

İÇİNDEKİLER

MODEL VIEW CONTROLLER NEDİR?	3
Model	3
View	3
Controller	4
ASP.NET MVC ÇALIŞMA MANTIĞI	4
ASP.NET MVC AVANTAJLARI DEZAVANTAJLARI	5
Neden Asp.Net Web Forms ?	5
ViewState Kullanımı	5
Zengin Sunucu Kontrolleri	5
Olay Güdümlü Programlama (Event Driven Programming)	5
Neden Asp.Net MVC ?	6
Performans	6
Yeniden Kullanılabilirlik (Reuseability)	6
Test Driven Development	6
Yazılım Geliştirme Süresi	6
İstemci Tarafli Geliştirme (Client-Side Development)	6
ASP.NET MVC PROJESİ OLUŞTURMA	6
1.Şablon Türleri	7
2.Proje Türü	7
3.Name	7
4.Location	7
5.Solution Name	7
6.Proje Türü	7
7.Framework Versiyonu	8
8.Sıralama Şekli	8
9.Create directory for solution	8
10.Add to Source Control	8
MODEL, VIEW, CONROLLER OLUŞTURMA	8
Controller	8
MVC 5 Controller – Empty :	9
MVC 5 Controller with read/write actions	10
MVC 5 Controller with views, using Entity Framework	10
Model Seçimi	11
Context Seçimi	11
Async Controller Action	11
Generate Views	11
Reference script libraries	11
Use a layout page	11
Layout Seçimi	11
Controller Name	12
Model	12
Views	12
1.View Name	12
2.Template Seçimi	12
3.Model Class	12
4.Data Context Class	13
5.Create as a partial view	13

6.Reference script libraries	13
7.Layout Seçimi	13
8.Layout Seçimi	13
VIEW ENGINE NEDİR ?	13
RAZOR VIEW ENGINE	13
@ OPERATÖRÜ	13
HTML HELPERS.....	15
HtmlHelper	16
TextBox Oluşturma.....	16
TextBox()	16
TextBoxFor	17
Custom Helper	17
Data Transfer To View	18
ViewBag	18
ViewData	19
TempData.....	20
Layout View.....	22
Partial View.....	23
Render Partial View	26
Html.Partial()	26
Html.RenderPartial().....	26
Html.RenderAction()	26
Section	27
App_Start.....	28
Minification	29
Bundle Types	30
ScriptBundle in ASP.NET MVC.....	30
CDN Kullanımı	30
StyleBundle in ASP.NET MVC.....	31
FilterConfig.....	31
Custom Filter Oluşturma	33
RouteConfig.....	34
Route.....	34
Route Constraints.....	35
Area.....	35
Multiple Routing.....	36
Yeni Area Ekleme.....	37
Grid MVC	37
Jquery ajax() method.....	40
Perform Ajax request.....	41
Perform Http POST request using ajax() method.....	42
State Management	43
Application State	43
Session State.....	44
Caching (Önbellekleme)	45
Data Caching (Veri Önbellekleme).....	46
Output Caching (Çıktı Önbellekleme)	48
Cookie.....	50

MODEL VIEW CONTROLLER NEDİR?

MVC, Model-View-Controller isimlerinin baş harflerinin oluşturduğu bir yazılım mimari deseni (architectural pattern)dir. Yazılım Mühendisliği'nde yeni bir teknoloji olarak görünse de geçmişi 1979 yılına dayanır. Tygve Reeskaug tarafından ortaya çıkan bu mimari PHP, Java gibi yazılım dillerinde aktif olarak kullanılmaktaydı. 2009 yılında Microsoft MVC Pattern'inin gelişiminin farkına vardı ve kendi framework'üne dahil etti. MVC 1.0 ile oraya çıkan yapı günümüzde MVC 5.1 ile devam etmekte ve hızla gelişmeye devam etmektedir.

Model

Bu katmanda projedeki veri akışı sağlanmaktadır. Veritabanı ile ilişki kuran katman olması ile birlikte alışverişini yaptığı tüm bilgileri kendi içerisinde işleyebilme olanağına sahip bir katmandır. OOP (Object Oriented Programming) ile hayatımıza giren Model yapısı MVC ile çok daha aktif ve işlevsel hale gelmiştir. Model içinde verilerin doğruluk, zorunluluk, okunabilirlik gibi kontrolleri yapılarak, veri türleri arasında bağlantı kurabilme olanağına sahip yapılar türetebiliriz.

Örneğin; Bir blog sitesi oluşturduunuz ve kullanıcıdan websitenize kayıt olmalarını istiyorsunuz. Böyle bir durumda karşınıza bir form çıkartmanız gerekir.

```
Models
├── AccountViewModels.cs
├── IdentityModels.cs
└── ManageViewModels.cs
```

```
public class IndexViewModel
{
    public bool HasPassword { get; set; }
    public IList<UserLoginInfo> Logins { get; set; }
    public string PhoneNumber { get; set; }
    public bool TwoFactor { get; set; }
    public bool BrowserRemembered { get; set; }
}
```

Bu durumda Model katmanında kendi oluşturmuş olduğunuz 'Uye' sınıf(class)'ı içerisinde 'Ad, Soyad, Yaş, E - Posta, Cinsiyet' gibi bilgileri zorunlu olarak girmesini isteyebiliriz hatta yaş alanına sayısal karakter girilmesini, e - posta alanına bir e - posta formatı girilmesini zorunlu tutabiliriz. Ayrıca bu class üzerinden diğer sınıf(class)lara ve oradaki verilere de erişebiliriz. Bir üyenin hangi şehirden üye olduğunu eklemek istediğimizde diğer bir sınıf olan 'Şehir'e ulaşarak oradaki şehir bilgisini üyenin bilgilerine kaydedebiliriz. (Bu konuyu ilerleyen bölümlerde detaylı olarak işleyeceğiz.)

View

Projede kullanıcıların gördüğü arayüzdür. Dış dünyada kullanıcılar ne görüyorlarsa view sayesinde. Proje yayına alındıktan sonra view içerisinde nasıl bir kodlama var ise kullanıcılar tarayıcıdan websitesini görüntülemek istediklerinde karşınıza o çıkacaktır.

Örneğin; www.blog.com sitesini bir tarayıcı üzerinden görüntülemek isteyen kullanıcı açılan sayfada en yeni 10 makalenin başlıklarını görüntüleyecektir. Siteyi yöneten blog sahibi yönetim panelinden her makale eklediğinde o makale en yeni olarak en üste gelecektir. Aynı zamanda en eski olan makale de kaldırılacaktır. Otomatik olarak en yeni 10 makalenin görüntülenmesini sağlayan kodlamayı view üzerinde gerçekleştirecektir. Client Side (istemci tarafı) kodlama HTML, CSS, Javascript, JQuery ile Server Side (sunucu tarafı) kodlama ise C# olarak yapılır.

```
Views
├── Account
├── Home
│   ├── About.cshtml
│   ├── Contact.cshtml
│   └── Index.cshtml
├── Manage
└── Shared
    ├── _Layout.cshtml
    ├── _LoginPartial.cshtml
    ├── Error.cshtml
    ├── Lockout.cshtml
    ├── _ViewStart.cshtml
    └── Web.config
```

Controller

Projenin merkez noktasıdır. View ile Model arasındaki işlemleri gerçekleştiren Controller katmanı ile kullanıcı web sitesini görüntülemek istediği anda, görüntülemek istediği View ile bağlantılı olan Controller'a gidilir orada gerekli veriler ve işlemler yapıldıktan sonra sayfa kullanıcıya gösterilir.

```
Controllers
├── AccountController.cs
├── HomeController.cs
└── ManageController.cs
```

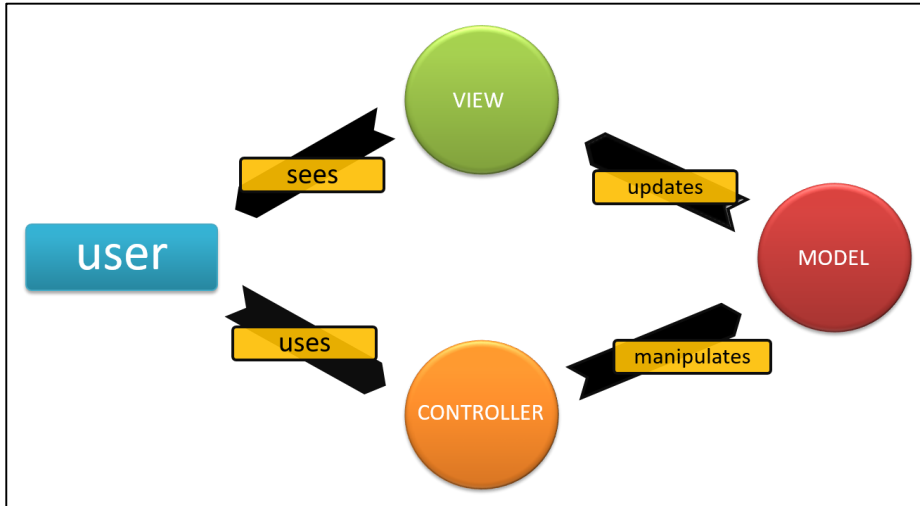
Controller'da işlemler Server Side (sunucu tarafı) olduğundan kodlama C# ile yapılır. Örneğin; Blog sitesi içerisinde makaleleri görüntülemek istiyorsak Model üzerinden aldığı verileri View'da listeletme işlemini burada yaparız.

ASP.NET MVC ÇALIŞMA MANTIĞI

Asp.Net ile web platformuna geçerken detaylı olarak ele aldığımız 'Page Life Cycle' (Sayfa yaşam döngüsü) aşamalarında bir kullanıcının web sayfasını görüntülemek istediği anda hangi aşamalardan geçtiğini görmüştük. Bu aşamaları inceleyecek olursak;

1. Kullanıcı görüntülemek istediği sayfanın 'Alan Adı(Domain)'ni tarayıcının adres çubuğuna yazar ve sayfayı görüntüleme isteği gönderilir.
2. Gönderilen istek ilk olarak DNS(İsim Sunucusu)'e gider Alan Adı'na karşılık gelen IP(İnternet Protokol)'yi bulur.
3. IP üzerinden gideceği Sunucu'ya isteği gönderir.
4. İsteği alan sunucu, içerisinde bulunan projeye gider ve istek yapılan sayfayı hazırlar.
5. Sunucu hazırladığı web sayfasının son halini istek yapan kullanıcıya(istemci) gönderir.
6. Kullanıcı sayfayı görüntüler.

Genel olarak bir web uygulamasının yaşam döngüsü bu şekildedir. MVC projesinin de kendi içerisinde bir yaşam döngüsü bulunmaktadır. Bu yaşam döngüsü ise aşağıdaki gibidir.



1. Kullanıcı görüntülemek istediği sayfanın Alan Adı(Domain)'ni tarayıcının adres çubuğuna yazarak istek gönderir.
2. Bu istek internet üzerinden projenin bulunduğu sunucuya DNS üzerinden gönderilir.
3. Gönderilen istek http protokolü üzerinden Controller'a iletilir.
4. İstek Controller'da işlenirken gerekli duyulan veriler için Model' gidilir.
5. Model istenilen verileri temin etmek için Veritabanı(Database)'e gider.
6. Veriler alınıp işlendikten sonra Controller'a gönderilir.
7. Controller'da gerekli işlemler yapıldıktan sonra veriler View'a aktarılır.

8. View içerisindeki HTML ve C# kodları gelen veriler ile çalıştırılır ve Http protokolü üzerinden kullanıcıya gönderilir.
9. Kullanıcı sayfayı görüntüler.

ASP.NET MVC AVANTAJLARI DEZAVANTAJLARI

Yazılım dünyasında Web Forms mu ? MVC mi ? soruları sıklıkla sorulmaktadır. Bu seçimi yaparken iki teknolojiyi de iyi tanımak gerektiğini vurgulamak gerekir. Her iki teknolojinin de kendine özel avantajları ve dezavantajları vardır. Bu ayrımları iyi yaparak projeyi hangi teknoloji ile sürdürmek gerektiğine karar verilmesi en doğrusudur.

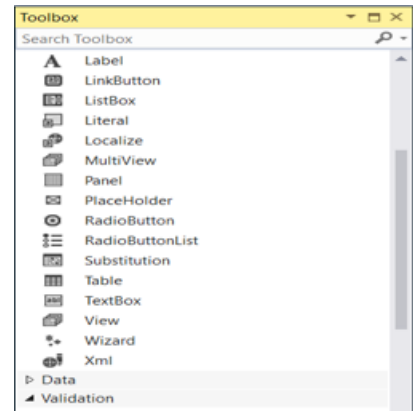
Neden Asp.Net Web Forms ?

ViewState Kullanımı

Web Forms için oldukça performans kaybı yaratabilen ViewState, doğru ve verimli kullanıldığında bu dezavantajı avantaja dönüştürülebilir.

Zengin Sunucu Kontrolleri

Web Forms'da büyük kolaylık sağlayan Asp.Net kontrolleri(button, textbox, checkbox, gridview, dropdownlist, listview...), çok büyük kolaylıklar sağlaması itibarı ile zor vazgeçilebilecek yapıların başında gelir. Javascript, Ajax gibi yapıları hazır olarak sunması, validation, editör gibi kullanıcıların sıklıkla kullandığı kontrolleri basit olarak sunması Web Forms'u zor vazgeçilir kılmaktadır. Birçok yazılımcı bu kontroller üzerinden işlem gerçekleştirdikleri için MVC teknolojisine soğuk bakmaktadır.



Olay Güdümlü Programlama (Event Driven Programming)

Asp.Net Web Forms teknolojisinin en önemli kolaylıklarından biri de olay güdümlü programlama yapısıdır. Geliştirici bir button'u işlevsel hale getirmek ve içerisinde bir kod bloğu çalıştırmak istediğinde tıklama olayı(click event)nın içerisine gitmesi yeterlidir. Ya da Bir listbox'da seçili elemanın bilgisini almak ve o an bir işlem yapmak istiyor ise seçili eleman değişme olayı(SelectedIndexChanged)na gidebilir.

```
<asp:Button ID="btnSave" runat="server" Text="Save" OnClick="btnSave_Click" />
```

```
public partial class About : Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }

    protected void btnSave_Click(object sender, EventArgs e)
    {

    }
}
```

Neden Asp.Net MVC ?

Performans

MVC teknolojisinde durum yönetimi otomatik olarak yapılmaz. ViewState kullanımı olmadığı için sayfa server'a giderken ve gelirken üzerinde herhangi bir kontrolün bilgisini ve diğer yük olacak verileri barındırmaz. Buna bağlı olarak da daha performanslı çalışır.

Yeniden Kullanılabilirlik (Reuseability)

Web Forms'da olay güdümlü programlama ve zengin kullanıcı kontrolleri faydalı görünse de MVC yapısına baktığımızda yeniden kullanılabilirlik açısından pek de faydalı değildir. Yazılan kodların yalnızca o sayfaya veya o kontrole özgü olması, her kontrol için ve her sayfa için ayrı ayrı kod yazmamızı gerektirir. MVC içerisinde böyle bir yapı bulunmadığından dolayı yazdığımız kodları birden fazla yerde kullanarak bir standart oluşturabilir, ileri tarihte projeyi geliştireceğimiz tüm süreçleri için önemli bir zaman kazanabiliriz.

Test Driven Development

Asp.Net MVC' de sorumlulukların ayrı olması prensibi itibari ile her katman ayrı bir iş gerçekleştirir. Böylece her işin kendine özgü test senaryoları ve süreçleri olabilir. Dolayısı ile proje Test Driven Development' a son derece uygun yapıdadır.

Yazılım Geliştirme Süresi

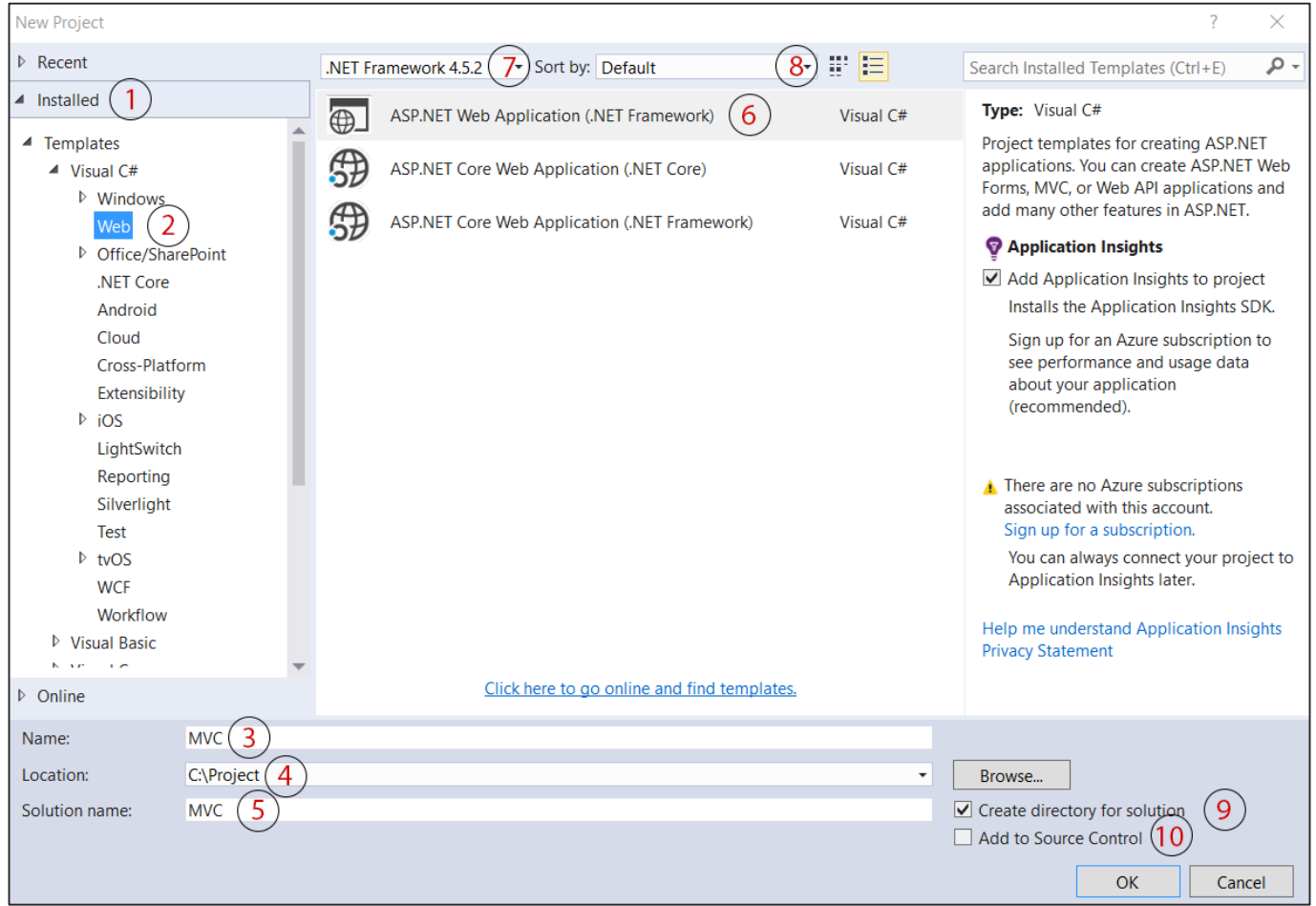
Bir projeyi birden fazla geliştirici üstlenebilir. MVC yapısında her katmanın birbirinden farklı sorumlulukları olduğundan aynı sürelerde farklı katmanlarda farklı geliştiriciler çalışabilir. Bu nedenle yazılım geliştirme süreci kat ve kat hızlı olacaktır.

İstemci Taraflı Geliştirme (Client-Side Development)

İstemci taraflı kontroller hazır olarak gelmediğinden tümü HTML, Javascript tabanlı yönetilir. Bu yapı itibarı ile sayfa sunucuya daha hızlı ve performanslı olarak taşınacaktır.

ASP.NET MVC PROJESİ OLUŞTURMA

Visual Studio içerisinde 2 seçenek üzerinden MVC projesi açılabilir. File menüsündeki New sekmesinden proje türü seçilir. Bu proje ekranından alt proje türleri arasından seçim yapılır ve aşama aşama MVC projemizin ayarlarını yaparak projeyi oluştururuz.



1.Şablon Türleri

Oluşturacak olduğumuz projenin hangi yazılım dili şablonunda olacağını seçtiğimiz alandır. Recent bölümünde son kullanılan şablonlar, Installed bölümünde yüklü şablonlar, Online bölümünde internetten şablon indirmemizi sağlayan seçenekler bulunmaktadır.

2.Proje Türü

Oluşturacağınız Proje türünü bu bölüm içerisinde filtreleyerek daha hızlı bir şekilde ulaşabilirsiniz

3.Name

Bu bölümde projenize bir isim vermeniz gerekmektedir.

4.Location

Proje dosyasının hangi bölümden açılacağını seçmeniz gerekmektedir

5.Solution Name

Eğer birden fazla katman ile çalışacaksanız burdaki bölümde gene kapsayıcı bir isim belirleyerek namespace mantığı ile çalışabilirsiniz

6.Proje Türü

Bu bölümden seçtiğiniz yazılım dili ve şablon üzerinden geliştirebileceğiniz proje türleri mevcuttur.

7.Framework Versiyonu

Oluşturacağımız projenin hangi Framework üzerinde çalışacağını belirlediğimiz alandır. Sistem üzerinde yüklü Framework'ler arasından seçim yapabiliriz.

8.Sıralama Şekli

Seçtiğimiz şablon türüne göre gelen proje türlerinin hangi sıraya göre listeleneceğini belirlediğimiz alandır. Default, Name Ascending, Name Descending seçenekleri üzerinden birini seçerek proje türlerini listeleyebiliriz.

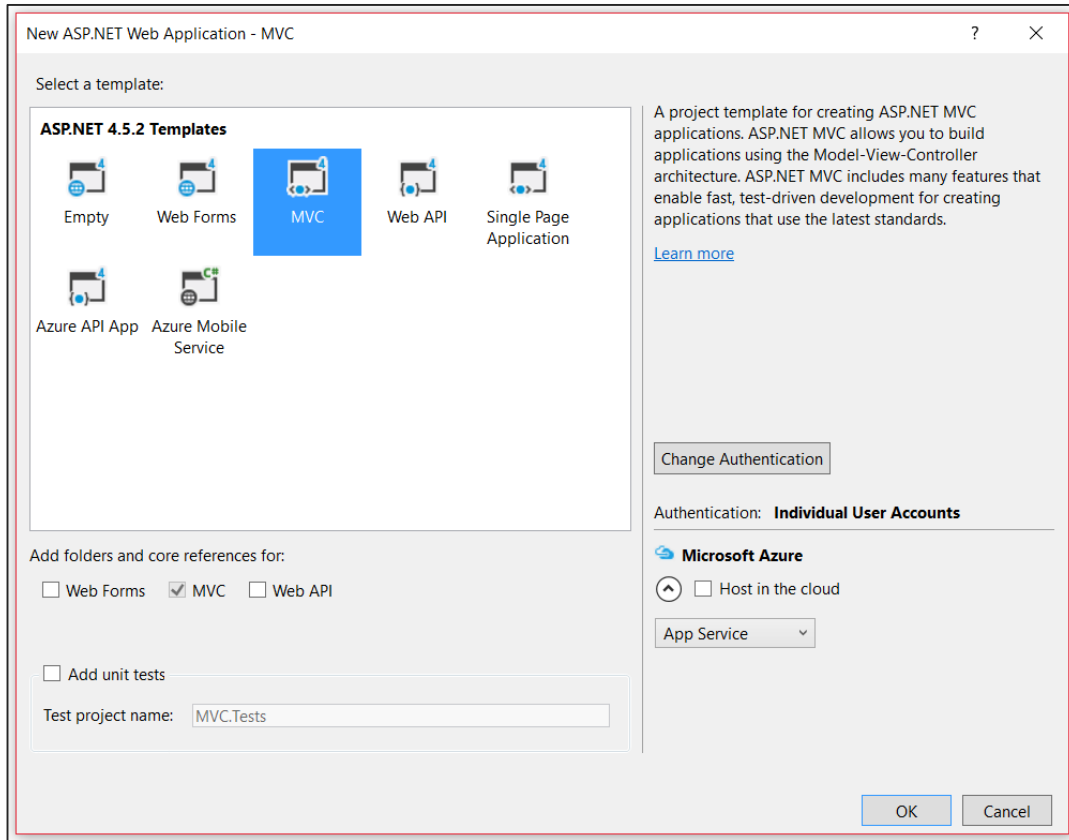
9.Create directory for solition

Bu bölüm seçili ise size Solition bölümünü ayrı oluşturmanıza izin verecektir, eğer Checkbox içerisinde işareti kaldırırsanız proje ismi ve solition ismi bir olacaktır.

10.Add to Source Control

Proje TFS (Team Foundation Server) ve ya Github üzerinde proje geliştirecekseniz. Burdaki alanı seçerek projenizi geliştirmeye devam edebilirsiniz.

Bu alandaki gerekli değişiklikleri ve seçimleri yaptıktan sonra, bir sonraki adıma geçip hangi Template üzerinden geliştirme yapacağımızı seçiyoruz.

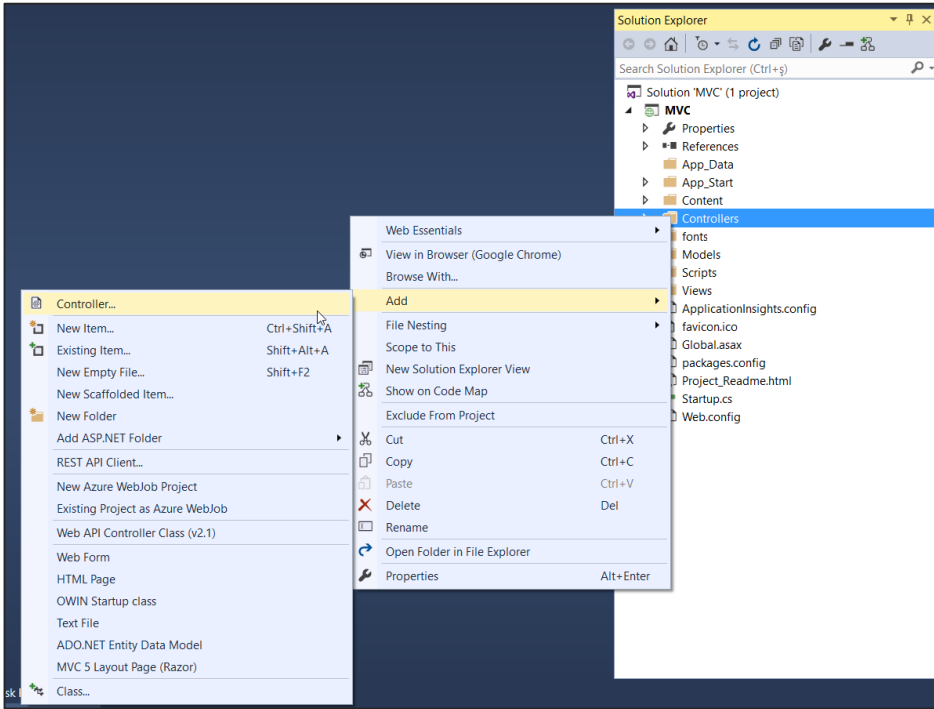


MODEL, VIEW, CONTROLLER OLUŞTURMA

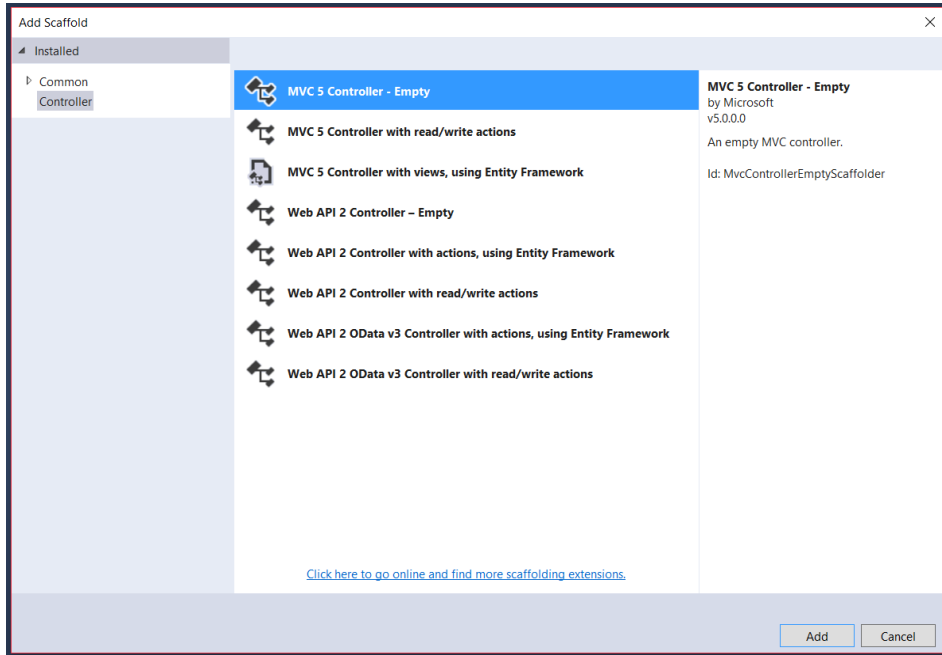
Model, View ve Controller' ın neler olduğunu ve ne amaçla kullanıldıkları hakkında detaylı bilgilere yer vermiştik. Bu bölümde bu üç yapının MVC projesinde nasıl oluşturulacaklarına yer vereceğiz.

Controller

Yeni bir Controller oluşturmak için Solution Explorer içerisinde bulunan Controller klasörüne sağ tıklayıp Add/Controller adımlarını izleyeceğiz.



Bu adımda Controller seçeneğini seçerek Projemize bir adet Controller ekliyoruz. Bir sonraki adım Controller Yapısı seçme.



Yukardaki resim içerisinde yer alan ilk 3 Controller yapısını bu bölüm içerisinde ele alacağız.

MVC 5 Controller – Empty :

Controller sınıfının kalıtımını alan bir Controller class'ı oluşturur. Önemli olan kıtas şudur ki Controller ismini verirken sonu mutlaka Controller ile bitmelidir. Örnek; HomeController. Ve içinde yalnızca Index adında bir ActionResult metodu oluşturur.

Add Controller

×

Controller name:

Add

Cancel

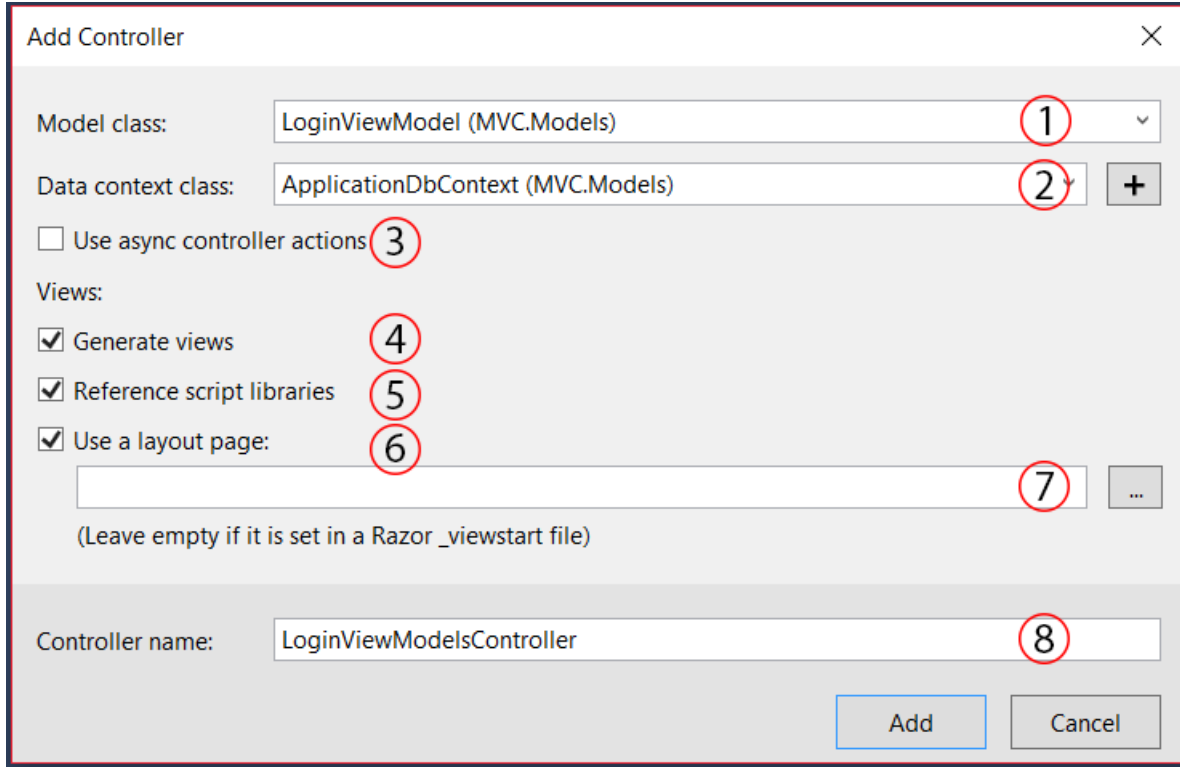
```
namespace MVC.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

MVC 5 Controller with read/write actions

Bu seçenekte oluşturduğunuz Controller çeşitli hazır yapılar ile gelecektir. Bunlar Index, Details, Create, Edit, Delete gibi Crud işlemlerini yapabileceğimiz metotlardır.

MVC 5 Controller with views, using Entity Framework

Database ile bağlantılı olan bir modelin tüm Crud işlemlerini (listele, detay, ekle, güncelle, sil) kodlanmış hali ile getiren yapıdır. Örneğin; müşteri bilgilerini tuttuğum bir Employee modelimiz olsun. Bu modelin tüm Crud işlemlerini kodlanmış halini tasarımı ile birlikte getirir. Bize sadece bu yapıyı kullanmak kalmış olur.



The image shows the 'Add Controller' dialog box in Visual Studio. It contains the following fields and options:

- Model class:** A dropdown menu showing 'LoginViewModel (MVC.Models)' with a red circle 1 next to it.
- Data context class:** A dropdown menu showing 'ApplicationDbContext (MVC.Models)' with a red circle 2 next to it.
- Use async controller actions:** An unchecked checkbox with a red circle 3 next to it.
- Views:**
 - Generate views:** A checked checkbox with a red circle 4 next to it.
 - Reference script libraries:** A checked checkbox with a red circle 5 next to it.
 - Use a layout page:** A checked checkbox with a red circle 6 next to it.
- Layout page:** A text box with a red circle 7 next to it, containing a red circle 7 next to it.
- Controller name:** A text box showing 'LoginViewModelsController' with a red circle 8 next to it.

At the bottom right, there are 'Add' and 'Cancel' buttons.

Açılan ekran içerisinde;

Model Seçimi

Oluşturduğunuz Controller için bir Model seçmeniz istenilir. Seçilen modele göre Tüm CRUD metotları hazırlanacaktır.

Context Seçimi

Seçilen Context Class'ına göre Veritabanı bağlantı işlemleri gerçekleştirilecektir.

Async Controller Action

Bu alan seçili olarak yapı oluşturulur ise, Controller içerisindeki tüm Action'lar Async olarak oluşturulacaktır.

Generate Views

Bu alan seçili olarak yapı oluşturulur ise, Controller içerisindeki tüm Action'ların View'ları Seçilen modele göre oluşturulacaktır.

Reference script libraries

İlgili alan seçilir ise ve Layout içerisinde default olarak eklenen script dosyaları yok ise, her bir view içerisinde bunlar render edilecektir.

Use a layout page

Bir layout (master page) kullanmak istemiyorsanız bu alanı seçimsiz bırakabilirsiniz.

Layout Seçimi

Custom olarak bir layout oluşturmadıysanız default olarak viewstart içerisinde tanımlı olan Layout seçili olarak gelecektir.

Controller Name

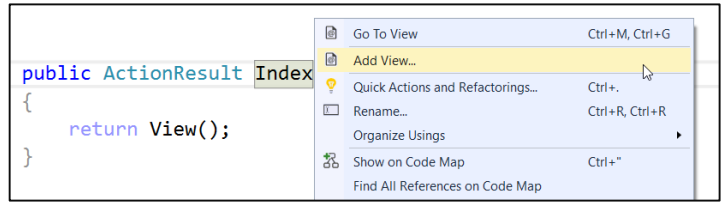
Seilen model'e gre custom bir controller name ataması yapılır. Farklı bir controller name seebilirsiniz.

Model

Proje ierisinde kullanılacak olan class'lar Model klasr ierisinde yer alır. Custom olarak oluřturacaėınız ya da **Edmx** dosyası olarak da ekleyedeėiniz Data blm bu alanda yer alacaktır.

Views

Controller ierisinde oluřturdumuz ActionResult metotları iin geriye dnř tipi olan View'ları eklemek iin farklı yntemler kullanabiliriz. İlk olarak Controller ierisindeki ActionResult'ın zerinden ekleme yntemini ele alalım. Saėdaki resimde bulunan iřlemlerden sonra Aılan ekran ierisinde;

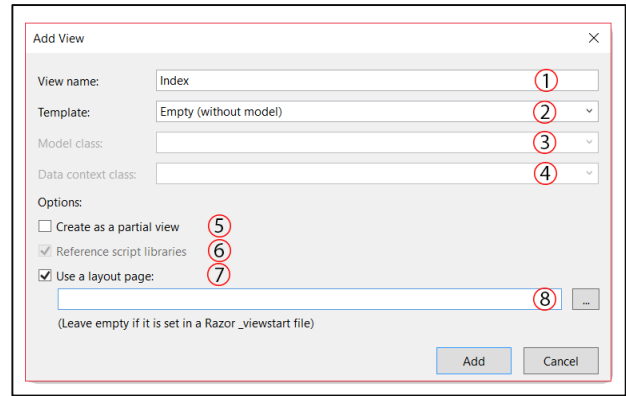


1.View Name

Oluruřturacaėınız View iin bir isim semeniz isteecektir. Bu alan ierisinde vereceėiniz isin Action ile aynı olmalıdır.

2.Template Seėimi

Oluruřturulan View'ın hangi iřlem iin oluruřturulacaėını belirten propertydir. View zerindeki html yapısına kolaylık saėlamak iin template seėimi yapmaktayız. Template detaylarını ařaėıdaki tablodan inceleyebilirsiniz.



Create :	Seėtiėiniz modele gre kayıt ekleme ekranı oluřturur
Delete :	Seilen Model iin kayıt silme ekranı tasarımı teslim eder
Details :	Model ierisindeki deėer iin detay sayfası tasarımı teslim eder.
Edit :	Seilen Model iin Dzenleme ekranı teslim eder
Emptyt :	Seilen Model iin , sayfa tasarımını kullanıcının oluřturabileceėi boř bir sayfa teslim eder
Empty (without model) :	Boř bir sayfa teslim eder.
List :	Seilen model iin listeleme sayfası teslim eder.

3.Model Class

Hazır Template kullanılacak ise, Bu alan ierisinden Model seilmesi gerekmektedir.

4.Data Context Class

Veri tabanı işlemleri için kullanılacak olan Context sınıfını seçmenizi ister.

5.Create as a partial view

WebForm'daki WebUserController'ün karşılığı olan Partial View(parçalı sayfa),sizden sayfanın normal bir web sayfasını yoksa PartialView olup olmadığını sorar

6.Reference script libraries

İlgili alan seçilir ise ve Layout içerisinde default olarak eklenen script dosyaları yok ise, her bir view içerisinde bunlar render edilecektir.

7.Layout Seçimi

Custom olarak bir layout oluşturmadıysanız default olarak viewstart içerisinde tanımlı olan Layout seçili olarak gelecektir.

8.Layout Seçimi

Kendi oluşturduğunuz bir Layout var ise bu alanı kullanarak seçebilirsiniz.

View eklemek için diğer bir yöntem ise, Views klasörü içerisinde, açtığınız Controller için kendi adında bir klasör yok ise manuel ekleyip içerisinde Add view sekmesini takip ederek gerçekleştirebilirsiniz.

Not : Eklediğini tüm view'ler kendi Controller isminde klasörler içerisinde yer almaktadır.

VIEW ENGINE NEDİR ?

Asp.Net MVC ile karşımıza çıkan bir yenilik de View Engine teknolojisidir. Web Form'da da gerekli bölgelerde kullandığımız bu yapı MVC ile birlikte olmazsa olmaz bir hale gelmiş bulunmaktadır. HTML içerisine yazdığımız C# kodlarını HTML olarak render eden, istemci taraflı ve sunucu taraflı kodları birlikte yazmamızı sağlayan yapıdır. Asp.Net MVC'de kullanılan View Engine'ler;

- Razor
- Web Form
- Spark
- Nhaln

MVC müfredat içeriğinde ki tüm işlemleri ve örnekleri Razor View Engine kullanarak gerçekleştireceğiz.

RAZOR VIEW ENGINE

Asp.Net MVC 3 ile hayatımıza giren Razor View Engine, View Engine'ler arasında en kullanışlı olanıdır. HTML ile C# kodlarını iç içe yazarken birbirinden rahatça ayırt edebileceğimiz, karmaşık yapıda olmayan View Engine'dir. Örneğin; tanımladığımız bir değişkeni HTML tagları ile kalın, italik yazdırabiliriz. Veya bir karar yapısında kontrol ederek kod bloklarının içerisine HTML kodları yazabiliriz.

@ OPERATÖRÜ

Asp.Net Web Forms'da <%....%> bu taglar arasına C# kodları yazabiliyorduk. Fakat bu kullanım bizlere karışık bir kod düzeni oluşturmaktaydı.

MVC ile Razor View Engine yapısını kullanmaya başladığımızda Client Side(İstemci Taraflı) kod yazmak bizler için daha rahat hale gelmiş bulunmaktadır

Razor View Engine ile Client Side tarafında C# kodlarının yazımını "@" operatörü ile sağlamaktayız. Böylece HTML tagları içerisinde rahat bir şekilde C# kodları yazabilir ve istediğimiz tüm işlemleri karmaşadan kurtarabiliriz.

Razor View Engine ile Kod Örneği	Ekran Çıktısı
<pre>@{ string companyName = "Bilge Adam"; } <div class="jumbotron" > <h1>@companyName</h1> </div></pre>	

Razor View Engine içerisinde, karar yapıları, döngüler gibi yapıları kolaylıkla kullanabiliriz. View Engine kullanımı ile ilgili birkaç örnek inceleyelim.

Razor View Engine ile View içerisinde Döngü Kullanım Örneği

```
@model List<MVC.Models.Category>
<div class="row">
    <div class="panel panel-default">
        <div class="panel-body">
            <table class="table table-hover">
                <thead>
                    <tr>
                        <td>Kategori Id</td>
                        <td>Kategori Adı</td>
                        <td>Açıklama</td>
                    </tr>
                </thead>
                <tbody>
                    @foreach (var category in Model)
                    {
                        <tr>
                            <td>category.CategoryId</td>
                            <td>category.CategoryName</td>
                            <td>category.Description</td>
                        </tr>
                    }
                </tbody>
            </table>
        </div>
    </div>
</div>
```

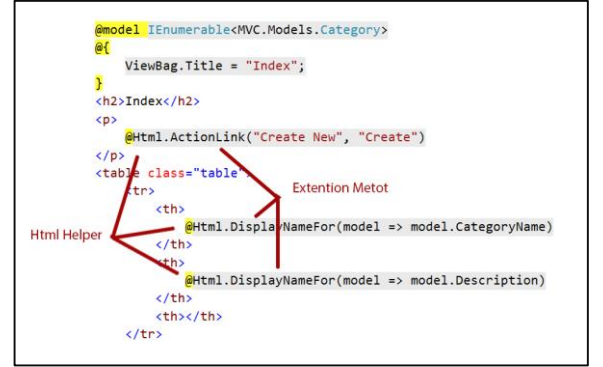
Yukarıdaki düzenlemeyi yaptıktan sonra projemizi çalıştırıyoruz. Sayfamız açıldığında resimdeki sonucu elde edebiliriz.

Kategori Id	Kategori Adı	Açıklama
1	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
3	Confections	Desserts, candies, and sweet breads
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, crackers, pasta, and cereal
6	Meat/Poultry	Prepared meats
7	Produce	Dried fruit and bean curd
8	Seafood	Seaweed and fish

HTML HELPERS

HtmlHelper sınıfı, metot görünümünde model sınıf nesnesini kullanarak html öğeleri üretir. Model nesnesini model öğelerinin değerlerini html öğelerine görüntülemek için html öğelerine bağlar ve web formu gönderirken html öğelerinin değerini de model özelliklerine atar. HtmlHelper sınıfını elle html etiketleri yazmak yerine metot görünümünde kullanın. Sağdaki resim, HtmlHelper sınıfının metot görünümündeki kullanımını göstermektedir.

Aşağıdaki tabloda Html Helpers metotlarının html karşılıklarını inceleyebilirsiniz.



HtmlHelper	Strogly Typed HtmlHelpers	Html Control
Html.ActionLink		Anchor link
Html.TextBox	Html.TextBoxFor	Textbox
Html.TextArea	Html.TextAreaFor	TextArea
Html.CheckBox	Html.CheckBoxFor	Checkbox
Html.RadioButton	Html.RadioButtonFor	Radio button
Html.DropDownList	Html.DropDownListFor	Dropdown, combobox
Html.ListBox	Html.ListBoxFor	multi-select list box
Html.Hidden	Html.HiddenFor	Hidden field
Password	Html.PasswordFor	Password textbox
Html.Display	Html.DisplayFor	Html text
Html.Label	Html.LabelFor	Label
Html.Editor	Html.EditorFor	Generates Html controls based on data type of specified model property e.g. textbox for string property, numeric field for int,

double or other numeric type.

HtmlHelper

TextBox Oluşturma

HtmlHelper sınıfı, bir textbox() oluşturan iki uzantı yöntemini içerir. TextBox() ve TextBoxFor(). TextBoxFor () kesinlikle yazılmış bir yöntem iken TextBox () yöntemi basit biçimde yazılmıştır. Aşağıdaki Student modeli ile TextBox () ve TextBoxFor () yöntemlerini kullanacağız.

Örnek : Student Model

```
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int Age { get; set; }
    public string Password { get; set; }
}
```

TextBox()

Html.TextBox () yöntemi, belirtilen ad, değer ve html nitelikleriyle <input type = "text"> ögesi oluşturur.

Textbox metodu için kullanılacak metod imzası

MvcHtmlString Html.TextBox(string name, string value, object htmlAttributes)

Örnek : Html.TextBox() – Razor View

```
@model Student
@Html.TextBox("FirstName", null, new { @class = "form-control" })
```

Html Çıktısı

```
<input class="form-control" id="FirstName" name="FirstName" type="text" value=""/>
```

Yukarıdaki örnekte, ilk parametre öğrenci model sınıfının "ÖğrenciAdı" özelliğidir ve metin kutusunun bir adı ve kimliği olarak ayarlanır. İkinci parametre, TextBox () yöntemi otomatik olarak metin kutusundaki StudentName özelliğinin değerini görüntüleyeceği için yukarıdaki örnekte boş olan bir metin kutusuna görüntülenecek bir değerdir. Üçüncü parametre sınıf niteliği olarak ayarlanacaktır. HtmlAttributes parametresi bir nesne türü, bu nedenle anonim nesne olabilir ve nitelikler adı, @ işareti ile başlar.

Örnek : Html.TextBox() - Razor View

```
@Html.TextBox("FirstName", "Öğrenci Adı", new { @class = "form-control" })
```

Html Çıktısı

```
<input class="form-control" id="FirstName" name="FirstName" type="text" value="Öğrenci Adı" />
```


TextBoxFor

TextBoxFor yardımcı yöntemi kesinlikle yazılmış bir uzantı yöntemidir. Bir lambda ifadesi kullanılarak belirtilen model özelliği için bir metin girdi ögesi üretir. TextBoxFor yöntemi, belirtilen bir model nesnesi mülkiyetini girdi metnine bağlar.

TextBoxFor() method Signature

MvcHtmlString TextBoxFor(Expression<Func<TModel,TValue>> expression, object htmlAttributes)

Örnek : Html.TextBoxFor() – Razor View

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

Html Çıktısı

```
<input class="form-control" id="FirstName" name="FirstName" type="text" value="Öğrenci Adı" />
```

Custom Helper

Helper sınıfının yetmediği yani yetersiz kalığı durumlarda kendi ihtiyaçlarımız için Custom olarak helper'lar oluşturabiliriz. Aşağıdaki örnek içerisinde normal string olarak geriye döndürülmüştür.

Örnek : CustomTextBox() Code

```
public string CustomTextBox(string name)
{
    return $"<input type='text' name='{name}' id='{name}' class='form - control' />";
}
```

Metot geriye string olarak döndüğünden dolayı, sayfa çıktısı olarak bir input değil yazı olarak size teslim edecektir.

String değer olarak hazırlana Html kodlarını render edip html kontrolü olarak kullanabilmemiz için Html.Raw metodundan yararlanıyoruz.

Örnek : CustomTextBox() – Razor View

```
@model Student
@Html.Raw(CustomTextBox("FirstName"))
```

Html Çıktısı

```
<input type="text" name="FirstName" id="FirstName" class="form-control" />
```

Custom Html Helper oluştururken, Geriye dönüş tipini MvcHtmlString olarak belirlersek, herhangi bir render işlemine gerek duymadan kontrollerimizi kullanabiliriz.

Örnek : CustomTextBox() Code

```
public static MvcHtmlString CustomTextBox(this HtmlHelper helper, string name)
{
```

```
var builder = new TagBuilder("input");
builder.Attributes.Add("type", "text");
builder.Attributes.Add("id", name);
builder.Attributes.Add("name", name);
return MvcHtmlString.Create(builder.ToString(TagRenderMode.SelfClosing));
}
```

Örnek : CustomTextBox() – Razor View

```
@model Student
@Html.CustomTextBox("FirstName")
```

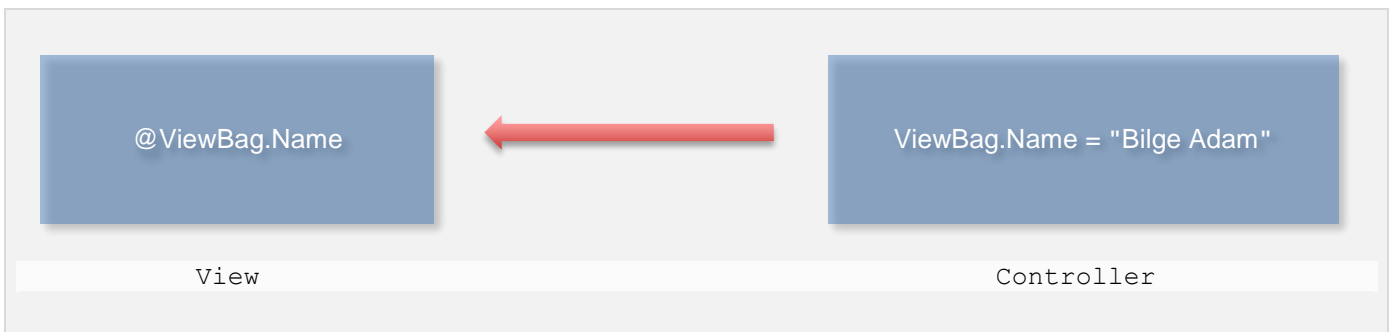
Html Çıktısı

```
<input type="text" name="FirstName" id="FirstName" class="form-control" />
```

Data Transfer To View

ViewBag

ViewBag, denetleyiciden görünüm için geçici verileri (modelde bulunmayan) aktarmak istediğinizde kullanışlı olabilir. ViewBag, tüm denetleyicilerin temel sınıfı olan ControllerBase sınıfının dinamik bir türü özelliğidir.



ViewBag'e Action içerisinde değer atama

```
namespace MVC.Controllers
{
    public class StudentController : Controller
    {
        Student student new Student()
        {
            StudentID=1, StudentName="Bilge Adam", Age = 20
        }

        public ActionResult Index()
        {
            ViewBag.TotalStudents = StudentName;

            return View();
        }
    }
}
```

Örnek : View içerisinde ViewBag'e Erişim

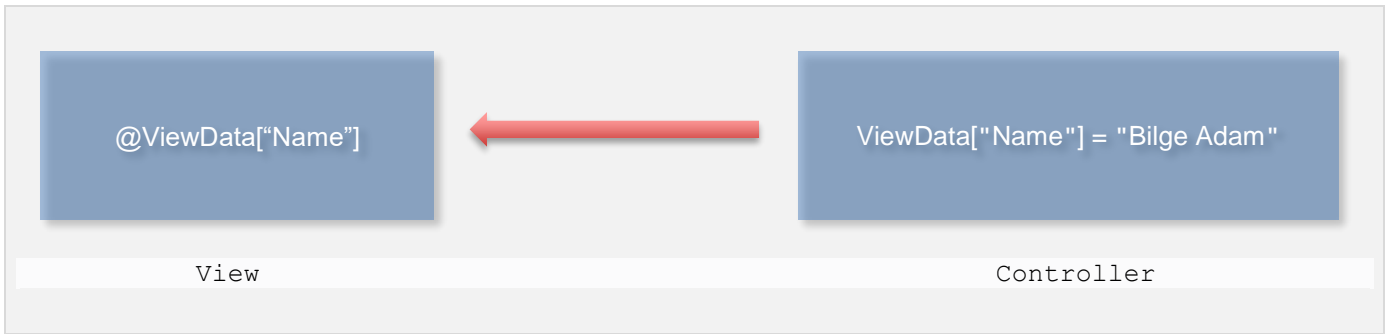
```
<label>Students: </label> @ViewBag.StudentName
```

Bilinmesi Gerekenler :

1. ViewBag, verileri denetleyiciden görünüme, ideal olarak bir modele dahil olmayan ideal geçici verilere aktarır.
2. ViewBag, C # 4.0'daki yeni dinamik özelliklerden yararlanan dinamik bir özelliktir.
3. ViewBag'a herhangi bir sayıda özellik ve değer atayabilirsiniz ViewBag'ın ömrü, yalnızca geçerli http isteği süresince sürer.
4. Yönlendirme gerçekleşirse, ViewBag değerleri boş olur.
5. ViewBag aslında ViewData çevresindeki bir sarmalayıcıdır.

ViewData

ViewData ViewBag'a benzer. Denetleyiciden Görünüm'e veri aktarımında yararlıdır. ViewData, her anahtarın dize olması gereken anahtar / değer çifti içerebilen bir sözlüktür.

**Örnek : ViewData'ya Action içerisinde değer atama**

```
namespace MVC.Controllers
{
    public class StudentController : Controller
    {
        public ActionResult Index()
        {
            IList<Student> studentList = new List<Student>();
            studentList.Add(new Student(){ StudentName = "Bilge Adam 1" });
            studentList.Add(new Student(){ StudentName = "Bilge Adam 2" });
            studentList.Add(new Student(){ StudentName = "Bilge Adam 3" });

            ViewData["students"] = studentList;

            return View();
        }
    }
}
```

Örnek : View içerisinde ViewData'ya ulaşmak

```
<ul>
    @foreach (Student student in ViewData["students"] as IList<Student>)
    {
        <li>
```

```
@ student.StudentName
</li>
}
</ul>
```

Örnek : KeyValuePair Kullanımı

```
public ActionResult Index()
{
    ViewData.Add("Id", 1);
    ViewData.Add(new KeyValuePair<string, object>("Name", "Bilge Adam"));
    ViewData.Add(new KeyValuePair<string, object>("Age", 20));

    return View();
}
```

Bilinmesi Gerekenler

- ViewData, veriyi Controller'den View'e aktarır, tersi değil.
- ViewData bir sözlük türü olan ViewDataDictionary'den türetilmiştir.
- ViewData'nın ömrü yalnızca geçerli http isteği süresince sürer. Yönlendirme gerçekleşirse ViewData değerleri silinir.
- ViewData değeri kullanılmadan önce cast edilmelidir.
- ViewBag dahili olarak ViewData sözlüğüne veri ekler. Dolayısıyla ViewData ve ViewBag özelliğinin anahtarları eşleşmemelidir.

TempData

ASP.NET MVC TempData, sonraki istekte kullanılabilecek geçici verileri depolamak için kullanılabilir. TempData, sonraki bir talebin tamamlanmasından sonra silinir. TempData, hassas olmayan verileri bir işlem yönteminden aynı ya da farklı bir denetleyicinin başka bir eylem yöntemine aktarmak istediğinizde ve yeniden yönlendirme yaparken yararlıdır. TempDataDictionary'ten türetilen sözlük türüdür.

Örnek : TempData

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        TempData["name"] = "Bilge Adam ";
        TempData["age"] = 20;

        return View();
    }

    public ActionResult About()
    {
        string userName;
        int userAge;

        if(TempData.ContainsKey("name"))
            userName = TempData["name"].ToString();

        if(TempData.ContainsKey("age"))
            userAge = int.Parse(TempData["age"].ToString());

        return View();
    }
}
```

Yukarıdaki örnekte, TempData'ya veri ekledik ve aynı veriye başka bir eylem yöntemi içinde bir anahtar kullanarak eriştik. TempData object olarak tuttuğundan dolayı veriyi Convert ve ya Cast etmek durumundayız.

TempData kullanırken ilk olarak TempData'nın tanımlandığı Action'ın derlenmesi gerekmektedir. Sonrasında diğer View'lar içerisinde veriye ulaşabilirsiniz.

Örnek : TempData

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        TempData["name"] = "Bilge Adam";
        return View();
    }

    public ActionResult About()
    {
        string data;

        if(TempData["name "] != null)
            data = TempData["name "] as string;

        TempData.Keep();

        return View();
    }

    public ActionResult Contact()
    {
        string data;

        if(TempData["name "] != null)
            data = TempData["name "] as string;

        return View();
    }
}
```

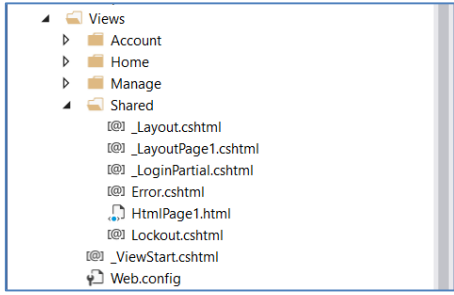
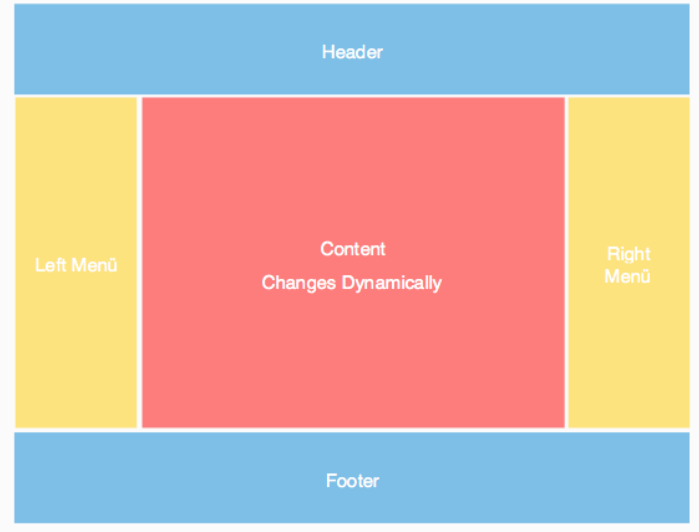
Bilinmesi Gerekenler :

1. TempData iki ardışık istek arasındaki verileri depolamak için kullanılabilir. TempData değerleri yönlendirme sırasında korunur.
2. TempData bir TempDataDictionary türüdür.
3. TempData dahili olarak verileri depolamak için Oturumu kullanır. Dolayısıyla, kısa ömürlü bir oturum olarak düşünün.
4. Kullanmadan önce TempData değeri döküm olmalıdır. Çalışma zamanı hatasından kaçınmak için boş değerleri kontrol edin.
5. TempData hata iletileri, doğrulama iletileri gibi yalnızca bir kez iletileri depolamak için kullanılabilir.
6. TempData'nın tüm değerlerini üçüncü bir isteğin içinde tutmak için TempData.Keep() ögesini çağırın.

Layout View

Bir uygulama, kullanıcı arayüzünde logo, üstbilgi, sol gezinme çubuğu, sağ çubuk veya altbilgi bölümü gibi uygulama boyunca aynı kalan ortak parçaları içerebilir. ASP.NET MVC, bu ortak UI parçalarını içeren bir düzen görünümü başlattı, böylece her sayfada aynı kodu yazmamız gerekmez. Düzen görünümü, ASP.NET webform uygulamasının ana (master page) sayfasıyla aynıdır. Örneğin, bir uygulama kullanıcı arayüzü, her sayfada aynı kalacak Üstbilgi, Sol menü çubuğu, sağ çubuk ve altbilgi bölümünü içerebilir ve yalnızca orta bölüm dinamik olarak aşağıda gösterildiği gibi değişir.

Düzen görünümü, bir uygulamanın birden fazla sayfasında tutarlı bir görünüm ve his sağlamak için birden fazla görünümde devralınabilen ortak bir site şablonu tanımlamanıza izin verir.



Düzen görünümü, yinelenen kodlamayı ortadan kaldırır ve geliştirme hızını ve kolay bakımını geliştirir. Yukarıdaki örnek kullanıcı arayüzü için düzen görünümü, Başlık, Sol Menü, Sağ Çubuk ve Altbilgi bölümlerini içerir.

Tarayıcı düzeni görünümü, diğer görünüm, .cshtml veya .vbhtml ile aynı uzantıya sahiptir. Sabit görünüm, birden çok görünümle paylaşılır, bu nedenle Paylaşımlı klasörde saklanması gerekir. MVC uygulaması içerisinde oluşturduğumuzda, aşağıda gösterildiği gibi _Layout.cshtml'yi Paylaşımlı klasörde de oluşturur.

Örnek Layout Sayfası Tasarımı

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>@ViewBag.Title - My ASP.NET Application</title>
  <link href="~/Content/bootstrap.css" rel="stylesheet" />
</head>
<body>
  <div class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
      <div class="navbar-header">
        <button type="button" class="navbar-toggle" data-toggle="collapse" data-
target=".navbar-collapse">
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
        </button>
      </div>
    </div>
  </div>
</body>
</html>
```

```
@Html.ActionLink("Application name", "Index", "Home", new { area = "" }, new {
@class = "navbar-brand" })
</div>
<div class="navbar-collapse collapse">
    <ul class="nav navbar-nav">
        <li>@Html.ActionLink("Home", "Index", "Home")</li>
    </ul>
</div>
</div>
</div>
<div class="container body-content">
    @RenderBody()
    <hr />
    <footer>
        <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
    </footer>
</div>
</body>
</html>
```

Partial View

Kısmi görünüm, çoklu diğer görünümelerde bir alt görünüm olarak kullanılabilen yeniden kullanılabilir bir görünümdür. Aynı kısmi görünümü birden fazla yerde tekrar kullanarak yinelenen kodlamayı ortadan kaldırır. Yerleşim görünümünde kısmi görünümü ve diğer içerik görünümelerini kullanabilirsiniz.

Herhangi bir model için kayıt ekranı hazırladığımızda, aynı model için birde düzenleme ekranı hazırlamamız gerekmektedir. Bu gibi durumlarda birbirini tekrar eden tasarımlar yerine PartialView içerisinde bunları tekilleştirerek tekrar tekrar kullanabiliriz.

Örnek : Kategori Classı

```
public partial class Category
{
    public int CategoryID { get; set; }
    public string CategoryName { get; set; }
    public string Description { get; set; }
}
```

Örnek : Kategori Ekleme Ekranı

```
@model MVC.Models.Category

@{
    ViewBag.Title = "Create";
}
<h2>Create</h2>
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Category</h4>
        <hr />
```

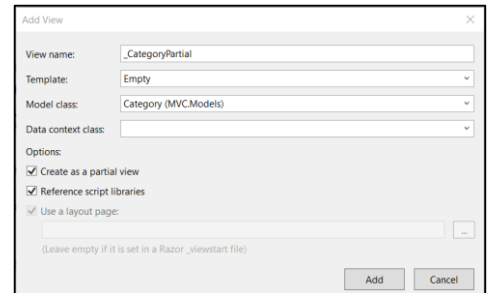
```
@Html.ValidationSummary(true, "", new { @class = "text-danger" })
<div class="form-group">
    @Html.LabelFor(model => model.CategoryName, htmlAttributes: new { @class = "control-label col-md-2" })
    <div class="col-md-10">
        @Html.EditorFor(model => model.CategoryName, new { htmlAttributes = new { @class = "form-control" } })
        @Html.ValidationMessageFor(model => model.CategoryName, "", new { @class = "text-danger" })
    </div>
</div>
<div class="form-group">
    @Html.LabelFor(model => model.Description, htmlAttributes: new { @class = "control-label col-md-2" })
    <div class="col-md-10">
        @Html.EditorFor(model => model.Description, new { htmlAttributes = new { @class = "form-control" } })
        @Html.ValidationMessageFor(model => model.Description, "", new { @class = "text-danger" })
    </div>
</div>
<div class="form-group">
    <div class="col-md-offset-2 col-md-10">
        <input type="submit" value="Create" class="btn btn-default" />
    </div>
</div>
</div>
}
<div>
    @Html.ActionLink("Back to List", "Index")
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

Yukarıdaki tasarım kategori eklemek için oluşturulmuştur.

Düzenleme işlemi için de aynı tasarım oluşturulması yerine partial view kullanarak aynı alanların tekrar edilmesini engelleyebilirsiniz.

Views klasörü içerisindeki Category klasörüne yeni bir partial ekliyoruz.

The image shows a 'Add View' dialog box in a web application. It has fields for 'View name' (set to '_CategoryPartial'), 'Template' (set to 'Empty'), 'Model class' (set to 'Category (MVC.Models)'), and 'Data context class' (empty). Under 'Options', there are three checked boxes: 'Create as a partial view', 'Reference script libraries', and 'Use a layout page'. There is a text box for 'Use a layout page' with a minus sign button. At the bottom, there are 'Add' and 'Cancel' buttons.

Örnek : Kategori Ekleme Ekranı

```
@model MVC.Models.Category

@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Category</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
    </div>
}
```



```

        <div class="form-group">
            @Html.LabelFor(model => model.CategoryName, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.CategoryName, new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.CategoryName, "", new { @class = "text-danger" })
            </div>
        </div>
        <div class="form-group">
            @Html.LabelFor(model => model.Description, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Description, new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Description, "", new { @class = "text-danger" })
            </div>
        </div>
        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Create" class="btn btn-default" />
            </div>
        </div>
    </div>
}

```

Oluşturduğumuz `_partial` içerisine tekrar eden tasarımı ekliyoruz. Ve sonra sayfadan kesip aldığımız olan bölüme `Html.Helper` sınıfı içerisinde yer alan `Html.Partial()` metodu ile, oluşturduğumuz partial View'ı ekliyoruz.

Örnek : Kategori Ekleme Ekranı

```

@model MVC.Models.Category

@{
    ViewBag.Title = "Create";
}
<h2>Create</h2>

@Html.Partial("_categoryPartial")

<div>
    @Html.ActionLink("Back to List", "Index")
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}

```

Render Partial View

Kısmi görünümü, üst düzey görünümde html yardımcı yöntemlerini kullanarak işleyebilir: `Partial ()` veya `RenderPartial ()` veya `RenderAction ()`.

Her yöntem farklı amaçlara hizmet eder. Her bir yöntem hakkında genel bir bilgiye sahip olmak ve bu yöntemleri kullanarak kısmi görünümü nasıl oluşturacağımıza bakalım.

Html.Partial()

@ `Html.Partial ()` yardımcı yöntemi belirtilen kısmi görünümü oluşturur. Kısmi görünüm adını bir dize parametresi olarak kabul eder ve `MvcHtmlString` değerini döndürür. Oluşturmadan önce html'yi modifiye etme şansınız olması için html dizesini döndürür. Aşağıdaki tabloda, Kısmi yardımcı yöntemin aşırı yükleri listelenmiştir:

Helper Metot	Açıklama
<code>MvcHtmlString Html.Partial(string partialViewName)</code>	Belirtilen kısmi görüntü içeriğini, gönderme görünümünde döndürür.
<code>MvcHtmlString Html.Partial(string partialViewName, object model)</code>	Gönderilen viewModel parametresindeki kısmi görüntü içeriğini, model nesnesini kısmi görünüme geçirir.
<code>MvcHtmlString Html.Partial(string partialViewName, ViewDataDictionary viewData)</code>	Kısmi görüntü içeriğini gönderme görünümünde döndürür. Veri parametresini görüntüleme, veri sözlüğünü kısmi görünüme geçirir.
<code>MvcHtmlString Html.Partial(string partialViewName, object model, ViewDataDictionary viewData)</code>	Kısmi görüntü içeriğini gönderme görünümünde döndürür. Model parametresi model nesnesini iletir ve Görünüm verileri, veri sözlüğünü kısmi görünüme geçirir.

Html.RenderPartial()

`RenderPartial` yardımcı yöntemi kısmi yöntemle aynıdır; bunun dışında, void döndürür ve sonuçta belirtilen kısmi görünümün html'sini doğrudan bir http yanıt akışına yazar.

Helper Metot	Açıklama
<code>RenderPartial(String partialViewName)</code>	Belirtilen kısmi görünümü döndürür
<code>RenderPartial(String partialViewName, Object model)</code>	Belirtilen kısmi görünümü döndürür ve belirtilen model nesnesini ayarlar
<code>RenderPartial(String partialViewName, ViewDataDictionary viewData)</code>	Belirtilen kısmi görünümü, ViewData mülkünü belirtilen ViewDataDictionary nesnesi ile değiştirerek render eder.
<code>RenderPartial(String partialViewName, Object model, ViewDataDictionary viewData)</code>	Kısmi Görünüm ViewData özelliğini belirtilen ViewDataDictionary nesnesiyle değiştirerek belirtilen kısmi görünümü döndürür ve belirtilen model nesnesini ayarlar

Html.RenderAction()

`RenderAction` yardımcı yöntemi, belirtilen bir denetleyiciyi ve eylemi çağırır ve sonucu kısmi bir görünüm olarak işler. Belirtilen Action yöntemi, Kısmi () yöntemini kullanarak `PartialViewResult` ögesini döndürmelidir.

Ad	Açıklama
----	----------

RenderAction(String actionName)	Belirtilen alt öge eylemi yöntemini çağırır ve sonucu üst görünümde oluşturur.
RenderAction(String actionName, Object routeValue)	Belirtilen alt eylem yöntemini belirtilen parametreleri kullanarak çağırır ve sonucu üst görünüme satır içi olarak işler.
RenderAction(String actionName, String controllerName)	Belirtilen denetleyici adını kullanarak belirtilen alt eylem yöntemini çağırır ve sonucu üst görünüme satır içi yapar.
RenderAction(String actionName, RouteValueDictionary routeValues)	Belirtilen alt eylem yöntemini belirtilen parametreleri kullanarak çağırır ve sonucu üst görünüme satır içi yapar.
RenderAction(String actionName, String controllerName, Object routeValue)	Belirtilen alt eylem yöntemini belirtilen parametreleri ve denetleyicinin adını kullanarak çağırır ve sonucu üst görünüme satır içi olarak işler.
RenderAction(String actionName, String controllerName, RouteValueDictionary routeValues)	Belirtilen alt eylem yöntemini belirtilen parametreleri ve denetleyicinin adını kullanarak çağırır ve sonucu üst görünüme satır içi olarak işler.

Örnek : Html.Partial()

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

Örnek : Html.RenderPartial()

```
@{
    Html.RenderPartial("_CategoryPartial")
}
```

Örnek : Html.RenderAction()

```
@Html.RenderAction("_CategoryPartial")
```

Section

Asp.Net MVC yapısında Layout Page yapısında RenderBody() metodu ile belirlediğimiz değişecek alan bizlere yalnızca tek bir alan sağlamaktadır.

Sayfa içerisinde eğer birden fazla değişecek alan istiyor isek bunu nasıl sağlayacağız? Bu soruyu sordüğümüzda karşımıza "Layout Section" yapısı çıkacaktır.

Bu yapı Layout Section olarak belirlediğimiz farklı bölgelerde değişiklikler yapmamızı sağlar. Örneğin sitenin başlığı ve sitenin içerik alanı farklılık gösterecek alanlar olacaktır. Bu durumda iki adet değişecek alan bulundurmamız gerekeceği için Layout Section yapısını kullanacağız.

Örnek : Section() - _Layout

```
<div class="container body-content">
    @RenderBody()
    <hr />
    <footer>
        <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
    </footer>
</div>
@RenderSection("scripts", required: false)
```

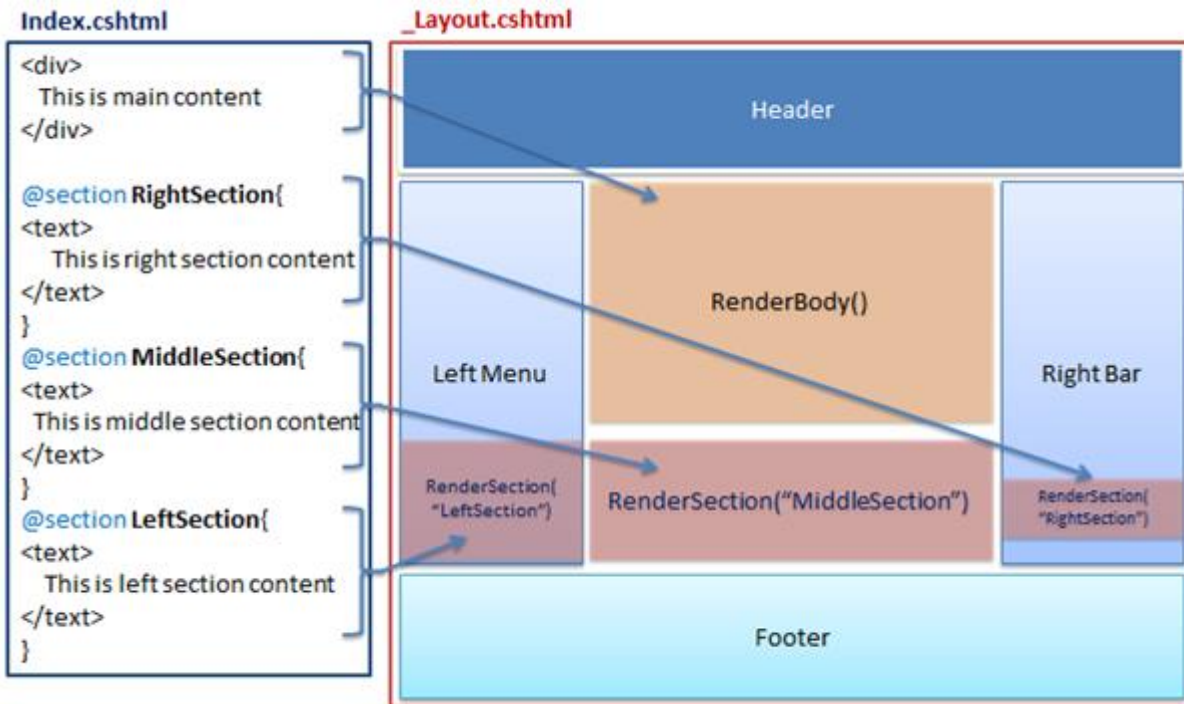
RenderSection içerisinde birinci parametrede ulaşmamız için bir isim belirlememizi ister.

İkinci parametrede ise, required alanında eğer true verilir ise bu layout'dan beslenen tüm sayfalarda tanımlanması zorunlu olarak belirlenir. Eğer false olarak verilir ise opsiyonel olarak atanması yapılır.

Örnek : Section() - View

```
@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

Sayfa içerisinde herhangi bir alan içerisinde tanımladığınız Section'ı sayfa derlendiğinde hangi alanda yer almasını istiyorsanız yapmanız gereken Layout içerisinde tanımlamasını gerçekleştirmeniz gerekmektedir.

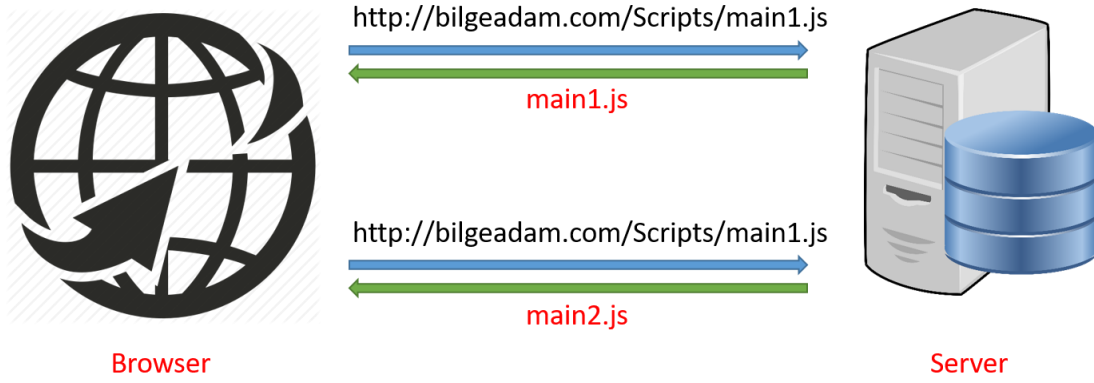


App_Start

Aşağıdaki configuration dosyaları, Global.asax içerisinde App_Start altında register edilmektedir.

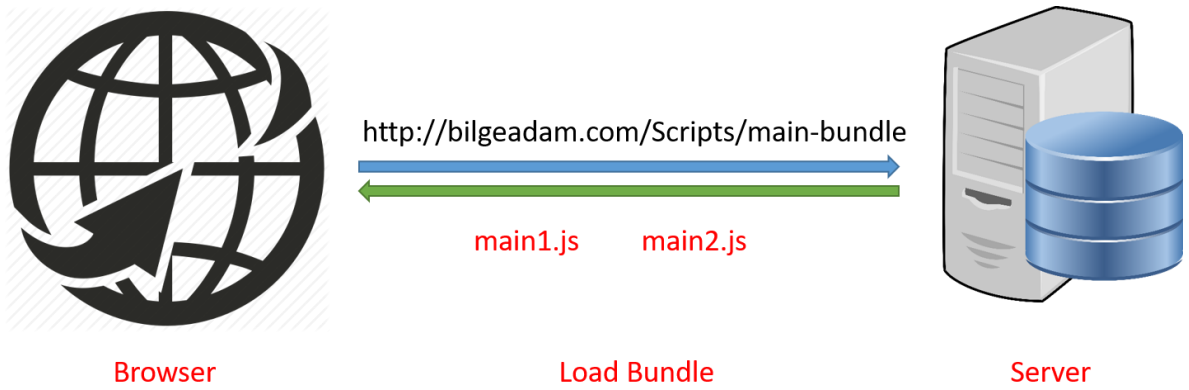
BundleConfig

İstek yükleme süresini iyileştirmek için MVC 4'te toplama ve indirme teknikleri kullanıldı. Paketleme, sunucudaki statik dosyaları bir http isteğine yüklememize izin verir.



Yukarıdaki şekilde tarayıcı, iki farklı JavaScript dosyası olan `main.js` ve `main2.js`'yi yüklemek için iki ayrı istek gönderir.

MVC 4'te paketleme tekniği, aşağıda gösterildiği gibi bir istekte birden fazla JavaScript dosyası, `main1.js` ve `main2.js`'yi yüklememize izin verir.



Minification

Minification tekniği, gereksiz boşluk ve açıklamaları kaldırarak ve değişken adları bir karaktere kıskarak komut dosyası veya css dosya boyutunu optimize eder.

Örnek : JavaScript

```
sayHello = function(name){  
    //this is comment  
    var msg = "Hello" + name;  
    alert(msg);  
}
```

Örnek : Minified JavaScript

```
sayHello=function(n){var t="Hello"+n;alert(t)}
```

Yukarıda da görebileceğiniz gibi, gereksiz boşlukları, açıklamaları kaldırdı ve değişken adları kısaltarak karakterleri azaltarak JavaScript dosyasının boyutunu azaltacak. Paketleme ve küçültme sayfanın yüklenmesi üzerindeki etkisini, dosyanın boyutunu ve isteklerin sayısını en aza indirgeyerek sayfayı daha hızlı yükler.

Bundle Types

MVC 5 System.web.Optimization ad alanında aşağıdaki paket sınıflarını içerir:

ScriptBundle : ScriptBundle, tekli veya çoklu komut dosyaları için JavaScript'in küçültülmesinden sorumludur.

StyleBundle : StyleBundle, tekli veya çoklu stil sayfası dosyalarının CSS minifikasyonundan sorumludur.

DynamicFolderBundle : ASP.NET'in aynı türdeki dosyaları içeren bir klasörden oluşturduğu bir Bundle nesnesini temsil eder.

Toplama ve Minification, statik betiği veya css dosyalarını en aza indirir, böylece yükleme süresi en aza indirir. MVC çerçevesi ScriptBundle, StyleBundle ve DynamicFolderBundle sınıfları sağlar. ScriptBundle, JavaScript dosyalarının küçültülmesini sağlar. StyleBundle, CSS dosyalarının küçültülmesini sağlar.

ScriptBundle in ASP.NET MVC

ASP.NET MVC API, JavaScript küçültme ve paketleme yapan ScriptBundle sınıfını içerir. Burada, tek bir http isteğinde birden çok JavaScript dosyasının nasıl oluşturulacağını öğreneceğiz.

App_Start \ BundleConfig.cs dosyasını MVC klasörlerinde açın. BundleConfig.cs dosyası varsayılan olarak MVC çerçevesi tarafından oluşturulur. Paket kodunuzu BundleConfig.RegisterBundles () yönteminde yazmalısınız. (BundleConfig sınıfını kullanmak yerine kendi özel sınıfınızı oluşturabilirsiniz, ancak standart uygulamayı izlemeniz önerilir.)

Örnek : BundleConfig.RegisterBundle()

```
public class BundleConfig
{
    public static void RegisterBundles(BundleCollection bundles)
    {
        bundles.Add(new ScriptBundle("~/bundles/jquery").Include(
            "~/Scripts/jquery-{version}.js"));
    }
}
```

Şimdi, bir projeye eklenen jquery dosyasını alacak. JQuery-1.7.1.js dosyasını eklediyseniz, bu dosyayı oluşturacak ve jquery dosyasını jquery-1.10.2.js dosyasına yükselttiğinizde 1.10 sürüm dosyasını kod değiştirmeden veya derlemeden otomatik olarak oluşturacaktır.

CDN Kullanımı

Komut dosyalarını yüklemek için İçerik Dağıtım Ağı'nı da kullanabilirsiniz. Örneğin, aşağıda gösterildiği gibi CDN'den jquery kitaplığını yükleyebilirsiniz.

Örnek : Load files from CDN

```
public class BundleConfig
{
    public static void RegisterBundles(BundleCollection bundles)
    {
        var cdnPath = "http://ajax.aspnetcdn.com/ajax/jquery/jquery-1.7.1.min.js";

        bundles.Add(new ScriptBundle("~/bundles/jquery", cdnPath)
            .Include("~/Scripts/jquery-{version}.js"));
    }
}
```

Örnek : Scripts.Render()

```
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
<title>@ViewBag.Title - My ASP.NET Application</title>
@Scripts.Render("~/bundles/jquery")
</head>
```

StyleBundle in ASP.NET MVC

ASP.NET MVC CSS küçültme ve paketlenme yapan StyleBundle sınıfını içerir. StyleBundle da bir Bundle sınıfından türetilir, böylece ScriptBundle ile aynı yöntemleri destekler.

App_Start -> BundleConfig.cs dosyasında bulunan BundleConfig sınıfının RegisterBundles () yönteminde komut dosyası ve css dosyaları paketleri oluşturmanız gerekir.

Css dosyaları eklemek için ScriptsInclude veya IncludeDerictory yöntemini kullanın.

Örnek : BundleConfig.RegisterBundle()

```
public class BundleConfig
{
    public static void RegisterBundles(BundleCollection bundles)
    {
        bundles.Add(new StyleBundle("~/Content/css").Include(
            "~/Content/bootstrap.css",
            "~/Content/site.css"));
    }
}
```

Örnek : Style.Render()

```
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>@ViewBag.Title - My ASP.NET Application</title>
@Style.Render("~/Content/css")
</head>
```

FilterConfig

ASP.NET MVC'de, bir kullanıcı isteği uygun denetleyiciye ve eylem yöntemine yönlendirilir. Bununla birlikte, bir eylem yönteminin çalıştırılmasından önce veya sonra bazı mantık yürütmek istediğiniz durumlar da olabilir. ASP.NET MVC bu amaçla filtreler sunar.

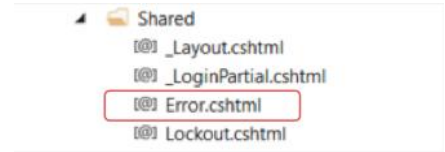
ASP.NET MVC Filtresi, bir eylem yönteminin çalıştırılmasından önce veya sonra çalıştırmak için özel mantığı yazabileceğiniz özel bir sınıftır. Filtreler bir eylem yöntemine veya denetleyiciye bildirimsel veya programlı bir şekilde uygulanabilir. Bildirge, bir eylem yöntemine veya denetleyici sınıfına ve programlı araçlara karşılık gelen bir arabirim uygulayarak bir filtre özneteliği uygulayarak anlamına gelir.

MVC farklı filtreler türleri sağlar. Aşağıdaki tabloda, filtre türleri, özel filtre sınıfı oluşturmak için uygulanması gereken tür ve arayüz için yerleşik filtreler gösterilmektedir.

Filter Type	Description	Built-in Filter	Interface
Authorization filters	İşlem yöntemini uygulamadan önce kimlik doğrulama yapar ve yetkilendirir.	[Authorize], [RequireHttps]	IAuthorizationFilter
Action filters	Bir eylem yöntemi yürütülmeden önce ve sonra bazı işlemleri gerçekleştirir.		IActionFilter
Result filters	İzleme sonucunun yürütülmesinden önce veya sonra bazı işlemleri yapar.	[OutputCache]	IResultFilter

Exception filters	ASP.NET MVC bağlantı hattının yürütülmesi sırasında atılan bir işlenmeyen özel durum varsa bazı işlemleri gerçekleştirir.	[HandleError]	IExceptionHandler
--------------------------	---	---------------	-------------------

Uygulamanızda işlenmeyen bir özel durum oluştuğunda bir istisna filtresi yürütülür. HandleErrorAttribute ([HandlerError]) sınıfı, MVC çerçevesinde yerleşik özel durum filtresi sınıfıdır. Bu yerleşik HandleErrorAttribute sınıfı, işlenmeyen bir özel durum oluştuğunda, Error.cshtml ögesini varsayılan olarak Paylaşılan klasörüne ekler.



HandleError filter örnek :

```
[HandleError]
public class HomeController : Controller
{
    public ActionResult Index()
    {
        throw new Exception("Beklenmedik Bir Hata !");
        return View();
    }

    public ActionResult About()
    {
        return View();
    }

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

Yukarıdaki örnekte, [HandleError] özniteliğini HomeController'a uyguladık.

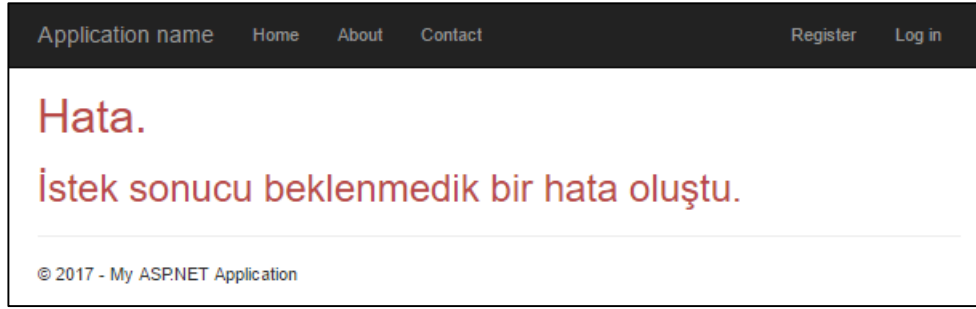
HomeController herhangi bir eylem yöntemi işlenmeyen özel durum atmak Şimdi Yani, Hata sayfası görüntüler. İşlenmemiş istisna, try-catch bloğu tarafından ele alınmayan bir istisnadır.

Denetleyiciye uygulanan filtreler, bir denetleyicinin tüm eylem yöntemlerine otomatik olarak uygulanır.

HandleErrorAttribute işlevinin düzgün çalışabilmesi için web.config dosyasının System.web bölümünde CustomError modunun açık olduğundan emin olun.

CustomError in web.config:

```
<system.web>
  <customErrors mode = "On" />
</system.web>
```

Custom Filter Oluşturma

Özel bir filtre oluşturmak istediğiniz uygun bir filtre arabirimini uygulayarak özel filtre nitelikleri oluşturabilir ve bu sınıfı bir özellik olarak kullanabilmeniz için bir FilterAttribute sınıfı türetebilirsiniz

Örnek : Custom Exception Filter

```
public class LogError : HandleErrorAttribute
{
    public override void OnException(ExceptionContext filterContext)
    {
        GetLog(filterContext.Exception);

        base.OnException(filterContext);
    }

    private void GetLog(Exception exception)
    {
        //loglama işlemleri bu alanda yer alacak.
    }
}
```

Örnek : Custom Exception Filter

```
[LogError]
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }
}
```

Home Controller içerisinde tanımlanan tüm Action'lar için Log işlemi yapacak bir Attribute tanımladık.

Bilinmesi Gerekenler :

- MVC Filtreleri, eylem yöntemini uygulamadan önce veya sonra özel mantık yürütmek için kullanılır.
- MVC Filtre türleri
 - Yetkilendirme filtreleri
 - İşlem filtreleri
 - Sonuç filtreleri
 - İstisna filtreleri
- Filtreler, ControllerConfig sınıfında, denetleyici seviyesinde veya eylem yöntemi düzeyinde genel olarak uygulanabilir.
- Özel filtre sınıfı, FilterAttribute sınıfı ve ilgili arayüzü uygulayarak oluşturulabilir.

RouteConfig

ASP.NET Web Forms uygulamasında, her URL belirli bir .aspx dosyasıyla eşleşmelidir. Örneğin, bir <http://bilgeadam.com/Index.aspx> URL'si, tarayıcıya yanıt vermek için kod ve biçimlendirme içeren Index.aspx dosyasıyla eşleşmelidir.

Yönlendirme, MVC çerçevesine özgü değildir. ASP.NET Webform uygulaması veya MVC uygulaması ile kullanılabilir.

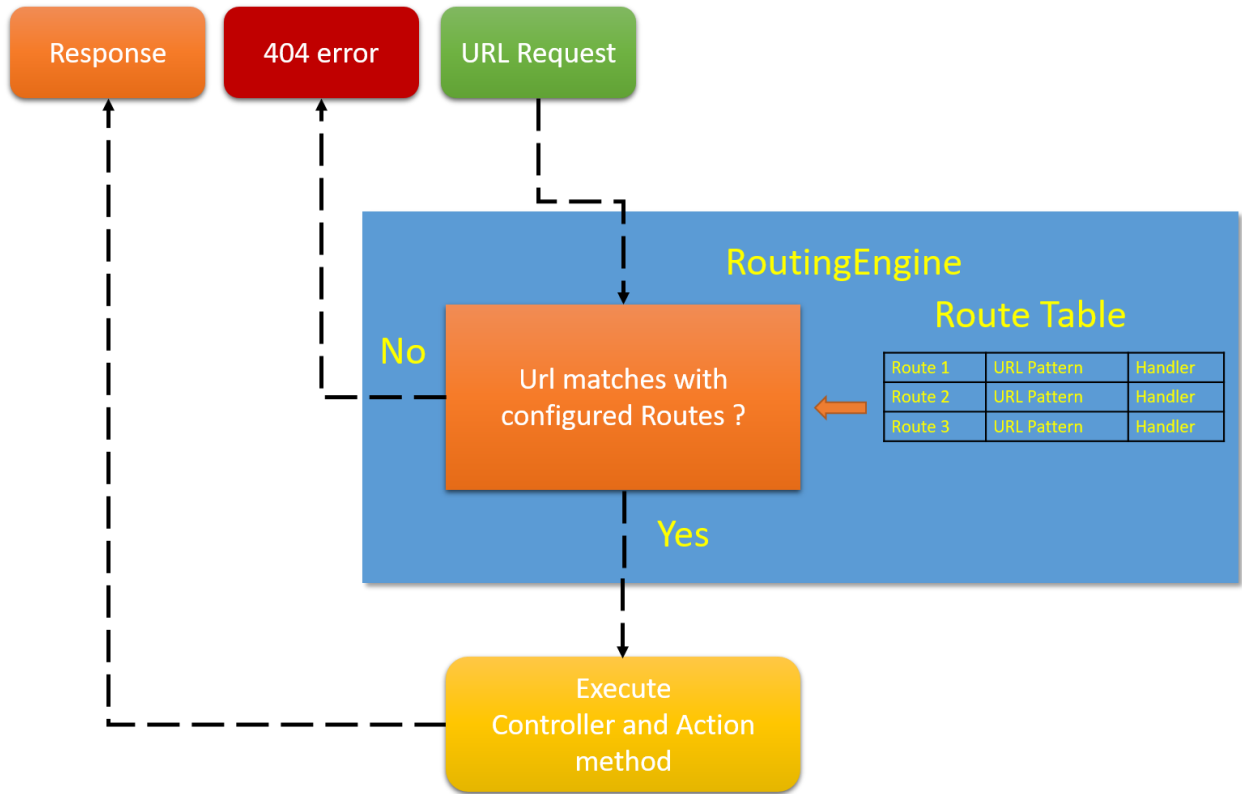
ASP.NET, her bir URL'yi fiziksel bir dosyayla eşleme gereksinimlerini gidermek için Yönlendirme'yi kullanıma sundu. Yönlendirme, istek işleyicisine eşleşen URL modelini tanımlamamızı sağlar.

Bu istek işleyici, bir dosya veya sınıf olabilir. ASP.NET Webform uygulamasında istek işleyicisi .aspx dosyasıdır ve MVC'de Denetleyici sınıfı ve Eylem yöntemi. Örneğin, [http:// etki alanı / ogrenciler](http://etki alanı / ogrenciler) ASP.NET Webforms'ta

<http://domain/students.aspx> olarak eşleştirilebilir ve aynı URL MVC'de Öğrenci Denetleyicisi ve Dizin eylem yöntemiyle eşleştirilebilir.

Route

Rota URL kalıbını ve işleyici bilgilerini tanımlar. RouteTable'da saklanan bir uygulamanın yapılandırılmış tüm rotaları ve gelen istek için uygun işleyici sınıfını veya dosyasını belirlemek için Yönlendirme altyapısı tarafından kullanılacaktır



Routing

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

        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "Home", action = "Index", id=UrlParameter.Optional }
        );
    }
}
```

```
    );  
}  
}
```

App_Start klasörü içerisindeki RouteConfig class'ı içerisinde default olarak bir adet route mekanizması yer alır.

Yapı içerisinde isteğe göre birden fazla Route yapısı yer alabilir.

Multiple Routing

```
public class RouteConfig  
{  
    public static void RegisterRoutes(RouteCollection routes)  
    {  
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");  
  
        routes.MapRoute(  
            name: "Product",  
            url: "Product/Detail/{catId}/{id}",  
            defaults: new {  
                controller = "Product",  
                action = "Detail",  
                catId = UrlParameter.Optional,  
                id = UrlParameter.Optional  
            }  
        );  
  
        routes.MapRoute(  
            name: "Default",  
            url: "{controller}/{action}/{id}",  
            defaults: new {  
                controller = "Home",  
                action = "Index",  
                id = UrlParameter.Optional  
            }  
        );  
    }  
}
```

Route Constraints

Ayrıca, yol kısıtlamalarını yapılandırarak parametrenin değerine kısıtlamalar uygulayabilirsiniz. Örneğin, aşağıdaki yol id parametresinin sayısal olmasını gerektiren id parametresine bir sınırlama uygular.

Multiple Routing

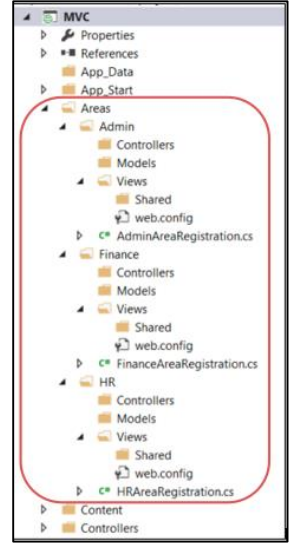
```
routes.MapRoute(  
    name: "Student",  
    url: "student/{id}/{name}/{standardId}",  
    defaults: new {  
        controller = "Student",  
        action = "Index",  
        id = UrlParameter.Optional,  
        name = UrlParameter.Optional,  
        standardId = UrlParameter.Optional },  
    constraints: new { id = @"\d+" }  
);
```

Area

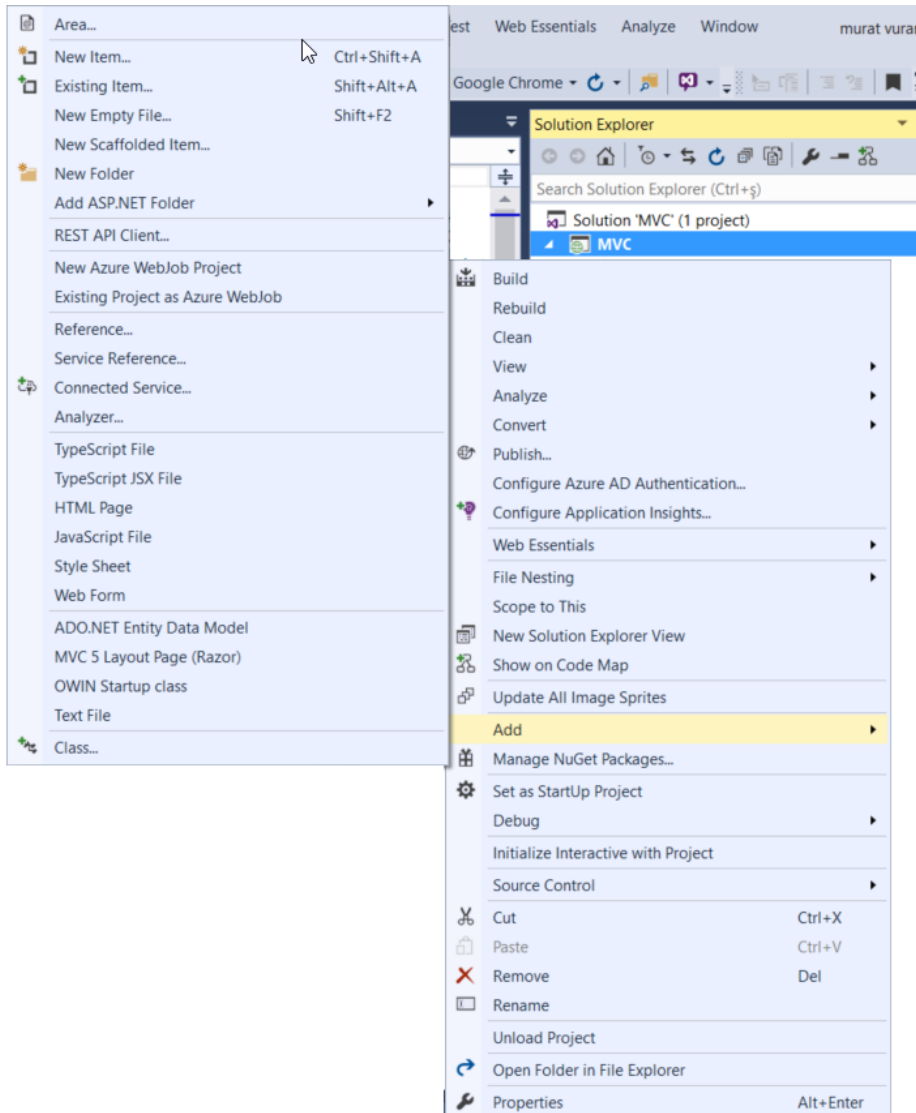
MVC içerisinde büyük uygulama, çok sayıda denetleyici, görünüm ve model sınıfı içerebilir. Varsayılan ASP.NET MVC proje yapısıyla çok sayıda görünüm, model ve denetleyiciyi korumak için yönetilemez hale gelebilir

ASP.NET MVC 2 Alan tanıttı. Alan, büyük bir uygulamanın, her birimin ayrı MVC klasör yapısını, varsayılan MVC klasör yapısıyla aynı olan daha küçük birimlere bölmelerini sağlar. Örneğin, büyük kurumsal uygulama, Yönetim, Finans, İK, pazarlama vb. Gibi farklı modüllere sahip olabilir. Dolayısıyla Bir Alan, aşağıda gösterildiği gibi tüm bu modüller için ayrı MVC klasör yapısını içerebilir.

Yeni bir area eklemek için, aşağıdaki resimdeki yolu takip edebilirsiniz.



Multiple Routing



Yeni Area Ekleme

Add Area

Area name:

Add
Cancel

Area Route Ayarlama

```
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional },
    namespaces: new[] { "MVC.Controllers" }
    // Area içerisindeki yapıların tanımlanabilmesi için
);
```

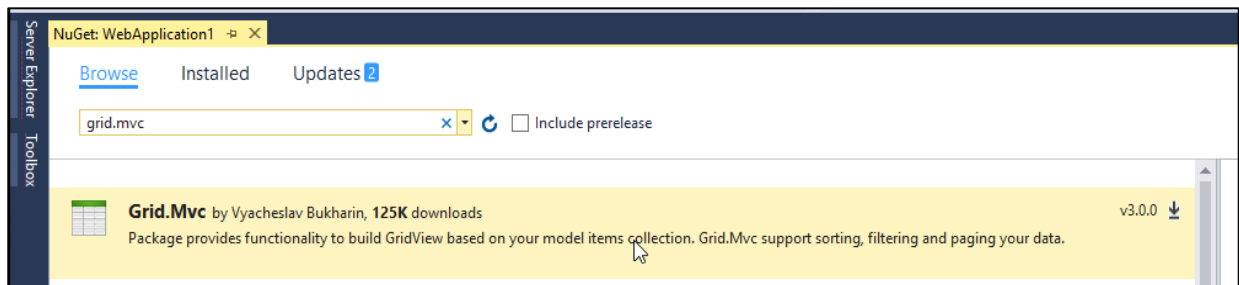
Grid MVC

Grid.Mvc, ASP.NET MVC web uygulamasında GridView denetimlerini oluşturmak için işlevsellik ekler. Model nesnelerinin bir koleksiyonundaki verileri görüntüleme, sayfalama, filtreleme ve sıralama için HTML tablolarını kolayca oluşturmanızı sağlayan bir bileşendir.

Grid.Mvc kullanabilmemiz için referanslara dll'i indirmemiz gerekmektedir.

Grid.Mvc referanslara eklemek için aşağıdaki resmi inceleyebilirsiniz. Referans işleminden sonra Grid.Mvc'yi sayfalarınızda kullanabilirsiniz.

Number	Date	Company	Is Vip	Order details
Edit 11074	0	Type: Equals	No	Order number: 11071
Edit 11075	0		No	Order date: 05.05.1998
Edit 11076	0	September 2013	Yes	Freight: 0.9300
Edit 11077	0	Su Mo Tu We Th Fr Sa	No	Ship address: 3508, Barquisimeto, Barquisimeto, Carrera 52 con Ave. Bolívar #65-98 Llano Largo
Edit 11070	0	1 2 3 4 5 6 7	No	Customer: LILA-Supermercado
Edit 11071	0	8 9 10 11 12 13 14	No	Employee: Davolio Nancy
Edit 11072	0	15 16 17 18 19 20 21	No	Edit order
Edit 11073	0	22 23 24 25 26 27 28	No	
Edit 11073	0	29 30 1 2 3 4 5	No	
Edit 11067	04.05.1998	Pencies Comidas clásicas	No	
Edit 11068	04.05.1998	Drachenblut Delikatessen	No	
Edit 11069	04.05.1998	Queen Cozinha	No	
Edit 11064	01.05.1998	Tortuga Restaurante	No	
Edit 11065	01.05.1998	Save-a-lot Markets	No	
Edit 11066	01.05.1998	LILA-Supermercado	No	
Edit 11060	30.04.1998	White Clover Markets	No	
Edit 11060	30.04.1998	Franchi S.p.A.	No	



Views Folder web.config

```
<pages pageBaseType="System.Web.Mvc.WebViewPage">
  <namespaces>
    <add namespace="System.Web.Mvc" />
    <add namespace="System.Web.Mvc.Ajax" />
    <add namespace="System.Web.Mvc.Html" />
    <add namespace="System.Web.Routing" />
    <add namespace="MVC_Grid" />
    <add namespace="GridMvc.Html"/>
  </namespaces>
</pages>
```

Views Folder _Layout

```
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>@ViewBag.Title - My ASP.NET Application</title>
  <link href="~/Content/bootstrap.css" rel="stylesheet" />
  <link href="~/Content/Gridmvc.css" rel="stylesheet" />
  <script src="~/scripts/jquery-1.10.2.js"></script>
  <script src="~/scripts/bootstrap.js"></script>
  <script src="~/scripts/gridmvc.js"></script>
</head>
```

Grid.Mvc Codebehind

```
public ActionResult Index()
{
    List<Category> categories = _northwindService.Categories.ToList();
    return View(categories);
}
```

Grid MVC Html

```
@Html.Grid(Model).Columns(c =>
{
    c.Add(ct => ct.CategoryID)
      .Titled("Kategori ID")
      .Encoded(false)
      .Sanitized(false);

    c.Add(ct => ct.CategoryName)
      .Titled("Kategori Adı")
      .Sortable(true)
      .Filterable(true)
      .Encoded(false)
      .Sanitized(false);

    c.Add(ct => ct.Description)
      .Titled("Açıklama")
      .Sortable(true)
      .Encoded(false)
      .Filterable(true)
      .Sanitized(false);
}).WithPaging(4)
```

```
.Selectable(true)
```

Property name	Description	Example
Titled	Kolon başlığını değiştirme	<code>Columns.Add(x=>x.Name).Titled("Name of product");</code>
Encoded	Kodlama sütun değerlerini etkinleştirir veya devre dışı bırakır	<code>Columns.Add(x=>x.Name).Encoded(false);</code>
Sanitized	Kodlama devre dışıysa, XSS saldırılarından sütun değerini sterilize edin	<code>Columns.Add(x=>x.Name).Encoded(false).Sanitize(false);</code>
SetWidth	Geçerli sütunun ayar genişliği	<code>Columns.Add(x=>x.Name).SetWidth("30%");</code>
RenderValueAs	Sütun değerlerini oluşturmak için temsilci ayarlayın	<code>Columns.Add(x=>x.Name).RenderValueAs(o => o.Employees.FirstName " " o.Employees.LastName)</code>
Sortable	Geçerli sütun için sıralama özelliğini etkinleştirin veya devre dışı bırakın	<code>Columns.Add(x=>x.Name).Sortable(true);</code>
SortInitialDirection	Sütunun ilk sıralama deirectionini ayarlayın (sıralama özelliğini etkinleştirmeniz gerekir)	<code>Columns.Add(x=>x.Name).Sortable(true).SortInitialDirection(GridSortDirection.Descending);</code>
SetInitialFilter	Sütunun ilk filtresini ayarlayın	<code>Columns.Add(x=>x.Name).Filterable(true).SetInitialFilter(GridFilterType.Equals, "some name");</code>
ThenSortBy	Kurulum Daha sonra mevcut sütun sıralama	<code>Columns.Add(x=>x.Name).Sortable(true).ThenSortBy(x=>x.Date);</code>
ThenSortByDescending	Kurulum Ardından Geçerli sütunun azalan sıralanmasıyla	<code>Columns.Add(x=>x.Name).Sortable(true).ThenSortBy(x=>x.Date).ThenSortByDescending(x=>x.Description);</code>
Filterable	Sütun filtreleme özelliğini etkinleştirme veya devre dışı bırakma	<code>Columns.Add(x=>x.Name).Filterable(true);</code>
SetFilterWidgetType	Özel filtreleme kullanıcı arabirimini oluşturmak için kurulum filtresi widget tipi	<code>Columns.Add(x=>x.Name).Filterable(true).SetFilterWidgetType("MyFilter");</code>
Format	Sütun değerini formatla	<code>Columns.Add(o => o.OrderDate).Format("{0:dd/MM/yyyy}")</code>
Css	Sütuna css sınıfları uygulayın	<code>Columns.Add(x => x.Number).Css("hidden-xs")</code>

Jquery ajax() method

JQuery ajax () yöntemi, jQuery'de Ajax'ın temel işlevselliğini sağlar. Sunucuya asenkron HTTP istekleri gönderir.

Syntax

```
$.ajax(url);  
$.ajax(url,[options]);
```

Options	Description
url	Verileri göndermek veya almak istediğiniz bir dize URL'si
options	Ajax isteği için yapılandırma seçenekleri. Bir options parametresi JSON formatını kullanarak belirtilebilir. Bu parametre isteğe bağlıdır.
accepts	İstek üstbilgisinde gönderilen ve sunucuya karşılığında ne tür bir yanıt kabul edeceğini bildiren içerik türü.
async	Varsayılan olarak, tüm istekler zaman uyumsuz olarak gönderilir. Eşzamanlı yapmak için yanlış ayarlayın.
beforeSend	Ajax isteği gönderilmeden önce yürütülecek bir geri arama işlevi.
cache	Tarayıcı önbelleğini belirten bir boolean. Varsayılan doğrudur.
complete	Talep bittiğinde yürütülecek bir geri arama fonksiyonu.
contentType	MIME içeriğini sunucuya gönderirken bir içerik türü içeren bir dize. Varsayılan "application / x-www-form-urlencoded; charset = UTF-8"
crossDomain	Bir isteğin bir alanlar arası olup olmadığını gösteren bir boolean değeri.
data	Sunucuya gönderilecek bir veri. JSON nesnesi, dizi veya dizi olabilir.
dataType	Sunucudan geri beklediğiniz veri türü.
error	İstek başarısız olduğunda çalıştırılacak geri arama işlevi.
global	Genel bir Ajax istek işleyicisinin tetiklenip tetiklenmeyeceğini gösteren bir Boolean. Varsayılan doğrudur.
headers	Talep ile birlikte gönderilecek ek başlık tuşu / değer çiftlerinin bir nesnesi.
ifModified	İsteğin ancak son istekten sonra yanıt değiştiğinde isteğin başarılı olmasına izin verin. Bu, Last-Modified başlığını işaretleyerek yapılır. Varsayılan değer false değeridir.
isLocal	Mevcut ortamın yerel olarak tanınmasına izin verin.
jsonp	Bir JSONP isteğinde geri arama işlev adını geçersiz kılın. Bu değer 'geri arama' yerine kullanılacak 'geri arama =' Url'deki sorgu dizesinin bir parçası.
jsonpCallback	JSONP isteği için geri arama işlev adını içeren dize.
contentType	XMLHttpRequest mime türünü geçersiz kılmak için bir mime türü içeren dize.

password	Bir HTTP erişim kimlik doğrulama isteğine yanıt olarak XMLHttpRequest ile kullanılacak bir parola.
processData	Veri seçeneğine atanan verilerin bir sorgu dizisine dönüştürüleceğini belirten bir Boolean. Varsayılan doğrudur.
statusCode	Sayısal HTTP kodları ve karşılık gelen kod olduğunda yanıtlanacak işlevleri içeren bir JSON nesnesi.
success	Ajax isteği başarılı olduğunda çalıştırılacak geri arama işlevi.
timeout	İstek zamanaşımı için milisaniye cinsinden bir sayı değeri.
type	Bir tür http isteği ör. POST, PUT ve GET. Varsayılan GET'tir.
url	İsteğin gönderileceği URL'yi içeren bir dize.
username	Bir HTTP erişim kimlik doğrulama isteğine yanıt olarak XMLHttpRequest ile kullanılacak bir kullanıcı adı.
xhr	XMLHttpRequest nesnesini oluşturmak için bir geri arama.
xhrFields	Nesneli XMLHttpRequest nesnesinde ayarlamak için fieldName-fieldValue çiftlerinin bir nesnesi.

Perform Ajax request

Ajax () yöntemleri eşzamansız http isteği gerçekleştirir ve verileri sunucudan alır. Aşağıdaki örnek, basit bir Ajax isteği göndermenin nasıl yapıldığını göstermektedir.

Örnek : jQuery Ajax request

```
<script>
$.ajax('http://localhost:7053/Categories/Get/', // request url
{
    success: function (data, status, xhr) { // success callback function
        $('p').append(data);
    }
});
</script>
```

Yukarıdaki örnekte, ajax () yönteminin ilk parametresi ' <http://localhost:7053/Categories/Get/> ', veriyi almak istediğimiz bir url'dir.

İkinci parametre, istek başarılı olduğunda çalıştırılacak geri çağırım işlevi tanımladığımız JSON formatındaki options parametresidir. Diğer seçenekleri yukarıdaki tabloda belirtildiği gibi yapılandırabilirsiniz. Aşağıdaki örnek, ajax () yöntemini kullanarak JSON verilerini nasıl elde edebileceğinizi gösterir.

Örnek : get JSON data

```
<script>
$.ajax('http://localhost:7053/Categories/Get/',
{
    dataType: 'json', // istekte bulunulan data tipi
    timeout: 500, // timeout milliseconds cinsinden
    success: function (data, status, xhr) { // başarılı olması durumunda çalışacak.

        $('p').append(data.firstName + ' ' + data.middleName + ' ' + data.lastName);

    },
},
```

```
error: function (jqXHR, textStatus, errorMessage) { //hata alması durumunda çalışacak
    $('p').append('Error: ' + errorMessage);
}
});
</script>
```

Yukarıdaki örnekte, ilk parametre JSON verilerini döndürecek bir istek url'dir. Options parametresinde, dataType ve timeout seçenekleri belirttik. DataType seçeneği yanıt verisinin türünü belirtir, bu durumda JSON olur.

Timeout parametresi istek zaman aşımı değerini milisaniye cinsinden belirtir. Ayrıca, hata ve başarı için geri arama işlevleri de belirledik. Ajax () yöntemi jQuery XMLHttpRequest nesnesini döndürür.

Aşağıdaki örnek, jQuery XMLHttpRequest nesnesinin nasıl kullanılacağını göstermektedir.

Örnek : ajax() method

```
<script>
var ajaxReq = $.ajax('GetJsonData', {
    dataType: 'json',
    timeout: 500
});

ajaxReq.success(function (data, status, jqXHR) {
    $('p').append(data.FirstName + ' ' + data.MiddleName + ' ' + data.LastName);
});

ajaxReq.error(function (jqXHR, textStatus, errorMessage) {
    $('p').append('Error: ' + errorMessage);
});
</script>
```

Perform Http POST request using ajax() method

Ajax () yöntemi her tür http isteğini gönderebilir. Aşağıdaki örnek, sunucuya http POST isteği gönderir.

Örnek : Send POST request with ajax() method

```
<script>
$.ajax('/jquery/submitData', {
    type: 'POST',
    data: { myData: 'İstek Sonucu Gelen Data' },
    success: function (data, status, xhr) {
        $('p').append('status: ' + status + ', data: ' + data);
    },
    error: function (jqXHR, textStatus, errorMessage) {
        $('p').append('Error' + errorMessage);
    }
});
</script>

<p></p>
```

Yukarıdaki örnekte, ilk parametre, verileri göndermek için kullanılan bir url'dir. Options parametresinde, POST olarak bir tür seçeneği belirttik, bu nedenle ajax () yöntemi http POST isteği gönderir. Ayrıca, sunucuya gönderilecek verileri içeren bir JSON nesnesi olarak veri seçeneğini belirttik.

Böylece ajax () yöntemini kullanarak GET, POST veya PUT isteklerini gönderebilirsiniz.

State Management

State Managment asp.net platformunda verilerin tek bir alandan gidip gelmiyor olması nedeniyle ortaya çıkmış bir kavramdır. Asp.net tarafında veriler sunucu (server) ve istemci (client) olmak üzere iki taraf üzerinden gidip gelmektedir. Bir web form üzerindeki veriler client tarafından server tarafına gönderilir, işlenir ve çıktısı geri client tarafına gönderilir.

ASP.NET State Managment tarafında bahsetmiş olduğumuz iki teknik vardır bunlar istemci taraflı ve sunucu taraflı state managment teknikleridir.

- Sunucu taraflı state management teknikleri
 - Session State (Oturum Durumu)
 - Application State (Uygulama Durumu)
- İstemci (Client) taraflı state management teknikleri
 - ViewState (Durum Görünümü)
 - QueryString (Sorgu Kelimeleri)
 - Cookies (Çerezler)

Sırasıyla tekniklerimizi inceleyelim..

Application State

Session state gibi değer verilebilen ve içerisindeki değerleri kullanabildiğimiz nesnelerdir. Ancak session state, başlatıldığı andan kullanıcının uygulamayı yani web sayfamızı kapatana kadar geçirdiği süre zarfında saklanırken, application state oluşturulduğu andan itibaren web sayfası yada sunucu kapatılana kadar (iis restrat işlemleri için de geçerlidir) korunacaktır. Kullanım olarak aynı yapıya sahip olan application state, ziyaretçi sayılarınızı tutabileceğiniz, anlık ziyaretçi bilgisine erişebileceğiniz bir tekniktir. Diğer bir söylemle session kullanıcıya özel olmakla birlikte application uygulama bazlıdır, tüm kullanıcılara içerisindeki değer istenilirse gösterilebilir veya değiştirmelerine olanak tanınabilir.

Application State için aşağıdaki gibi bir kullanım gerçekleştirebilirsiniz; Application["EtiketAdı"] = "EtiketinDeğeri";

Yukarıdaki şekilde istediğiniz sayfaların codebehind bölümlerinde kullanım gerçekleştirebilirsiniz. Bununla beraber application başlangıcından uygulama durdurulana yada kapatılana kadar gerçekleşen süreçler

Global.asax (ASAX => Active Server Application X) içerisinde ki metotlar içerisine yazılmaktadır.

Bazılarını inceleyecek olursak; Application_Start : Uygulamanızın başlangıcı esnasında çalışacak olan metottur.

Bu metot sayfa kullanıcı tarafından açıldığında değil web sayfası ilk olarak yayına açıldığında çalışmaktadır.

Session_Start :Uygulama bir kullanıcı tarafından açıldığı anda session_start metodu çalışmaktadır.

Application_BeginRequest : FormsAuthentication kullanıldığında kullanıcının sisteme başarılı şekilde giriş yapması durumunda tetiklenecek olan olaydır.

Application_Error : Uygulama bir hata ile karşılaşması durumunda tetiklenecek eventdir.

Bu event uygulamaya erişim kesildiğinde yani sunucu kapandığında çalışmayacaktır.

Session_End : Kullanıcı siteyi terk ettiği veya session timeout yaşandığı esnada tetiklenen eventdir.

Application State kullanarak web sayfamız da kaç kişinin online olabildiğini görebileceğimiz örnek bir uygulama yapalım,

Asp.net projemize sağ tıklayarak add > new item > Global Application Class ögesini seçtikten sonra projemize Global.asax ismiyle ekliyoruz. Dosyamız oluşturulduktan sonra içerisinde yer alan eventler üzerinden online kullanıcılarımızın bilgileri için ilk olarak Application_Start eventimiz içerisine web sayfamız yayına başladığı anda oluşturulacak olan bir application nesnesi oluşturuyoruz.

Global.asax – Application Start

```
protected void Application_Start()
{
    AreaRegistration.RegisterAllAreas();
    FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
    RouteConfig.RegisterRoutes(RouteTable.Routes);
    BundleConfig.RegisterBundles(BundleTable.Bundles);

    Application.Add("OnlineUsers", 0);
}
```

Oluşturulan OnlineKullanıcılar etiketine sahip olan application nesneminin varsayılan değerini 0 olarak belirledik. Bu işlemi aşağıdaki yöntemle yapabildik;

```
Application["OnlineKullanıcılar"] = 0;
```

Uygulamamızın aktif olduğu esnadaki online kullanıcı sayımızı 0 olarak belirledik, artık uygulamamızda her yeni bir oturum açıldığında online kullanıcı sayımızı bir arttıracaktır. Her yeni bir kullanıcı, yeni bir oturum olarak kabul edildiğinden dolayı kullanıcı sayımızın artırma işlemini session_start event'i içerisinde gerçekleştirmeliyiz.

Global.asax – Session Start

```
protected void Session_Start(object sender, EventArgs e)
{
    Application.Lock();
    Application["OnlineKullanıcılar"] = ((int)Application["OnlineKullanıcılar"]) + 1;
    Application.Unlock();
}
```

Yukarıdaki örnek kodda, Application["OnlineKullanıcılar"]'dan dönen değeri int değerine cast ettik ve değeri bir artırarak Application nesneminin içerisine tekrar teslim ettik. Bu şekilde online kullanıcı sayımız + 1 yaparak güncel online kullanıcı sayımızı application nesnesi içerisinde tutabiliriz. Kod örneğinde bulunan Application.Lock() metodu turnike görevi görerek o esnada application nesnesinin başka bir session tarafından kullanılmasını engelleyecektir. Kullanıcı sayımızdaki artışı tamamladık şimdi geldi sıra kullanıcı yani oturum (session) sonlandırılırsa bu kullanıcı online kullanıcı sayısından düşmeliyiz. Bunun için oturum bittiğinde, session_end eventini kullanabiliriz;

Global.asax – Session End

```
protected void Session_End(object sender, EventArgs e)
{
    Application.Lock();
    Application["OnlineKullanıcılar"] = ((int)Application["OnlineKullanıcılar"]) - 1;
    Application.Unlock();
}
```

Yukarıda da aynı kullanıcı sayımızı arttırdığımız gibi, session sonlandığı anda yani kullanıcı timeout olduğu yada web sayfasını kapattığı zaman online kullanıcı sayımızın bir düşmesini sağladık.

Artık gerekli artış ve azalış işlemlerini gerçekleştirdiğimize göre istediğimiz sayfadan web sayfamızı ziyaret eden kişilere kaç online kullanıcımız olduğunu gösterebiliriz.

Html

```
<label> Kullanıcı Sayısı :@HttpContext.Current.Application["OnlineKullanıcılar"] </label>
```

Session State

Web sunucusu, kendisi ile iletişime geçildiği anda, kendisine talepte bulunan browser (tarayıcı) ve talebin geldiği IP adreslerini kullanarak kullanıcıya özel, benzersiz bir değer üretir... Üretilen bu değere ASPNET_SessionId adi verilmektedir. Üretilen bu SessionId bilgisinin yer aldığı bir cookie'de client tarafa yerleştirilir...

Daha sonra, kullanıcı sayfalar arasında gezinirken, Sunucudaki ASPNET_SessionId bilgisi ile client taraftaki cookie aracılığı ile tutulan ASPNET_SessionId bilgisi karşılaştırılır. Her Session için RAM'de ayrı ayrı bellek bölümleri kullanılmaktadır. Oldukça performanslı olan bu yapı, yanlış ve bilinçsiz bir kullanıma uygulamanızın sürekliliğini bile bozabilir.

Session Tanımlama

```
Session["EtiketIsmi"] = "Etiket içerisindeki değer";
```

Yukarıdaki kullanımdaki gibi kullanabilirsiniz. Session nesneleri içeriye değer olarak object türünde kabul etmektedir bu nedenle tüm değer ve referans tiplerini session içerisinde saklayabilirsiniz. Session web sayfalarının giriş yapılan kullanıcı numaralarını saklamak, sayfalar içerisinde gezinti sağlanırken tekrar kullanıcı adı şifre sormamak için ve genellikle e-ticaret sitelerinde sepet içerisindeki ürünleri tutmak için kullanılır.

Session nesnesi timeout yani zaman aşımı oluncaya kadar saklanmaktadır. Sunucuların içerisinde IIS (Internet Information Server) yani web sitesini yayınlayan web server servisinde varsayılan olarak 20 dakika olarak ayarlanmıştır. Bir kullanıcının giriş yaptığı esnada bir session içerisinde giriş yapan kullanıcının veritabanındaki ID numarasını tuttuğunuzu düşünürsek, kullanıcı bir sayfa dışına çıkmaz yani aynı sayfada sabit olarak kalır ise 20 dakika sonra otomatik olarak session kapatılacaktır. Kullanıcı sayfalar arasında gezintiye devam ettiği sürece, her yeni bir sayfaya giriş yaptığında session süresi tekrar 20 dakika olarak ayarlanacaktır.

Web sayfanızın web.config dosyasından session timeout süresini varsayılan dışında bir değer olarak belirleyebilirsiniz. Değer tipi olarak saniye kabul etmektedir, bu değişikliği aşağıdaki kod örneği ile gerçekleştirebilirsiniz.

Web.config

```
<configuration>
  <system.web>
    <httpRuntime executionTimeout="110"></httpRuntime>
  </system.web>
</configuration>
```

Html

```
<label>Toplam Sayfa Ziyaret Sayısı : @Session["PageCount"]</label>
```

Codebehind

```
public ActionResult Index()
{
    if (Session["PageCount"] != null)
        Session["PageCount"] = Convert.ToInt32(Session["PageCount"]) + 1;
    else
        Session.Add("PageCount", 1);

    return View();
}
```

Caching (Önbellekleme)

Önbellekleme, sayfalarınızda ihtiyacınız olan verilerin her request (istek) esnasında tekrar tekrar kaynaktan çekilmesinin önüne geçmek ve bu şekilde performansı arttırmak amacıyla ortaya çıkmıştır. Örnek olarak bir e-ticaret siteniz var ve ana sayfada bulunan ürünleri her ziyaretçi siteye girdiğinde veritabanından çekerek gösteriyorsunuz. Ana sayfada bulunan ürünlerinizin sürekli veritabanından tekrar tekrar çekilmesi doğru bir yöntem değildir. Bu yöntem yerine önbellekleme yaparak verilerinizi bir sonraki değişikliğe kadar önbellekte saklayabilir, kullanıcılarınız web sayfanıza girdiğinde direkt olarak ürünleri önbellekten çekerek onlara gösterebilirsiniz. Bu

kaynađınızı, veritabanınızı yormayacađı gibi aynı zamanda hız anlamında da ekstra olarak bir performans sađlayacaktır.

Cache (önbellek) ikiye Data Caching ve Page Output Caching olarak ikiye ayrılmaktadır.

Data Caching (Veri Önbellekleme)

Herhangi bir veri kaynađından gelen verilerinizi önbellekleyerek, yukarıdaki örnekte bahsetmiř olduđumuz gibi veriler için performans kazanmanıza olanak tanımaktadır.

Class

```
public class FoodList
{
    public int Id { get; set; }
    public string Name { get; set; }
    public DayOfWeek Day { get; set; } // DayOfWeek is enum
    public int Week { get; set; }
}
```

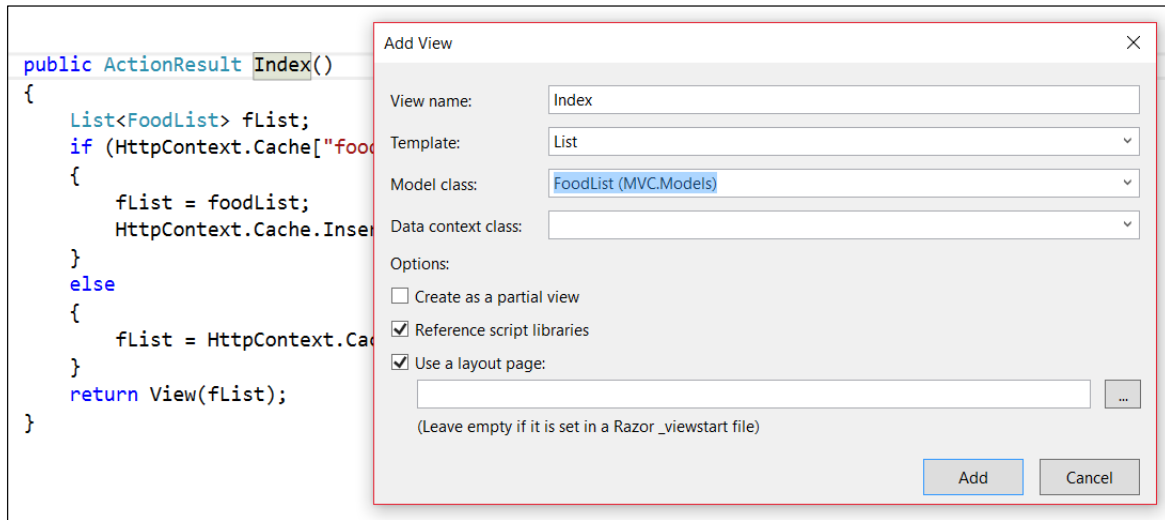
Data

```
List<FoodList> foodList = new List<FoodList>()
{
    new FoodList {Id=1,Name = "Pizza",Day =DayOfWeek.Monday,Week =1},
    new FoodList {Id=2,Name = "Pizza",Day =DayOfWeek.Tuesday,Week =1},
    new FoodList {Id=3,Name = "Pizza",Day =DayOfWeek.Wednesday,Week =1},
    new FoodList {Id=4,Name = "Pizza",Day =DayOfWeek.Thursday,Week =1},
    new FoodList {Id=5,Name = "Pizza",Day =DayOfWeek.Friday,Week =1}
};
```

Action

```
public ActionResult Index()
{
    List<FoodList> fList;
    if (HttpContext.Cache["foodList"] == null)
    {
        fList = foodList;
        HttpContext.Cache.Insert("foodList", fList);
    }
    else
    {
        fList = HttpContext.Cache["foodList"] as List<FoodList>;
    }
    return View(fList);
}
```

View oluşturmak için ařađıdaki resmi inceleyebilirsiniz.



View

```
<table class="table">
  <tr>
    <th>
      @Html.DisplayNameFor(model => model.Name)
    </th>
    <th>
      @Html.DisplayNameFor(model => model.Day)
    </th>
    <th>
      @Html.DisplayNameFor(model => model.Week)
    </th>
  </tr>
  @foreach (var item in Model) {
    <tr>
      <td>
        @Html.DisplayFor(modelItem => item.Name)
      </td>
      <td>
        @Html.DisplayFor(modelItem => item.Day)
      </td>
      <td>
        @Html.DisplayFor(modelItem => item.Week)
      </td>
      <td>
        @Html.ActionLink("Edit", "Edit", new { /* id=item.PrimaryKey */ }) |
        @Html.ActionLink("Details", "Details", new { /* id=item.PrimaryKey */ }) |
        @Html.ActionLink("Delete", "Delete", new { /* id=item.PrimaryKey */ })
      </td>
    </tr>
  }
</table>
```

Sonuç olarak ekranda aşağıdaki çıktıyı görüntüleyebilirsiniz.

Name	Day	Week	
Pizza	Monday	1	Edit Details Delete
Pizza	Tuesday	1	Edit Details Delete
Pizza	Wednesday	1	Edit Details Delete
Pizza	Thursday	1	Edit Details Delete
Pizza	Friday	1	Edit Details Delete

Output Caching (Çıktı Önbellekleme)

Sayfanızın tüm html ve css çıktısını önbelleğe alarak kullanıcılarınıza bir sonraki önbellekleme zamanına kadar aynı görüntünün gösterilmesini sağlayan önbelleme mekanizmasıdır.

OutputCache

```
[OutputCache(  
    CacheProfile = "",  
    Duration = 1,  
    Location = new System.Web.UI.OutputCacheLocation(),  
    NoStore = true,  
    SqlDependency = "",  
    VaryByCustom = "",  
    VaryByHeader = "",  
    VaryByParam = ""  
)]
```

CacheProfile

Webconfig içerisinde belirleyeceğiniz profillere göre farklı zamanlama ayarları kullanabilirsiniz.

CacheProfile

```
< caching >  
  < outputCacheSettings >  
    < outputCacheProfiles >  
      < add name="Long" duration="60" varyByParam="none" />  
      < add name="Medium" duration="60" varyByParam="none" />  
      < add name="Short" duration="10" varyByParam="none" />  
    < /outputCacheProfiles >  
  < /outputCacheSettings >  
< / caching >
```

CacheProfile

```
[OutputCache(CacheProfile = "Long")]  
public ActionResult List()  
{  
    return View();  
}
```


Duration

Saniye cinsinden ilgili ekranın cache'de kalmasını sağlar

Duration

```
[OutputCache(Duration = 60)]
public ActionResult List()
{
    return View();
}
```

Location

Cache'in hangi alanda saklanacağını belirleyebilirsiniz. İçerisine aldığı parametreler ise;

Location

```
[OutputCache(Location = System.Web.UI.OutputCacheLocation.Server)]
public ActionResult List()
{
    return View();
}
```

Any : Çıktı önbellek tarayıcı istemcisinde (isteğin kaynağı olduğu yerde) bulunabilir, İsteğine katılan bir proxy sunucudaki (veya başka herhangi bir sunucuda) veya

İsteğin işlendiği sunucu.

2) Clint : Çıktı önbelleği, isteğin kaynaklandığı tarayıcı istemcisinde bulunur. Bu değer, System.Web.HttpCacheability.Private numaralandırma karşılık gelir

Downstream : Çıktı önbellek, HTTP 1.1 önbellek özellikli diğer cihazlarda saklanabilir.

Kökenli sunucu. Bu, proxy sunucuları ve isteği yapan istemciyi içerir.

None : Çıktı önbelleği, isteğin işlendiği Web sunucusunda bulunur. Bu değer, System.Web.HttpCacheability.Server numaralandırma karşılık gelir.

Server : İstenen sayfa için çıktı önbelleği devre dışı bırakıldı. Bu değer, System.Web.HttpCacheability.NoCache numaralandırma değeri.

ServerAndClient : Çıktı önbelleği, yalnızca orijin sunucusunda veya istekte depolanabilir Müşteri. Proxy sunucularının yanıtın önbelleğe alınmasına izin verilmez. Bu değer, System.Web.HttpCacheability.Private ve System.Web.HttpCacheability.Server kombinasyonuna Numaralandırma değerleri.

NoStore

Bilgilerin kalıcı olarak saklanmaması için de NoStore özelliğinin true olarak set edilmesi gerekmektedir.

NoStore

```
[OutputCache(NoStore = true)]
public ActionResult List()
{
    return View();
}
```

SqlDependency

Sql tablonuza bir değişiklik olduğunda kendiliğinden cache yapısını bozup tekrar cache yapısını kurar.

Not :SqlDependency yapısı için web.config içerisinde ve CommandPromp üzerinden ayarlar yapmanız gerekmektedir.

SqlDependency

```
[OutputCache(SqlDependency = "Northwind:FoodList")]
public ActionResult List()
{
    return View();
}
```

VaryByCustom

Bu alt özellik ; kendi yazacağımız kodumuza göre Cahcleme yapabiliriz. Bu özelliğin kullanması ise özel bir kod yapısı kullanması gerekmektedir..

VaryByHeader

Bu alt özellik ise HTML başlığınızın değişimine göre Cachleme yapılmasını sağlar.

VaryByParam

Parametrede gönderilen değere tanımlamasına göre Cache'in bozulmasını sağlayabilirsiniz.

VaryByParam

```
[OutputCache(VaryByParam="id")]
public ActionResult List()
{
    return View();
}
```

Cookie

Türkçe karşılığı "çerezler" olarak bilinen, istemci taraflı bir state management tekniğidir. Cookies tamamen web sayfasına bağlanan kullanıcının browser yani tarayıcısı tarafında tutulan bir yöntemdir. Cookie dosyaları kullanıcının bilgisayarında onunla ilgili bilgileri tutup, gerektiği zaman yine istemci tarafından o bilgiye erişebilmek için kullanılır. Kullanımı oldukça kolay, sunucu tarafını yormayan, veri saklama esnasında geliştiricinin sorumluluk almak zorunda olmadığı bir yöntemdir. Cookie nesnelerinin çok fazla kullanıldığı web sayfalarında üyelik girişlerinde bulunan beni hatırla özelliğinin yapımını aşağıda inceleyelim.

Class

```
public class User
{
    public int Id { get; set; }
    public string UserName { get; set; }
    public string Password { get; set; }
    public bool Remember { get; set; }
}
```

Login Action

```
[HttpGet]
public ActionResult Login()
{
    if (Request.Cookies["login"] != null)
    {
        HttpCookie userCookie = Request.Cookies["login"];
    }
}
```

```
        User user = new User();
        user.UserName = userCookie["userName"];
        user.Password = userCookie["password"];
        return View(user);
    }
    return View();
}
```

Login Action

```
[HttpPost]
public ActionResult Login(User user)
{
    if (user.Remember)
    {
        HttpCookie cerez = new HttpCookie("login");
        cerez.Values.Add("userName", user.UserName);
        cerez.Values.Add("password", user.Password);
        cerez.Expires = DateTime.Now.AddDays(15);
        Response.Cookies.Add(cerez);
        return RedirectToAction("Index");
    }
    else
    {
        return RedirectToAction("Index");
    }
}
```

Clear Cookie

```
[HttpGet]
public ActionResult ClearCookie()
{
    if (Request.Cookies.AllKeys.Contains("login"))
    {
        HttpCookie cookie = Request.Cookies["login"];
        cookie.Expires = DateTime.Now.AddDays(-1);
        Response.Cookies.Add(cookie);
    }
    return RedirectToAction("Index");
}
```