduğumuz **Test Account**'a ait **API Credentials** menüsünden elde ediyoruz. **Environment** ise geliştirme ortamını temsil etmekte. Değer olarak PayPal geliştirme merkezi olan sandbox değerini veriyoruz.

`Index.cshtml` adlı View dosyamız içerisine de aşağıdaki kodları ekliyoruz:

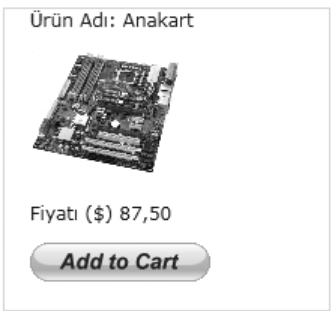
```
@using Microsoft.Web.Helpers;
@{
     ViewBag.Title = "Index";
}
```

```
@{ var payPalButton = PayPal.ButtonManager.AddToCartButton.Create(
         business: "PayPal test account ismi",
         itemName : "Anakart",
         amount : "87.50");
         HtmlString payPalButtonHtml = new
              HtmlString(payPalButton.WebSiteCode);
}
<fieldset>
     @Html.Label("Ürün Adı: Anakart")
     <img src="@Href("~/Content/anakart.gif")" alt="anakart"
          width="100" height="100"/>
     <p>
  @Html.Label("Fiyatı ($)")
          @Html.Label("87,50")
     </p>
     @payPalButtonHtml
</fieldset>
```

Kodumuz içerisinde `PayPal.ButtonManager.AddToCartButton.Create` metodu ile `AddToCartButton (Sepete Ekle)` şeklinde bir html nesnesi oluşturuyoruz.

```
HtmlString payPalButtonHtml = new HtmlString(payPalButton.WebSiteCode);
```

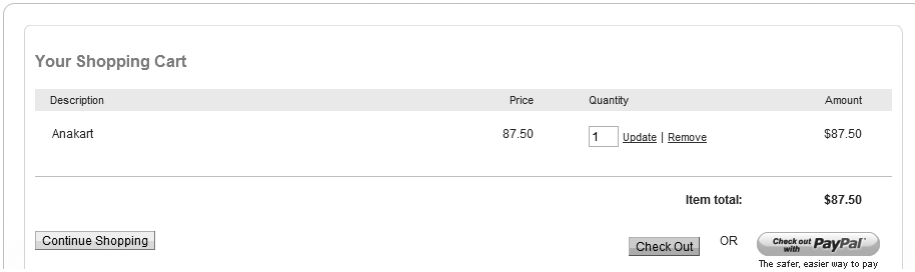
... satırında ise `@PayPalButtonHtml` tarafından çağrılacak HTML içeriği oluşturulur.



Uygulamamızı çalıştırdığımızda ürün resmimizin altına **Add to Cart** butonunun eklendiğini görüyoruz.

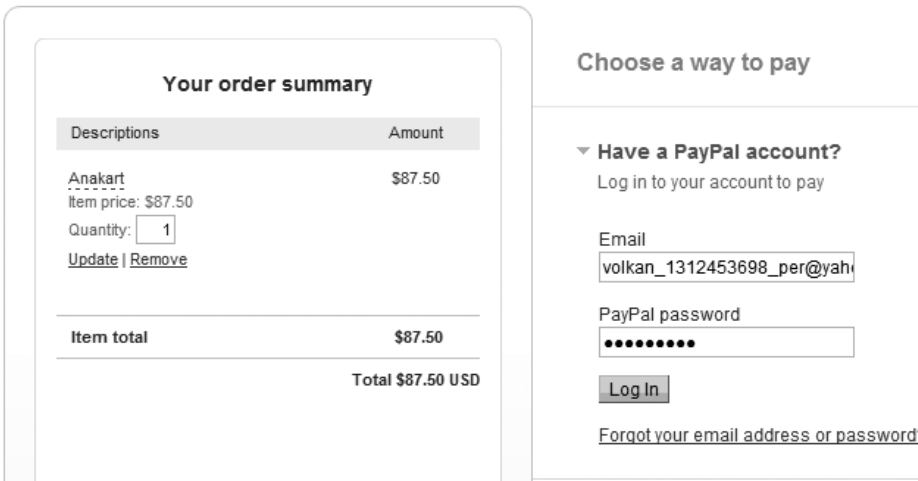
Add to Cart butonuna tıkladığımızda alışveriş sepetimizin içeriğinin görüntülendiği sayfa görüntülenir.

volkan aktaş's Test Store

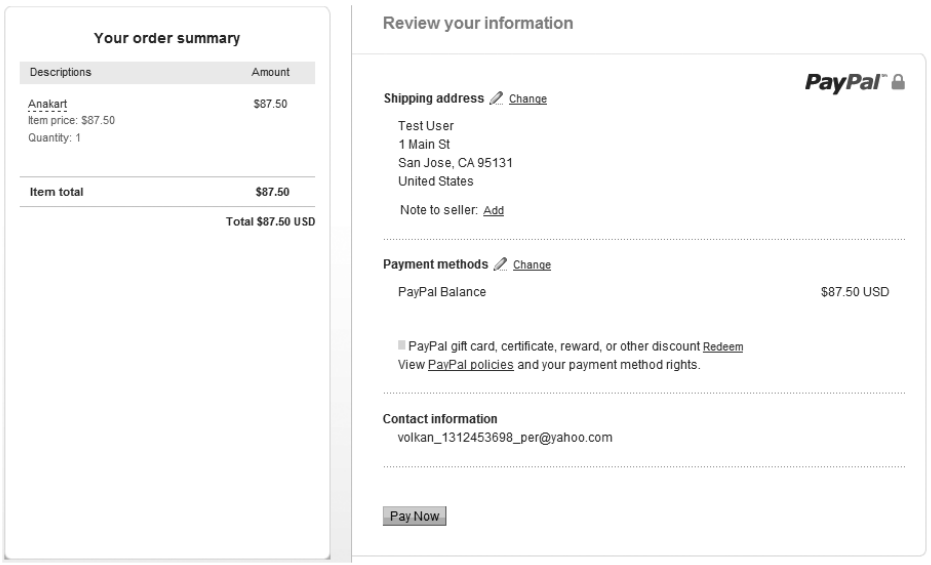


Bu sayfada **Quantity** bölümünde ürün adedini değiştirebilir, yeni ürün adedini **Update** linkine tıklayarak güncelleyebilir, **Remove** linkine tıklayarak da ilgili ürünü silebiliriz.

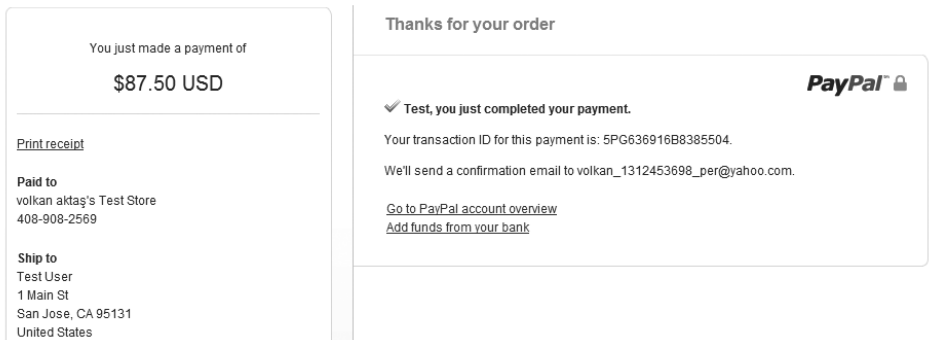
Continue Shopping butonuna tıklayarak alışverişe devam edebilir ya da **Check out with PayPal b**utonuna tıklayarak ödeme giriş sayfasına geçebiliriz.



E-mail adresine daha önce almış olduğumuz **Buyer Account** adresini, şifre kısmına da **Buyer Account** şifresini giriyoruz. **Login** butonuna tıkladığımızda ödeme sayfasına geçebiliriz.



Pay Now butonuna tıklayarak ödeme işlemini tamamlarız.

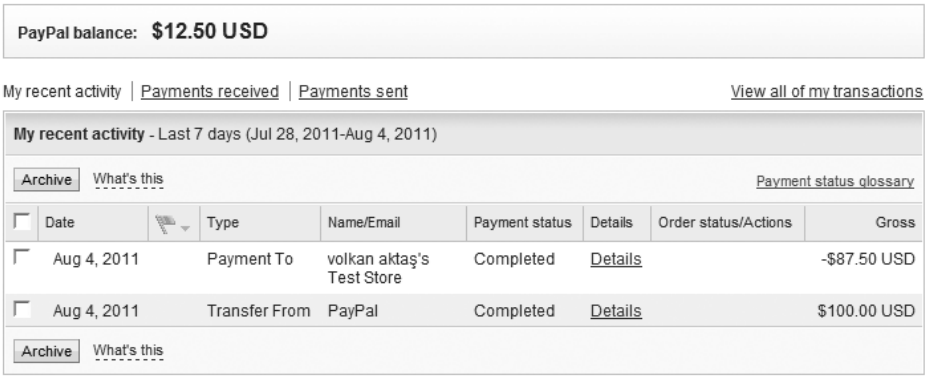


https://developer.paypal.com adresinden test account hesabımıza girip **Test Email** başlığı altında ödeme ile ilgili bilgi maili gelip gelmediğini kontrol ediyoruz.

Inbox

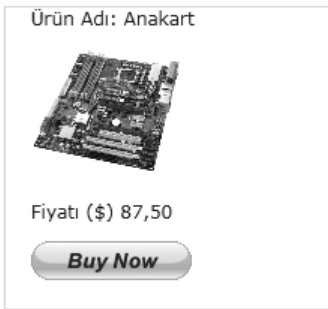
To	From	Subject	Date
volkan_1312190588_biz@yahoo.com	volkan_1312453698_per@yahoo.com	Notification of payment received	Aug. 4, 2011 03:48:22 PDT
volkan_1312453698_per@yahoo.com	service@paypal.com	Receipt for Your Payment to volkan aktaş's Test Store	Aug. 4, 2011 03:47:46 PDT
volkanaktas@yahoo.com	service@intl.paypal.com	Bir kredi kartı eklediniz	Aug. 1, 2011 01:52:30 PDT
volkanaktas@yahoo.com	service@intl.paypal.com	Bir kredi kartı eklediniz	Jul. 30, 2011 13:15:37 PDT
volkanaktas@yahoo.com	service@intl.paypal.com	Yeni bir PayPal şifresini nasıl oluşturursunuz?	Jul. 30, 2011 10:54:03 PDT

Görüldüğü gibi ödeme bilgisi mailimize gelmiş. Yine test account hesabımız içerisinde **Enter Sandbox Test Site** butonuna tıklayarak yapılan ödeme sonuçlarını görebiliriz.



PayPal Helper'ı kullanırken **AddToCartButton** dışında başka butonları da kullanabilirsiniz.

- **`BuyNow:`** Sepete eklemeden her ürünün tek tek satın alındığı ödeme türüdür. **BuyNow** butonu için `index.cshtml` dosyamız içerisindeki kodu aşağıdaki gibi değiştiriyoruz.




```
var payPalButton = PayPal.ButtonManager.BuyNowButton.Create(
        business: "volkan_1312190588_biz@yahoo.com",
        itemName: "Anakart",
        amount : "87.50");
HtmlString payPalButtonHtml = new HtmlString(payPalButton.WebSiteCode);
```

- **Subscribe:** Taksitler halinde yapılan ödeme türüdür. **Subscribe** butonu ile beraber taksitlendirme için aşağıdaki seçenekler kullanılabilir:

  Günler (“D”): 1 ile 90 arasında değer alır.

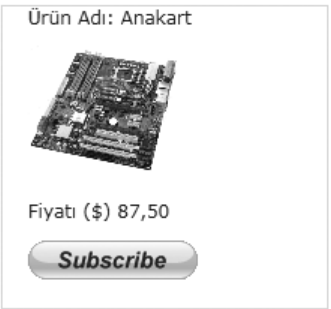
  Haftalar (“W”): 1 ile 52 arasında değer alır.

  Aylar (“M”): 1 ile 24 arasında değer alır.

  Yıllar (“Y”): 1 ile 5 arasında değer alır.

  Subscribe butonu için `index.cshtml` dosyamız içerisindeki kodu aşağıdaki gibi değiştiriyoruz.

```
var payPalButton = PayPal.ButtonManager.SubscribeButton.Create(
        “volkan_1312190588_biz@yahoo.com”,
        “Anakart”,
        “15.00”, “6”, “M”);
HtmlString payPalButtonHtml = new HtmlString(payPalButton.WebSiteCode);
```



  Kodumuz ile 6 aylık bir taksitlendirme yaparak, aylık taksit oranını 15 $ olarak belirledik.

- **Donate:** Belirtilen tutar kadar bağış yapılabilen ödeme türüdür. **Donate** butonu için `index.cshtml` dosyamız içerisindeki kodu aşağıdaki gibi değiştiriyoruz.

```
@using Microsoft.Web.Helpers;
@{
     ViewBag.Title = “Index”;
}
@{
     var payPalButton = PayPal.ButtonManager.DonateButton.Create(
       “volkan_1312190588_biz@yahoo.com”,
```

```
        "vbnetnedir.com için 10$",
        "10.00");
    HtmlString payPalButtonHtml =
          new HtmlString(payPalButton.WebSiteCode);
}
 <fieldset>
    <img src="@Href("~/Content/vbnetnedir.gif")" alt="vbnetnedir" />
    @payPalButtonHtml
</fieldset>
```



Örneğimizde **vbnetnedir.com** sitesi için `10 $`'lık bir bağış yapılmaktadır.

Kullanici Tanimli Helper Oluşturmak

MVC Framework kendi Helper metotlarımızı oluşturmaya izin vermektedir. Bu sayede kendi ihtiyaçlarımıza uygun biçimde Helper metotlarını tanımlamamız mümkün olmaktadır.

Bunu yapmanın iki yöntemi vardır.

- **Static metot**
- **Extension metot**

Static Metot

Kendi oluşturacağımız sınıf içerisinde tanımlayacağımız bir static metot kullanarak yeni bir Html Helper metodu oluşturabiliriz. Örnek olarak girilen bir mail adresinin geçerli olup olmadığını bildiren bir Helper oluşturalım. Örneğimiz için projemiz içerisine `MailKontrolClass` isimli yeni bir class ekliyoruz. İçerisine yazacağımız kodumuz aşağıdaki gibi olacak:

```
using System;
using System.Web;
using System.Text.RegularExpressions;
namespace Helper_MailKontrol
{
```

```
    public static class MailKontrolClass
    {
      public static bool mailkontrol(string mailadresi)
      {
        Regex mailDeseni = new Regex
              ("[\\w-]+(?:\\.[\\w-]+)*@(?:[\\w-]+\\.)+[a-zA-Z]{2,7}");
        if (mailDeseni.Match(mailadresi).Success)
        {
            return true;
        }
        else return false;
      }
    }
}
```

Gerek sınıfımızı, gerekse metodumuzu `static` olarak tanımlıyoruz. `mailkontrol` isimli metodumuzu `bool` türünden tanımlayıp, okunan `mailadresi` isimli mail değeri eğer geçerli bir mail adresi ise geriye `true` değeri, geçerli değilse geriye `false` değeri döndürülmesini sağlıyoruz.

Daha sonra `HomeController` isimli controller'ımızı oluşturup kodumuzu aşağıdaki gibi yazıyoruz.

```
using System;
using System.Web;
using System.Web.Mvc;
using Helper_MailKontrol;
namespace MailKontrolHelper.Controllers
{
  public class HomeController : Controller
  {
      // GET: /Home/
      public ActionResult Index()
      {
        return View();
      }
      [HttpPost]
      public ActionResult MailKontrol()
      {
        string mail = Request["GirilenMail"];
```

```
        if (MailKontrolClass.mailkontrol(mail))
        {
          return Content("Mail Adresi Geçerli");
        }
        else
            return Content("Mail Adresi Geçersiz");
      }
  }
}
```

`MailKontrol` isimli `ActionResult` içerisinde index sayfasından gelecek olan `GirilenMail` değeri okunarak `MailKontrolClass` isimli sınıfımız içerisindeki `mailkontrol` metodu çalıştırılmakta ve mail adresinin geçerli olup olmadığı kontrol edilmektedir. Kontrol sonucunun olumlu ya da olumsuz oluşuna göre index sayfası içerisinde görüntülenecek olan sonuç geriye döndürülmektedir. `MailKontrolClass` sınıfı içerisindeki `mailkontrol` metodunu kullanabilmemiz için...

```
using Helper_MailKontrol;
```

... şeklinde `Helper_MailKontrol` namespace'ini projemize dahil etmemiz gerekmektedir.

Verilerimizi index sayfası içerisine ajax yardımıyla görüntüleyeceğimiz için `_Layout.cshtml` dosyası içerisinde `<head> </head>` tagları arasına...

```
script src="@Url.Content("~/Scripts/jquery.unobtrusive-ajax.js")"
type="text/javascript"></script>
```

... satırını ekliyoruz.

Son olarak `Index.cshtml` isimli View sayfasının kodlarını yazıyoruz. Kodumuz aşağıdaki gibi olacak:

```
@{
     ViewBag.Title = "Index";
}
@{
     HtmlHelper.UnobtrusiveJavaScriptEnabled = true;
}
<fieldset>
<legend>Mail Kontrol Helper</legend>
 @using (Ajax.BeginForm("MailKontrol",
```

```
        new AjaxOptions { HttpMethod = "POST",
             UpdateTargetId = "sonuc",
             InsertionMode = InsertionMode.Replace
        }))
 {
    @Html.Label("Mail adresini giriniz:")
    <br />
    @Html.TextBox("GirilenMail")
    <br />
    <input type="submit" value="İşlem Yap" />
 }
</fieldset>
<div id="sonuc"/>
```

Örneğimizde...

```
HtmlHelper.UnobtrusiveJavaScriptEnabled = true;
```

... satırında unobtrusive JavaScript özelliğini aktifleştiririz. `UnobtrusiveJavaScript` yapısı Html ve JavaScript kodlarını birbirinden ayırmak amacıyla kullanılır.

```
@using (Ajax.BeginForm("MailKontrol",
        new AjaxOptions { HttpMethod = "POST",
            UpdateTargetId = "sonuc",
            InsertionMode = InsertionMode.Replace
        }))
 {
```

---

satırlarında `HomeController` içerisinde bulunan `MailKontrol` isimli ActionResult metodunu çağırıyoruz. Dönen değerleri kodumuzun en altında...

```
<div id="sonuc"/>
```

... satırı ile oluşturmuş olduğumuz

sonuc isimli div tag'ı içerisinde görüntülüyoruz.

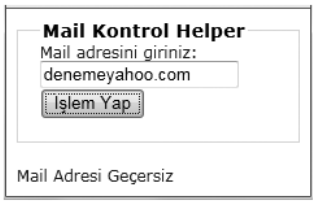
```
InsertionMode = InsertionMode.Replace
```

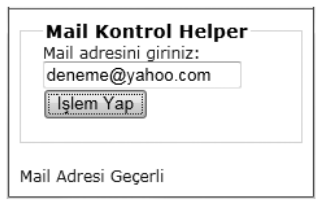
... satırında ise dönen değerlerin her defasında ne şekilde ekleneceğini belirliyoruz. `Replace` şeklinde belirttiğimizde her görüntüleme işlemi öncesinde var olan değerler silinerek, yerine yeni değerler yazılır.

`InsertionMode` özelliği ayrıca `InsertBefore` ve `InsertAfter` şeklinde iki değer

daha alır. `InsertBefore` seçildiğinde yeni değer, görüntülenen değerden daha önce yazılır. `InsertAfter` seçildiğinde ise daha sonra yazılır.

Örneğimizi çalıştırıp, TextBox içerisine örnek bir mail adresi girelim. Formun alt tarafında sonucu görebiliriz.





Extension Metot

Var olan Html Helper metotlarına **Extension Methods** (*Genişletme Metotları*) kullanarak ek özellikler ekleyebiliriz. Bu sayede oluşturduğumuz Html Helper metotlarını `Html.Metotadı` şeklinde kolayca çağırabiliriz. Örnek olarak girilen bir string değerin tüm karakterlerini, büyük harf, küçük harf ya da kelimelerin ilk harfi büyük şeklinde değiştirebilecek bir Helper metodu, extension yöntemiyle oluşturalım.

Projemize öncelikle `MyHtmlHelper` ismiyle yeni bir class ekliyoruz. Üzerinde ekleme yapacağımız Helper metodu `TextBox Helper`. TextBox Helper'ın kullanımı aşağıdaki gibidir:

```
public static MvcHtmlString TextBox(this HtmlHelper htmlHelper,
       string name)
```

Buradan da görüldüğü gibi Html Helper metodu yazarken birkaç hususa dikkat etmek gerekmektedir. Bunlar;

- İlk parametre...

  `this HtmlHelper htmlHelper`

  ... şeklinde `this` ifadesi ile Html Helper üründen tanımlanmalıdır.

- Tanımlanacak metot `static` türünden tanımlanmalıdır.

Örneğimizde temel olarak `TextBox Html Helper`'ı alıp üzerinde değişiklik yapacağız. `MyHtmlHelper` isimli class kodumuz aşağıdaki gibi olacak:

```
using System;
using System.Web;
```

```
using System.Web.Mvc;
namespace AdvTextHelper
{
  public static class MyHtmlHelper
  {
    /// <summary>
    /// Extension TextBox Helper.
    /// ceviri_tipi parametresi
    /// "TumuBuyuk,TumuKucuk veya ilkHarflerBuyuk"
    /// şeklinde değer alır
    /// </summary>
       public static string TextBox(this HtmlHelper htmlHelper,
              string name, string ceviri_tipi)
    {
        if (!String.IsNullOrEmpty(ceviri_tipi))
        {
            if (ceviri_tipi==”TumuBuyuk”)
            {
                name=name.ToUpper();
            }
            else if (ceviri_tipi==”TumuKucuk”)
            {
                name=name.ToLower();
            }
            else if (ceviri_tipi==”ilkHarflerBuyuk”)
            {
                name=System.Globalization.CultureInfo.
                       CurrentCulture.TextInfo.
                       ToTitleCase(name.ToLower());
            }
            return (name);
        }
        return null;
    }
  }
}
```

`TextBox` isimli Helper metodumuza `ceviri_tipi` isimli yeni bir parametre ekledik. Bu şekilde kullanıcının belirttiği çeviri türüne göre metodumuz içerisinde işlem yaptırıp, sonucu geri döndürüyoruz. Ayrıca kodumuz içerisinde Html Helper metodumuzu tanımlarken XML yorum satırlarını da kullandık. Bu şekilde hem ex-

tension metodumuzun ayırt edilmesini, hem de metota yeni eklediğimiz parametre hakkında bilgi vererek, kullanıcıların ilgili parametreye hangi değerleri gireceği hakkında bilgi sahibi olmasını sağladık.

`HomeController` isimli Controller'ı ekleyip, `Index.cshtml` isimli View dosyamızı oluşturuyoruz. `Index` View içerisine aşağıdaki kodları ekliyoruz.

```
@using AdvTextHelper;
@{
    ViewBag.Title = "Index";
}
@Html.TextBox("deneme cümlesi",ceviri_tipi:"ilkHarflerBuyuk")
```

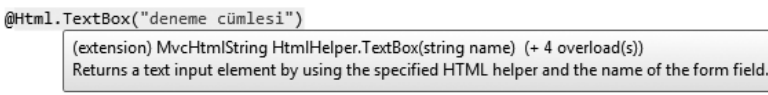
Görüldüğü gibi kodumuz içerisinde...

```
@using AdvTextHelper;
```

... ifadesi ile `AdvTextHelper` isimli namespace'in projemize dahil edilmesini sağladık.

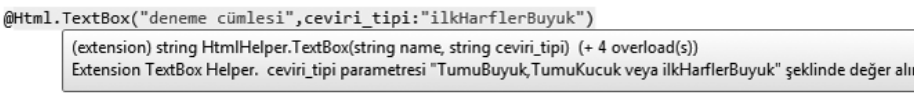
`ceviri_tipi` isimli parametremize `ilkHarflerBüyük` değerini aktardık.

`Html.TextBox` Helper metodunu...



... şeklinde standart hali ile kullanıp, farenin imlecini `TextBox` metodu üzerine getirdiğimizde standart xml yorum bilgilerini görebiliriz.

Extension metot halinde kullandığımızda ise kendi hazırladığımız xml yorum bilgilerini görebiliriz.



Uygulamamızı çalıştırdığımızda `Html TextBox Helper` içerisindeki ifadenin `Deneme Cümlesi` şeklinde kelimelerin ilk harfleri büyük olarak yazdırıldığını görebiliriz.

FİLTRELER

8

Filtreler, web sitesine gelen her bir istek sırasında devreye sokup çeşitli işlemler gerçekleştirebileceğiniz özel sınıflardır. MVC içerisinde bir Controller ya da bir Action metodu çalıştırılmadan önce veya çalıştırıldıktan sonra istenilen işlemler, filtreler yardımıyla gerçekleştirilebilir.

MVC içerisinde dört çeşit filtre kullanılabilmektedir. Filtreler ve implement edildiği sınıflar ve interface'ler aşağıdaki gibidir.

Filter	Interface	Default implementation
Authorization Filter	`IAuthorizationFilter`	`AuthorizeAttribute`
Action Filter	`IActionFilter`	`ActionFilterAttribute`
Result Filter	`IResultFilter`	`ActionFilterAttribute`
Exception Filter	`IExceptionFilter`	`HandleErrorAttribute`

- **`Authorization:`** Diğer filtreler veya Action metotlarından önce çalıştırılır. Bir Controller ya da Conroller Action'a erişimi kısıtlamak amacıyla çalıştırılır.
- **`Exception:`** Yalnızca diğer bir filtre, bir `Action` metot ya da `Action Result` tarafından oluşturulan istisna durumu sonucunda çalıştırılır.
- **`Action:`** Action metottan önce ve sonra çalıştırılır.
- **`Result:`** Action Result çalıştırılmadan önce ve çalıştırıldıktan sonra gerçekleştirilir.

Filtre tanımlanırken Filter **attribute**'ı (*niteliği*) Action metot veya Controller ile birlikte kullanabilirsiniz. Eğer Filter niteliği bir Controller için kullanılırsa, Action Filter Controller içerisindeki bütün Action metotlar için geçerlidir.

```
[Authorize]
public class HomeController : Controller
{
     public ActionResult Index()
     {
          ViewBag.Message = "KODLAB ASP.NET MVC!";
          return View();
     }
     public ActionResult About()
     {
          return View();
     }
}
```

Yukarıdaki örnekte `HomeController` sınıfının varsayılan tanımlaması görülmektedir. Örnekte `Authorize` niteliği Controller ile birlikte kullanılarak `Authorize` Filter'ın Controller içerisindeki bütün Action metotlara uygulanması sağlanmıştır.

İstenildiği takdirde herhangi bir Action metot için ayrıca filtre uygulanabilir.

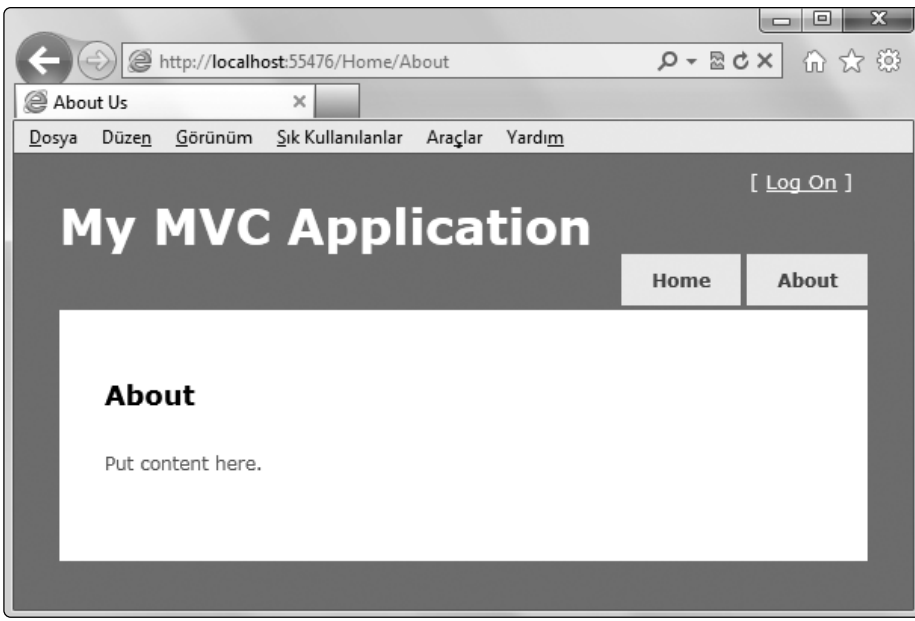
```
[Authorize]
public class HomeController : Controller
{
     public ActionResult Index()
     {
          ViewBag.Message = "Welcome to ASP.NET MVC!";
          return View();
     }
     [HandleError]
     public ActionResult About()
     {
          return View();
     }
}
```

Yukarıdaki örnekte `Authorize` Filter, Controller içerisindeki bütün Action metotlara uygulanırken, `HandleError` Filter yalnızca `About` isimli Action metoda uygulanmıştır.

Authorization Filter

`Authorization` Filter, bir Controller ya da Action metoda güvenli bir şekilde erişilmesini sağlar. Oturum açmış bir kullanıcının ya da erişim hakkı verilmiş özel bir kullanıcının erişim sağlaması gibi izinler tanımlanabilir ya da login olmamış kullanıcıların otomatik olarak login sayfasına yönlendirilmesini sağlar. `Authorization` Filter, Windows tabanlı ve web tabanlı doğrulama işlemlerinin her ikisini de destekler.

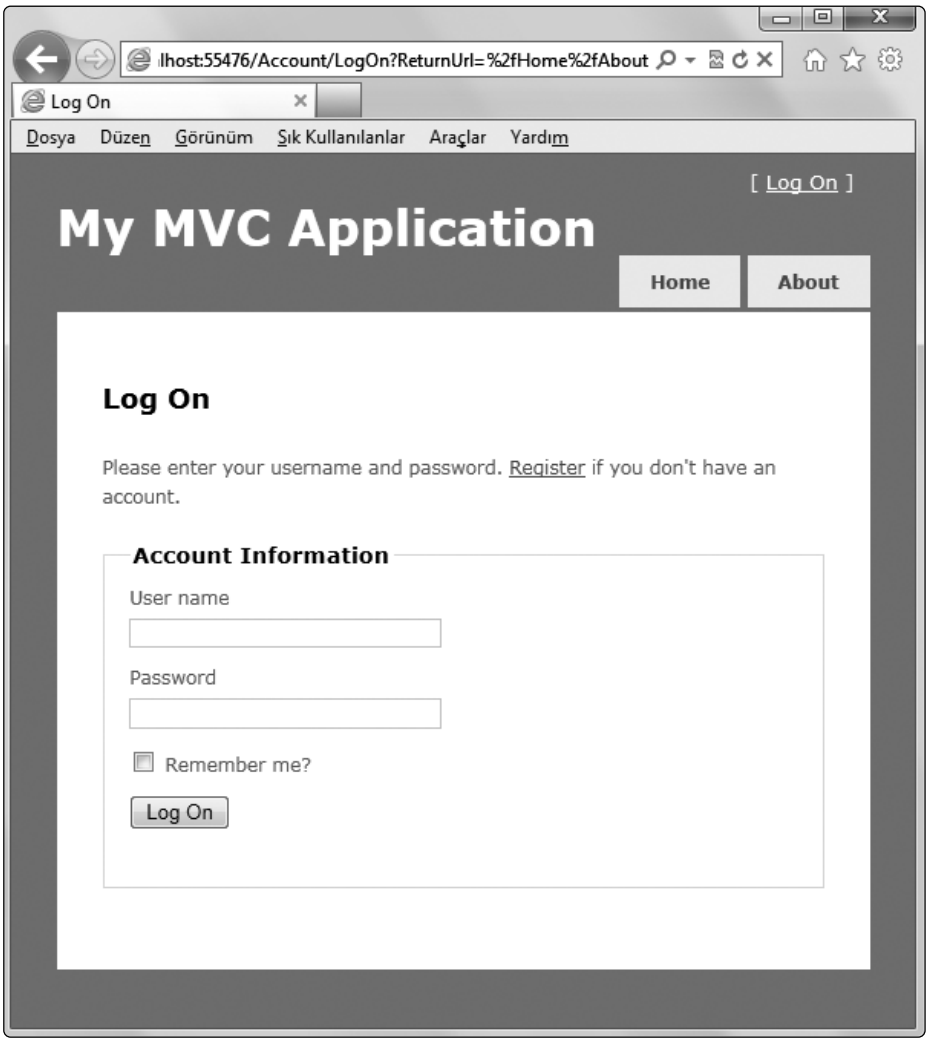
Örnek olarak **Internet Application** şablonunu kullanarak hazır bir **MVC 3 Razor** projesi açalım. Internet Application proje örneğini çalıştırıp, About sayfasını açmaya kalktığımızda herhangi bir uyarı ya da engelleme olmadan About sayfasını görebiliriz.



`Home` Controller'un içerisini açarak `About Action` metodumuzun baş tarafına `[Authorize]` niteliğini ekleyelim.

```
[Authorize]
public ActionResult About()
{
    return View();
}
```

Uygulamamızı yeniden çalıştırıp `About` sayfamızı görüntülemeye kalktığımızda yetkili kullanıcı olarak ilgili sayfayı görüntüleyebilmemiz için login sayfası görüntülenmektedir.



Bir Action metoda erişimi kısıtlamak için kullanılacak yöntemleri `AuthorizeAttribute` sınıfını kullanarak belirleyebiliriz. `AuthorizeAttribute` sınıfının iki temel özelliğini kullanarak yetkilendirme yapabiliriz.

- **`Users:`** Action metoda erişebilecek kullanıcıların listesini aralarına virgül karakteri koyarak yazabiliriz.
- **`Roles:`** Kullanıcıların Action metoda erişim seviyesini belirleyen rolleri içerir. Her bir kullanıcı için bu rollerden en az biri tanımlanmalıdır.

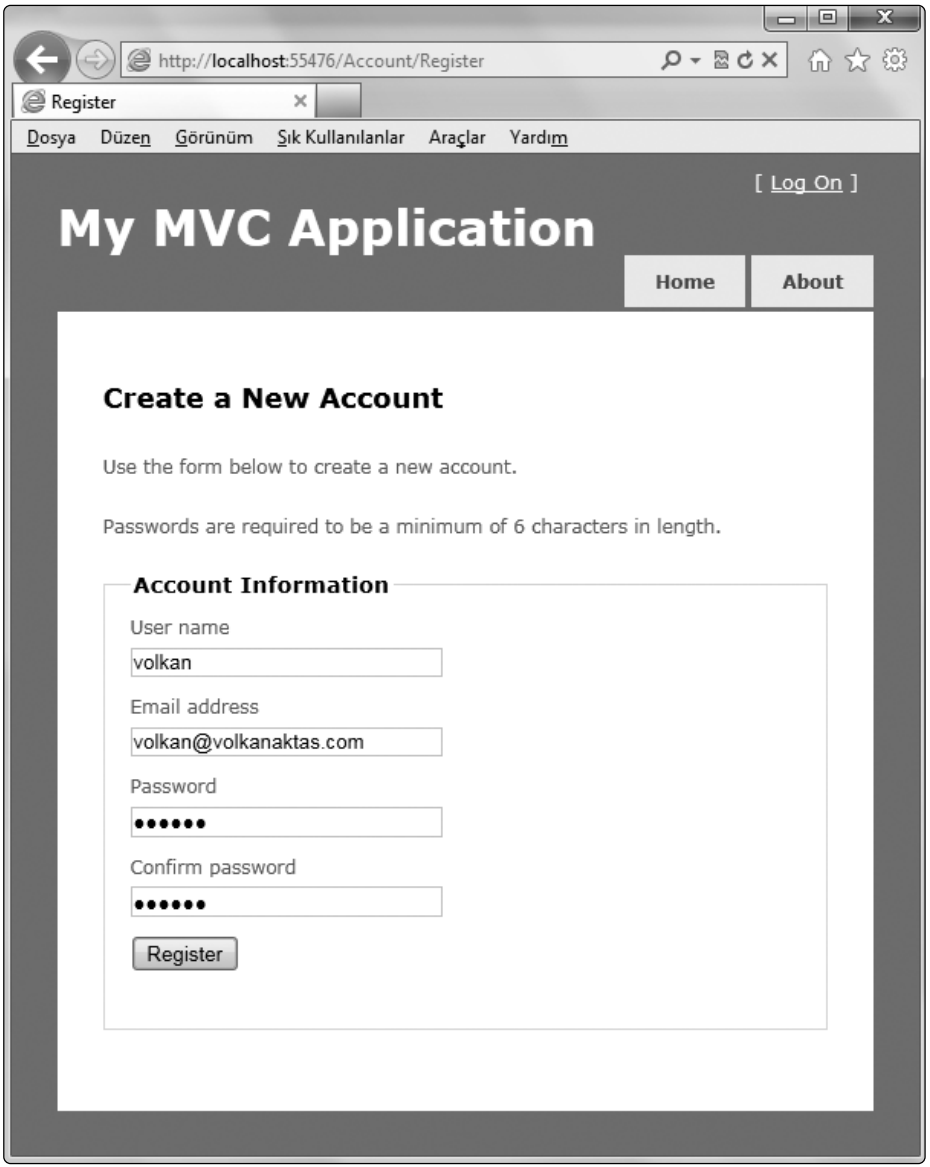
`Users` özelliğine örnek olarak; yukarıdaki örneğimiz üzerinde değişiklik yaparak, yalnızca belirlediğimiz bir kullanıcının ilgili Action metodu çalıştırmasını sağlayalım.

`Home` Controller içerisini açarak `About` Action metodumuzun baş tarafına `[Authorize]` niteliğini ekleyelim.

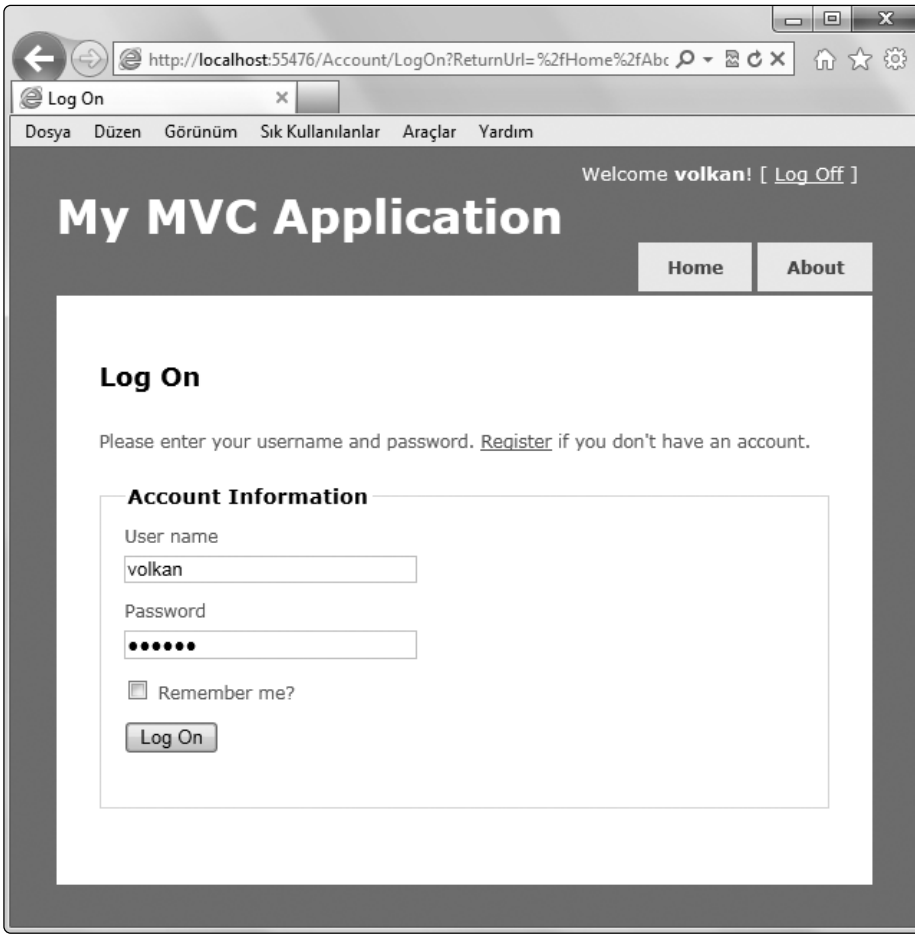
```
[Authorize(Users="volkan")]
public ActionResult About()
{
    return View();
}
```

Kodumuzda yalnızca `volkan` kullanıcısının `About` Action metodunu çalıştırabileceğini belirledik.

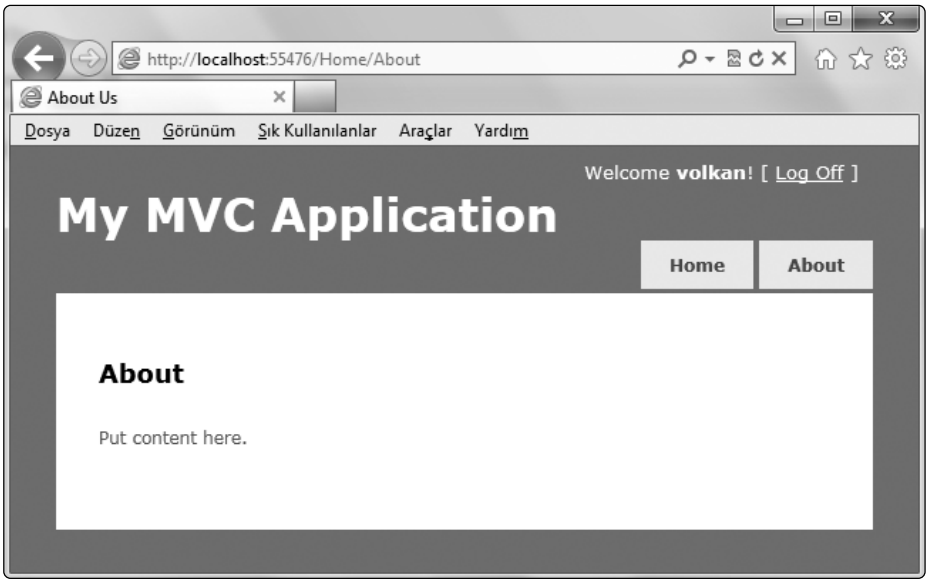
Uygulamamızı yeniden çalıştırıp `About` sayfamızı görüntülemeye kalktığımızda, yetkili kullanıcı olarak ilgili sayfayı görüntüleyebilmemiz için **Login** sayfası görüntülenmektedir. Bu sayfadaki **Register** linkine tıklayarak yeni bir kullanıcı kaydı yapabileceğimiz Register sayfasını görüntüleyelim.



Burada `volkan` isimli yeni bir kullanıcı kaydı yapıp, `About` sayfasını yeniden açarak volkan kullanıcısı için login olalım.



Volkan kullanıcısı ile login olduğumuzda artık `About` sayfasını görüntüleyebiliriz.



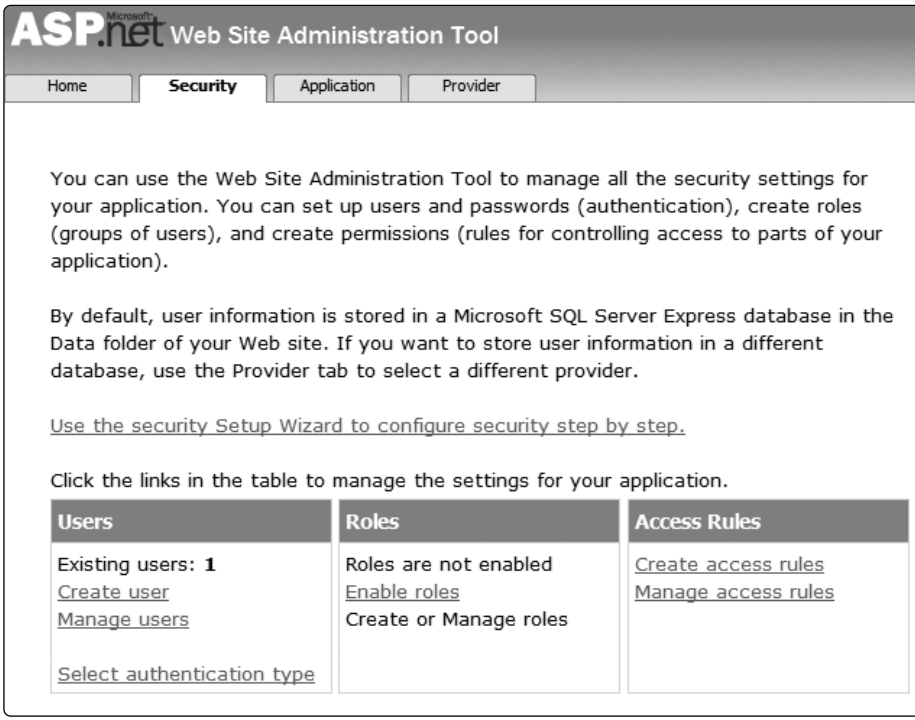
Roles özelliğine örnek olarak iki ayrı kullanıcıya erişim hakları tanımlayalım.



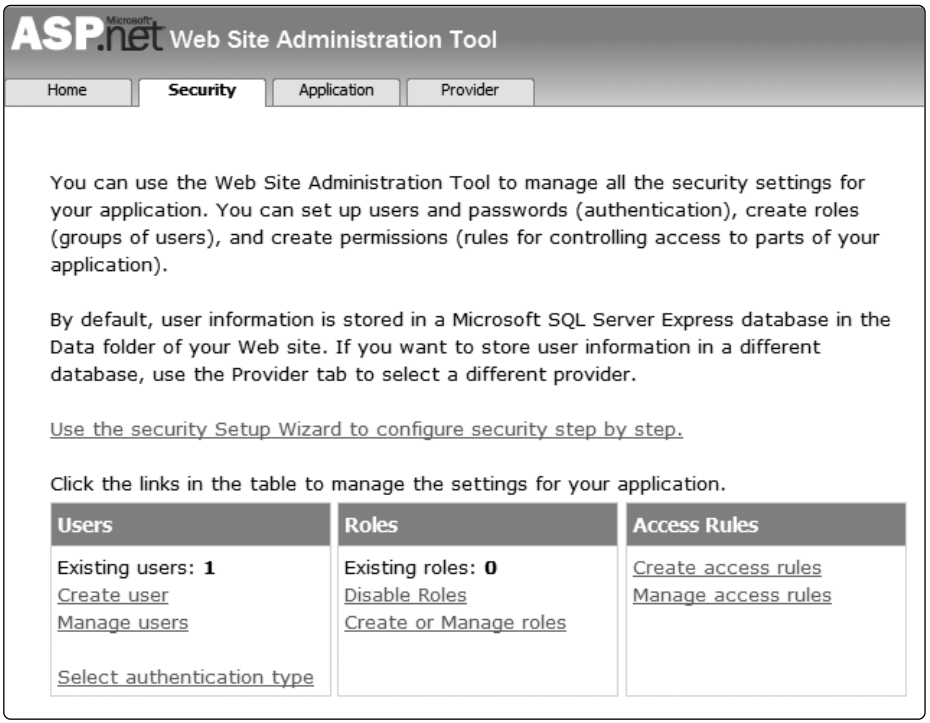
Erişim hakkı tanımlama işlemlerini **Visual Studio** içerisinde **Project** menüsünden **ASP.NET Configuration** alt menüsü ile ya da **Solution Explorer** içerisinde **ASP.NET Configuration** butonu ile gerçekleştirebiliriz.

Karşımıza işlemlerimizi gerçekleştirebileceğimiz **ASP.NET Web Site Administration Tool** isimli uygulama penceresi gelmektedir. Kullanıcı tanımlama ve rol ekleme gibi düzenleme işlemlerini bu penceredeki **Security** sekmesi altında gerçekleştireceğiz.

Öncelikle erişim rollerini aktifleştirmek için **Roles** başlığı altındaki **Enable Roles** linkine tıklamamız gerekiyor.



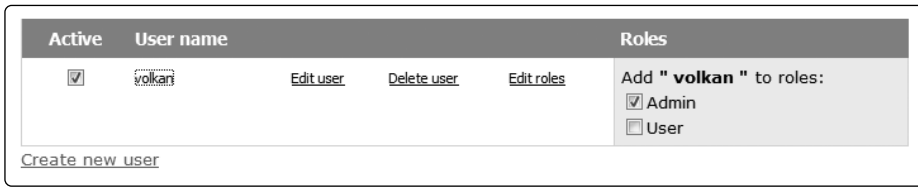
Bu işlemin ardından aktifleşen **Create or Manage** roles linkine tıklıyoruz.



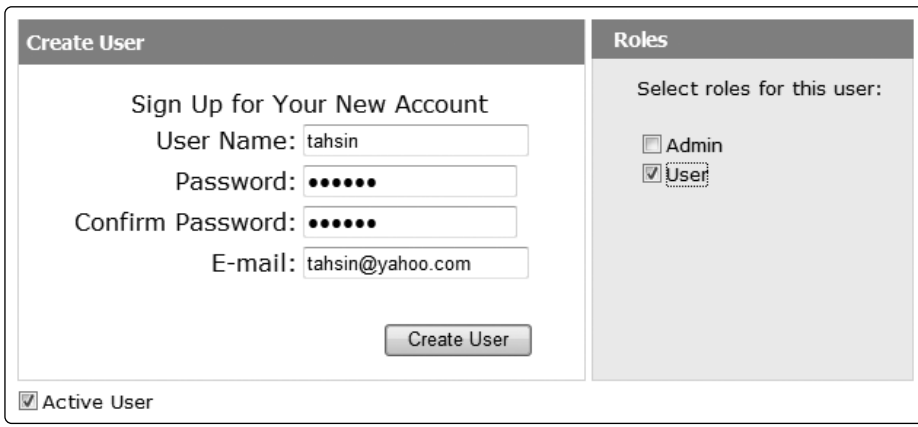
Karşımıza gelen pencerede tanımlamak istediğimiz role isimlerini yazarak **Add Role** butonuna tıklıyoruz.



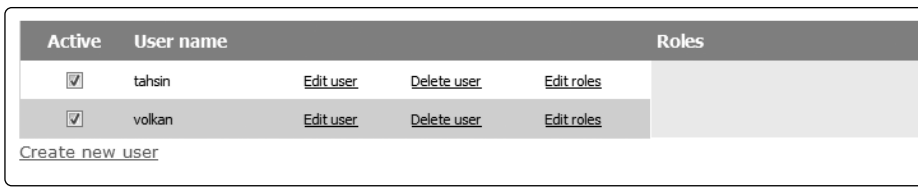
Daha sonra **Security** sekmesine geri dönüp **Users** başlığı altındaki **Manage users** linkine tıklıyoruz.



Burada mevcut olan kullanıcının isminin yan tarafında eklediğimiz Role isimleri listelenmektedir. Uygun Role ismini işaretliyoruz. Daha sonra örneğimiz için yeni bir kullanıcı daha tanımlamak amacı ile **Create New user** linkine tıklayarak yeni bir kullanıcıyı daha uygun rol ile oluşturuyoruz.



Örneğimiz için `volkan` kullanıcısını **Admin** olarak tanımlarken, tahsin kullanıcısını **User** olarak tanımladık.

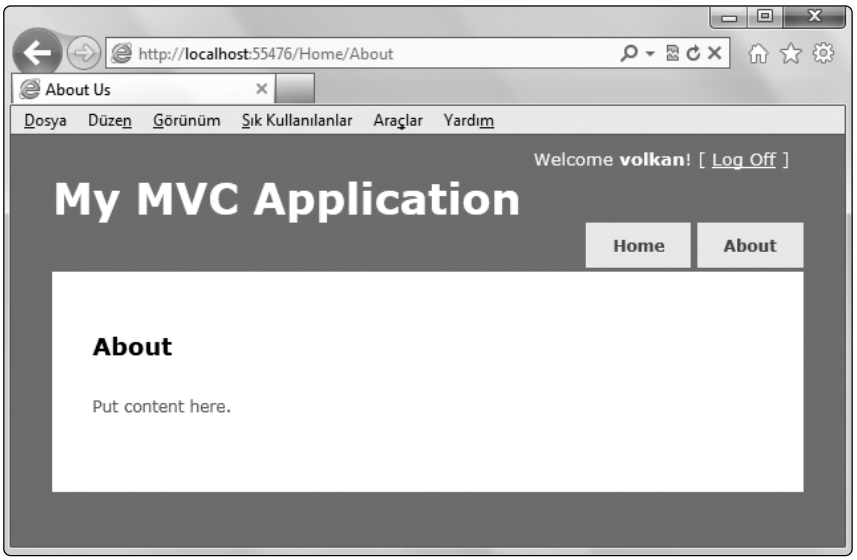


Kullanıcı ve role tanımlamalarını bitirdikten sonra artık proje kodumuza dönebiliriz. `Home` Controller içerisini açarak `About Action` metodumuzu aşağıdaki gibi düzenleyelim.

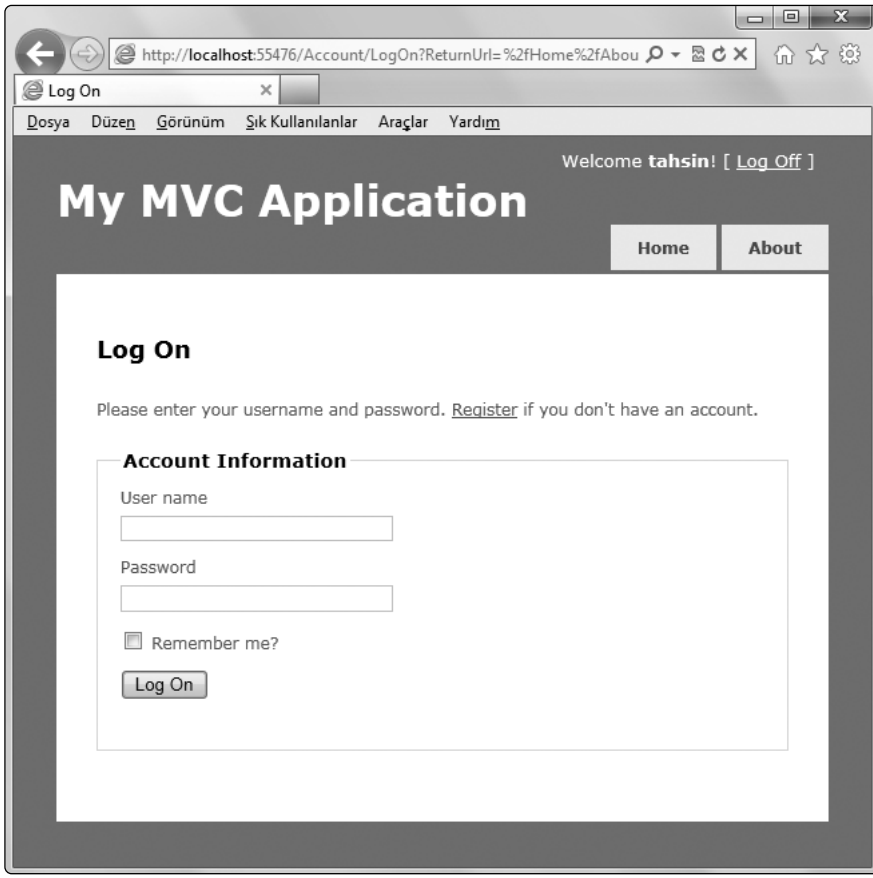
```
[Authorize(Users="volkan,tahsin")]
[Authorize(Roles = "Admin")]
```

```
public ActionResult About()
{
    return View();
}
```

Uygulamamızı yeniden çalıştırıp `About` sayfamızı açalım ve ilk olarak `volkan` kullanıcısı için giriş yapalım.



Görüldüğü gibi `volkan` kullanıcısı olarak `About` sayfasını görüntüledik. Şimdi de çıkış yaparak `tahsin` kullanıcısı için giriş yapalım.



Tahsin kullanıcısı ile giriş yapmamıza rağmen `About` sayfasını görüntüleyemedik. Çünkü kodumuz içerisinde...

```
[Authorize(Roles = "Admin")]
```

... satırı ile `About` sayfasını yalnızca Admin rolüne sahip olan kullanıcıların görüntüleyebileceğini belirtmiştik. `tahsin` kullanıcısı ise User rolüne sahip olduğu için `About` sayfasını görüntüleyemedi.



Kullanıcı ve rol tanımlaması yaptığımız

```
[Authorize(Users="volkan,tahsin")]
    [Authorize(Roles = "Admin")]
```

... satırlarını...

```
[Authorize(Users="volkan,tahsin",Roles="Admin")]
```

... biçiminde yazarak da kullanabiliriz.

`System.Web.Security` sınıfının `Roles.GetAllRoles` metodunu kullanarak tanımlanmış olan rollerin listesini elde edebiliriz. Örneğimiz üzerinde gerekli değişiklikleri yapalım.

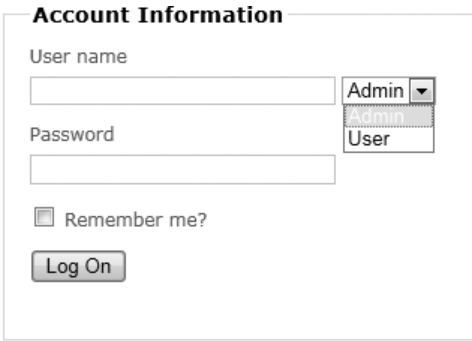
AccountController.cs isimli Controller dosyasını açarak `LogOn` isimli `ActionResult` kodunu aşağıdaki gibi yeniden düzenleyelim:

```
public ActionResult LogOn()
{
    SelectList rl = new SelectList(System.Web.Security.Roles.GetAllRoles(),
       "roleList");
    ViewData["roleList"] = rl;
    return View();
}
```

Daha sonra **Account** klasörü içerisindeki **LogOn.cshtml** isimli **View** dosyasını açarak `editör-field` isimli `div` kodunu aşağıdaki gibi yeniden düzenleyelim.

```
<div class="editor-field">
    @Html.TextBoxFor(m => m.UserName)
    @Html.ValidationMessageFor(m => m.UserName)
    @Html.DropDownList("roleList")
</div>
```

Görüldüğü gibi artık login işlemi sırasında mevcut Role isimleri de listelenmektedir.



Exception Filter

Yalnızca diğer bir filtre, bir Action metot ya da Action result tarafından oluşturulan istisna durumu sonucunda çalıştırılır. MVC içerisinde `HandleError` niteliği varsayılan `Exception` Filter olarak kullanılmaktadır.

Oluşan istisna durumu sırasında yönlendirme işlemleri için `View` ve `Master` özellikleri kullanılabilir.

- **`View`** özelliği, Filter işlemi sırasında kullanılacak View ismini belirtir. Eğer herhangi bir isim belirtilmezse varsayılan değeri `Error` olacaktır.
- **`Master`** özelliği, Filter işlemi sırasında kullanılacak layout ismini belirtir. Eğer herhangi bir isim belirtilmezse varsayılan layout sayfası kullanılacaktır.

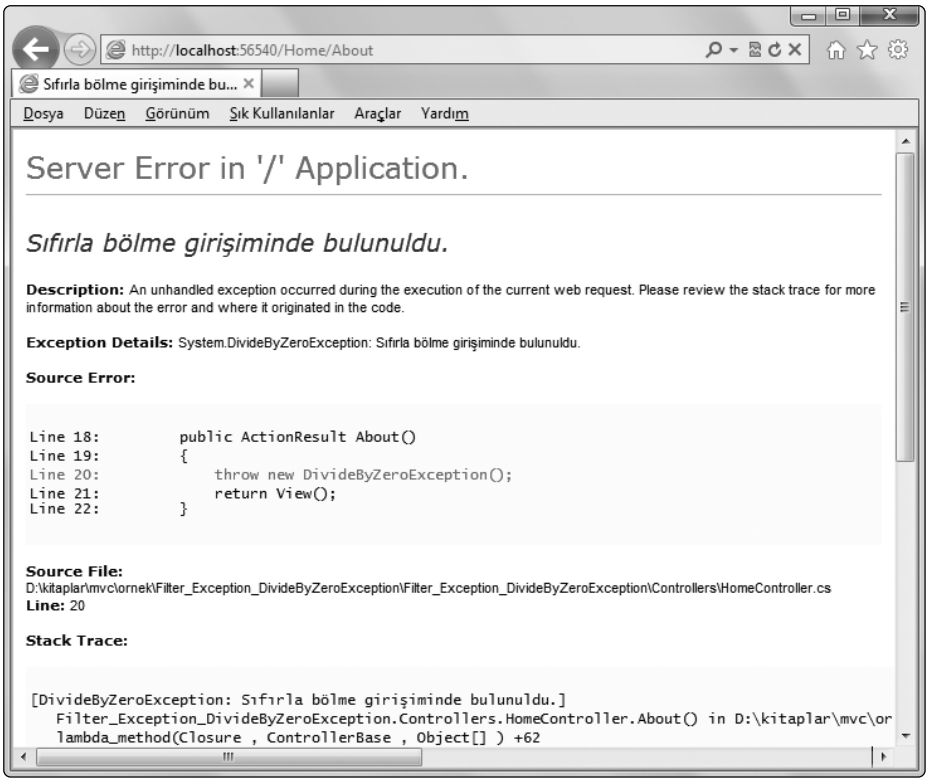
Örnek olarak **Internet Application** şablonunu kullanarak hazır bir **MVC 3 Razor** projesi açalım. **HomeController.cs** dosyasını açıp `About` isimli Action kodunu aşağıdaki gibi değiştirelim.

```
public ActionResult About()
{
    throw new DivideByZeroException();
    return View();
}
```

`throw` metodu belirtilen hata sınıfından yeni bir `Exception` nesnesi oluşturarak bir hata durumu gerçekleştirmek amacı ile kullanılır. Kodumuz içerisinde `DivideByZeroException` (*sıfıra bölme*) hatası oluşturduk.

Hata ayıklaması yapılmadan uygulamamızı hata durumu ile birlikte çalıştırabilmek

için **Ctrl+F5** tuşlarını kullanıyoruz. `About` sayfasını görüntülediğimizde aşağıda görüldüğü gibi About sayfası hata kodlarını ve mesajını görüntüleyerek açılacaktır.

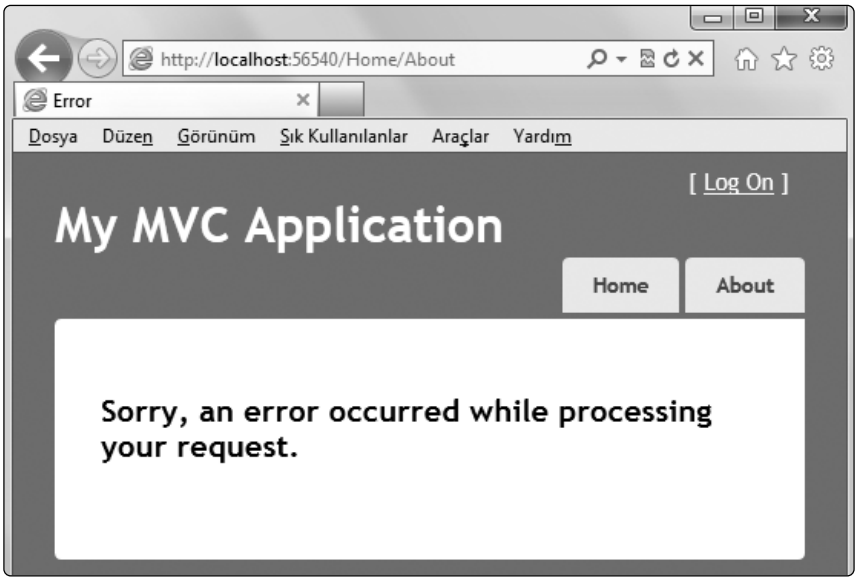


MVC'nin varsayılan `Error View` template'ini kullanarak oluşan hata durumlarında sayfamızın daha düzenli görünmesini sağlayabiliriz. Bunun için **web.config** dosyası içerisini açıp `system.web` başlığı altına...

```
<customErrors mode="On"/>
```

... satırını ekliyoruz.

Uygulamamızı yeniden çalıştıralım.



Varsayılan görünüm ve hata mesajı ile kullanıcı bilgilendirilerek `About` sayfası görüntülendi. Uygulama içerisinde oluşan hatalar `Exception` nesnesinden türetilmektedir. `Exception` nesnesi altında bulunan başlıca hatalar şunlardır:

`Exception`

- `ApplicationException`
  - `Your own exception types`
- `SystemException`
  - `ArgumentException`
    - `ArgumentNullException`
    - `ArgumentOutOfRangeException`
- `DivideByZeroException`
- `IndexOutOfRangeException`
- `NullReferenceException`
- `RankException`
- `StackOverflowException`
- `IOException`
    - `EndOfStreamException`
    - `FileNotFoundException`
    - `FileLoadException`

Oluşan istisna durumlarında View ve Master özellikleri ile özel olarak View ve Layout'lar biçiminde hata bilgilendirme sayfaları oluşturabiliriz. Örnek olarak yukarıda yaptığımız uygulamayı, oluşan hata durumunda bizim oluşturduğumuz View sayfasına yönlenecek şekilde değiştirelim.

Örneğimizde varsayılan olarak **Shared** klasörü altında **Error.cshtml** isimli View'in mevcut olması gerekmekte. Eğer mevcut değilse, **Solution Explorer** içerisinde **Views/Shared** klasörüne Mouse'un sağ tuşu ile tıklayarak **Add View** seçeneği ile `Error` isimli yeni bir View ekleyelim.

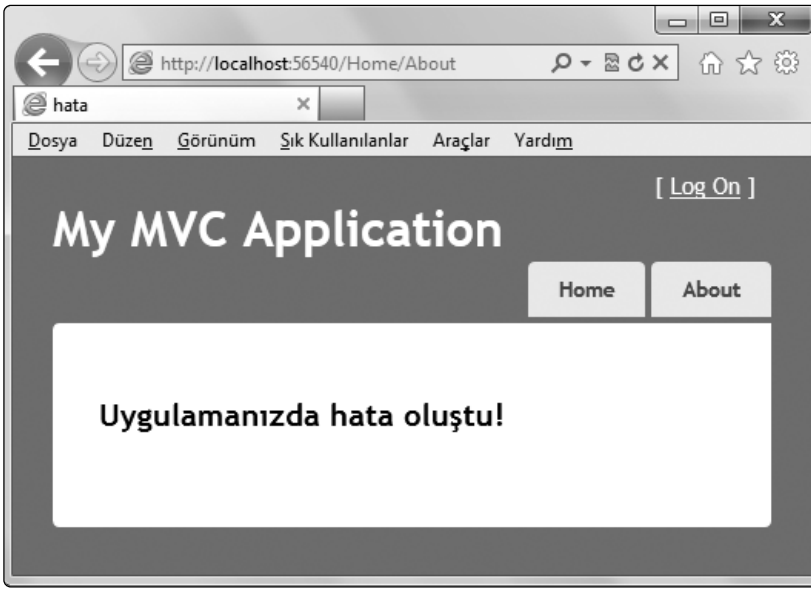
View kodu aşağıdaki gibi olacak:

```
@{
    ViewBag.Title = "hata";
}
<h2>Uygulamanızda hata oluştu!</h2>
```

HomeController.cs dosyasını açıp `About` isimli Action kodunu aşağıdaki gibi değiştirelim.

```
[HandleError(View = "Error")]
public ActionResult About()
{
    throw new DivideByZeroException();
    return View();
}
```

`HandleError` Filter kullanarak hata durumunda varsayılan `Error View` template'i yerine kendi düzenlediğimiz `Error` isimli View'ın görüntülenmesini sağlıyoruz.



`Exception` nesnesinin özellikleri kullanılarak oluşan hata ile ilgili detaylı bilgiler alınabilir.

Bu özellikler şunlardır:

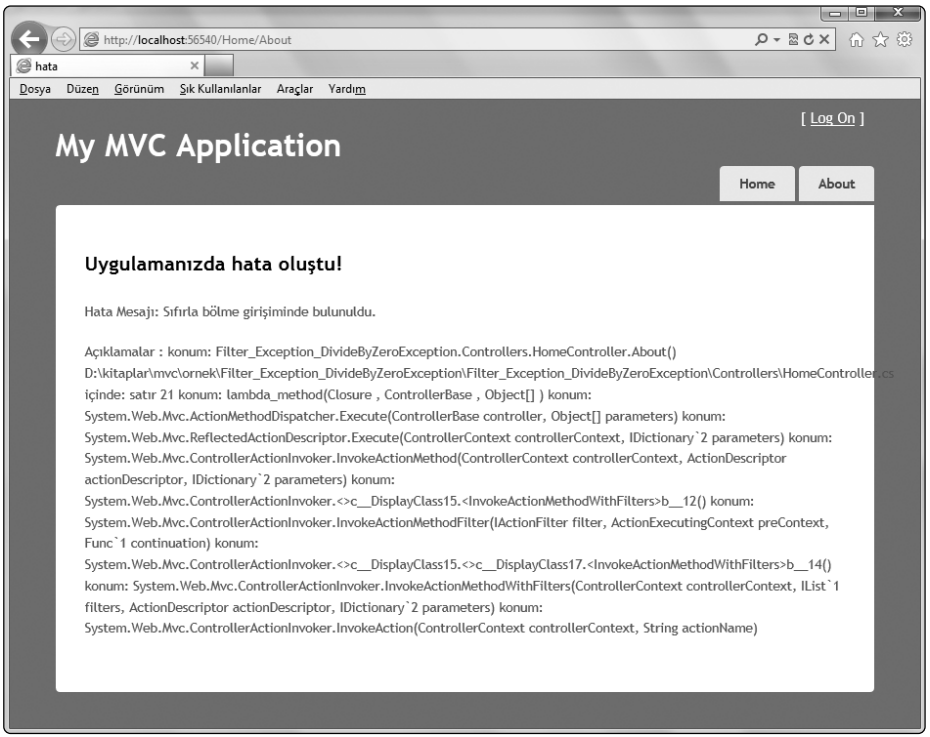
- **`Message:`** Oluşan hata ile ilgili açıklamaları içerir.
- **`Source:`** Hatanın oluştuğu nesne ya da sınıf hakkında bilgi içerir.
- **`TargetSite:`** Hatayı oluşturan metot hakkında bilgi içerir.
- **`StackTrace:`** Hataya neden olan çağrıları içeren **stack** (*yığın*) bilgisini içerir.
- **`InnerException:`** Oluşan iç hataları yakalamak amacıyla kullanılır. Eğer hiçbir iç hata oluşmazsa bu özellik null değer içerir.

MVC içerisinde `System.Web.Mvc.HandleErrorInfo` özelliğini kullanarak `Exception` nesnesinin özellikleri ile ilgili bilgi alınabilir. **Error.cshtml** isimli View kodumuzu aşağıdaki gibi düzenliyoruz.

```
@{
     ViewBag.Title = “hata”;
}
<h2>Uygulamanızda hata oluştu!</h2>
<p>
```

```
 Hata Mesajı: @Model.Exception.Message
</p>
<p>
 Açıklamalar : @Model.Exception.StackTrace
</p>
```

Uygulamamızı çalıştırdığımızda artık hata görüntüleme sayfamız içerisinde daha ayrıntılı bilgiler verildiğini görebiliriz.



`ExceptionType` özelliğini kullanarak oluşan hatanın türüne göre farklı işlemler gerçekleştirmemiz mümkündür.

Örneğin;

```
[HandleError(View = "Error")]
public ActionResult About()
{
```

```
    throw new DivideByZeroException();
    return View();
}
```

... şeklindeki kodumuzu...

```
[HandleError(ExceptionType = typeof(DivideByZeroException),
View = "Error")]
public ActionResult About()
{
    throw new DivideByZeroException();
    return View();
}
```

... şeklinde değiştirerek yalnızca `DivideByZeroException` türündeki istisna durumlarının yakalanmasını sağlıyoruz.

Action ve Result Filter

`Action` Filter, `Action` metotlarından; `Result` Filter ise `ActionResult` metotlarından önce ve sonra çalıştırılırlar. `Action` Filter olayların gerçekleşmesi esnasında uygulanırken; `Result` Filter ise olayların sonuçlarının görüntülenmesi esnasında uygulanır. `Action` Filter özellikleri `IActionFilter` arabirimini, `Result` Filter özellikleri ise `IResultFilter` arabirimini kullanır.

`ActionFilterAttribute` niteliğini kullanarak kendi özel `Action` ve `Result` Filter'larınızı oluşturabilirsiniz. `ActionFilterAttribute` niteliği ile aşağıdaki dört yöntemi kullanabilirsiniz.

- **`public override void OnActionExecuted(ActionExecutedContext filterContext):`** `Action` metot çalıştırıldıktan sonra devreye girer.
- **`public override void OnActionExecuting(ActionExecutingContext filterContext):`** `Action` metot çalıştırılmadan önce devreye girer.
- **`public override void OnResultExecuted(ResultExecutedContext filterContext):`** `Action` metot çalıştırılıp, Action result sonucu döndürüldükten sonra devreye girer.
- **`public override void OnResultExecuting(ResultExecutingContext filterContext):`** `Action` metot çalıştırılıp, Action result sonucu döndürülmeden önce devreye girer.

Örnek olarak internet sitemize Firefox ile bağlanıldığında kullanıcıyı uyaran bir uygulama yapalım. Yeni bir **Internet Application** projesi açarak **Solution Explorer** içerisinde `Browser_Filter` isminde yeni bir class oluşturalım.

Oluşturduğumuz class kodu aşağıdaki gibi olacak:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
namespace Filter_Result_Action
{
  public class Browser_Filter : ActionFilterAttribute,
       IActionFilter
  {
    public override void OnActionExecuting
       (ActionExecutingContext filterContext)
    {
        var browser = filterContext.HttpContext.
       Request.Browser.Browser;
        if (browser.ToLowerInvariant() == "firefox")
        {
            filterContext.Result = new ContentResult{Content = firefox};
        }
    }
    public string firefox
    {
      get
      {
        return @"<html>
              <head><title>Firefox Desteklenmiyor</title>
              <style>
              h1 {
                  font-size: 35px;
                  font-weight: bold;
                  line-height: 2;
                  margin: 80px 20px 20px;
                  text-align: center;
              }
```

```
                </style>
            </head>
            <body>
              <h1>Web sitesi Firefox'u desteklemiyor</h1>
            </body>
          </html>";
    }
  }
 }
}
```

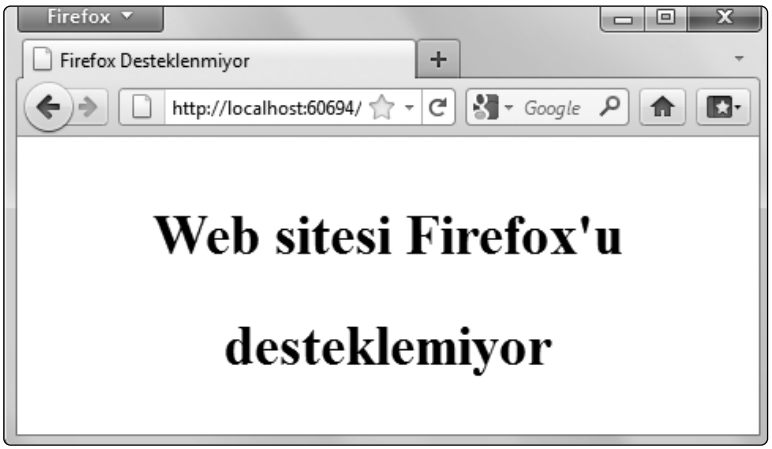
Kodumuz içerisinde `OnActionExecuting` metodu içerisinde `Webbrowser`'ın niteliği araştırılmakta ve eğer Firefox web tarayıcısı ise `firefox` isimli metod içerisinde tanımladığımız HTML içerik görüntülenecektir. `OnActionExecuting` metodunun kullanılmasından dolayı `Browser_Filter` isimli filtre, başına yazıldığı `ActionResult` metodu henüz çalışmadan önce devreye girerek, gerekli yönlendirme işlemini gerçekleştirecektir.

`HomeController` kodunu ise aşağıdaki gibi yazıyoruz:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using Filter_Result_Action;
namespace Filter_Result_Action.Controllers
{
    [Browser_Filter]
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
          ViewBag.Message = "Welcome to ASP.NET MVC!";
          return View();
        }
        public ActionResult About()
        {
          return View();
        }
    }
}
```

`[Browser_Filter]` niteliğini `class` isminin hemen üzerine yazdığımız için class içerisinde mevcut olan `Index` ve `About` isimli her iki `Action result` metoduna da uygulanacaktır.

Uygulamamızı çalıştırıp Firefox ile görüntülemeye kalktığımızda kullanıcıyı uyaran bir mesaj görüntülenecektir.



Filtrelerde Sıralama

Filtreler varsayılan olarak aşağıdaki gibi sıralanır:

1. Authorization Filter
2. Action Filter
3. Result Filter
4. Exception Filter

Aynı türe sahip filtreler ise yazıldığı sırada çalıştırılır. Ancak bu sıralamayı değiştirmek istediğimiz durumlar olabilir. Bu sıralamayı değiştirmek için `Order` özelliğini kullanırız.

Örnek olarak boş bir **MVC 3 Razor** uygulaması oluşturarak **Solution Explorer** içerisinde mesaj isimli yeni bir class oluşturalım.

Class kodu aşağıdaki gibi olacak:

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Web;
using System.Web.Mvc;
namespace Filter_Order
{
  [AttributeUsage(AttributeTargets.Method, AllowMultiple = true)]
  public class mesaj : ActionFilterAttribute, IActionFilter
  {
    public string Mesaj { get; set; }
    public void OnActionExecuting
        (ActionExecutingContext filterContext)
    {
     filterContext.HttpContext.Response.Write(string.Format("<h1>{0}</h1>",
       Mesaj));
      }
  }
}
```

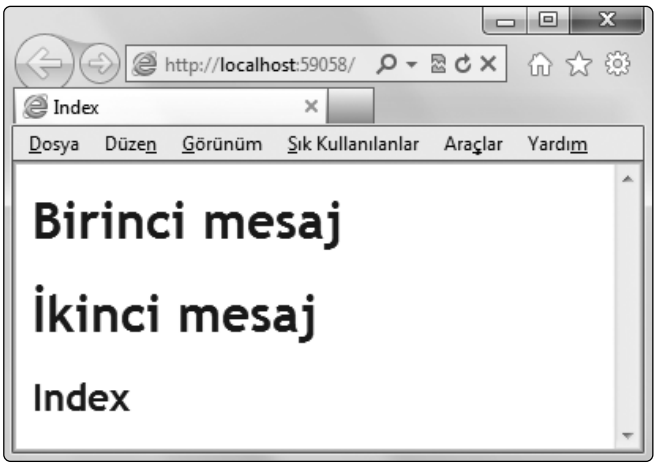
Kodumuz içerisine `OnActionExecuting` metodu içerisinde Mesaj değişkenine gelen değer, belirtilen formatta ilgili `ActionResult` metoduna geri döndürülmektedir.

`HomeController` kodu ise aşağıdaki gibi olacak:

```
using System;
using System.Web;
using System.Web.Mvc;
using Filter_Order;
namespace Filter_Order.Controllers
{
  public class HomeController : Controller
  {
    [mesaj(Mesaj = "Birinci mesaj")]
    [mesaj(Mesaj = "İkinci mesaj")]
    public ActionResult Index()
    {
        return View();
    }
  }
}
```

View kodu üzerinde ise herhangi bir değişiklik yapmıyoruz.

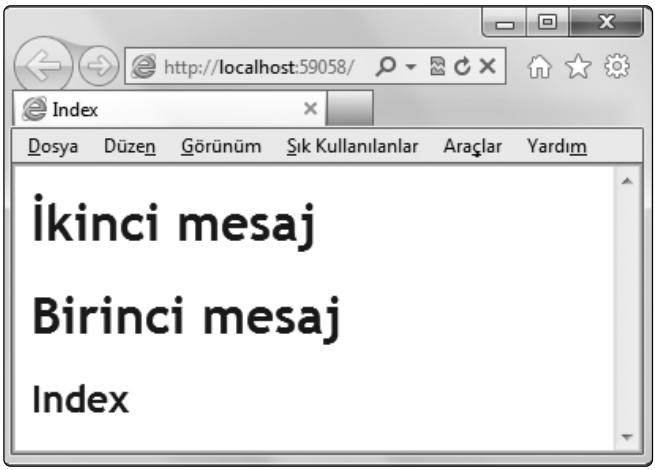
Kodumuzu çalıştırdığımızda filtreler için herhangi bir sıralama işlemi uygulamadığımız için standart olarak yazdığımız sıralama ile görüntülendi.



`HomeController` kodu üzerinde sıralama için gerekli değişiklikleri yapalım. `Index` isimli `ActionResult` kodumuz aşağıdaki gibi olacak:

```
[mesaj(Mesaj = "Birinci mesaj",Order=2)]
[mesaj(Mesaj = "İkinci mesaj",Order=1)]
public ActionResult Index()
{
    return View();
}
```

Kodumuzu bu hali ile çalıştırdığımızda ise `Order=1` ifadesi ile ilk sırada *"İkinci mesaj"* içeriğinin görüntülenmesini belirttiğimiz için sıralama değiştirilerek görüntülendi.


http://localhost:59058/
Index
Dosya Düzen Görünüm Sık Kullanılanlar Araçlar Yardım
İkinci mesaj
Birinci mesaj
Index

AJAX

9

Ajax, **Asynchronous Javascript and Xml** kelimelerinin baş harflerinden oluşan, Türkçe karşılığı ise **Eş Zamansız Javascript ve Xml** olan bir teknolojidir. Web sayfalarında **JavaScript** ve **XMLHttpRequest** ile etkileşimli uygulamalar oluşturmamızı sağlayan bir teknolojidir. Ajax ile sayfanın tamamının yüklenmesine gerek kalmadan belli kısmında değişiklikler ve güncellemeler yapılabilir.

Ajax'ın çalışma sistemi, sayfada yapılan bir istekle veya belirli zamanlarda olan tetiklemelerle arkaplanda sunucuyla etkileşime geçerek istek yollar ve aldığı veriyi tüm sayfanın yenilenmesine gerek kalmadan belirtilen yerde gösterir. Ajax ile sayfanın sadece bir kısmı yüklendiği için hem daha çok hızlı çalışır hem de daha az bant kullanır.

MVC ve Ajax

MVC'de Ajax işlemleri için Controller bölümünü kullanacağız. MVC'de View içerisinde JavaScript ile Controller içerisindeki metodu tetikleyeceğiz ve dönen veriyi sayfada göstereceğiz. MVC Framework'ünde Ajax işlemleri için geliştirilmiş özel yapılar bulunmaktadır. Bu yapılar **Html helper** ve **JavaScript** kütüphanelerinden oluşmaktadır. MVC 3'e kadar bu yapılar çok gelişmiş değildi. MVC 3.0 ile gelen **Unobtrusive JavaScript** yapısı ile JavaScript kütüphaneleri hem sadeleştirilip hem de işlevselliği arttırıldı.

Ajax'ın temel yapısında sayfa yenilenmeden server ile alışveriş olduğu için MVC'de bu işlemler için controller bölümünü kullanacağız.

MVC 3 Ajax İşlemleri

MVC3 ile gelen bu özellikte, MVC Ajax kütüphaneleri Jquery ile desteklenerek daha sade bir **Ajax Helper** kütüphanesi oluşturuldu. **Unobtrusive JavaScript** kütüphanesini kullanmak için öncelikle **web config** içerisindeki `AppSettings` altındaki `UnobtrusiveJavaScriptEnabled` değeri `true` olmalıdır.

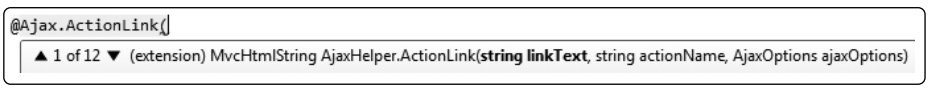
```
<appSettings>
  <add key="webpages:Version" value="1.0.0.0"/>
  <add key="ClientValidationEnabled" value="true"/>
  <add key="UnobtrusiveJavaScriptEnabled" value="true"/>
</appSettings>
```

Layout.cshtml içerisine **jquery-1.5.1.min.js** ve **jquery.unobtrusive-ajax.min.js** JavaScript kütüphanelerini eklememiz gerekmektedir.

```
<script src="@Url.Content("~/Scripts/jquery-1.5.1.min.js")" type="text/javascript"></script>
<script src="@Url.Content("~/Scripts/jquery.unobtrusive-ajax.min.js")" type="text/javascript"></script>
```

Ajax.ActionLink AjaxHelper

`Ajax.ActionLink` bir linke tıklandığında Ajax metodunu çağırmada kullanılır. Kullanımı gayet basit olduğu gibi işlevselliği de yüksektir. 12 adet `overload` metodu vardır.



En sık kullanılan metot `@Ajax.ActionLink("Link metni","Action adı",Ayarlar)` şeklinde olandır. Bu metodu kullanırken **Ajax.Action** linkinin çalıştırdığı Action ile View'in Action'ı aynı Controller içerisinde olmalıdır. **Ajax.Action** linkinin çalıştıracağı Controller ile View'in Controller'ı aynı değilse Action adı yazıldıktan sonra Controller adı parametre olarak yazılır.

`AjaxOptions` kısmını açıklayacak olursak...

- **`UpdateTargetId:`** Ajax ın çalıştırdığı Action'dan geri döndürülen değerin işlendiği span veya div tag'ının ID'si.
- **`HttpMethod:`** `Post` ve `Get` olarak iki değer alır.

View kısmına aşağıdaki kodları yazalım.

```
@Ajax.ActionLink("Güncel Saat", "GetStatus",
       new AjaxOptions { UpdateTargetId="status",
```

```
        HttpMethod = "Post" })
<div id="status">Saat Güncellenmemiş</div>
```

Controller'a ekleyeceğimiz kod:

```
public string GetStatus()
{
    return "Güncel Saat " + DateTime.Now;
}
```

Kodu çalıştırdığımızda öncelikle *"Saat Güncellenmemiş"* yazarken linke tıklandığında sayfa yenilenmeden `Güncel Saat :[Tarih-saat]` yazacaktır.





Farklı bir örnek ile konumuza devam edelim. Bir önceki örneğimizde her tıklamada güncel saati getiriyordu. Bu sefer Controller içerisinde bir List yapısı hazırlayıp, bu yapıdan seçeceğimiz rasgele kayıtları geri döndürelim. Listemizin yapısını `Ogrenci` isimli bir class'ımız oluşturacak. `Ogrenci` adında bir sınıf oluşturup `Ad`, `Soyad` ve `Sinif` adında üç özellik atayacağız.

```
public class Ogrenci
{
    public string Ad { get; set; }
    public string Soyad { get; set; }
    public string Sinif { get; set; }
    public Ogrenci(string ad, string soyad, string sinif)
    {
        Ad = ad;
        Soyad=soyad;
        Sinif=sinif;
    }
}
```

Controller kısmında örnek bir öğrenci listesi oluşturup linke tıklandığında rasgele olarak bu isimlerden biri gelecek.

View sayfamıza aşağıdaki kodlarımızı ekliyoruz:

```
@Ajax.ActionLink("Kişi çek", "KisiCek",
      new AjaxOptions() { UpdateTargetId="Kisi",
```

```
      HttpMethod = "Post"  })
<div id="Kisi">Kişi Bilgisi</div>
```

Linkimizin yazısı *"Kişi çek"*, linkimize tıklandığında Controller'da çalışacak metot `KisiCek` ve güncellenecek HTML tag'ımızın ID'si `Kisi` olacak. Controller'a aşağıdaki kodlarımızı ekliyoruz:

```
public string KisiCek()
{
       Random rnd=new Random();
       int sayi=rnd.Next(0, 3);
       List<Ogrenci> kisiler = new List<Ogrenci>()
       {
            new Ogrenci("Yavuz Selim","Çelik","1-A"),
            new Ogrenci("Esra","Özcan","3-B"),
            new Ogrenci("Hakan","Sarı","2-C")
       };
       return kisiler[sayi].Ad+"-"+kisiler[sayi].Soyad ;
}
```

Metodumuzda önce `0` ile `3` arasında bir sayı oluşturuyoruz ve `kisiler` adlı listemize üç eleman ekliyoruz. Son olarak `kisiler` listesinden rasgele oluşturduğumuz sayıyı kullanarak kaydımızı geri döndürüyoruz.



Örneğimizi biraz değiştirerek random sayımızı View sayfamızda oluşturup bunu Controller sayfamızdaki metodumuza parametre olarak gönderelim. Ama unutmayalım ki rastgele sayımız View sayfamız oluşurken oluşturulacağı için linke tıklandığında değişmeyecektir. Bu örnekte `Ajax.ActionLink` helper'ımız ile parametre yollayacağız.

View sayfamıza ekleyeceğimiz kodlar şöyledir:

```
@Ajax.ActionLink("Kişi çek", "KisiCek2",
      new { id=sayi},
      new AjaxOptions() { UpdateTargetId="Kisi2",
      HttpMethod = "Post" })
<div id="Kisi2">Kişi Bilgisi</div>
```

Controller sayfamıza ekleyeceğimiz kodlar:

```
public string KisiCek2(int id)
{
      List<Ogrenci> kisiler = new List<Ogrenci>()
      {
            new Ogrenci("Yavuz Selim","Çelik","1-A"),
            new Ogrenci("Esra","Özcan","3-B"),
            new Ogrenci("Hakan","Sarı","2-C")
     };
     return kisiler[id].Ad+"-"+kisiler[id].Soyad;
}
```

`Ajax.ActionLink` helper'ı ile parametre gönderirken Controller sayfamızdaki ilgili metodumuzun adını yazdıktan sonra, `new` ifadesinden sonra `{}` içerisinde önce metodumuz içerisinde kullanılan parametre adını yazıp = koyduktan sonra göndereceğimiz değeri yazıyoruz.

Ajax.BeginForm

ActionLink ile sadece linke tıklama ile tetikleme yapılıyordu. `BeginForm` ile bu tetikleme işlemini buton aracılığı ile yaptırabiliriz. Ayrıca form içerisindeki elemanlara girilmiş verileri çalışacak metoda gönderebiliriz. `Ajax.BeginForm` ile ActionLink'in aldıkları parametreler benzerlik göstermektedir.

`Ajax.BeginForm` metodunun kullanımı aşağıdaki gibidir:

```
@using(Ajax.BeginForm("Çalıştırılacak metot",Ajax Ayarları))
```

Bu kullanımı bir örnekle inceleyelim...

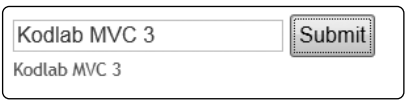
```
@using(Ajax.BeginForm("UpdateForm",
       new AjaxOptions { UpdateTargetId="textEntered"}))
{
       @Html.TextBox("textBox1", "Lütfen Yazı Girin");
       <input type="submit" value="Submit"/><br />
       <span id="textEntered">Yazı Girilmemiş</span>
}
```

Yukarıdaki kodu incelediğinizde Form içerisindeki elemanlara (örnekte sadece TextBox mevcut) girilen değerler `UpdateForm` adlı bir metoda gönderiliyor. Geri döndürülen değer ise `textEntered` adlı `div`'e yazdırılıyor.

Controller içerisindeki `UpdateForm` metodunun kodları ise aşağıdaki gibidir:

```
public string UpdateForm(string textBox1)
{
    return textBox1.ToString();
}
```





Submit butonuna tıklandığında normal MVC'deki form yapısı gibi veriler post edilecektir. Fakat farklı olarak form elemanlarımız View'ın ait olduğu Action'a değil `Ajax.BeginForm` içerisinde belirtilen metota gönderilecektir. Örneğimizde ise `TextBox`'a girilen değer aynen geri döndürülmekte ve örneğimizde `AjaxOptions` ile belirtilen div içerisine aynen yazdırılmaktadır. Ayrıca gönderim işlemi Ajax ile yapıldığından ve sayfa yenilenmediği için `TextBox`'lar da boşalmayacaktır. Fakat bu durum ASP.NET'teki `ViewState` yapısı ile karıştırılmamalıdır. `ViewState` yapısında her form elemanı için `HiddenField` oluşturulur ve formların verileri bu `HiddenField`'lar içinde saklanır. Ajax işlemi sonrasında `TextBox`'ta verinin silinmemesinde ise böyle bir durum söz konusu değildir. Burada sayfa yenilenmediği için `TextBox` boşalmamıştır.

Konumuza farklı bir örnek ile devam edelim. Örneğimiz için bir mesaj formu oluşturacağız. Form içerisinde mail adresi, mesajın başlığı ve mesajın içeriğinin bulunacağı üç alanımız olacak. Bir de formumuzu post etmek için butona ihtiyacımız olacak. Butona tıklandığında yazılarımız Ajax ile ilgili div'imizin içeriğine yazdırılacak.

Uygulamamıza farklı olarak birkaç özellik ekleyelim. Örneğin; butonumuza tıklandığında formumuz kaybolsun. Ve yeni oluşan div'imiz altında *"Yeni mesaj eklemek için tıklayın"* şeklinde bir yazımız olsun. Bu yazıya tıklandığında sayfa ilk haline gelsin.

View kısmına yazacağımız kodlar:

```
<script type="text/javascript">
    $(document).ready(function () {
       $('#Bilgi').hide();
    });
    function MesajIletimi() {
        $('#Bilgi').show('slow');
        $('#iletisimFormu').hide('slow');
```

```
        $('#Gonderilen').show('slow');
    }
    function YeniMesaj() {
        $('#Bilgi').hide('slow');
        $('#iletisimFormu').show('slow');
        $('#Gonderilen').hide('slow');
    }
</script>
@using (Ajax.BeginForm("SendForm",
       new AjaxOptions { OnSuccess = "MesajIletimi",
       UpdateTargetId="Gonderilen" }))
{
<fieldset id="iletisimFormu">
    <legend>Mesaj Gönderim Formu</legend>
    <p>
        @Html.Label("Mail", "E-mail"): @Html.TextBox("Email")
    </p>
    <p>
        @Html.Label("Baslik", "Başlık"): @Html.TextBox("Baslik")
    </p>
    <p>
        @Html.Label("Konu", "Konu"): @Html.TextArea("Mesaj")
    </p>
    <p>
        <input type="submit" value="Gönder" />
    </p>
</fieldset>
<div id="Gonderilen"></div>
<p id="Bilgi" onclick="YeniMesaj()">
    Mesajınız Gönderildi..! Yeni Mesaj için tıklayınız.
</p>
}
```

Önce üst kısımdaki JavaScript kodlarımızı geçerek `Ajax.BeginForm`'u açıklayalım.

```
@using (Ajax.BeginForm("SendForm",
       new AjaxOptions { OnSuccess = "MesajIletimi",
       UpdateTargetId="Gonderilen" }))
```

- **`SendForm:`** Çalıştırılacak metodumuzun adı.
- **`OnSuccess:`** `OnSuccess` ile Ajax metodumuz çalışıp verilerimiz sayfaya yüklendikten sonra çalışacak JavaScript fonksiyonumuzu belirtiyoruz.

- **UpdateTargetID:** `UpdateTargetID` ile dönen verilerimizin yazdırılacağı HTML elemanının ID'sini belirtiyoruz.

`AjaxBegin` formumuz içerisinde önce mesaj gönderimi için formumuzu oluşturuyoruz. Ardından Ajax işlemi sonrası dönecek verinin yazdırılacağı div'i ekliyoruz. En son olarak da mesajımız gönderildiğini belirten yazıyı ekliyoruz. Fakat burada yazıya tıklandığında JavaScript kodunu çalıştırması için `onclick` olayını ekliyoruz.



JavaScript kodlarımıza bakacak olursak, yaptığımız işlem mesaj gönderim işleminin durumuna göre HTML elemanlarının gösterilmesini veya gizlenmesini ayarlıyoruz.

Mesaj gönderim formumuz üst resimdeki gibi olacaktır. Buraya bilgilerimizi girdikten sonra **Gönder** butonuna bastığımızda Controller içerisinde bulunan `SendForm` metodumuz çalışacak.

Controller içerisindeki `SendForm` metodumuz aşağıdaki gibidir:

```
public string SendForm(string Email, string Baslik, string Mesaj)
{
  return "Gonderen :"+Email+"<br> Başlık :"+Baslik+"<br>Mesaj :"+Mesaj;
}
```

Burada yapılan işlem gelen verileri geri döndürmektir. Son olarak View sayfamızda formumuz kaybolacak ve aşağıdaki şekilde bir görünüm olacaktır.



"Mesajınız gönderildi! Yeni mesaj için tıklayınız." yazısına tıkladığımızda tekrar yazılar kaybolacak ve formumuz gösterilecektir. Fakat formumuzun içerisindeki yazılar silinmeden gelecektir. Bunun sebebi; yaptığımız işlemlerde bu form ele-

manlarının yeniden yüklenmesi söz konusu olmamasıdır. Sadece form elemanlarının gösterilebilirliğini değiştirdik.

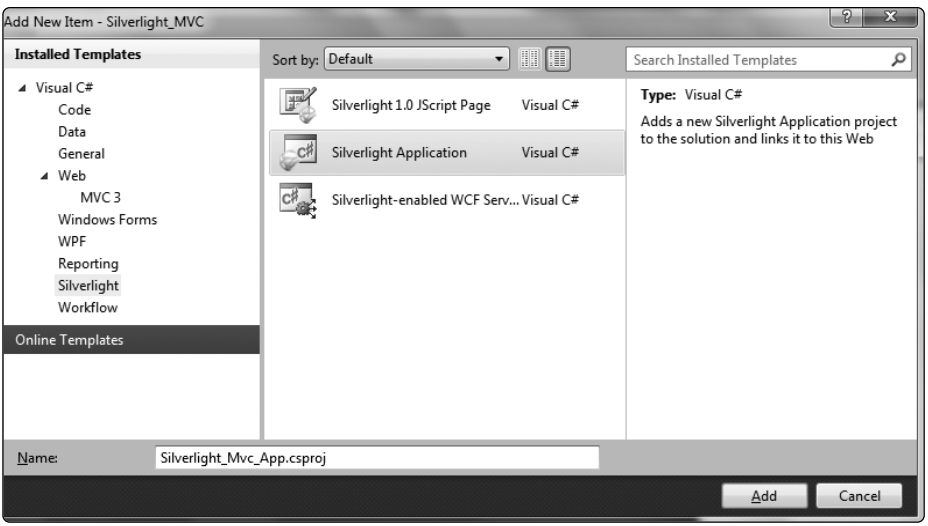


MVC VE SILVERLIGHT

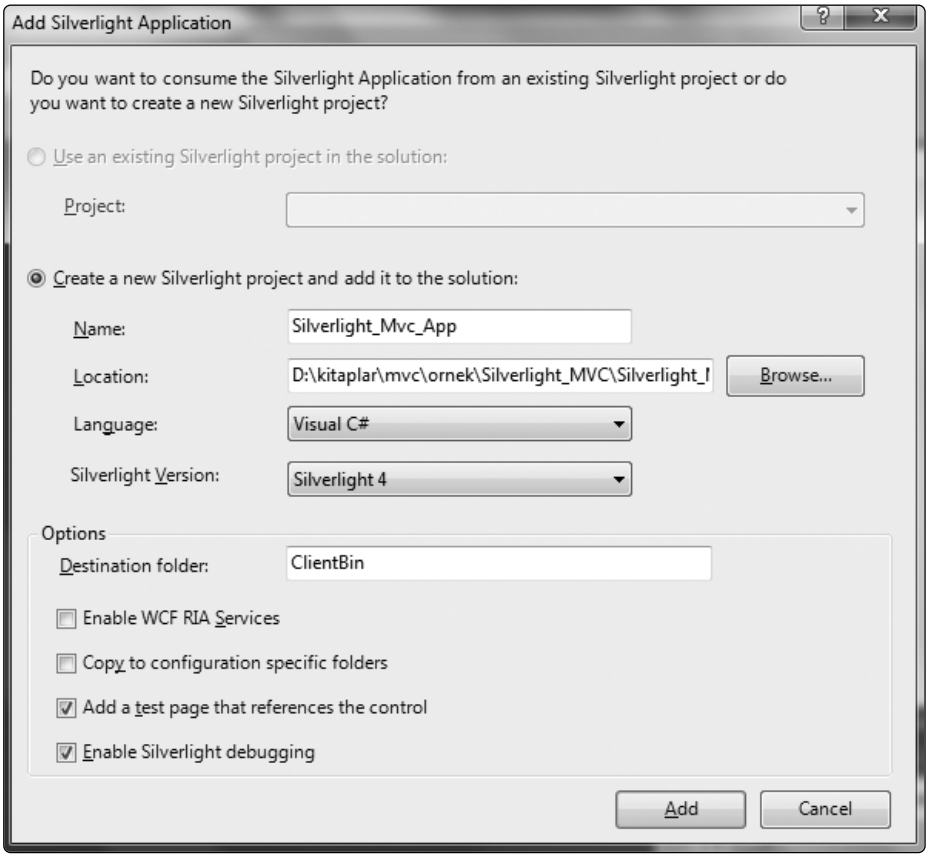
10

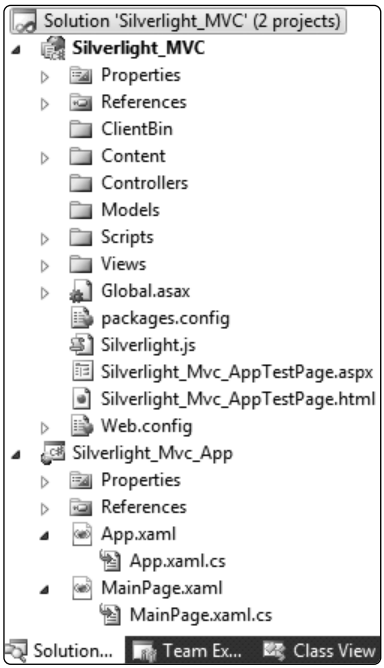
MVC içerisinde Silverlight teknolojisini kullanarak Silverlight tabanlı web uygulamaları geliştirmeniz mümkündür. Bunun için sisteminizde **SilverlightTools for Visual Studio** yüklü olmalı. Gerekli uygulamayı *http://www.silverlight.net/* adresinden indirebilirsiniz.

Örneğimiz için boş bir **MVC 3 Razor** projesi oluşturuyoruz. Daha sonra **Solution Explorer** içerisinde proje ismine Mouse'un sağ tuşu ile tıklayarak ya da **Project** menüsünden **Add/New Item** seçeneği ile gelen pencerede **Silverlight Application** seçeneğini seçerek **Add** butonuna tıklıyoruz.



Karşımıza gelen pencerede Silverlight uygulaması için gerekli ayarları aşağıdaki gibi yaparak **Add** butonuna tıklıyoruz.





Bu işlemin ardından gerekli Silverlight uygulama dosyaları projemize eklenecek ve proje dosyalarının son hali yandaki gibi olacaktır.

Silverlight içerisinde tasarım dosyalarının uzantıları **xaml** iken, kod dosyalarının uzantısı ise C# kullandığımızdan dolayı **cs**'dir.

Bu dosyalardan;

• **App.xaml ve App.xaml.cs:** Silverlight uygulamamızı düzenlemek için kullanılan dosyalardır.

• **Page.xaml ve Page.xaml.cs:** Kullanıcı arabiriminin tasarım ve kod dosyalarıdır.

Öncelikle ana sayfamızı tasarlayarak işe başlayalım. Ana sayfa içerisinde **Stack** panel kullanarak kontrollerimizi yerleştireceğiz.

Stack panel; bir ya da birden fazla kontrolü içerisinde barındırarak, içerisindeki nesneleri belirtilen bir düzen içerisinde görüntüler

Ana sayfamızı oluşturacak olan **MainPage.xaml** dosyasının kodu aşağıdaki gibi olacak:

```
<UserControl x:Class="Silverlight_Mvc_App.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d"
  d:DesignHeight="1024" d:DesignWidth="768">
  <Grid x:Name="LayoutRoot" Background="White">
    <Grid.RowDefinitions>
      <RowDefinition Height="110"></RowDefinition>
      <RowDefinition Height="*"></RowDefinition>
    </Grid.RowDefinitions>
    <StackPanel Grid.Row="0" Margin="5">
```

```
    <Image Height="104" Width="1024" Stretch="Fill"
       HorizontalAlignment="Center" VerticalAlignment="Center"
      Source="/Silverlight_Mvc_App;component/Images/banner.png"/>
  </StackPanel>
  <Grid x:Name="colColumns" Grid.Row="1">
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="200"></ColumnDefinition>
      <ColumnDefinition Width="*"></ColumnDefinition>
      <ColumnDefinition Width="200"></ColumnDefinition>
    </Grid.ColumnDefinitions>
     <StackPanel Background="Azure" Grid.Column="0" Margin="5">
       <Button Content="Ana Sayfa"
            Name="btnAnaSayfa" Margin="3" >
       </Button>
       <Button Content="Makalelerim"
            Name="btnMakale" Margin="3">
       </Button>
       <Button Content="Programlar"
            Name="btnProgramlar" Margin="3">
       </Button>
       <Button Content="Ziyaretçi Defteri"
            Name="btnZiyaretciDefteri" Margin="3" >
       </Button>
       <Button Content="Hakkımda"
            Name="btnHakkımda" Margin="3" >
       </Button>
    </StackPanel>
    <StackPanel x:Name="myContent" Grid.Column="1" Margin="5">
       <StackPanel.Background>
          <LinearGradientBrush EndPoint="0.40,3" StartPoint="0.9,3">
            <GradientStop Color="#9999FF"/>
            <GradientStop Color="#0066CC" Offset="1.5"/>
          </LinearGradientBrush>
       </StackPanel.Background>
       <TextBlock Margin="15"  FontSize="25"
            FontWeight="300" Foreground="White"
            Text="Sayfama Hoş Geldiniz!"
            HorizontalAlignment="Center" >
       </TextBlock>
    </StackPanel>
```

```
        <StackPanel Background="Azure" Grid.Column="2" Margin="5">
        </StackPanel>
    </Grid>
  </Grid>
</UserControl>
```

Kodumuz içerisinde...

```
d:DesignHeight="1024" d:DesignWidth="768"
```

... satırı ile tasarım alanımızın genişlik ve yükseklik değerlerini belirledik.

Tasarım alanına eklediğimiz nesneler, kodumuzda da görüldüğü gibi `<Grid>` `</Grid>` arasına yazılırlar.

Grid'i...

```
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="200"></ColumnDefinition>
    <ColumnDefinition Width="*"></ColumnDefinition>
    <ColumnDefinition Width="200"></ColumnDefinition>
</Grid.ColumnDefinitions>
```

... satırlarında `3` sütuna ayırıyoruz. Sütunlardan soldaki `200`, sağdaki `200`, ortadaki ise toplam genişlikten bu genişlik değerleri çıkarıldıktan sonra kalan genişlik değerini alıyor.

Stack panel zemin rengini...

```
<StackPanel.Background>
  <LinearGradientBrush EndPoint="0.40,3" StartPoint="0.9,3">
      <GradientStop Color="#9999FF"/>
      <GradientStop Color="#0066CC" Offset="1.5"/>
   </LinearGradientBrush>
</StackPanel.Background>
```

.... satırlarında belirliyoruz. Zemin renginde gradient'ler kullanıyoruz. Burada birbiri üzerine binen iki adet gradient belirliyoruz. Bu sayede formumuz tek renk olmak yerine bir köşesinden itibaren farklı bir renkle tonlama yapılıyor.

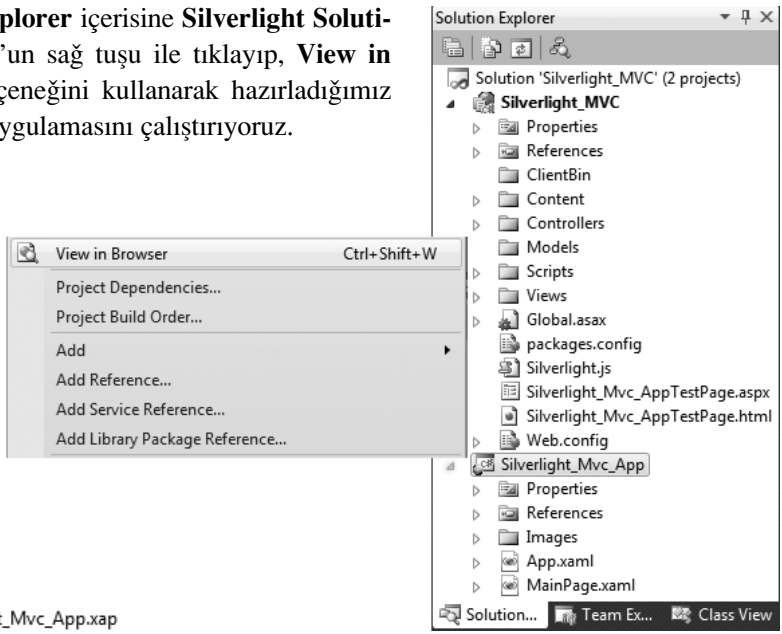
Yazılarımızı `<TextBlock></TextBlock>` arasına yazıyoruz.

Buton ise...

```
<Button Content="Makalelerim"
        Name="btnMakale" Margin="3">
</Button>
```

... örneğinde görüldüğü gibi tanımlanır. `Name` özelliği ile butonumuzun ismini belirlerken, `Content` özelliği ile butonumuzun üzerinde görünecek yazıyı belirleriz.

Solution Explorer içerisine **Silverlight Solution**'a Mouse'un sağ tuşu ile tıklayıp, **View in Browser** seçeneğini kullanarak hazırladığımız Silverlight uygulamasını çalıştırıyoruz.



Bu şekilde **ClientBin** klasörü içerisine **xap** uzantılı Silverlight uygulama dosyasının oluşturulmasını sağlıyoruz. Bu uygulama dosyasını kullanarak MVC içerisinden bu uygulama dosyasının çalıştırılmasını sağlayacağız.

MVC içerisinde **HomeController**'ı oluşturup ardından **Index** isimli **View**'i oluşturuyoruz. Index View içerisine Silverlight uygulamamız ile bağlantı kurmak için aşağıdaki kod satırlarını ekliyoruz.

```
<object data="data:application/x-silverlight-2,"
  type="application/x-silverlight-2" width="1024" height="768">
    <param name="source" value="ClientBin/Silverlight_Mvc_App.xap"/>
    <param name="background" value="blue" />
```

```
    <param name="minRuntimeVersion" value="4.0.60129.0" />
    <param name="autoUpgrade" value="true" />
    <a href="http://www.microsoft.com/getsilverlight/get-started/install/"
       style="text-decoration:none">
      <img src="http://go.microsoft.com/fwlink/?LinkId=161376"
           alt="Get Microsoft Silverlight" style="border-style:none"/>
    </a>
</object>
```

`</object></object>` tag'ini kullanarak Index View içerisinde Silverlight uygulamasının render edilerek görüntülenmesini sağlıyoruz.

- `source` parametresi ile Silverlight uygulamasının yolunu belirtiyoruz.
- `background` parametresi ile Silverlight uygulamasının içerisinde görüntüleneceği ortamın zemin rengini belirliyoruz. Bu zemin rengi Silverlight uygulaması çalıştırılana kadar görüntülenecektir.
- `minRuntimeVersion` parametresi ile Silverlight uygulamasının çalıştırılması için gerekli olan en düşük Silverlight versiyonu belirtilir.
- `autoUpgrade` parametresi ile Silverlight programının yeni versiyonu çıktığında otomatik olarak güncelleme yapılması sağlanır.

Ayrıca Silverlight programının bilgisayarda yüklü olmadığı durumlarda yüklenebilmesi için link veriyoruz.

Web sayfalarımızı oluştururken MVC tabanlı (*.cshtml*) ya da Silverlight tabanlı (*.xaml*) olarak hazırlayabiliriz. Sayfalar arası yönlendirme kodumuz da buna göre değişiklik gösterecektir.

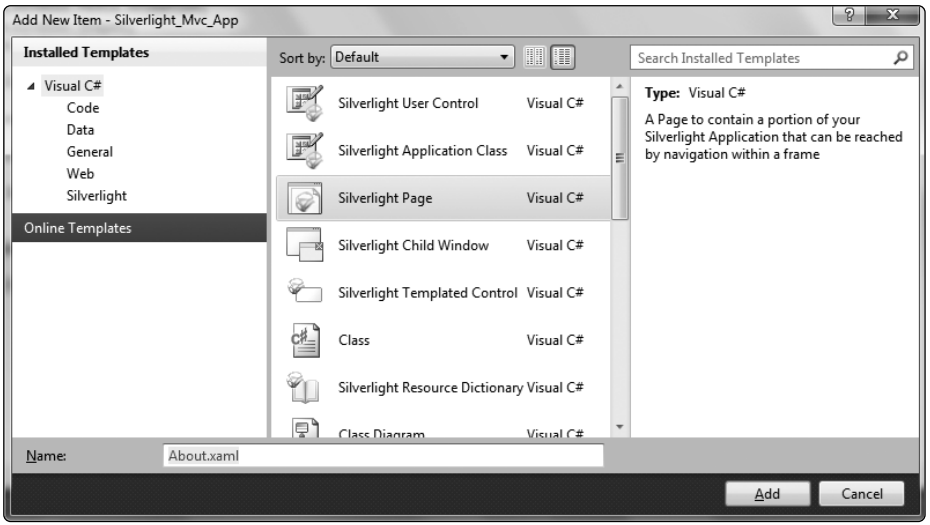
Öncelikle MVC tabanlı (*.cshtml*) olarak bir sayfa oluşturup bu sayfaya yönlendirme yapalım. Örnek olarak **About.cshtml** isimli bir View sayfası oluşturalım. **MainPage.xaml** sayfası içerisinde Hakkımda isimli butona çift tıklayarak **MainPage.xaml.cs** isimli kod sayfası içerisindeki **Hakkımda** butonunun `Click` olayını açalım.

`Click` olayının kodu aşağıdaki gibi olacak:

```
private void btnHakkımda_Click(object sender, RoutedEventArgs e)
{
    HtmlPage.Window.Navigate(new Uri("http://localhost:59635/Home/About"));
}
```

NOT Uygulamamızı local host'ta çalıştırdığımız için local host adresini yazdık. Siz de kendi uygulamanız için uygun değişikliği yaparsınız.

Şimdi de Silverlight tabanlı (*.xaml*) olarak bir sayfa oluşturup, bu sayfaya yönlendirme yapalım. **Solution Explorer** içerisinde **Silverlight Solution**'a Mouse'un sağ tuşu ile tıklayarak **Add New Item** seçeneği ile **About.xaml** isimli bir Silverlight sayfası ekleyelim.



Projemizin ana sayfasındaki **Videolar** butonuna tıklandığında **videolar.xaml** sayfasının görüntülenmesi için **App.xaml.cs** dosyası içerisindeki...

```
private void Application_Startup(object sender, StartupEventArgs e)
{
    this.RootVisual = new MainPage();
}
```

... kod satırlarını...

```
private static Grid root;
private void Application_Startup(object sender, StartupEventArgs e)
{
    root = new Grid();
    root.Children.Add(new MainPage());
```

```
    this.RootVisual = root;
}
public static void Navigate(UserControl yeniSayfa)
{
    UserControl eskiSayfa= root.Children[0] as UserControl;
    root.Children.Add(yeniSayfa);
    root.Children.Remove(eskiSayfa);
}
```

… şeklinde değiştiriyoruz. **MainPage.xaml** sayfası içerisinde Hakkımda isimli butona çift tıklayarak **MainPage.xaml.cs** isimli kod sayfası içerisindeki **Hakkımda** butonunun `Click` olayını açalım. `Click` olayının kodu aşağıdaki gibi olacak.

```
private void btnHakkımda_Click(object sender, RoutedEventArgs e)
{
    App.Navigate(new About());
}
```

Yukarıdaki kod satırlarında bir **User Control** oluşturarak, root elementi vasıtası ile oluşturmuş olduğumuz **About.xaml** dosyasını görüntüledik.



Silverlight hakkında detaylı bilgi edinmek için Kodlab'tan çıkmış olan Silverlight kitabını kullanabilirsiniz.

JQUERY

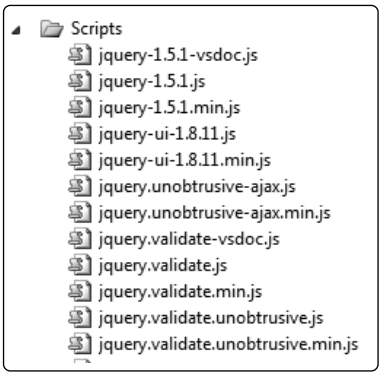
11

JQUERY'Yİ TANIMAK

2006 yılında John Resig tarafından duyurulan **JQuery**, 15 kişilik bir ekip tarafından geliştirilmeye başlanan ve hala geliştirilmesine devam edilen bir JavaScript kütüphanesidir. Web yazılımında server-side tarafı ne kadar gelişse de sayfaların hızlı yüklenme sorunu her zaman var olmuştur. Ajax ile bu duruma büyük oranda çare bulunsa da iş yükünü ne kadar çok Client-Side tarafına aktarırsak, bizim için o kadar avantajlı olacaktır. Bu durumda ihtimal olarak JavaScript tarafının geliştirilmesi kalmaktadır. Fakat JavaAcript yazması zahmetli olduğu kadar browserlar arası farklılık gösterebilmektedir.

JQuery ile hızlı anlaşılır ve browserlar arası çalışma farklılığı göstermeyen bir framework hayatımıza girmiş oldu. JQuery'nin mantığında daha az kod, daha çok iş yatmaktadır. Kısaca; JQuery ile bizim yazdığımız kod azalmakta, işin büyük çoğunluğu framework içerisindeki fonksiyonlarla çözülmektedir.

JQuery, Visual Studio ile oluşturduğumuz projelerde otomatik olarak projeye eklenmektedir. Fakat projemizde aynı isimde iki tane kütüphane bulunmaktadır. Biri normal hali, diğeri minimum halidir. Bunlardan ikisi de aynı içeriğe sahip olmakla birlikte minimum halinde boşluklar kaldırıldığı için daha az yer kaplamaktadır. Bundan dolayı projelerimizde minimum halini kullanmamız daha avantajlıdır. Fakat JQuery kütüphanesinin kodlarını incelemek isterseniz, normal olan JavaScript dosyasından rahatça inceleyebilirsiniz.



`Layout.cshtml` altında JQuery kütüphanemiz projemize eklenmiş durumdadır. Diğer kütüphaneleri kullanmak için de aşağıdaki satırı kodumuzun içerisine eklememiz gerekmektedir.

```
<script src="@Url.Content("~/Scripts/JQuery-1.5.1.min.js")"
type="text/javascript"></script>
```



JQuery ile ilgili detaylı ve sürekli bilgiyi Kodlab'tan daha önce çıkmış olan JQUERY kitabından bulabilirsiniz.

Temel Seçiciler

MVC uygulamalarında JQuery kullanımı ASP.NET web form uygulamalarına göre çok daha kolaydır. Bunun en büyük sebebi ASP.NET web form'da ASP.NET elementleri kullanılmasına karşın, MVC'de HTML elementleri kullanılmasıdır.

JQuery işlemlerinde sayfa elemanları üzerinde sıklıkla çalışma ihtiyacı hissedeceğimiz sayfa elemanlarına ulaşmak için beş yöntem bulunmaktadır.

Seçicisi

Sayfa elemanlarını `ID` veya `Name` özelliği ile seçmemizi sağlar.

```
<script type="text/javascript">
  $(document).ready(function () {
    $("#ornek1").html("Kodlab yayınları MVC 3");
    $("#txtOrnek").val("deneme");
  });
</script>
```

```
<h2>
  TemelSeciciler</h2>
  <div id="ornek1"></div>
  @Html.TextBox("txtOrnek")
```

Örneğimizde...

```
$(document).ready(function (){};
```

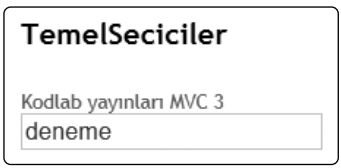
... ile kodumuzun sayfa yüklenirken çalışmasını sağlıyoruz.

```
$("#ornek1").html("Kodlab yayınları MVC 3");
```

... ile `ornek1` ID'li div'imizin içeriğini değiştiriyoruz.

```
$("#txtOrnek").val("deneme");
```

... ile `txtOrnek` name'ine sahip `TextBox`'ımıza değer atıyoruz.

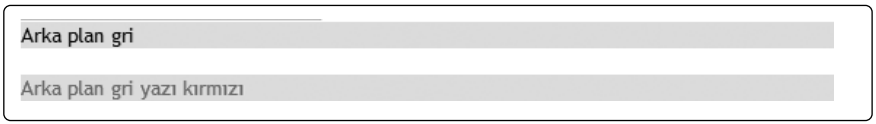


. Class Seçicisi

Sayfa elemanlarını class özelliği ile seçmemizi sağlar.

```
<script type="text/javascript">
  $(document).ready(function () {
    $(".gri").css("background-color", "gray");
    $(".kirmizi").css("color","red")
  });
</script>
  <div class="gri">Arka plan gri</div><br />
  <div class="gri kirmizi">Arka plan gri yazı kırmızı</div>
```

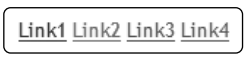
Örneğimizde JQuery ile seçtiğimiz class'lara ait elemanların CSS özelliğini değiştiriyoruz. Css class'ı `gri` olan elemanların arkaplanı gri, `kirmizi` olanların ise yazı rengi kırmızı oluyor. Örneğimizde üstteki div'e sadece gri CSS'i atanmışken, altta bulunan div'e hem `gri` hem `kirmizi` CSS'leri atanmıştır. Kodumuzu çalıştırdığımızda üstteki div'imizin arkaplanı gri, alttaki div'imizin ise arkaplanı gri, yazı rengi kırmızı olacaktır.



[] Öz Nitelik Seçicisi

Html elementlerinin sahip oldukları `Attribute`'lara göre seçim yapar.

```
    $("a[href='http://www.kodlab.com/']").css("color", "green");
    $("a[href^='ftp://']").css("color", "red");
    $("a[href$='net']").css("color", "blue");
    $("a[href*='JQuery']").css("color", "gray");
  <a href="http://tahsin.sevinch.net">Link1</a>
  <a href="ftp://deneme.com">Link2</a>
  <a href="http://JQuery.com">Link3</a>
  <a href="http://www.kodlab.com/">Link4</a>
```



Örnekleri incelediğimizde `a` tagına ait `href` öz niteliğine farklı şekillerde ulaşmayı göreceksiniz. İlk kodumuzda `href` attribute'u, yazıldıktan sonra sadece eşittir konulmuştur. Böylece sadece içeriği yazılan içeriğe eşit olan seçilir. İkinci örnekte eşittir işaretinden önce `^` ifadesi eklenmiştir. Böylece belirtilen içerikle başlayanlar seçilir. Üçüncü örnekte `$` işareti koyulmuştur bunun manası belirtilen içerik ile bitenleri seç demektir. Son örnekte ise `*` ifadesi kullanılmıştır. Böylece içerisinde belirlenen ifadenin geçtiği elemanlar seçilmiştir.

Element Seçicisi

Yazılan tag isminde olan tüm elementleri seçer. Yani tüm `div`, `p` veya `TextBox`'ları seçmemizi sağlar.

```
<script type="text/javascript">
  $(document).ready(function () {
    $("#ornek1").html("Kodlab yayınları MVC 3");
    $("#txtOrnek").val("deneme");
    $(".sari").css("background-color", "yellow");
    $(".kirmizi").css("color", "red");
  $("input[type='submit']").click(function () {
    $("div").hide("slow");
  });
  });
</script>
```

```
<h2>
  TemelSeciciler</h2>
  <div id="ornek1"></div>
  @Html.TextBox("txtOrnek")
  <br />
  <div class="sari">Arka plan sarı</div><br />
  <div class="sari kirmizi">Arka plan sarı yazı kırmızı</div>
<input type="submit" value="Kaydol" />
```

Örneğimizde dikkat etmemiz gereken kısım şudur:

```
$("input[type='submit']").click(function () {
  $("div").hide("slow");
});
```

Burada öncelikle tüm `input` tag'ını taşıyan elemanlar içerisinde type'ı `submit` olan elemanlar seçiliyor ve bu elemanların `click` event'i yazılıyor. Yapılacak işlem ise tüm `div` tag'ını taşıyan elemanların gizlenmesi olarak belirtiliyor.

: Form Elemanı Seçicisi

Form elemanlarını seçmek için kullanırız.

- **`text:`** metin alanları,
- **`file:`** dosya alanları,
- **`password:`** şifre alanları,
- **`radio:`** radio butonları,
- **`checkbox:`** onay kutularını,
- **`submit:`** gönderme işlemi yapan butonlarını seçerek, kısaca `input[type='']` yazmak yerine **`:`** koyup form elemanımızın `type` değerini yazmamız yeterli olacaktır.

Bir önceki örnekte...

```
$("input[type='submit']").click(function () {
  $("div").hide("slow");
});
```

... yerine...

```
$(":submit").click(function () {
  $("div").hide("slow");
})
```

... yazılması aynı işi görecektir.

Animasyon Efektleri

JQuery ile uygulamamızda elemanların gösterimi, gizlenmesi, saydamlığının değişmesi gibi işlemleri yapabiliriz.

Gösterip Gizleme

Gösterip gizleme işlemleri için JQuery'nin `show` ve `hide` metotlarını kullanıyoruz.

```
<script type="text/javascript">
  $(document).ready(function () {
    $("span").css("display", "block").css("color","red");
    $("#sp1").click(function () {
      $("#mvc").show();
      $("#wf").hide();
    });
    $("#sp2").click(function () {
      $("#wf").show("slow");
      $("#mvc").hide(5000);
    });
  });
</script>
<h2>Animasyon</h2>
<span id="sp1">MVC 3</span>
<div id="mvc">Asp.net MVC framework hakkında bilgilendirme</div>
<span id="sp2">Web Form</span>
<div id="wf">Asp.net web form hakkında bilgilendirme</div>
```

Yukarıdaki örneğimizi incelediğimizde öncelikle `span` tag'larına CSS ekliyoruz. Renklerini kırmızı yapıyoruz. Ardından iki `span`'ımızın `click` event'lerine kod yazıyoruz. `sp1` id'li span'a tıklandığında id'si `mvc` olan div gösterilip, id'si `wf` olan div kaybolacak. `sp2` id'li span'a tıklandığında id'si `wf` olan div gösterilip, `mvc` olan div kaybolacak. Fakat dikkat etmelisiniz ki ilk seferde yazdığımız `show` ve `hide` metotlarında parantez içleri boş iken ikinci yazdıklarımız da dolu.

`show` ve `hide` metotlarını kullanırken efektli bir şekilde gösterim veya gizleme için efekt hızını belirleyebiliriz. Bunu ister `slow`, `normal`, `fast` olarak yazı ile ifade edebiliriz ya da milisaniye cinsinden belirtebiliriz (`5000 ms=5 saniye`).

Görünürlüğünü Değiştirme

JQuery ile HTML elemanlarının görünürlüğünü değiştirebilir, elemanları görünür veya görünmez yapabiliriz. Bu işlemleri `fadeIn` ve `fadeOut` metotları aracılığı ile yapıyoruz.

fadeIn metodunu opacity değerini arttırma işlemi yaparken, fadeOut metodunu ise opacitiy değerini azaltır.

```
<script type="text/javascript">
  $(document).ready(function () {
    $("span").css("display", "block").css("color","red");
    $("#sp3").click(function () {
      $("#jq").fadeIn();
      $("#js").fadeOut();
    });
    $("#sp4").click(function () {
      $("#jq").fadeOut("slow");
      $("#js").fadeIn(5000);
    });
  });
</script>
<h2>Animasyon</h2>
<span id="sp3">JQuery</span>
<div id="jq">JQuery hakkında bilgilendirme</div>
<span id="sp4">JavaScript</span>
<div id="js">Javascript hakkında bilgilendirme</div>
```

Örneğimizi çalıştırdığımızda div'lerin opacity değerlerinin yavaşça düşerek kaybolduğunu, diğerinin de opacity değerinin yavaşça yükselerek görünmeye başladığını göreceğiz.

SlideUp/SlideDown

Div'lerimizin açılış ve kapanışının dikey slayt efekti şeklinde yapılmasını sağlar. Kodun efekti show ve hide metodunun değer alan metotlarına benzemektedir.

```
<script type="text/javascript">
  $(document).ready(function () {
    $("span").css("display", "block").css("color","red");
    $("#sp5").click(function () {
      $("#css3").slideUp();
      $("#html5").slideDown();
    });
    $("#sp6").click(function () {
      $("#css3").slideDown("slow");
      $("#html5").slideUp(5000);
    });
```

```
    });
</script>
<h2>Animasyon</h2>
<span id="sp5">CSS 3</span>
<div id="css3">CSS hakkında bilgilendirme</div>
<span id="sp6">HTML 5</span>
<div id="html5">HTML 5 hakkında bilgilendirme</div>
```

Örneğimizi çalıştırdığımızda gizleme ve gösterme işleminin efektli bir şekilde olduğunu göreceksiniz.

JQUERY UI

JQuery UI, JQuery kütüphanesi kullanılarak geliştirilen kontrolleri barındırır. Ayrıca kontroller için çeşitli temalar da geliştirilmiştir. Konu hakkında detaylı bilgi için *http://JQueryui.com/* sayfasını ziyaret edebilirsiniz. Konudaki anlatımda JQuery UI kütüphanesi'nin tanıtımı ve kullanımı ile ilgili genel örnekler verilecektir.

JQueryUI kütüphanesinde bulunan eklentiler şunlardır:

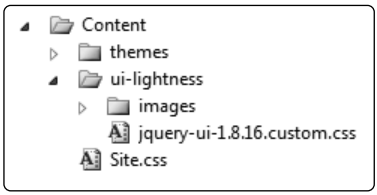
- Accordion menu
- Otomatik tamamlamalı TextBox
- Buton
- Datepicker
- Dialogbox
- Progressbar
- Slider
- Tab'lı yapı

Bu eklentiler için geliştirilmiş temaları *http://JQueryui.com/themeroller/* sayfasından indirebilirsiniz.

Öncelikle `Layout.cshtml` sayfamıza JQuery.UI ile ilgili javascript kütüphanemizi ekliyoruz.

```
<script src="@Url.Content("~/Scripts/JQuery-ui-1.8.11.min.js")"
type="text/javascript"></script>
```

Tema sayfamızdan bir tema indirip temanın **css** klasöründe bulunan dosyaları **Content** klasörü içerisine taşıyoruz.



Son olarak CSS'imizin yolunu `Layout.cshtml` sayfamızdan projemize ekliyoruz.

```
<link href="@Url.Content(
      "~/Content/ui-lightness/JQuery-ui-1.8.16.custom.css")"
      rel="stylesheet" type="text/css" />
```

Artık **JQueryUI** eklentilerini kullanabiliriz.

Öncelikle **Datepicker**'ı kullanalım. Bunun için View içerisine **TextBox** ekliyoruz.

```
<input type="text" id="datepicker" />
```

Ardından JavaScript kodumuzu yazıyoruz.

```
<script type="text/javascript">
  $(function () {
    $("#datepicker").datepicker();
  });
</script>
```

`TextBox`'ımızı seçip `datepicker` metodunu yazıyoruz ve sonuç şu şekilde oluyor:



Örnekte tarih formatı **ay-gün-yıl** şeklindedir. Bunu değiştirmek için kodumuzda aşağıdaki gibi bir değişiklik yapmamız gerekiyor.

```
$("#datepicker").datepicker({ dateFormat: 'dd-mm-yy' });
```



Diğer UI eklentilerini kullanmak da `Datepicker` kullanmak kadar kolay. `Accordion Menu` örneğimizi inceleyelim.



Bu tip bir `Accordion Menu` oluşturmak için öncelikle html kısmını yazıyoruz:

```
<div id="dvaccordion" style="width:180px;">
<h3><a href="#">Tab-1</a></h3>
<div><p> içerik-1</p></div>
<h3><a href="#">Tab-2</a></h3>
<div><p>içerik-2</p></div>
<h3><a href="#">Tab-3</a></h3>
<div><p>içerik-3</p></div>
</div>
```

Dikkat etmemiz gereken şey; <h3> tag'ları Accordion başlıklarımız oluyor. <h3> tag'ları altındaki div'ler üstteki tag'ın alt alanı olarak oluşuyor. JavaScript kodumuz ise aşağıdaki gibi olacaktır.

```
$("#dvaccordion").accordion();
```

Gördüğünüz gibi kapsayıcı div'imizin sonuna `accordion()` metodunu ekliyoruz.

JQueryUI'da bulunan `Dialog Box`'ı kullanmak için önce `Dialog Box`'ımızın div'ini oluşturalım.

```
<div id="dvdialog" title="Mesaj Kutusu">
Merhaba, bu dialog box JQuery UI script'inin bir eklentisidir.
</div>
```

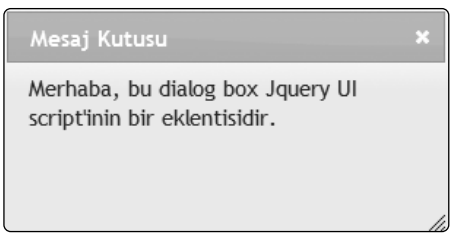
`Dialog Box`'ımızı bir butona tıkladığımızda açalım. Bunun için öncelikle butonumuzu oluşturalım.

```
<button id="btnAc">Dialog Box'ı Aç</button>
```

Son olarak JQuery kodlarımızı yazıyoruz.

```
$("#dvdialog").dialog({
  autoOpen: false,
  show: "blind",
  hide: "explode"
});
$("#btnAc").click(function () {
  $("#dvdialog").dialog("open");
  return false;
});
```

Öncelikle div'imizin `Dialog Box` olduğunu belirtip, sayfa yüklenirken açılma durumunu `false` yapıyoruz. Ve butonun `Click` event'ına `Dialog Box`'ımızı açacak kodlarımızı yazıyoruz.



JQueryUI ile son olarak tab'lı yapıya örnek verelim. Tab'lı yapı olarak öncelikle html yapımızı oluşturalım.

```
<div id="tabs">
       <ul>
              <li><a href="#tab1">Tab 1</a></li>
              <li><a href="#tab2">Tab 2</a></li>
              <li><a href="#tab3">Tab 3</a></li>
       </ul>
       <div id="tab1">
              <p>tab içeriği-1</p>
       </div>
       <div id="tab2">
              <p>tab içeriği-2</p>
       </div>
       <div id="tab3">
              <p>tab içeriği-3</p>
       </div>
</div>
```

Öncelikle tab yapımızı `ul>li` list şeklinde oluşturuyoruz ve `li` tag'larımıza link veriyoruz. Fakat linklerimizin çapa link olarak adlandırılan sayfadaki div'lere yönelmesini sağlıyoruz. `Ul>li` listemizi oluşturduktan sonra ta'blara tıkladığımızda açılacak alanları yani div'leri oluşturuyoruz. Tekrar hatırlatalım ki linklere verdiğimiz çapa adresi ile div'lerin idleri aynı olmalıdır.

Bir sonraki adımda ise JQuery kodlarımızı yazıyoruz. Diğer örneklerimizde olduğu gibi tab'lı yapımıza ait JQuery kodu da tek metottur.

```
$("#tabs").tabs();
```



JSON

JSON veri taşımak için oluşturulmuş bir standarttır. Açılımı **JavaScript Object Notation** demektir. Yapı olarak XML'e benzeyen JSON, JavaScript içerisinde kolayca işlenebilen bir yapıdır. JSON yapılarında kullanılan veri tipleri `string`, `number`, `bool`, `array`, `object` ve `null`'dur.

JSON özellikle AJAX platformunda tercih edilir. Bunun nedeni ise; JSON ile düzenlenen bir veri yapısının JavaScript komutları ile çok daha hızlı ve kolay bir şekilde erişilmesinden kaynaklanıyor.

JSON, yapı olarak XML'e benzese de XML'e göre çok hızlı parse edilir ve fiziksel boyut olarak da az yer kaplar. Bunun sebebi JSON da XML'deki gibi tag açıp kapama yoktur. Satırlar ve elementler birbirlerinden noktalama işaretleri ve parantezler ile ayrılırlar.

JSON yapısına örnek olarak aşağıdaki gibi bir desene sahiptir:

```
{Kullanici:[
  {
       "Adi": "Tahsin", "No":153, "Durum":true
  }
  {
       "Adi": "Volkan Aktaş", "No":187, "Durum":false
  }
]}
```

Öncelikle **Controller** sayfamızdan gönderilen JSON verisinin **View**'de gösterimi ile ilgili örneğimizi inceleyelim.

```
public ActionResult JSON()
{
    return View();
}
public JsonResult IsimleriVer()
{
    List<ListItem> list = new List<ListItem>()
    {
      new ListItem() { Value = "1", Text = "Tahsin Sevinç" },
      new ListItem() { Value = "2", Text = "Volkan Aktaş" },
```

```
        new ListItem() { Value = "3", Text = "Emrah Çapar" }
    };
    return this.Json(list, JsonRequestBehavior.AllowGet);
}
```

View için oluşturduğumuz `ActionResult`, gördüğünüz gibi boştur. Bununla birlikte bize JSON verisini döndüren `IsimleriVer` adlı metodumuz var. Metodumuzun sonunda dönen veriye dikkat edelim.

```
return this.Json(list, JsonRequestBehavior.AllowGet);
```

Bu kod, list yapısını JSON a döndürüyor.

```
<script type="text/javascript">
 $(function () {
   $('#btnDoldur').click(function () {
     $.getJSON("/Home/IsimleriVer", null, function (data) {
       JQuery.each(data, function (i, eleman) {
         $("#dvdata").append(eleman.Value+"-"+ eleman.Text+ "<br/>");
       });
     });
   });
 });
</script>
<h2>JSON</h2>
<input type="button" id="btnDoldur" value="Doldur" />
<div id="dvdata"></div>
```

View sayfamızdaki kodlarımızı incelediğimizde butonun tıklama event'ine kod yazıyoruz.

`getJSON` ile JSON datamızı alıyoruz. `getJSON` metodumuzun birinci parametresi JSON döndüren metodumuz, ikinci parametresi ise bu metoda göndereceğimiz parametredir. `function(data)` ile JSON verimiz data parametresine yükleniyor.

`JQuery.each`, C# ve VB .NET içerisinde kullanılan `foreach` döngüsüne benzemektedir. Bu kod sayesinde liste halinde gelen yapımızı ayrıştırıyoruz ve `dvData` id'li div'in içerisine verilerimizi ekliyoruz.



Server tarafından JSON datası aldığımız gibi JSON datası göndererek işlem de yapabiliriz. Öncelikle bir model oluşturuyoruz.

```
public class Kisi {
    public string Ad { get; set; }
    public string Soyad { get; set; }
}
```

Ardından View sayfamızda form oluşturuyoruz.

```
<div>Adı:@Html.TextBox("Ad")</div>
<div>Soyadı:@Html.TextBox("Soyad")</div>
<input type="button" id="btnKaydet" value="Kaydet" />
```

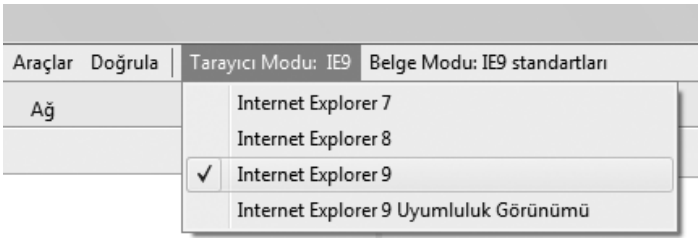
Oluşturduğumuz form yapısına uygun olarak JQuery kodlarımızı yazıyoruz.

```
$('#btnKaydet').click(function () {
  var Kisi = {
    Ad: $('#Ad').val(),
    Soyad: $('#Soyad').val()
  };
  $.ajax({
    url: '/Home/Kaydet',
    type: "POST",
    data: JSON.stringify(Kisi),
    dataType: "json",
    contentType: "application/json; charset=utf-8"
  });
  return false;
});
```

Ve Ajax ile datamızı post ediyoruz.

Burada dikkat etmemiz gereken husus; Ajax yapımızın ayarlarını uygun yapmamız. Ayrıca JQuery yapısındaki `JSON.stringify()` metodu IE9 ve üzeri browserlarda çalışmaktadır.

Internet Explorer üzerinde bu ayarı **`F12`**'ye basıp, tarayıcı uyumluluğunu Internet Explorer 9 olarak seçip yapıyoruz.



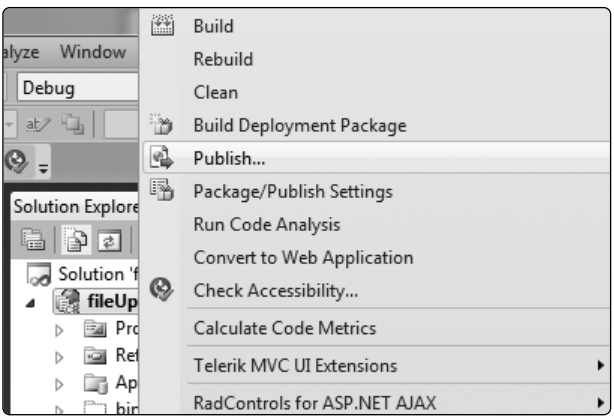
Uygulamamızı çalıştırdığımızda JQuery ile `TextBox`'lara girdiğimiz verilerin Controller'a aktarıldığını göreceksiniz.

```
[HttpPost]
public ActionResult Kaydet(Kisi pkisi)
{
    return View();
}
```

MVC 3 PROJESİNİN YAYINLANMASI

12

MVC 3 projemizi yayınlamak için **Publish** etmemiz gerekmektedir. Bu işlem için **Visual Studio**'nun **Solution Explorer** penceresinde bulunan projemize sağ tıklayıp **Publish**'e tıklıyoruz.

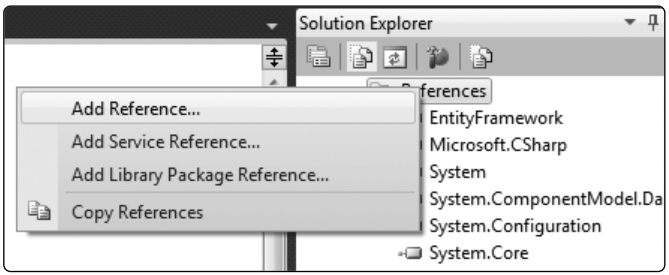


Daha sonra belirlenen yönergeleri izliyoruz. Bu yönergeler aşağıdaki referans ekleme işlemlerinden sonraki işlemlerin aynısı olduğu için referans ekleme işlemlerinden sonra detaylı olarak anlatılmıştır. Publish edilen dosyaları host'umuza yüklediğimizde sorunsuzca çalışması için, host'umuzun MVC3'ü desteklemesi gerekmektedir.

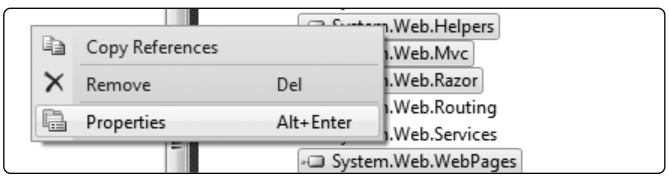
Peki, host'umuz MVC3'ü desteklemiyorsa ne yapmamız gerekiyor? Bunun için projemize...

- `Microsoft.Web.Infrastructure`
- `System.Web.Helpers`
- `System.Web.Mvc`
- `System.Web.Razor`
- `System.Web.WebPages`
- `System.Web.WebPages.Deployment`
- `System.Web.WebPages.Razor`

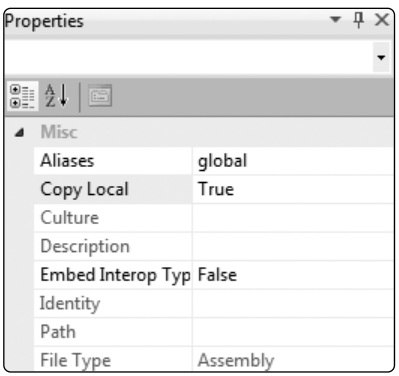
... referanslarını eklememiz gerekmektedir.



Referanslarımızı ekledikten bu referansları seçerek, sağ tıklayıp **Properties**'e tıklıyoruz.



Properties penceresinde bu dll'lerin **Copy Local** özelliğini `true` yapıyoruz.

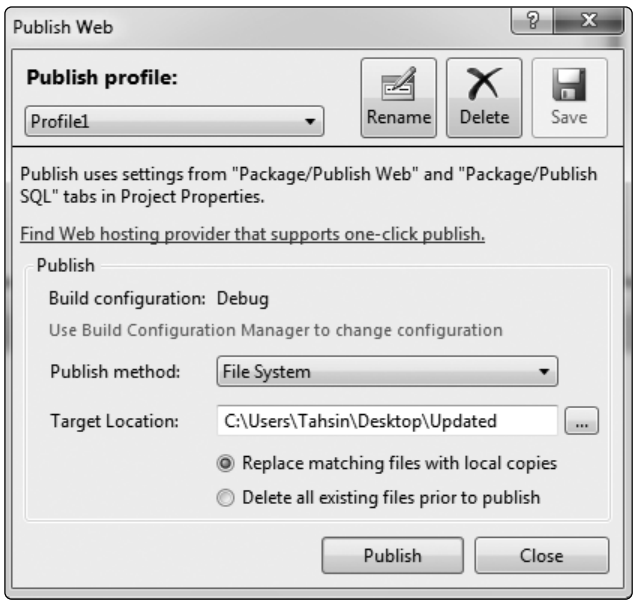


Bu sayede bu dll'ler publish edildiğinde `BIN` klasörümüz içerisinde oluşacak. Artık projemiz üzerinde sağ tıklayıp **Publish** seçeneğini tıklıyoruz. Açılan pencerede **Publish method** kısmında dört seçenek görececeğiz.

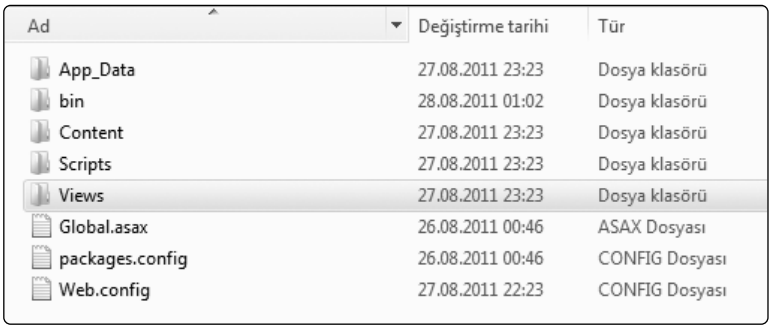
Bunlar:

- **Web Deploy:** Deployment servisi üzerinden publish edilir.
- **FTP:** FTP bilgileri girilerek, FTP üzerine publish edilmesi sağlanır.
- **File System:** Kullandığımız bilgisayar üzerinde seçilen klasöre publish edilir.
- **FPSE:** FrontPage sunucu uzantısı üzerinden publish edilir.

Biz **File System** seçeneğini seçiyoruz ve **Target Location** seçeneğinden klasörümüzü seçiyoruz. Ardından **Publish** butonuna tıklıyoruz.



Publish işlemi sonucunda ilgili klasörümüzde projemiz yayınlamaya hazırdır. Bu dosyaları server üzerine yükleyerek uygulamanızı çalıştırabilirsiniz.





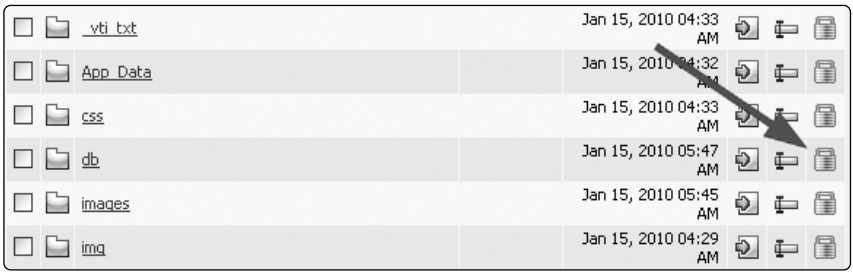
Bu işlemler sonucu dosya izinlerinden kaynaklı veya veritabanı bağlantısından kaynaklı problemler ile karşılaşabilirsiniz. Uygulamanızı host üzerine attığınızda bu ayarları kontrol etmeyi unutmayın.

Veritabanı bağlantı hatasını çözmek için **Web.config** üzerinden **connection string** üzerindeki bilgiler güncellenmelidir.

```
<connectionStrings>
<add name="sevinchnConnectionString"
connectionString="Data Source=xxx.xx.xx.xx;
Initial Catalog=xxx;
User ID=xx; Password=zzzz"
providerName="System.Data.SqlClient" />
</connectionStrings>
```

Güncellenecek alanlar, **data source** (*database adresi*), **catalog** (*veritabanı adı*), **user id** (*kullanıcı adı*) ve **password** (*şifre*) alanlarıdır.

Dosya yazma izinlerini ise hosting firmasının bize sağladığı panel aracılığı ile yapabiliriz. Windows Server üzerinde panel olarak genelde **Plesk Panel** kullanılır. Anlatımdaki örnek **Plesk Panel** üzerinden verilmiştir. Öncelikle hosting firmasından tahsis edilen kullanıcı adı ve şifre ile giriş yapıyoruz. Dosya dizinini açıyoruz.



İzin vereceğimiz dosyanın yanındaki kilit işaretine tıklayarak izin yazma sayfasını açıyoruz.



Kullanıcıları seçerek izin verip, kaydederek işlemimizi tamamlıyoruz.

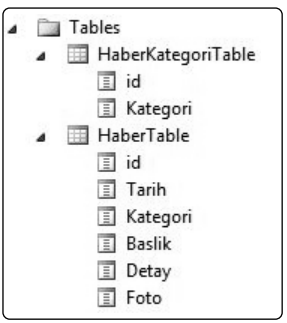
HABER PORTALI UYGULAMASI

13

Kitabımızda şu ana kadar işlediğimiz konulara örnek olması açısında bir uygulama yapacağız. Uygulamamız basit bir haber portalı olacaktır. Uygulama içerisinde haber eklenebilecek ve bu haberler kategorize edilebilecek bir yapıda olacaktır. Uygulama içerisinde girilen kelimenin aranacağı bir arama yapısı ve haberin yayınlandığı tarihe göre arşivleneceği bir arşivleme yapısı bulunacaktır.

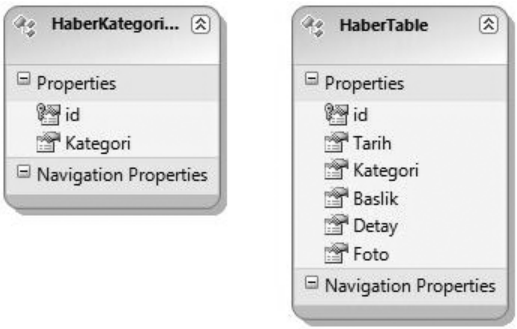
Model Yapisi

Uygulamamızda öncelikle model yapımızı oluşturacağız. Uygulamamızda haberler ve kategoriler olacak. Bu yüzden bu bilgilerin tutulacağı tabloları oluşturmamız gerekiyor.



Kategori tablomuzda **kategori adı** ve **id**'sinin bulunması yeterli olacaktır. Haber bilgilerinin tutulduğu tablomuzda ise **id**, **haberin tarihi**, **haberimizin kategorisi**, **başlık bilgisi**, **haberimizin içeriği** ve **fotoğraf** bilgisinin tutulması gerekecektir.

Veritabanımızı oluşturduktan sonra modelimizi **Entity Framework** ile modelliyoruz.



Modelimiz üzerinde son olarak `Haber` tablomuzu **validate** ediyoruz. Bu sayede zorunlu alanlar ve modelimizdeki alanların View'de gösterimini düzenliyoruz.

```
public int id { get; set; }
[Required(ErrorMessage = "Tarih Girilmesi Zorunludur!")]
[Display(Name = "Haber Tarih:")]
public DateTime Tarih { get; set; }
[Required(ErrorMessage = "Lütfen Kategori Seçiniz!")]
[Display(Name = "Haber Kategorisi:")]
public int Kategori { get; set; }
[Required(ErrorMessage = "Lütfen Haber Başlığı Giriniz!")]
[Display(Name = "Haber Başlığı:")]
public string Baslik { get; set; }
[Display(Name = "Haber Fotoğrafı:")]
public string Foto { get; set; }
[Required(ErrorMessage = "Lütfen Haber Detayını Giriniz!")]
[Display(Name = "Haber Detayı:")]
public string Detay { get; set; }
```

Haber ve Kategori Ekleme

Uygulamamıza öncelikle haber ve kategori ekleme işlemlerini yapacağımız Controller ve View'leri ekliyoruz. Öncelikle Controller'larımızı oluşturuyoruz. Kategorilerimizi `DropDownList` ile seçeceğimiz için kategorilerimizi `SelectList` yapısı ile bir `ViewData`'ya atıyoruz.

```
public ActionResult Ekle()
{
    ViewData["Kategori"] = new SelectList(db.HaberKategoriTable, "id",
       "Kategori", 0);
```

```
    return View();
}   //yeni haber eklemek için kullanılacak form action
```

Aşağıda ise gelen veri `HaberTable` modeline atanıyor. Foto variable'ına resmimizin uzantısı atanıyor ve datamız veritabanına kaydediliyor. Resmimiz klasöre **haberin ID'si+uzantısı** şeklinde eklenmektedir.

```
string filePath =
    Path.Combine(HttpContext.Server.MapPath("../Content/image"),
    Path.GetFileName(habertable.id.ToString() +
    Path.GetExtension(haberform.FileName)));
haberform.SaveAs(filePath);
return RedirectToAction("Index");
```

Resmimiz **Content** klasörü içerisindeki **image** klasörünün içerisine **Haberin ID'si+Resmin Uzantısı** formatında kaydediliyor.

```
[ValidateInput(false)]
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult Ekle(HttpPostedFileBase haberform)
{
    if (haberform.ContentLength>0)
    {
        HaberTable habertable = new HaberTable();
        habertable.Tarih=Convert.ToDateTime(Request.Form["Tarih"]);
        habertable.Kategori =Convert.ToInt32(Request.Form["Kategori"]);
        habertable.Baslik = Request.Form["Baslik"];
        habertable.Detay = Request.Form["Detay"];
        habertable.Foto = Path.GetExtension(haberform.FileName);
        db.HaberTable.AddObject(habertable);
        db.SaveChanges();
        string filePath =
           Path.Combine(HttpContext.Server.MapPath("../Content/image"),
           Path.GetFileName(habertable.id.ToString() +
           Path.GetExtension(haberform.FileName)));
        haberform.SaveAs(filePath);
        return RedirectToAction("Index");
    }
    return RedirectToAction("Index");
}
```

View'imizi oluşturduktan sonra bazı yerleri değiştiriyoruz. Öncelikle kategorilerimizi `DropDownList` ile seçeceğiz.

```
<div class="editor-label">
    Haber Kategorisi:
</div>
<div class="editor-field">
    @Html.DropDownList("Kategori", "Lütfen Kategori Seçiniz")
</div>
```

Resimlerin yüklemesi için ilgili yeri **input file** ile değiştiriyoruz.

```
<div class="editor-label">
   Haber Fotoğrafı:
</div>
<div class="editor-field">
   <input name="haberform" type="file" />
</div>
```

Haberimizin detayını yazacağımız giriş alanı html editör olacağı için **CK Editör**'ü projemize ekliyoruz. **CK Editörü** *http://ckeditor.com/download* adresinden indiriyoruz ve **CK Editör** klasörünü projemizin bulunduğu dizine kopyalıyoruz. `Layout.cshtml` sayfamıza script'imizi ekliyoruz.

```
<script type="text/javascript" src="../../ckeditor/ckeditor.js"></script>
```

View'imizde detay bölümünü aşağıdaki gibi değiştiriyoruz.

```
<div class="editor-label">
   Haber Detayı:
</div>
<div class="editor-field">
   <textarea class="ckeditor" cols="80"
id="editor1" name="Detay" rows="10"></textarea>
</div>
```

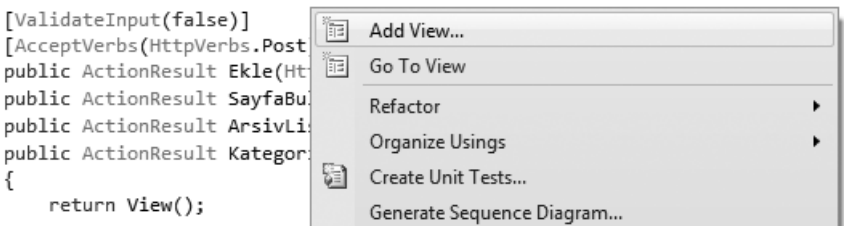
Ve son olarak tarihi gireceğimiz alana **JqueryUI** ile `DatePicker` ekliyoruz.

```
<script type="text/javascript">
  $(function () {
    $("#Tarih").datepicker({ dateFormat: 'dd-mm-yy' });
  });
</script>
```

Haber ekleme ile ilgili yapacaklarımız bu kadar. Artık kategori ekleme işlemine geçebiliriz. Öncelikle kategori ekleme işlemi için Controller'a `ActionResult`'larımızı yazıyoruz.

```
public ActionResult KategoriEkle()
{
    return View();
}
[HttpPost]
public ActionResult KategoriEkle(FormCollection collect)
{
    HaberKategoriTable kategori=new HaberKategoriTable();
    kategori.Kategori=Request.Form["Kategori"].ToString();
    db.HaberKategoriTable.AddObject(kategori);
    db.SaveChanges();
    return View();
}
```

Ardından **Add View** diyerek View'imizi ekliyoruz.



Global.asax Sayfamızda Route Ayarları

Route işlemleri, uygulamamızın link yönetimini esnekleştiren ve SEO yapısına uygun linkler üretmemizi sağlayan yapıdır. Haber uygulamamızda özel olarak ayarlayacağımız Route yapıları aşağıda verilmiştir.

```
routes.MapRoute(
    "ArsivRoute", // Route ismi
    "Arsiv/{tarih}", // URL parametresi
    new { controller = "Haber", action = "Arsiv", tarih = "" },
    // Varsayılan parametreler
    new { tarih = new tarihConstraint() }
);
```

Arşiv gösterimi için /Arsiv/12.09.2011 gibi bir url yapımız olacaktır.

```
routes.MapRoute(
    "Arama", // Route ismi
    "Aranan/Arama", // URL parametresi
    new { controller = "Haber", Action="Ara"}
    // Varsayılan parametreler
);
```

Arama işlemini gerçekleştirecek url /Aranan/Arama şeklinde olacaktır.

```
routes.MapRoute(
     "Detay", //Route ismi
     "{id}",
     new{ controller="Haber", action= "Detay", id= ""},
     new{ id=@"\d+"}
);
```

Haberin detayını okumak için sadece /1 şeklinde haberin ID'sinin yazılması yeterli olacaktır.

```
routes.MapRoute(
     "Kategori",//Route ismi
     "kategori/{kid}",
     new { controller = "Haber", action = "Kategori", kid = "" },
     new { kid = @"\d+" }
);
```

Kategorilere ait haberleri görmek için /Kategori/2 gibi kategori ID'sinin girilmesi yeterli olacaktır.

Haberlerin Görüntülenmesi

Ekleme işlemlerini yaptıktan sonra gösterim ile ilgili sayfalarımızı hazırlayabiliriz. Öncelikle master sayfamız olan layout sayfamızı hazırlayalım. Sayfamızın kenarında kategorilerimizin listelendiği bir menü olacak. Bu menüyü hazırlamak için layout sayfamıza aşağıdaki kodu ekliyoruz.

```
<ul class="menuana" style="left: -31px; top: 6px; width: 115px">
@{
      HaberPortal.Models.DataHaberEntities db =
             new HaberPortal.Models.DataHaberEntities();
```

```
      foreach (var l in db.HaberKategoriTable)
      {
            <li class=”menu” onmouseout=”this.className='menu'”
             onmouseover=”this.className='menuactive'”>@Html.ActionLink(
             l.Kategori, l.id.ToString(), “Kategori”)
           </li>
      }
}
```

CSS kodlarına, uzunluğu nedeniyle burada yer vermiyoruz. Örnek uygulama içerisinden erişebilirsiniz.

```
<li class="menu" onmouseover="this.className='menuactive'"
  <a href="/Kategori/1">
    Metin - Güncel
<li class="menu" onmouseover="this.className='menuactive'"
  <a href="/Kategori/2">
    Metin - Siyaset
<li class="menu" onmouseover="this.className='menuactive'"
  <a href="/Kategori/3">
    Metin - Ekonomi
<li class="menu" onmouseover="this.className='menuactive'"
  <a href="/Kategori/4">
    Metin - Dünya
<li class="menu" onmouseover="this.className='menuactive'"
  <a href="/Kategori/5">
    Metin - Eğitim
```

Kodumuzu incelediğimizde yukarıdaki gibi bir html çıktımız olacaktır. Kategori sayfamız, `kategori` ID'sini parametre olarak alacak ve bize o kategorideki yazıları gösterecek sayfamız olacaktır.

Verilen kategoriye ait haberlerin listelendiği sayfamızı oluşturabiliriz. Öncelikle ilgili sayfanın `ActionResult`'ını yazalım.

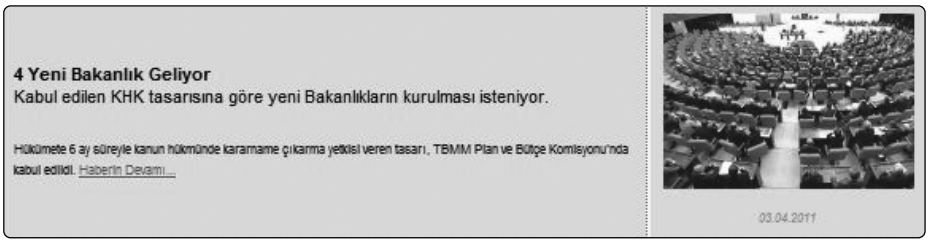
```
public ActionResult Kategori(int kid)
{
     IList<HaberTable> kategori_haberleri = new List<HaberTable>();
     kategori_haberleri = (from p in db.HaberTable
                           where p.Kategori == kid
                           select p).ToList();
     return View(kategori_haberleri);
} //Haberleri kategorilerine göre listeleyen action
```

LINQ ile tablomuzdaki kategorimize ait haberleri çekiyoruz. Sorgumuzda kullandığımız...

```
where p.Kategori == kid
```

... ile `kategori` ID'ye göre filtrelememizi yapıyoruz. Artık View'imizi oluşturabiliriz.

View'imizi oluşturduktan sonra haberimizin görüntülenme şeklinin aşağıdaki gibi olması için bazı değişiklikler yapacağız.



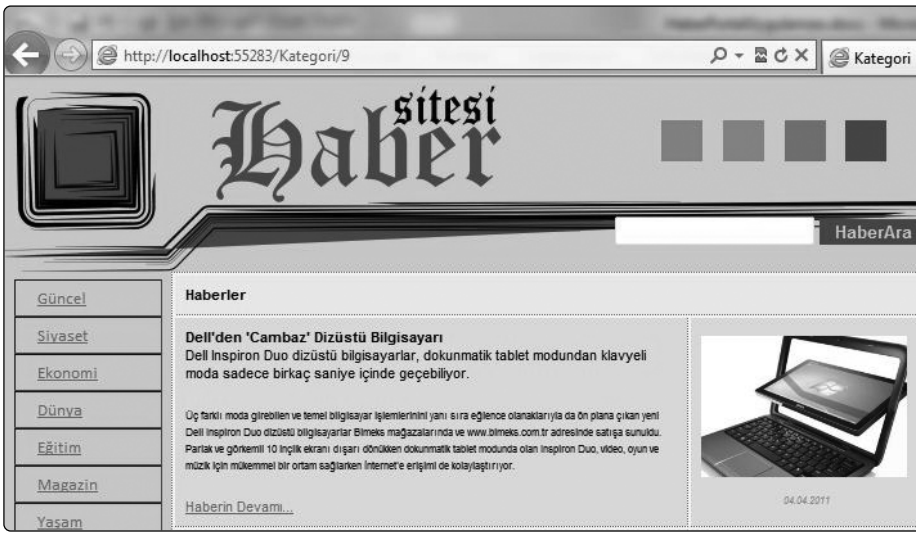

```
@foreach (var item in Model)
{
  <tr class="row-alternating">
       <td>
        <b>@item.Baslik</b>
       @{
            string haber = item.Detay;
            if (haber.Contains("page-break-after"))
            {
               haber = haber.Substring(0,
                (haber.IndexOf("page-break-after") - 12));
                 haber += "<a href='/" + item.id + "'> Haberin
                      Devamı...</a>";
            }
        }
        @MvcHtmlString.Create(haber)
       </td>
       <td>
          <div class="photo-content"> <img src=
             "../content/image/@item.id@item.Foto" alt="@item.Baslik"
             height="110"/>
          </div>
```

```
            <div class="photo-caption">
                @item.Tarih.ToString("dd.MM.yyyy")
            </div>
        </td>
    </tr>
}
```

Yukarıdaki kodumuzu inceleyelim.

Öncelikle başlığımızı yazdırıyoruz ve haberimizin detayında **page-break-after** olan kısma kadar gösterip, buna *"haberin detayı"* adı altında link ekliyoruz. Page-break-after, CK Editör içerisinde bir özelliktir. Bu özelliğin kullanım amacı, sayfamızın özet kısmına kadar olan kısmı belirlememizi sağlıyor olmasıdır. Bu özelliği kod ile burada kullanıyoruz.

İçeriği istediğimiz şekilde yayınladıktan sonra yan kısımda resmi gösteriyoruz ve resmin altında tarih yayınlanıyor. Kategori gösterimimiz aşağıdaki gibi olacaktır.



Kategori işlemlerini bu şekilde tamamlıyoruz ve artık `Index` sayfamızı oluşturabiliriz. `Index` sayfamız da kategori sayfamızdaki gibi bir görünüme sahip olacak. Fakat farklı olarak burada tüm haberler listelenecektir.

Index sayfamıza ait `ActionResult` aşağıdaki gibi olacaktır.

```
public ActionResult Index()
{
    return View(db.HaberTable.ToList());
} //Sitemizin Anasayfası
```

View'imizi oluşturuyoruz ve içeriğini aşağıdaki gibi değiştiriyoruz.

```
@foreach (var item in Model)
{
    <tr class="row-alternating">
        <td>
         <b>@item.Baslik</b>
        @{
             string haber = item.Detay;
             if (haber.Contains("page-break-after"))
             {
               haber = haber.Substring(0,
                   (haber.IndexOf("page-break-after") - 12));
               haber += "<a href='/" + item.id + "'> Haberin Devamı...</a>";
             }
         }
         @MvcHtmlString.Create(haber)
        </td>
        <td>
            <div class="photo-content"> <img src=
             "../content/image/@item.id@item.Foto" alt="@item.Baslik"
             height="110"/>
            </div>
            <div class="photo-caption">
             @item.Tarih.ToString("dd.MM.yyyy")
            </div>
        </td>
    </tr>
}
```

Haber portalımızın index sayfasını ve kategorilerimizin bulunduğu sayfalarımızı hazırladıktan sonra haberimizin gösterileceği detay sayfamızı hazırlayabiliriz. Detay sayfamız, gönderilen ID'ye ait haberin tüm içeriğini gösterecektir. Öncelikle `ActionResult`'ımızı oluşturalım.

```
db.HaberTable.Single(p => p.id == id)
```

... ile gelen ID'nin veritabanında karşılığı olan kaydı seçiyoruz ve bu kaydı View'e gönderiyoruz.

```
public ActionResult Detay(int id)
{
  HaberTable haber_detay = db.HaberTable.Single(p => p.id == id);
  return View(haber_detay);
} //Haberlerin detaylarının çağrıldığı action
```

View'imiz ise aşağıdaki gibi olacaktır.

```
<div class="content-main"><h1 class="first">@Model.Baslik</h1>
  <div class="photo-container align-right" style="width:202px;">
    <div class="photo-content"> <img src=
      "../content/image/@Model.id@Model.Foto" alt="@Model.Baslik"
            height="110"/>
    </div>
    <div class="photo-caption">@Model.Tarih.ToString("dd.MM.yyyy")</div>
  </div>
  @MvcHtmlString.Create(Model.Detay)
</div>
```

Haberimizin başlığı, içeriği, tarihi ve resmini detay sayfamızda gösteriyoruz. Detay sayfamızın görünümü aşağıdaki gibi olacaktır.



Haber portalımızın arama kısmını yapmak için öncelikle arama işleminin gerçekleşeceği **Partial View**'i oluşturarak, içerisine arama için kelime yazacağımız `TextBox` ve `Button`'umuzu koyuyoruz. `BeginForm`'umuzu `RouteMap`'imize göre yapılandırıyoruz ve aranan kelimeyi **Query String** ile gönderiyoruz.

```
@using(Html.BeginForm("Arama","Aranan",new { arama=
HttpContext.Current.Request.QueryString["arama"] },
FormMethod.Get))
{
      <input type="text" name="arama"/>
      <input type="submit" value="HaberAra" id="arabutton"/>
}
```

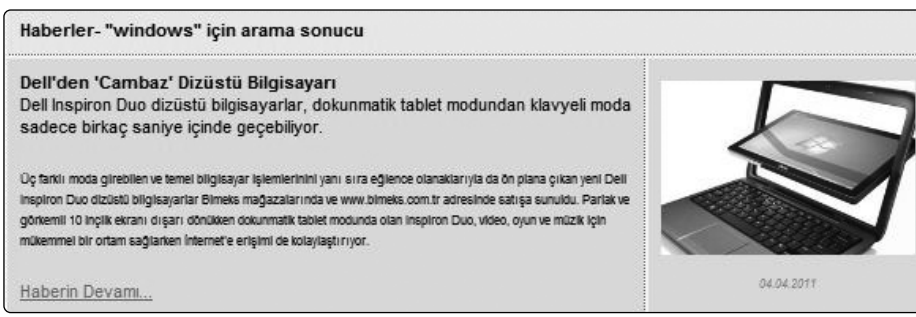
Butonumuza tıklandığında **Global.asax**'da belirlediğimiz Route yapısına uygun olarak linkimiz oluşur ve aranacak ifade Query String ile `Arama` isimli `ActionResult`'ımıza gönderilir.

```
public ActionResult Ara(string arama)
{
  arama=Request.QueryString["arama"];
  IList<HaberTable> aramasonucu = new List<HaberTable>();
  if (!String.IsNullOrEmpty(arama))
  {
       ViewData["arananKelime"] = arama;
       aramasonucu = (from p in db.HaberTable
                      where p.Detay.Contains(arama)
                      select p).ToList();
  }
  return View(aramasonucu);
} //Haberler içerisinde arama yapan action
```

Öncelikle `aramasonucu` listesine tüm haberler listeleniyor ve aranan kelime boş değilse, haberlerin detayları içerisinde kelimemiz aranıyor. `ActionResult` içerisinde girilen kelimenin geçtiği tüm haberleri listeleyerek View sayfamıza gönderiyoruz. View'imizi ekliyor ve içeriğini aşağıdaki gibi değiştiriyoruz. View sayfamızın içeriği de kategori ve index sayfasındaki gibi özet gösterim şeklinde olacaktır.

```
<table class="data-table" width="589">
  <tr>
       <th colspan="2" >
```

```
            Haberler -
            "@ViewData["arananKelime"]" için arama sonucu
        </th>
    </tr>
@foreach (var item in Model)
{
    <tr class="row-alternating">
        <td>
            <b>@item.Baslik</b>
            @{
                string haber = item.Detay;
                if (haber.Contains("page-break-after"))
                {
                haber = haber.Substring
                    (0, (haber.IndexOf("page-break-after") - 12));
                haber += "<a href='/" + item.id + "'> Haberin Devamı...</a>";
                }
            }
            @MvcHtmlString.Create(haber)
        </td>
        <td>
            <div class="photo-content"> <img src=
                "../content/image/@item.id@item.Foto" alt="@item.Baslik"
                height="110"/>
            </div>
            <div class="photo-caption">
                @item.Tarih.ToString("dd.MM.yyyy")
            </div>
        </td>
    </tr>
}
```



Uygulamamız içerisinde son olarak arşivleme özelliğini ekleyeceğiz. Arşivleme özelliği girilen haberleri tarihlerine göre gruplayarak seçilen tarihteki haberleri gösterecek bir yapı olacaktır. Bunun için öncelikle tarihlerin listeleneceği sayfayı oluşturacağız. Tarihleri almak için aşağıdaki gibi bir `ActionResult` oluşturuyoruz.

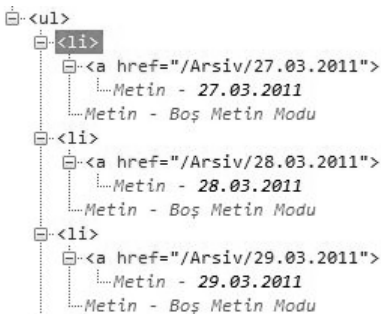
```
public ActionResult ArsivListe()
{
     List<DateTime> Haberler=
        (from p in db.HaberTable select p.Tarih).Distinct().ToList();
     ViewBag.ListeTarih=Haberler;
     return View();
}
```

`ActionResult`'ımızda LINQ ile tüm haberlerin yayınlandığı tarihlerimizi seçiyoruz ve `Distinct` ile birden fazla aynı kayıt varsa, bu kayıtları teke indiriyoruz. Listeyi `ViewBag` üzerine atıyoruz. Bu `ViewBag`'i View sayfamızda işleyeceğiz.

View'imizi ekliyor ve içeriğini aşağıdaki gibi düzenliyoruz.

```
<ul>
  @foreach(DateTime item in ViewBag.ListeTarih)
  {
    <li> @Html.ActionLink(item.ToString("dd.MM.yyyy"),
         "Arsiv", "Haber", new { tarih=item.ToString("dd.MM.yyyy") }, null)
    </li>
  }
</ul>
```

Tarihleri `ul` listemiz ile listeliyoruz. `ActionLink` ile tarihleri link yapısında gösteriyor ve tarihlere tıklandığında `Arsiv` sayfasında ilgili tarihe ait haberleri görüntülüyoruz.



Son olarak tarihe tıklandığında haberlerimizin gösterileceği sayfayı hazırlayacağız. Sayfamız dışarıdan gelen `tarih` parametresini alıp o tarihte yayınlanmış haberler listelenecektir.

`ActionResult`'ımız aşağıdaki gibi olacaktır.

```
public ActionResult Arsiv(string tarih)
{
     CultureInfo cuinfo = new CultureInfo("tr-TR");
     DateTime _tarih = DateTime.Parse(tarih, cuinfo);
     ViewData["tarih"] = _tarih;
     IList<HaberTable> arsivsonucu = new List<HaberTable>();
     arsivsonucu = (from p in db.HaberTable
                    where (p.Tarih == (_tarih))
                    select p).ToList();
     return View(arsivsonucu);
} // Tarihlere göre haberleri listeleyen action
```

Gelen tarihe göre arama işlemi LINQ yardımıyla...

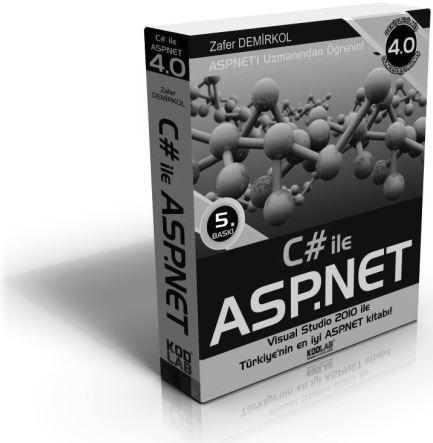
```
where (p.Tarih == (_tarih))
```

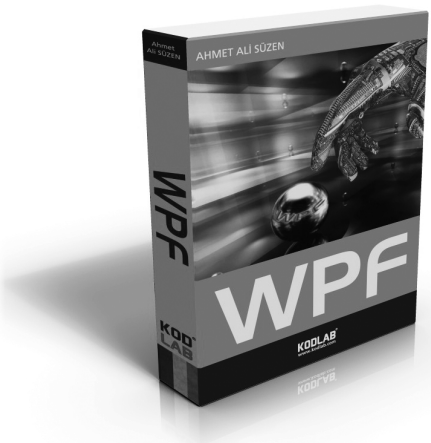
... haberleri filtreleyerek yapıyoruz ve listemizi View sayfamıza gönderiyoruz. View sayfamız diğer listeleme sayfalarında olduğu gibi özet gösterim olacaktır.

```
@foreach(var item in Model)
{
  <tr class="row-alternating">
    <td>
      <b>@item.Baslik</b>
      @{
         string haber = item.Detay;
        if(haber.Contains("page-break-after"))
        {
          haber = haber.Substring(0,
            (haber.IndexOf("page-break-after") - 12));
          haber += "<a href='/" + item.id + "'> Haberin Devamı...</a>";
        }
      }
      @MvcHtmlString.Create(haber)
    </td>
    <td>
```

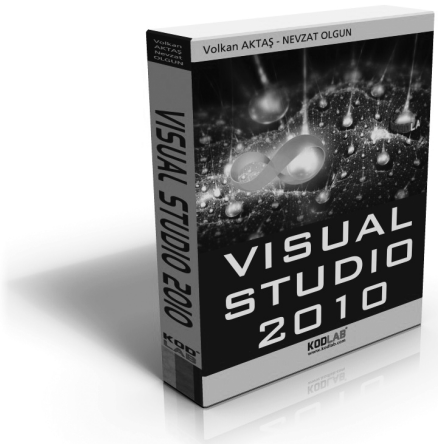
```
        <div class="photo-content">
          <img src="../content/image/@item.id@item.Foto" alt=
            "@item.Baslik" height="110"/>
        </div>
        <div class="photo-caption">
            @item.Tarih.ToString("dd.MM.yyyy")
         </div>
      </td>
    </tr>
}
```

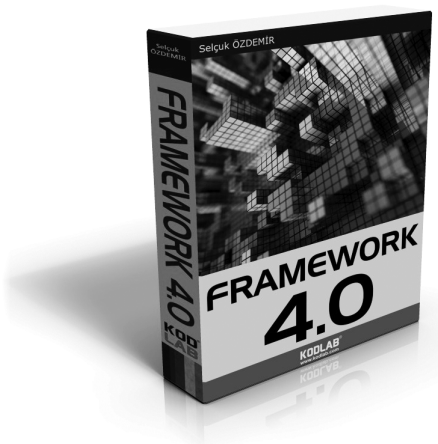

Ahmet Ali SÜZEN
4
SILVERLIGHT 4
C#.NET ve VB.NET ile Uygulamalı Olarak Silverlight Öğrenin!
Silverlight
KODLAB


C# ile ASP.NET 4.0
Zafer DEMİRKOL
C# ile ASP.NET
ASPNET'i Uzmanından Öğrenin!
4.0
5. BASKI
C# ile
ASP.NET
Visual Studio 2010 ile
Türkiye'nin en iyi ASPNET kitabı!
KODLAB


Ahmet Ali SÜZEN
AHMET ALİ SÜZEN
WPF
KODLAB


Volkan AKTAŞ
HERKES İÇİN!
C#
VISUAL STUDIO 2010 İLE
HER YÖNÜYLE
C# 4.0
KODLAB


Volkan AKTAŞ - NEVZAT OLGUN
VISUAL STUDIO 2010
KODLAB


Selçuk ÖZDEMİR
FRAMEWORK 4.0
KODLAB