

İÇİNDEKİLER

ASP.NET NEDİR?	2
Visual Studio Üzerinde Asp.NET Uygulaması Açmak	2
ASP.NET SAYFA YAŞAM DÖNGÜSÜ	2
Page Request	3
Start	3
Page Initialization	3
Load	3
Validation	3
Postback Event Handling	3
Rendering	3
Unload	3
ASP.NET WEB FORMLAR	3
Form Elementi ve Öznitelikleri	4
POSTBACK KAVRAMI	4
ASP.NET SUNUCU KONTROLLERİ	5
Sunucu Kontrol Türleri	6
View State	6
HTML KONTROLLERİ	6
Web Sunucu Kontrolleri Ekleme ve Yapılandırmak	7
VALIDATION KONTROLLER	7
Required Field Validator	8
Compare Validator	9
Range Validator	10
Regular Expression Validator	11
Custom Validator	12
DATA KONTROLLER	14
Repeater	14
GridView	16
GridView Üzerinde Update, Delete İşlemleri	19
DataList	21
ASP.NET WEB USER CONTROL	23
MASTER PAGE	25
STATE MANAGEMENT(DURUM YÖNETİMİ)	27
Session State	28
Application State	29
ViewState	32
Query String	33
Cookies	35
CACHING (ÖNBELLEKLEME)	37
Data Caching (Veri Önbellekleme)	37
Cache Parametreleri	38
Output Caching (Çıktı Önbellekleme)	39

ASP.NET NEDİR?

.Net Framework'ü içerisinde web tabanlı dinamik uygulamaları yazmak için kullandığımız Asp'den türetilmiş bir teknolojidir.

Asp.Net yapısını incelemeden önce front end ve back end kavramlarına bakalım. Front end, kullanıcının etkileşime girdiği, web sitesinin görselliğinin olduğu kısımdır. Back end ise kullanıcının görmediği, web sitesinin işlevselliği ile ilgili kodların yazıldığı kısımdır.

Asp.Net uygulamaları temelde front end'de Html, Css, JavaScript gibi Client-Side çalışan teknolojileri, back end'de Server-Side olarak çalışan C#, VB gibi yazılım dillerini kullanır. Böylelikle web tabanlı uygulamaları istediğimiz gibi programlayabiliriz.

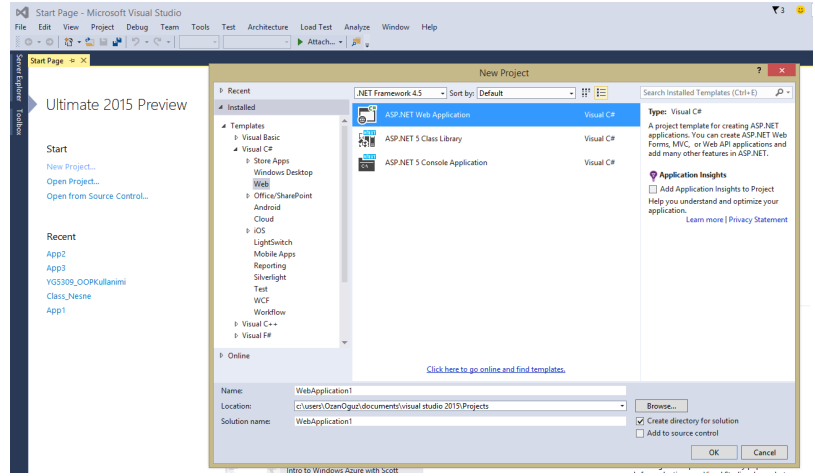
Visual Studio Üzerinde Asp.NET Uygulaması Açmak

Visual Studio üzerinden > New Project > Visual C# > Web > ASP.NET Web Application

Name: Oluşturulacak Asp.Net uygulamasına verilecek isim

Location: Proje dosyasının ve klasörlerinin oluşturulacağı konum..

Solution Name: Proje için oluşturulacak Solution'un klasörünün adı..



ASP.NET SAYFA YAŞAM DÖNGÜSÜ

Asp.Net sayfalarının oluşması esnasında bazı yaşam döngüleri vardır. Bu yapıya Page Life Cycle Management adı verilmektedir. Sayfaların ilk istekle başlayıp kullanıcının karşısına gelene kadar geçmiş olduğu yollar aşağıdaki gibidir.



Page Request

Page Life Cycle başlamadan önce bu aşama çalışır. Sayfa bir kullanıcı tarafından istendiği zaman Asp.NET, bu sayfanın parse ve compile edilip edilmeyeceğine veya sayfayı çalıştırmadan varsa cache'deki versiyonunu gönderip göndermeyeceğine karar verir.

Start

Yaşam döngüsünün ilk adımını temsil eden bu süreçte page nesnesinin Request ve Response gibi property'leri set edilir. Ayrıca page nesnesi, gönderilen request'in bir postback sonucu mu yoksa yeni bir request olarak gelip gelmediğini belirler ve ona göre IsPostBack property'sini set eder. IsPostBack, sayfanın postback modunu bildirir. Bu aşamada ek olarak sayfanın UICulture property'si de set edilir.

Page Initialization

Sayfanın oluşturulma aşamasında sayfa üzerindeki kontroller erişilebilir duruma gelir ve her kontrolün UniqueID property'si set edilir. Eğer o anki request, bir postback sonucu gerçekleşmişse bu aşamada postback data henüz yüklenmemiş ve kontrollerin değeri viewstate'e göre yenilenmemiştir.

Load

Load aşamasında eğer geçerli request bir postback ise kontrollerin değeri ve durumu viewstate içeriğine göre şekillenir.

Validation

Varsa sayfa üzerindeki validator kontrollerin Validate() metodu çağrılır. Validate() metodu, page nesnesi ve içindeki kontrollerin her birinin IsValid property'sini set eder. IsValid property'si, kontrolün doğrulama işleminin başarılı olup olmadığını bildirir.

Postback Event Handling

Eğer request bir postback sonucu gerçekleşmişse, postback'i gerçekleştirmiş veya durumu değişmiş kontrollerin ilgili eventleri tetiklenir.

Rendering

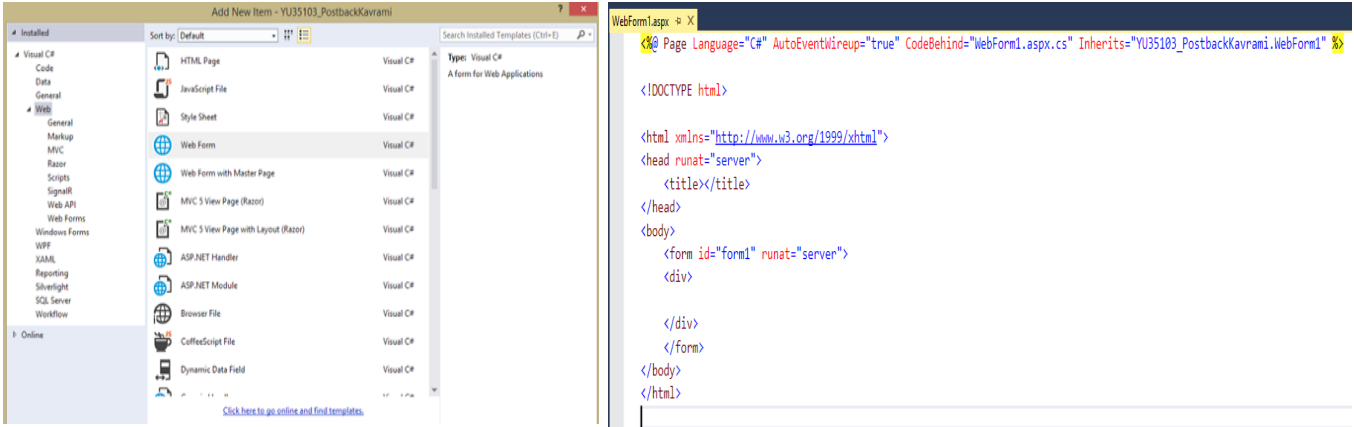
Render işleminden önce page ve kontroller için viewstate nesnesi kaydedilir. Render aşamasında page nesnesi, her kontrolün Render() metodunu çalıştırır. Render() metodu, text writer nesnesini kullanarak kontrolün output'unu page'in Response property'sine ait OutputStream'e (HTTP ile gidecek olan binary içerik) yazar.

Unload

Unload evresi, sayfa tümüyle render edildikten ve istemciye gönderildikten sonra çağrılır. Bu durumda sayfa, artık hafızadan kaldırılmaya (discard) hazırdır. Unload aşamasında, Request ve Response nesneleri temizlenmiş olur.

ASP.NET WEB FORMLAR

Web Formları tarayıcı üzerinde görüntülemek istediğiniz metin ve kontrolleri içeren kapsayıcı olarak düşünebiliriz. Aşağıda ki resimde projeye nasıl Web Form ekleneceğini görebilirsiniz. Oluşturulan web form sayfamızın kod dosyasının belirlendiği, hangi dil için oluşturulduğu, başlığının ne olacağı gibi bilgilerin belirlendiği alandır. Sağ tarafta da standart bir WebForm'un içeriği yer almaktadır.



Head elementi içerisinde sayfa başlığı, Css ve Javascript dosyaları yer alabilir. Web Form'un görünümünü özelleştirmek ve farklı tasarımlar kullanmak için Css dosyalarından yararlanılır. Hali hazırda yer alan Css Frameworklerini kullanabilir ya da kendi yazacağımız Css'ler ile de web form tasarımımızı özelleştirebiliriz.

Javascript ile kullanıcı tarafında (client-side) etkileşim sağlayabiliriz. Kullanıcı tarafı doğrulama işlemleri, html nesnelerinin hareket ettirilmesi, görsel etkileşim, resim, slayt işlemleri gibi işleri javascript üstlenmektedir.

Body elementinde, kullanıcının sayfa içeriğini görüntüleyebilmesi ve tarayıcı tarafından yorumlanabilmesi için görüntülenecek metin ve kontroller yer almalıdır. Gerek ASP.NET Web Form kontrolleri, gerekse html kontrolleri ve elementleri, h1, input, div gibi nesneleri içeren kapsayıcıdır diyebiliriz.

Form Elementi ve Özellikleri

Her ASP.NET Web Form içerisinde bir adet form elementi varsayılan olarak gelmektedir. Eğer biz sunucu tarafında bir işlem yapacaksak, kullandığımız kontrolün form elementi içerisinde yer alması gelmektedir. Form elementi içerisindeki her bir eleman server tarafta işlenebilir özelliğe sahiptir. Ayrıca her web form içerisinde yalnızca bir adet form elementi bulunmalıdır.

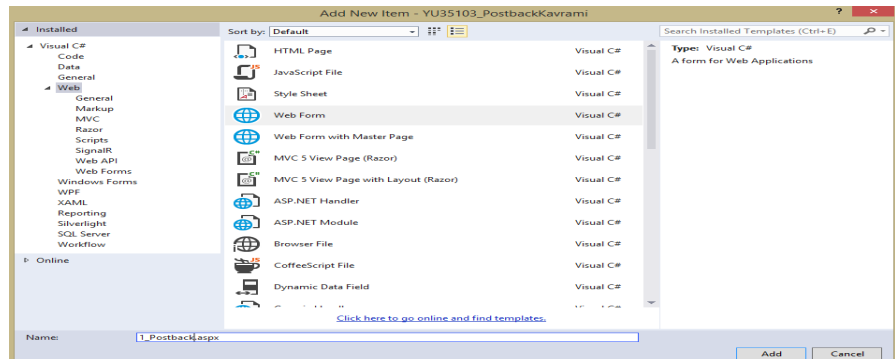
POSTBACK KAVRAMI

Server tarafından yollanan sayfa üzerindeki verilerin tekrar server'a gönderilmesi ve sayfanın tekrardan hazırlanıp kullanıcıya geri gönderilmesi sürecine "Postback" denir.

Sayfanızın ilk istekle (örneğin tarayıcıya adresi yazıp Enter'a basmak vs...) gönderilmesini sağlayan web metodu "GET"tir.

Bir Asp.Net projesi açtık ve bu projeye Web Form ekliyoruz.

Web Form'un adını
1_Postback.aspx olarak belirledik.



Web Form'un Design kısmını kodluyoruz. Bunun için aspx sayfasına bir label ve bir button kontrolü ekliyoruz. Buradaki amacımız button'a bastığımız zaman sayfanın postback sonucu mu yoksa Get metoduyla mı geldiğini öğrenmek.

1_Postback.aspx

```
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label ID="lblKarsilama" runat="server"></asp:Label>
            <br />
            <asp:Button ID="btnPostBackYap" runat="server" Text="PostBack Yap" />
        </div>
    </form>
</body>
```

Ardından F7 butonuyla code behind kısmına geçiyoruz ve sayfanın Load event'ine aşağıdaki gibi kodlarımızı yazıyoruz. Sayfanın postback sonucu geldiğini bize söyleyecek olan şey isPostBack kontrolüdür. Eğer sayfa postback olmuş ise isPostBack bize true değerini dönecektir. Aksi takdirde sayfanın ilk istekle yani Get metoduyla geldiğini belirlemiş olacağız.

1_Postback.aspx.cs

```
protected void Page_Load(object sender, EventArgs e)
{
    if (IsPostBack)
    {
        lblKarsilama.Text = "Sayfa postback sonucu gelmiştir.";
    }
    else
    {
        lblKarsilama.Text = "Sayfa ilk istek sonucu gönderilmiştir.";
    }
}
```

ASP.NET SUNUCU KONTROLLERİ

Bir ASP.NET sayfası oluşturduğumuzda body etiketinin içerisinde form etiketi olduğunu daha önce görmüştük. Bu form etiketinde runat="server" kullanılmaktadır. Eğer bir kontrolün sunucu tarafında işlem görmesini istiyorsak özniteliğinde runat="server" a sahip olması gerekmektedir. Bu özniteliği sadece form etiketine değil, sunucuda çalışacak tüm kontroller içerisine yazmamız gerekir.

Sunucu kontrolleri Html kontrollerine benzer ancak Html'de bir TextBox oluşturmak için input etiketi kullanılırken, ASP.NET'te aşağıdaki şekildedir:

Default.aspx

```
<body>
    <form id="form1" runat="server">
        <div>
            <asp:TextBox ID="TextBox1" runat="server" />
        </div>
    </form>
</body>
```

```
</div>

</form>

</body>
```

Sunucu kontrolleri <asp: ile başlar ve kontrol türü, özellikleri ile devam eder.

Sunucu Kontrol Türleri

Html kontrolleri tarayıcı üzerinde birebir render edilmektedir. Eğer ASP.NET web formumuz içerisinde herhangi bir html kontrol kullanmışsak, tarayıcı sayfamızı yorumlarken bu kontrolleri herhangi bir render işlemine tabi tutmaz. Fakat sunucu kontrollerini tarayıcının anlayabilmesi için bir render işlemine tabi tutar ve html elementleri, nesneleri haline getirir. Bunlar içerisinde standart kontroller, veri kontrolleri, doğrulama ve kullanıcı giriş kontrolleri, yönlendirme kontrolleri yer almaktadır. Yerleşik web kontrollerine ek olarak, ASP.NET kendimize özel denetimler oluşturmamıza olanak sağlar. Eğer şöyle durumlardan biriyle karşı karşıya kalırsak özel kontroller geliştirmek bizim için yararlı olabilir:

- İki veya daha fazla yerleşik web kontrolünün işlevselliğini birleştirmek gerekiyorsa,
- Bir yerleşik kontrolün işlevlerinin genişletilmesi gerekiyorsa,
- Tamamen farklı bir kontrole ihtiyacımız varsa.

View State

ViewState, Web Form üzerinde yer alan kontrollerin durumunu kaydeden gizli bir denetimdir diyebiliriz.

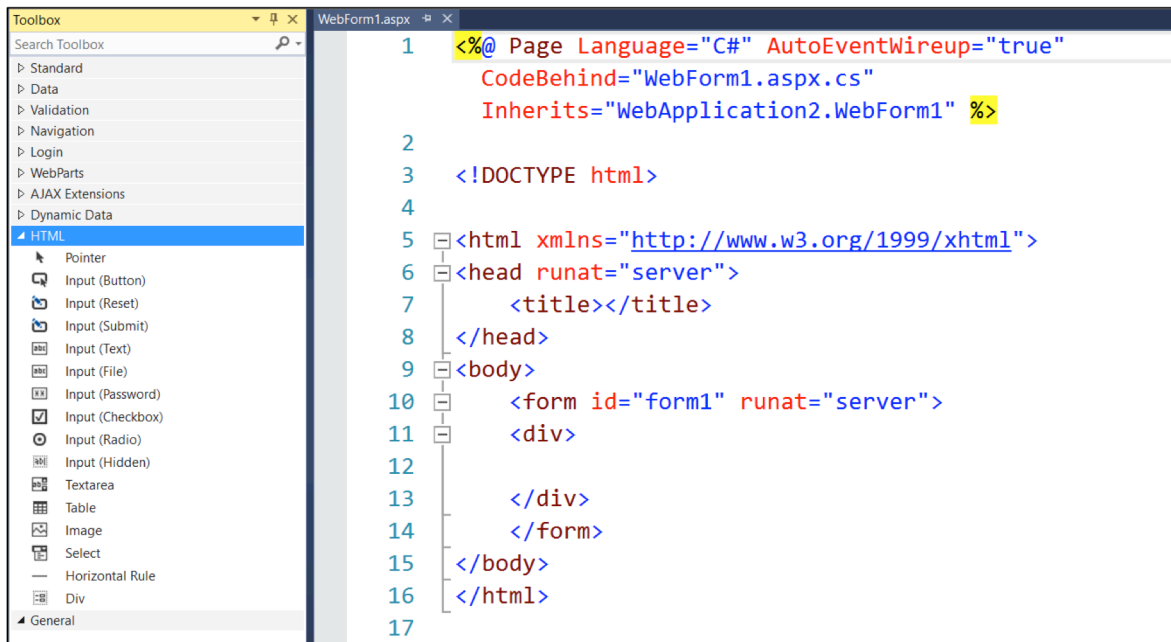
ASP.NET, sayfalar ile kontroller arasındaki değerleri korumak ve kullanabilmek için ViewState'i kullanır. Şu durumlarda ViewState'i kullanabiliriz:

- Sayfaya ait kontrollerin verilerini kalıcı kılmak,
- Kullanıcı sayfayı postback işlemine tabi tuttuğunda değerleri saklayabilmek.

İlerleyen sayfalarda ViewState konusuna detaylı olarak değineceğiz.

HTML KONTROLLERİ

Web form'a html kontrolü eklemek için toolbox'tan html section'ı altından istediğimiz herhangi bir html kontrolünü sürükleyip bırak yöntemiyle ekleyebiliriz.



Properties penceresini kullanarak kontrollerimizin özelliklerini ve aldığı değerleri ayarlayabiliriz.

Normal şartlar altında standart html kontrolleri sunucu tarafı çalışmazlar. Eğer bir html kontrolünün server tarafında işlenebilmesini istiyorsak `runat="server"` özniteliğini eklememiz gerekmektedir.

Html sunucu kontrolleri tarafından sağlanan özelliklerden bazıları şunlardır:

- İstemci tarafında çalışan komutlar yakalanabilir.
- Html sunucu kontrolleri `System.Web.UI.HtmlControls` kütüphanesi altında gruplandırılmış olarak yer almaktadır.
- Sayfa, istemci ve sunucu arasında istek ve dönüş yaptığında, html sunucu kontrolleri devreye girer, değerler otomatik olarak korunur ve tarayıcıya iade edilir.
- ASP.NET doğrulama kontrolleri ile etkileşimde bulunabilir. Bu etkileşim, kullanıcının bizim belirlediğimiz kontrole doğru verileri girmesini kontrol etmemize yardımcı olur.
- Css ile daha farklı görünüm kazandırılabilir.

Web Sunucu Kontrolleri Ekleme ve Yapılandırma

Web form üzerinde html kontrollerinden bahsettik. Eğer html kontrollerine `runat="server"` özniteliğini verirsek sunucu üzerinde işlem yapabileceğini gördük. Web Sunucu kontrolleri ise tamamen sunucu üzerinde çalışan kontrollerdir. Buna örnek olarak `TextBox`, `Button`, `ListBox`, `DropDownList`, `Label`, `Calendar` verilebilir.

Sunucu kontrollerinin hepsinde ID özniteliği mevcuttur. Bu ID değeri bize, code behind tarafında işlem yapmamız için seçici özellik sağlamaktadır. Hangi nesneyi seçeceğimizi, örneğin hangi `TextBox`'taki değeri alacağımız ve hangi `Label`'a değer vereceğimizi bilmemiz için gereklidir.

Sunucu Kontrolü Örnek Kullanımı

```
<body>
  <form id="form1" runat="server">
    <div>
      <asp:Button ID="btnSave" Text="Save" runat="server" />
    </div>
  </form>
</body>
```

Web sunucu kontrolleri programlama yeteneği zengin bir nesne modeli sağlar. Örneğin bir `DropDownList`'in seçilen `index`'i değiştirildiğinde, kullanıcı `TextBox`'a değer girip çıktıktan sonra hangi işlemin çalıştırılacağı, `Button`'a tıklanınca neler yapılacağı gibi. Web sunucu kontrolleri tarayıcıları otomatik olarak algılamaktadır. Hangi tarayıcıda nasıl davranması gerektiğini tespit eder ve uygun biçimde işler.

Css ile görünümü değiştirilebilir ve farklı tasarımlar uygulanabilmektedir. İstediğiniz kontrolü seçerek kendiniz için farklı bir kontrol oluşturabilirsiniz.

VALIDATION KONTROLLER

Web uygulamalarında, dışarıdan gelecek olan (client-side) verilerin bazı tutarlılık kontrollerinden geçmesi gerekmektedir. Bu tutarlılık kontrollerini gerçekleştiren kontrollerimize *Validation Kontroller* adını vermekteyiz. Tüm validation kontrolleri hem client-side hem de server-side kontroller gerçekleştirirler. Client-side kontroller JavaScript yardımıyla gerçekleştirilir. Öyle ki, eğer doğrulama işleminiz başarısız olursa verinin server'a gitmesi daha client'ta iken kesilir. Böylece veri tutarlılığını sağlamış olursunuz. Ancak olur da kullanıcı tarayıcısı aracılığı ile JavaScript'leri kapatırsa bu sefer devreye server-side kontrolleriniz girer ve verilerin yine server tarafta işlenmesine engel olur.

Validation kontrollerde güvenlik amacı ile her zaman server-side kontrollerin de gerçekleştirilmesi gerekmektedir. Özellikle sql ile ilgili işlemlerde, yani veritabanına dışarıdan gelecek olan parametrelerin insert edilmesi söz konusu ise buralarda validation kontrollerini hem client-side hem de server-side olarak kodlamamız gerekmektedir. Aksi takdirde tutarsız ve bozuk verilerle karşılaşırız. Bir başka problem de veri tutarsızlığıdır. Örneğin yaş bilgisinin

girilmesini istediğimiz textbox'a isim girilmesi sistemde parametre tipi uyumsuzluğu oluşturacağı için uygulamamız hataya düşebilir bu da istenmeyen bir durumdur. Bu gibi durumların önüne validation kontrollerle geçebiliriz.

Validation'ların ekstra kazandırdıklarına bakacak olursak:

Client-Side:

- İstemci tarafında doğrulama sağlayarak kodun server'da işlenmesini engeller.
- Anında geri bildirim sağlar.
- Postback döngülerini azaltır.

Server-Side:

- İstemci tarafından geçen kontrolün tekrardan server'da kontrol edilerek verinin tutarlılığının garanti altına alınmasını sağlar.

Required Field Validator

En çok karşılaştığımız tutarlılık kontrolü bir alanın boş geçilip geçilmemesidir. Bu durumu Required Field Validator ile aşmaya çalışacağız. Bazı özellikleri;

ControlToValidate: Hangi kontrol, bu validator'un sorumluluğu altındadır. Bir validator yalnızca tek bir kontrolden sorumlu olabilir!

ErrorMessage: Hata durumunda kullanıcıya ne mesaj verilecek?

Text: Eğer bu özelliğe atama yaparsanız, birkaç sayfa sonra göreceğimiz "Validation Summary" kontrolüne ihtiyacınız var demektir. Text özelliği ilgili validator'un yerine yazar.

RequiredFieldValidator.aspx

```
<body>
    <form id="form1" runat="server">
        <div>
            <asp:TextBox runat="server" ID="txtGirisAlani" />
            <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
                ErrorMessage="Bu alan boş bırakılamaz!" ControlToValidate="txtGirisAlani"
                CssClass="uyari"></asp:RequiredFieldValidator>
            <br />
            <asp:Button Text="Kaydet" ID="btnKaydet" runat="server" />
        </div>
    </form>
</body>
```

Uyarı mesajımızın kırmızı renkli olması için bir stil tanımlaması yapıp Required Field Validator'un CssClass özneliğine oluşturduğumuz class'ı atadık.

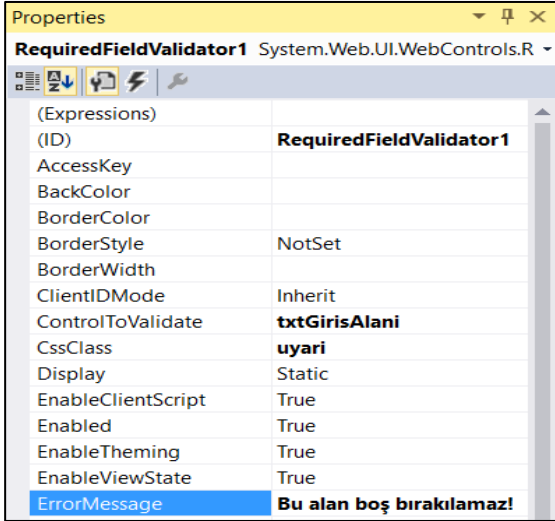
RequiredFieldValidator.aspx

```
<head runat="server">
    <title></title>
    <style type="text/css">
```



```
.uyari {
    color: Red;
}

</style>
</head>
```



Aynı zamanda Design sekmesinden Required Field Validator'un kontrol edeceği kontrolün atamasını, hata durumunda vereceği hata mesajını ve istersek başka özelliklerini properties penceresinden seçerek ayarlayabiliriz.

Kullanıcı ilgili alanı boş geçtiği zaman, alacağı hata (sayfa hiçbir şekilde postback'e uğramaz):

Bu alan boş bırakılamaz!

Compare Validator

Sayfanız üzerinde iki veri girişi kontrolündeki verilerin karşılaştırılması ve eğer karşılaştırma sonucu yanlış ise verilerin işlenmemesini sağlayan validation kontrolümüzdür. İki görünen değeri karşılaştırabildiğiniz gibi, veri girişi kontrolündeki veriyi sabit bir değerle de karşılaştırabilirsiniz. Bazı özellikleri;

ControlToCompare: Referans kontrolünüzü sizden ister. Yani karşılaştırılırken uyumlu olması gereken ana veri hangisidir?

ControlToValidate: Doğrulanacak (referans kontroldeki değerlerle aynı değere sahip) kontrol hangisidir?

ErrorMessage: Hata durumunda kullanıcıya gösterilecek mesaj nedir?

Type: Hangi tipte iki verinin karşılaştırılacağını belirtirsiniz. (Metinsel, sayısal, ondalıklı, tarihsel vs..)

Operator: Sadece eşitlik (Equal) durumu değil, eşit olmama (NotEqual), eşit ya da büyük olma (GreaterThanEqual), eşit ya da küçük olma (LessThanEqual), küçük (LessThan) ya da büyük (GreaterThan) olma ve veri tiplerinin uyumluluğu (DataTypeCheck) da kontrol edilebilir.

CompareValidator.aspx

```
<body>
    <form id="form1" runat="server">
        <div>
            <span>Şifrenizi Giriniz:</span>&nbsp;<asp:TextBox runat="server" ID="txtSifre" /><br />
            <span>Şifrenizi Tekrar Giriniz:</span>&nbsp;<asp:TextBox runat="server"
ID="txtTekrarSifre" />
            <asp:CompareValidator ID="CompareValidator1" runat="server" CssClass="uyari"
ErrorMessage="Şifreler birbiriyle uyumsuz!" ControlToCompare="txtSifre"
ControlToValidate="txtTekrarSifre"></asp:CompareValidator>
```

```
<br />
<asp:Button Text="Onayla" ID="btnOnayla" runat="server" />
</div>
</form>
</body>
```

Şifreler birbiriyle uyumsuz ise kullanıcının alacağı ekran görüntüsü:

Şifrenizi Giriniz:

Şifrenizi Tekrar Giriniz: **Şifreler birbiriyle uyumsuz!**

Range Validator

Bir veri girişi kontrolü içerisindeki verinin, istediğimiz bir değer aralığında olup olmadığını kontrol eden doğrulama kontrolümüzdür. Sadece sayısal değil; metinsel, tarihsel ifadeleri de aralık kontrolü içerisine dahil edebiliriz. Bazı özellikleri;

ControlToValidate: Hangi kontrol, bu validator'un sorumluluğu altındadır?

ErrorMessage: Hata durumunda kullanıcıya ne mesaj gösterilecek?

Type: Hangi tipte veriler kontrol edilecek? Metinsel, sayısal, tarihsel kontroller gerçekleştirilebilir. Metinsel kontrol esnasında, sözlük mantığına göre bir kontrol gerçekleştirecektir. Örneğin Max değere "D", Min değere "A" verirsiniz sözlükte A ile D arasında geçen tüm kelimeler (Ahmet, Burcu, Celal, Deniz) geçerli olurken D'den sonraki tüm kelimeler geçersiz olacaktır (Süleyman, Murat...)

Maximum - Minimum Value: Kontrol esnasında kriter alınacak maksimum ve minimum değerler nelerdir?

RangeValidator.aspx

```
<body>
  <form id="form1" runat="server">
    <div>
      <span>Lütfen yaşınızı giriniz:</span>&nbsp;
      <asp:TextBox runat="server" ID="txtYasBilgisi" />
      <asp:RangeValidator ID="RangeValidator1" runat="server" ErrorMessage="Sadece 18-25
yaş aralığındaki kullanıcılar giriş yapabilir!" CssClass="uyari"
ControlToValidate="txtYasBilgisi" Type="Integer" MinimumValue="18" MaximumValue="25">
      </asp:RangeValidator>
      <br />
      <asp:Button Text="Kaydet" ID="btnKaydet" runat="server" />
    </div>
  </form>
</body>
```

18-25 yaş aralığının dışında bir değer girildiğinde kullanıcının alacağı ekran görüntüsü:

Lütfen yaşınızı giriniz: Sadece 18-25 yaş aralığındaki kullanıcılar giriş yapabilir!

Regular Expression Validator

Regular Expression (Düzenli İfadeler – Regex) bir ifadenin, istediğiniz formata uygun olup olmaması durumunu kontrol etmektedir. Örneğin, web sitesi istediğiniz alanda http olması, e-mail istediğinizde @ olması, bir şifrede en az 3 adet rakam olması gibi kurallı ifadeleri bu validator ile kontrol edebilirsiniz. Dünya üzerinde birçok Regex Pattern'i (düzenli ifade dokusu) üreten geliştiriciler <http://regexlib.com> sitesinde buluşmakta ve pattern'lerini dünyayla paylaşmaktadır. Sizler de uygulamalarınızda bu pattern'leri kullanabilirsiniz. Bazı özellikleri;

ControlToValidate: Hangi kontrol, bu validator'un sorumluluğu altındadır?

ErrorMessage: Hata durumunda kullanıcıya ne mesaj gösterilsin?

ValidationExpression: Bu validator bağlı bulunduğu kontrolden gelecek olan veriyi hangi dokuya göre süzecek?

Aşağıda belirtilen ValidationExpression alanlarında istersek hazır expression'ları kullanabilir, ya da <http://regexlib.com> adresinden custom expression'ları kullanabiliriz. Aşağıdaki örnekte e-mail adresinin ve doğum tarihinin uygun formatta olup olmadığını kontrol eden expression deseni kullandık.

RegularExpressionValidator.aspx

```
<body>
    <form id="form1" runat="server">
        <div>
            <span>Lütfen Mail Adresinizi Giriniz:</span>&nbsp;<asp:TextBox runat="server"
ID="txtMailAdresi" />
            <asp:RegularExpressionValidator ID="RegularExpressionValidator1" CssClass="uyari"
                runat="server" ErrorMessage="Mail adresiniz uygun formatta değildir!"
ControlToValidate="txtMailAdresi"
                ValidationExpression="\w+([-+.'\w+)*@\w+([-.\w+)*\.\w+([-.\w+)*">
            </asp:RegularExpressionValidator>
            <br />
            <asp:Button Text="Kaydet" ID="btnKaydet" runat="server" />
            <hr />
            <span>Lütfen Doğum Tarihinizi Giriniz:</span>&nbsp;<asp:TextBox runat="server"
ID="txtDogumTarihi" />
            <asp:RegularExpressionValidator ID="RegularExpressionValidator2" CssClass="uyari"
                runat="server" ErrorMessage="Doğum tarihi formatı şu şekilde olmalıdır:
AA/GG/YYYY (1900 - 2099)" ControlToValidate="txtDogumTarihi" ValidationExpression="^(([1-
9])|([01-9])|([10-2]))\/((([0-9])|([0-2][0-9])|(3[0-1]))\/((([0-9][0-9])|([1-2][0,9][0-9][0-
9]))$"></asp:RegularExpressionValidator>
            <br />
            <asp:Button Text="Kaydet" ID="btnKontrolEt" runat="server" />
        </div>
    </form>
</body>
```

Eğer kullanıcı istenilen formatlarda giriş yapmaz ise karşılaşılabilecek hata ekranı:

Lütfen Mail Adresinizi Giriniz:	<input type="text" value="test.com"/>	Mail adresiniz uygun formatta değildir!
<input type="button" value="Kaydet"/>		
Lütfen Doğum Tarihinizi Giriniz:	<input type="text" value="1988"/>	Doğum tarihi formatı şu şekilde olmalıdır: AA/GG/YYYY (1900 - 2099)
<input type="button" value="Kaydet"/>		

Custom Validator

Eğer .NET kütüphanesinin size sunduğu validator'ların karşılayamadığı bir kontrol olduğunu düşünüyorsanız, sizin için bir validator yazma template'i açacaktır. Gerekli ayarlarına istediğiniz değerleri verdikten sonra kendi validator'unuzu yazmış olursunuz. Unutmamanız gereken, hem client- taraflı (JavaScript) hem de server- taraflı tüm kontrollerini kendinizin yapma gerekliliğidir. Bazı özellikleri;

ControlToValidate: Hangi kontrol, bu validator'un sorumluluğu altındadır?

ErrorMessage: Hata durumunda kullanıcıya ne tür bir mesaj gösterilsin?

ClientValidationFunction: JavaScript fonksiyonunu da kendimiz yazacağımız için, bu ayara o fonksiyonun adını vermelisiniz.

ServerValidate(Event): Client-side kontrolünü yaptığınız gibi bir de Server-Side kontrolü de kendiniz yapmalısınız. Bunun için de ilgili event'i doldurmalısınız.

CustomValidator.aspx

```
<body>
    <form id="form1" runat="server">
        <div>
            <span>En az altı karakterlik bir şifre giriniz:</span>&nbsp;
            <asp:TextBox runat="server" ID="txtSifreAlani" />
            <asp:CustomValidator ID="CustomValidator1" runat="server" ErrorMessage="Şifreniz en
az altı karakterden oluşmalıdır!" CssClass="uyari" ClientValidationFunction="Validate"
ControlToValidate="txtSifreAlani" OnServerValidate="CustomValidator1_ServerValidate">
            </asp:CustomValidator>
            <br />
            <asp:Button Text="Kaydet" ID="btnKaydet" runat="server" />
        </div>
    </form>
</body>
```

Client-Side çalışacak olan Validation kontrolünü JavaScript ile yazıyoruz.

CustomValidator.aspx

```
<head runat="server">
    <title></title>
    <style type="text/css">
        .uyari {
            color: Red;
        }
    </style>
</head>
```

```
</style>
<script type="text/javascript">
    //Validate adında bir function oluşturuyoruz
    function Validate(sender, args) {
        //txtSifreAlani id'sine sahip olan kontrolün üzerindeki değeri v adındaki değişkene
        atıyoruz.
        var v = document.getElementById("txtSifreAlani").value;
        //Eğer girilen değerin karakter uzunluğu 6'dan küçükse hata fırlatıyoruz.
        if (v.length < 6) {
            args.IsValid = false;
        }
        else {
            //Eğer 6 karaktere eşit ve büyükse isValid özelliğini true atayıp işleme izin
            veriyoruz.
            args.IsValid = true;
        }
    }
</script>
</head>
```

Server-Side çalışacak kontrolü ise code-behind üzerinden ServerValidate event'ine yazıyoruz.

CustomValidator.aspx.cs

```
protected void CustomValidator1_ServerValidate(object source, ServerValidateEventArgs args)
{
    //args size bu validator'la ilgili temel verileri döndürür. Örneğin, bağlı bulunduğunuz
    TextBox'a ne yazıldı gibi. IsValid özelliği ise server taraflı kontrolün bu değere geçit verip
    vermeyeceğini kontrol ettiğiniz özelliktir.
    if (args.Value.Length < 6)
    {
        args.IsValid = false;
    }
    else
    {
        args.IsValid = true;
    }
}
```

DATA KONTROLLER

Asp.NET'te verileri düzenli ve kolayca listeleyebileceğimiz hazır kontroller vardır. Bu kontrollere Data Kontroller adı verilmektedir. Bazı Data Kontroller sadece listeleme yaparken, bazıları veri güncellenmesi, silinmesi, sıralanması, sayfanın girilmesi gibi ekstra özellikleri de sağlamaktadır.

Repeater

Repeater, tasarlanan bir kalıp çerçevesinden belirlenen bir veri kaynağındaki tüm verilerin baştan sona okunarak, tekrarlı bir şekilde HTML çıktısı halinde ekrana basılmasını sağlayan ASP.NET kontrolüdür. Data kontroller arasında en hızlı veri gösterme kontrolüdür. Bu kontrol üzerinden Insert, Update, Delete işlemleri gerçekleştirilemez! Sadece veri gösterme işini yapar. Bir data kontrolünün, herhangi bir şablonuna veri kaynağının bir bölümünü (DataTable => DataColumnName, Class => Property, vs...) bağlamak için #Eval("bolumAdi") ifadesi kullanılır. Eval ile bind edilmiş (yüklenmiş) veri ilgili kontrole atanır ve # ile yazdırılır.

Repeater hiçbir şekilde size bir HTML desteği sunmaz. Tüm tasarımsal içerik yazılımcının sorumluluğu altındadır. Yalnızca bir takım template'leri kullanıma açacaktır. Bu template'ler;

ItemTemplate: Olmazsa olmaz template'dir. Repeater bağlanan veri kaynağındaki verilerin sıralı halde gösterilmesine olanak sağlar.

AlternatingItemTemplate: ItemTemplate'e alternatif olarak bir görünüm oluşturmamızı sağlar. Kullanımı tıpkı ItemTemplate gibidir.

FooterTemplate: Sadece bir defa çalışan template'dir ve tüm veriler listelendikten sonra çalışır. En son aşama olarak adlandırılabilir.

HeaderTemplate: En başta çalışan template'tir. Başlık görevi görür. İsterseniz bir başlık atayabilir ya da kısa bir açıklama gerçekleştirebilirsiniz.

SeperatorTemplate: Ayraç template'i olarak adlandırılır. Bir template'i diğerinden ayırmak için kullanabilirsiniz. (Kısa bir çizgi çekmek vs...)

Repeater.aspx

```
<body>
  <form id="form1" runat="server">
    <div>
      <asp:Repeater ID="rptUrunler" runat="server">
        <ItemTemplate>
          <div class="urunItem">
            <!-- Database'den çekilen ProductName'lerin sırayla buraya basılmasını
sağlıyoruz. --%>
            <%#Eval("ProductName") %>
            <br />
            <!-- Birim Fiyat'ın para formatında gözükmelerini sağladık. --%>
            Fiyat : <%#Eval("UnitPrice","{0:C2}") %>
            <br />
            Stok Miktarı : <%#Eval("UnitsInStock") %>
          </div>
        </ItemTemplate>
        <AlternatingItemTemplate>
          <div class="urunItem" style="background-color:aliceblue">
```

```
<%#Eval("ProductName") %>
<br />
Fiyat : <%#Eval("UnitPrice","{0:C2}") %>
<br />
Stok Miktarı : <%#Eval("UnitsInStock") %>
</div>
</AlternatingItemTemplate>
<SeparatorTemplate>
<hr />
</SeparatorTemplate>
</asp:Repeater>
</div>
</form>
</body>
```

Bu örnekte ItemTemplate ile tekrar edecek tasarımımızı oluşturduk. AlternatingItemTemplate içerisinde ise farklı bir arkaplan rengi vererek tasarıma alternatif bir görüntü oluşturduk. SeparatorTemplate sayesinde ise her bir veri arasına çizgi çekilmesini sağladık.

Repeater.aspx

```
<head runat="server">
  <title></title>
  <style>
    .urunItem{
      width:450px;
      height:70px;
      border:2px solid #808080;
      background-color:hotpink;
      font-size:13px;
      font-weight:bold;
      font-family:'Comic Sans MS';
      margin-bottom:5px;
    }
  </style>
</head>
```

urunItem class'ını oluşturarak stil eklemesi yaptık.

Repeater.aspx.cs

```
protected void Page_Load(object sender, EventArgs e)
```

```
{  
    SqlDataAdapter da = new SqlDataAdapter("Select * from Products",  
    "Server=.;Database=NORTHWND;Integrated Security=true");  
    DataTable dt = new DataTable();  
    da.Fill(dt);  
  
    rptUrunler.DataSource = dt;  
    rptUrunler.DataBind();  
}
```

Repeater'ın kaynağını code-behind tarafında belirliyoruz. Bunun için Products tablomuzdan tüm verileri çekerek rptUrunler id'sini verdiğimiz repeater'ımızın DataSource'unu dolduruyoruz.

Web ortamında, Windows ortamından farklı olarak, bir data kontrolüne vermiş olduğunuz veri kaynağındaki tüm verilerin yüklenmesi (Eval metotlarının çalıştırılması) için "DataBind()" metodu mutlaka kullanılmalıdır. Yukarıdaki kodlar çalıştırıldığında resimdeki sonuç ekranda görüntülenecektir.

Chai Fiyat : 18,00 ₺ Stok Miktarı : 39
Chang Fiyat : 19,00 ₺ Stok Miktarı : 17
Aniseed Syrup Fiyat : 10,00 ₺ Stok Miktarı : 13
Chef Anton's Cajun Seasoning Fiyat : 22,00 ₺ Stok Miktarı : 53
Chef Anton's Gumbo Mix Fiyat : 21,35 ₺ Stok Miktarı : 0

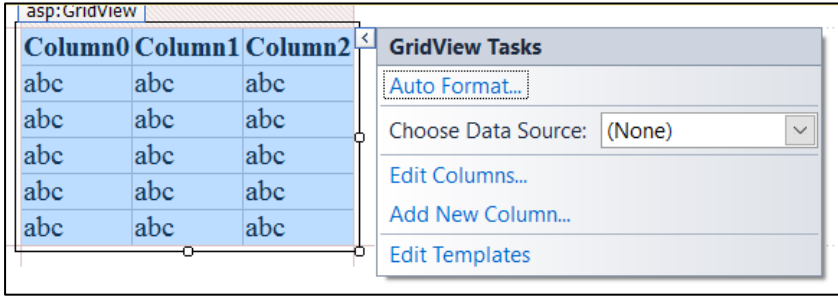
GridView

GridView kontrolü, .NET System.Windows kütüphanesindeki DataGridView kontrolünün web versiyonudur. Data kontrolleri arasında en güçlü ve en kullanışlı kontrollerden bir tanesidir. GridView üzerinde Update, Delete işlemlerini gerçekleştirebiliriz.

GridView.aspx

```
<body>  
    <form id="form1" runat="server">  
        <div>  
            <asp:GridView runat="server"></asp:GridView>  
        </div>  
    </form>  
</body>
```


Ekrana bir GridView attıktan sonra Design sekmesindeki smart tag'e tıklayarak Auto Format seçerseniz, hazır GridView tasarımıyla istediklerinizi kullanabilirsiniz.



Column0	Column1	Column2
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc

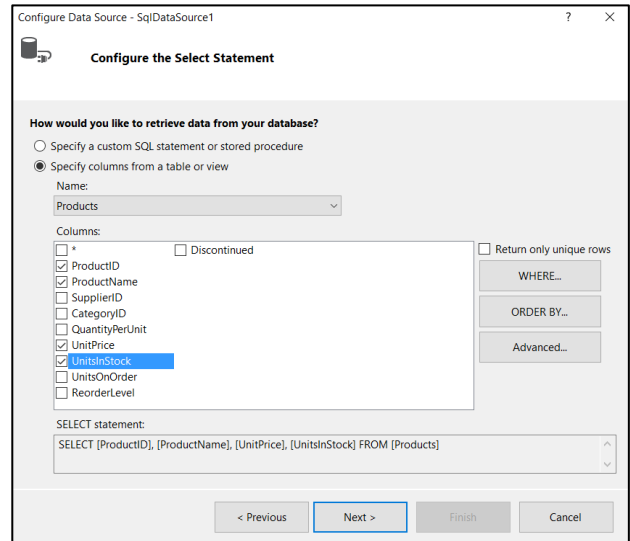
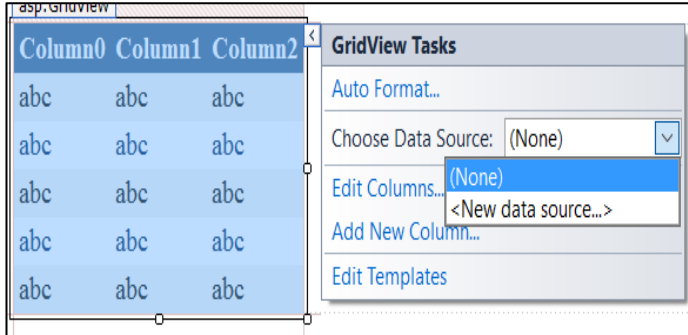
Bir tasarım seçtikten sonra tekrar Source kısmına dönerseniz, kodlarınızın otomatik oluşturulduğunu görebilirsiniz.

GridView.aspx

```
<body>
  <form id="form1" runat="server">
    <div>
      <asp:GridView runat="server" CellPadding="4" ForeColor="#333333" GridLines="None">
        <AlternatingRowStyle BackColor="White" ForeColor="#284775" />
        <EditRowStyle BackColor="#999999" />
        <FooterStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
        <HeaderStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
        <PagerStyle BackColor="#284775" ForeColor="White" HorizontalAlign="Center" />
        <RowStyle BackColor="#F7F6F3" ForeColor="#333333" />
        <SelectedRowStyle BackColor="#E2DED6" Font-Bold="True" ForeColor="#333333" />
        <SortedAscendingCellStyle BackColor="#E9E7E2" />
        <SortedAscendingHeaderStyle BackColor="#506C8C" />
        <SortedDescendingCellStyle BackColor="#FFFDF8" />
        <SortedDescendingHeaderStyle BackColor="#6F8DAE" />
      </asp:GridView>
    </div>
  </form>
</body>
```

Yine design sekmesinden New Data Source diyerek ekstra kod yazmadan GridView'inize kaynak belirtebilirsiniz.

Connection String'inizi oluşturduktan sonra çıkan ekranda istediğiniz tablo üzerinden istediğiniz sütunları çekerek bir Select sorgusunu otomatik oluşturursunuz. Kod kısmına döndüğünüzde artık GridView'inizin altına SqlDataSource eklenmiş olacaktır.



GridView.aspx

```
<body>
  <form id="form1" runat="server">
    <div>
      <asp:GridView runat="server" CellPadding="4" ForeColor="#333333" GridLines="None">
        <AlternatingRowStyle BackColor="White" ForeColor="#284775" />
        <EditRowStyle BackColor="#999999" />
        <FooterStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
        <HeaderStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
        <PagerStyle BackColor="#284775" ForeColor="White" HorizontalAlign="Center" />
        <RowStyle BackColor="#F7F6F3" ForeColor="#333333" />
        <SelectedRowStyle BackColor="#E2DED6" Font-Bold="True" ForeColor="#333333" />
        <SortedAscendingCellStyle BackColor="#E9E7E2" />
        <SortedAscendingHeaderStyle BackColor="#506C8C" />
        <SortedDescendingCellStyle BackColor="#FFFDF8" />
        <SortedDescendingHeaderStyle BackColor="#6F8DAE" />
      </asp:GridView>
      <asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString="<%%$
ConnectionStrings:NORTHWNDConnectionString %>" SelectCommand="SELECT [ProductID],[ProductName],
[UnitPrice],[UnitsInStock] FROM [Products]"></asp:SqlDataSource>
    </div>
  </form>
</body>
```

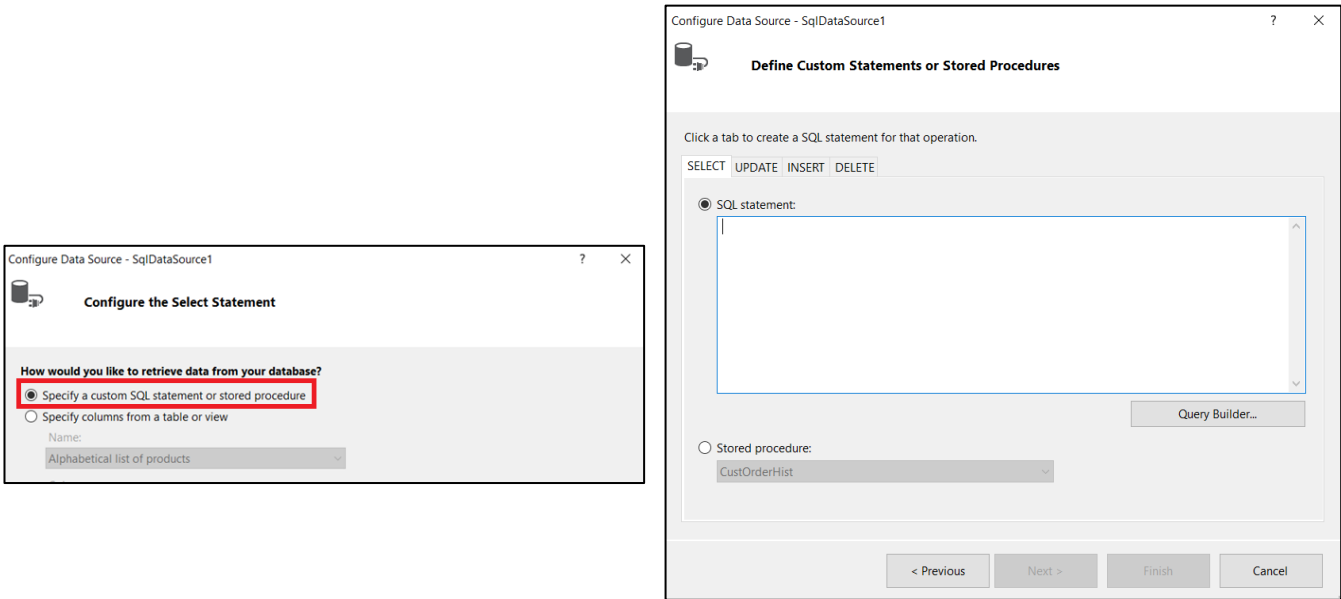
Yukarıdaki kodlar çalıştırıldığında ekranda resimde ki sonuç gösterilecektir.

ProductID	ProductName	UnitPrice	UnitsInStock
1	Chai	18,0000	39
2	Chang	19,0000	17
3	Aniseed Syrup	10,0000	13
4	Chef Anton's Cajun Seasoning	22,0000	53
5	Chef Anton's Gumbo Mix	21,3500	0
6	Grandma's Boysenberry Spread	25,0000	120
7	Uncle Bob's Organic Dried Pears	30,0000	15
8	Northwoods Cranberry Sauce	40,0000	6
9	Mishi Kobe Niku	97,0000	29
10	Ikura	31,0000	31

GridView Üzerinde Update, Delete İşlemleri

Ekrana bir GridView attıktan sonra yine Design kısmından tasarım seçip, yeni bir SqlDataSource ekliyoruz. Ancak bu sefer karşımıza çıkan ekranda özel olarak Select, Insert, Update, Delete sorgularını kendimiz yazacağız. Bu yüzden geçen sefer yaptığımız seçimi değiştiriyoruz.

Karşımıza çıkan pencerede Select, Update, Insert ve Delete için kendi sorgularımızı belirliyoruz.



Bu adımdan sonra artık kod tarafına döndüğümüzde aşağıdaki gibi Columns etiketleri arasında getirmiş olduğumuz sütun isimlerini görüyoruz. Aynı zamanda SqlDataSource içerisinde yukarıdaki ekranda yazdığımız kodlar otomatik oluşturuluyor.

GridView.aspx

```
<body>
    <form id="form1" runat="server">
        <div>
            <asp:GridView runat="server" AutoGenerateColumns="False" BackColor="White"
BorderColor="#CCCCCC" BorderStyle="None" BorderWidth="1px" CellPadding="3"
DataKeyNames="CategoryID" DataSourceID="SqlDataSource1">
```

```
<Columns>
    <asp:BoundField DataField="CategoryID" HeaderText="CategoryID"
InsertVisible="False" ReadOnly="True" SortExpression="CategoryID" />
    <asp:BoundField DataField="CategoryName" HeaderText="CategoryName"
SortExpression="CategoryName" />
    <asp:BoundField DataField="Description" HeaderText="Description"
SortExpression="Description" />
</Columns>
<FooterStyle BackColor="White" ForeColor="#000066" />
<HeaderStyle BackColor="#006699" Font-Bold="True" ForeColor="White" />
<PagerStyle BackColor="White" ForeColor="#000066" HorizontalAlign="Left" />
<RowStyle ForeColor="#000066" />
<SelectedRowStyle BackColor="#669999" Font-Bold="True" ForeColor="White" />
<SortedAscendingCellStyle BackColor="#F1F1F1" />
<SortedAscendingHeaderStyle BackColor="#007DBB" />
<SortedDescendingCellStyle BackColor="#CAC9C9" />
<SortedDescendingHeaderStyle BackColor="#00547E" />
</asp:GridView>

<asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString="<%"$
ConnectionStrings:NORTHWNDConnectionString %>" DeleteCommand="Delete from Categories where
CategoryID=@CategoryID" InsertCommand="Insert into Categories (CategoryName,Description) values
(@CategoryName,@Description)" SelectCommand="Select CategoryID,CategoryName,Description from
Categories" UpdateCommand="Update Categories Set CategoryName=@CategoryName,
Description=@Description where CategoryID=@CategoryID">
    <DeleteParameters>
        <asp:Parameter Name="CategoryID" />
    </DeleteParameters>
    <InsertParameters>
        <asp:Parameter Name="CategoryName" />
        <asp:Parameter Name="Description" />
    </InsertParameters>
    <UpdateParameters>
        <asp:Parameter Name="CategoryName" />
        <asp:Parameter Name="Description" />
        <asp:Parameter Name="CategoryID" />
    </UpdateParameters>
</asp:SqlDataSource>
</div>
</form>
</body>
```

	CategoryID	CategoryName	Description
Edit Delete Select	0	abc	abc
Edit Delete Select	1	abc	abc
Edit Delete Select	2	abc	abc
Edit Delete Select	3	abc	abc
Edit Delete Select	4	abc	abc
Edit Delete Select	5	abc	abc
Edit Delete Select	6	abc	abc
Edit Delete Select	7	abc	abc
Edit Delete Select	8	abc	abc
Edit Delete Select	9	abc	abc

1 2

SqlDataSource - SqlDataSource1

GridView Tasks
 Auto Format...
 Choose Data Source: SqlDataSource1
 Configure Data Source...
 Refresh Schema
 Edit Columns...
 Add New Column...
☒ Enable Paging
☒ Enable Sorting
☒ Enable Editing
☒ Enable Deleting
☒ Enable Selection
 Edit Templates

Düzenle, sil ve seç linklerini tasarımımıza eklemek için Design kısmından ilgili ComboBox'ları seçiyoruz. İsterseniz otomatik sayfalama yapmak için Paging özelliğini de aktif edebilirsiniz.

Bunun için PageSize özneliğini GridView'inize ekleyerek sayfa başına kaç kayıt gözükmesini istediğinizi belirleyebilirsiniz.

Artık istediğiniz veriyi düzenleye basarak açılan TextBox ile güncelleyebilirsiniz. Yine sil ve seç işlemlerini de buradan hızlıca yapabilirsiniz. Ayrıca GridView'inizin Source kısmında oluşan kodları değiştirerek tasarımınızı istediğiniz gibi düzenleyebilirsiniz.

	CategoryID	CategoryName	Description
Güncelleştir İptal	1	Beverages	Güzel içecekler, kahveler...
Düzenle Sil Sec	2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
Düzenle Sil Sec	3	Confections	Desserts, candies, and sweet breads
Düzenle Sil Sec	4	Dairy Products	Cheeses
Düzenle Sil Sec	5	Grains/Cereals	Breads, crackers, pasta, and cereal

1 2

GridView, DataSource'unu bu şekilde almak zorunda değildir. Code-behind üzerinden GridView'inize bir DataSource atayabilirsiniz. Bu yöntem büyük projelerde sizin için çok daha faydalı olacaktır.

DataKeyNames: Bir satırın arka planında bir ya da birden fazla değerin tutulabilmesi için kullanılan bir property'dir. Genellikle Primary Key alanlar için kullanılır. Ancak senaryoya göre bazen tüm kolonlar, bazen detay kolonlar anahtar kolon olarak kullanılabilir. Bilinmesi gereken en önemli özellik, bir satırın arka planında saklanan değere ulaşabilmek için o satırın mutlaka "selected" bir satır olması gerekmektedir.

DataList

Verilerinizi tablo formatında listelemeye olanak sağlayan oldukça güçlü bir kontroldür. Bu kontrol üzerinde de EditItemTemplate yardımıyla Update ve Delete gibi işlemleri de gerçekleştirebilirsiniz.

DataList.aspx

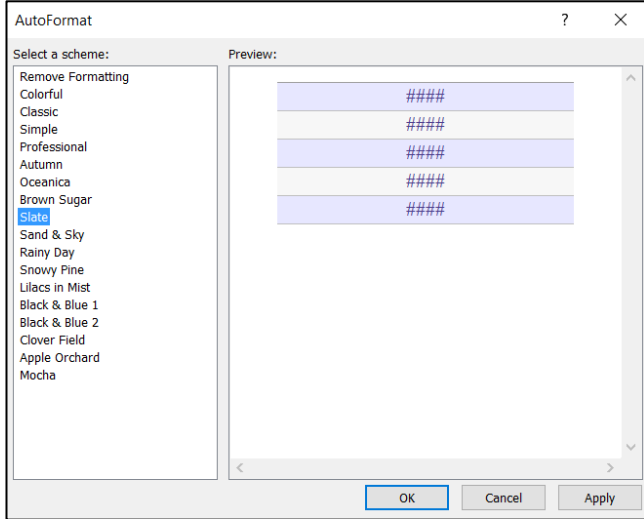
```

<body>
  <form id="form1" runat="server">
    <div>
      <asp:DataList runat="server">
        <ItemTemplate></ItemTemplate>
      </asp:DataList>
    </div>
  </form>

```

```
</body>
```

DataList te aynı GridView kontrolü gibi ItemTemplate ile gelecektir. Yine Design sekmesinden istediğimiz formatı seçebiliriz.



DataList kontrolünde de yine aynı sekme içerisinde SqlDataSource ekleyebiliriz.

Bu örneğimizde kendi veri kaynağımızı kod üzerinde oluşturalım.

Source kısmına geçtiğimiz anda tasarım kodlarının DataList'i etkilediğini görebiliriz.

DataList.aspx

```
<body>
  <form id="form1" runat="server">
    <div>
      <asp:DataList runat="server" BackColor="White" BorderColor="#E7E7FF" BorderStyle="None"
        BorderWidth="1px" CellPadding="3" GridLines="Horizontal">
        <AlternatingItemStyle BackColor="#F7F7F7" />
        <FooterStyle BackColor="#B5C7DE" ForeColor="#4A3C8C" />
        <HeaderStyle BackColor="#4A3C8C" Font-Bold="True" ForeColor="#F7F7F7" />
        <ItemStyle BackColor="#E7E7FF" ForeColor="#4A3C8C" />
        <ItemTemplate>
          <asp:Label Text='<%#Eval("AdSoyad") %>' runat="server" ID="lblAdSoyad" />
          <br />
          <asp:Label Text='<%#Eval("Unvan") %>' runat="server" ID="lblUnvan" />
        </ItemTemplate>
        <SelectedItemStyle BackColor="#738A9C" Font-Bold="True" ForeColor="#F7F7F7" />
      </asp:DataList>
    </div>
  </form>
</body>
```

ItemTemplate içerisine tasarımda tekrar etmesini istediğimiz kodları yazıyoruz. Bu örnekte Northwind üzerinden çalışanların Ad Soyad ve Unvan bilgilerini çektik.

Employees tablomuzdan TitleOfCourtesy, FirstName ve LastName'i birleştirerek AdSoyad isimlendirmesini yapıyoruz. #Eval içerisinde de yine AdSoyad isimlendirmesi ise birleştirdiğimiz kolonlarımıza ulaşıyoruz.

DataList.aspx.cs

```
protected void Page_Load(object sender, EventArgs e)
{
    if (IsPostBack) return;

    SqlDataAdapter da = new SqlDataAdapter("Select TitleOfCourtesy + ' ' + FirstName + ' ' + LastName as AdSoyad, Title as Unvan from Employees", "Server=.;Database=NORTHWND;Integrated Security=true");

    DataTable dt = new DataTable();
    da.Fill(dt);

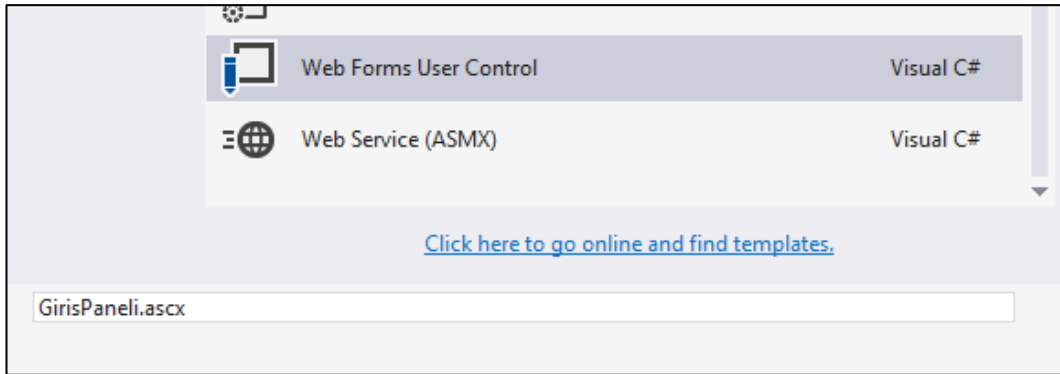
    dtlCalisanlar.DataSource = dt;
    dtlCalisanlar.DataBind();
}
```

Yukarıdaki kodlar çalıştırıldığında ekranda aşağıdaki sonuç gösterilecektir.

Ms. Nancy Davolio Sales Representative
Dr. Andrew Fuller Vice President, Sales
Ms. Janet Leverling Sales Representative
Mrs. Margaret Peacock Sales Representative

ASP.NET WEB USER CONTROL

Asp.Net Web User Control, aynı bir Asp.Net sayfası gibi davranan, kendi event'leri olan oldukça kullanışlı bir kontroldür. Web User Control bize Asp.Net sayfalarımızda tekrarladığımız, client-side (yani kullanıcı tarafında yorumlanan) Html ve Css kodlarımızın, server-side çalışan (Code Behind yani C#) kodlarımızı yazdığımız alanların tek bir yerden yönetilmesine olanak tanımaktadır. Örnek vermek gerekirse kullanıcı girişi yapılabilmesi için bir giriş paneli tasarladınız, bu giriş panelini ana sayfada ve başka bir sayfada göstermek istiyorsunuz. İki sayfaya birden aynı kodları uygulamak yerine oluşturacağınız bir Web User Control ile bu işlemi tek bir yerden gerçekleştirmeniz mümkündür.



GirisPaneli.ascx isminde bir Web User Control oluşturunuz. Oluşturduğumuzda boş bir sayfada sadece aşağıdaki Page Directive kısmı gelecektir.

```
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="GirisPaneli.ascx.cs"
    Inherits="WebUserControl.GirisPaneli" %>
```

Artık bu kodların altına sayfalarda ortak olarak kullanmak istediğimiz tasarımı oluşturabiliriz. Örnek olarak bir üye kayıt tasarımı oluşturalım.

GirisPaneli.ascx

```
<table border="1">
  <tr>
    <td>Üye adınızı giriniz:</td>
    <td>
      <asp:TextBox ID="txtUyeAdi" runat="server" />
    </td>
  </tr>
  <tr>
    <td>Şifrenizi giriniz:</td>
    <td>
      <asp:TextBox ID="txtSifre" runat="server" />
    </td>
  </tr>
  <tr>
    <td>
      <a href="#">Şifremi Unuttum</a>
    </td>
    <td>
      <asp:Button Text="Giriş Yap" ID="btnGirisYap" runat="server" />
    </td>
  </tr>
</table>
```


Oluşturduğumuz bu kontrol söylediğimiz gibi aynı bir Asp.Net sayfası gibi davranır ancak mutlaka bir Asp.Net sayfası içerisinde olmak zorundadır. Web User Controllerin de Code Behind bölümünün olduğundan bahsetmiştik. Böylece kontrolümüz C# kodlarını yazdığımız alanda gerekli işlemleri gerçekleştirebilir.

Web User Control'u bir Asp.Net sayfasına Solution Explorer'dan sürükleyip bırak yöntemiyle ekleyebilirsiniz.

```

1  <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
    Inherits="WebUserControl.Default" %>
2
3  <%@ Register Src="~/GirisPaneli.ascx" TagPrefix="uc1" TagName="GirisPaneli" %>

```

Ekledikten sonra sayfanızın en üstüne bir Page Directive ekleyecektir. Bu Web User Control'un yolunu belirtecek alandır. Page Directive içerisindeki TagName kısmı ise sayfanın body bölümünde bulunan Web User Control'un orada bulunmasını sağlayan etiketi belirlemektedir.

Aşağıdaki örnekte gösterilmiş olduğu gibi sürükleyip bırak ile kontrolü eklemiş olduğunuz alanda Web User Control içeriğiniz ve Code Behind bölümünde bulunan C# kodlarınız çalıştırılacaktır.

```

<body>
    <form id="form1" runat="server">
        <div>
            <uc1:GirisPaneli runat="server" id="GirisPaneli" />
        </div>
    </form>
</body>

```

Default.aspx sayfamızın code-behind tarafında eğer Web User Control'un verilerine ulaşmak istersek FindControl'u kullanmamız gerekir. FindControl içerisine verdiğiniz id'ye sahip kontrolü size getirir. Bu kontrol'ün özelliklerine ulaşabilmek içinse önce bir cast işlemine tabi tutmanız gerekir.

Default.aspx.cs

```

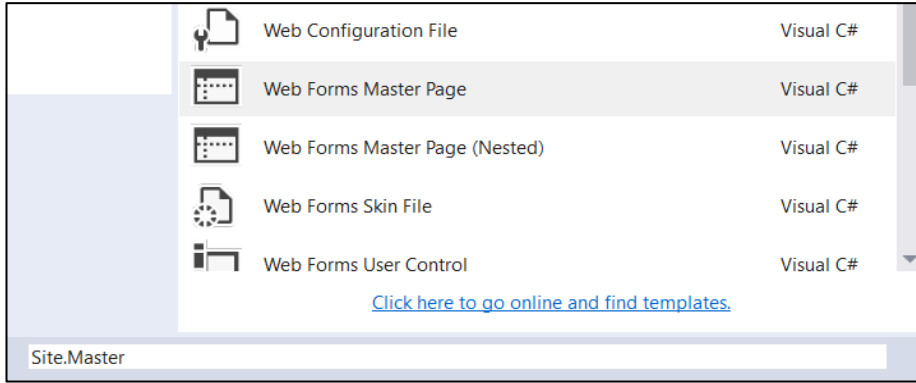
protected void btnGirisYap_Click(object sender, EventArgs e)
{
    string girilenAd = (GirisPaneli.FindControl("txtUyeAdi") as TextBox).Text;
    string girilenSifre = (GirisPaneli.FindControl("txtSifre") as TextBox).Text;
    Response.Write(girilenAd + " " + girilenSifre);
}

```

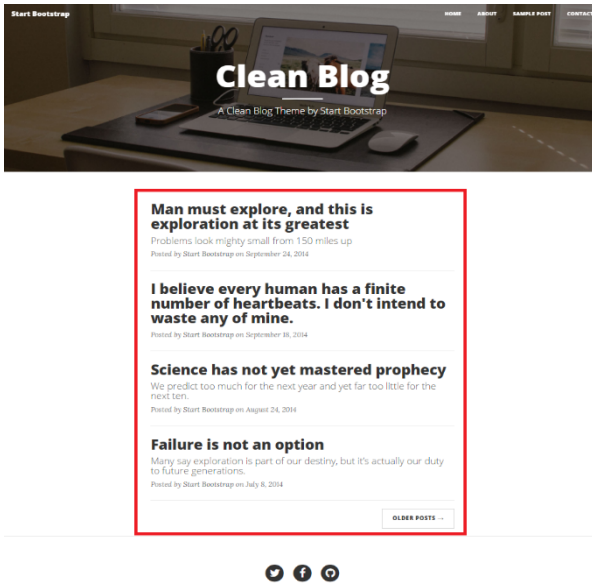
MASTER PAGE

Master Page kontrolü, sayfalarınıza ortak bir tasarım yapısı kazandırmanızı sağlayan kontroldür. Bu kontrolün en büyük faydası genel bir değişiklik gerçekleştireceğimiz zaman tek bir yerden tasarımlarımıza müdahale edebilmemizi sağlar. Ortak alanların değişikliğini yaparken çok kolay bir şekilde sadece Master Page dosyamızı değiştirerek diğer sayfaların etkilenmesini sağlayabiliriz. Bunlarla beraber her sayfada karmaşa yaratacak tasarım kodlarının sadece Master Page içerisinde bulunmasını sağlamış oluruz. Master Page dosyamızı kullanan Asp.Net sayfalarının içerisinde ise sadece ortak olmayan, her sayfa için farklı olabilen alanların kodları yer alır.

Bir Master Page oluşturduktan sonra bir Asp.Net sayfası eklerken, o sayfanın hangi Master Page'i kullanacağını seçebilirsiniz. Seçmiş olduğunuz Master Page üzerinde bulunan Content Place Holder alanları sayesinde her sayfa için, Master Page üzerindeki tasarım kullanılırken farklı içerikler yer almasını sağlayabilirsiniz. Kısacası Master Page tek bir tasarım şablonunun istediğiniz sayfalara etki etmesini ve bu sayfalarda sadece değişecek alanlar için işlem yapmanızı sağlar.



Bir Master Page dosyası oluşturmak için projemizin üzerine sağ tıklayarak, Add > New Item sekmesinden Web Forms Master Page seçeneğini takip edebiliriz.



Örnek olarak bu template için bir Master Page oluşturmak istiyorsak öncelikle diğer sayfalarda değişecek alanı belirlemeliyiz.

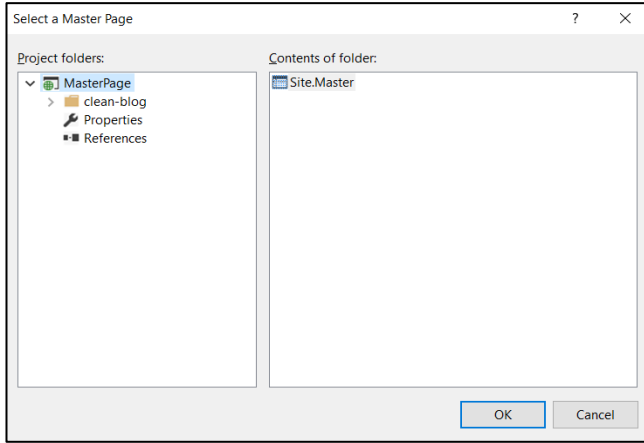
Seçtiğimiz içerik alanı her sayfada farklılık göstereceğinden bu alanı tasarım ekranında silerek yerine Content Place Holder atıyoruz.

Aynı zamanda head etiketleri arasına da bir Content Place Holder yerleştiriyoruz.

Site.Master

```
<!-- Main Content -->
<div class="container">
  <div class="row">
    <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">

    </asp:ContentPlaceHolder>
  </div>
</div>
```



Master Page dosyalarına herhangi bir şekilde web ortamından bir Asp.Net sayfasıymış gibi erişilmesi mümkün değildir.

Master Page dosyaları ancak bir Asp.Net sayfası tarafından kullanırsa bir anlam kazanacaktır.

Bir Master Page'i kullanan Asp.Net sayfası eklemek için projemize sağ tıklayarak Add > New Item bölümünden Web Form Using Master Page sekmesini seçebilirsiniz.

Master Page dosyasını oluşturduğunuzda kod kısmına geçtiğiniz zaman otomatik olarak gelen Content Place Holderlardan bir tanesinin de head bölümünde olduğunu görebilirsiniz. Head içerisinde yer alan Content Place Holder, Master Page kullanılan sayfalar için özel olarak Css, JavaScript gibi dosyaların kullanımına imkan tanır. Body kısmına gelen Content Place Holder ise diğer sayfalarda da değişebilen alanları farklı olarak kullanabilmenize olanak tanıyacaktır.

Master Page kullanılan bir Asp.Net sayfası ise aşağıdaki görünüme sahiptir. Bu sayfada tasarım şablonuyla ilgili hiçbir kodun olmadığını görebilirsiniz.

Default.aspx

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Site.Master" AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="MasterPage.Default" %>

<asp:Content ID="Content1" ContentPlaceHolderID="cphHead" runat="server">
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="cphIcerik" runat="server">
</asp:Content>
```

STATE MANAGEMENT(DURUM YÖNETİMİ)

State Management 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 Management tarafında bahsetmiş olduğumuz iki teknik vardır. Bunlar istemci taraflı ve sunucu taraflı State Management teknikleridir.

- 1) Sunucu Taraflı State Management Teknikleri
 - a. Session State (Oturum Durumu)
 - b. Application State (Uygulama Durumu)
- 2) İstemci (Client) Taraflı State Management Teknikleri
 - a. ViewState (Durum Görünümü)
 - b. Query String (Sorgu Kelimeleri)
 - c. Cookies (Çerezler)

Sırasıyla tekniklerimizi inceleyelim.

Session State

Session, kelime anlamı oturum olan bir State Management nesnesidir. Web sayfalarına giriş yapılan kullanıcı numaralarını saklamak, sayfalar içerisinde gezinti sağlanırken tekrar kullanıcı adı şifre sormamak ve genellikle e-ticaret sitelerinde sepet içerisindeki ürünleri tutmak için kullanılır.

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 adı 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 kullanımla uygulamanızın sürekliliğini bile bozabilir.

Session tekniğinin kullanımı şu şekildedir:

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

Session nesneleri içeriye değer olarak object türünde kabul etmektedir bu nedenle tüm değer ve referans tiplerini session içerisinde saklayabilirsiniz.

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 içerisine giriş yapan kullanıcının veri tabanınızdaki 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 yaptığı anda 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>
```

Ornek1.aspx sayfası açalım. Bu sayfaya bir textbox ve bir button ekleyelim. Amacımız kullanıcı adını girip button'a bastığında adını Session içerisine atarak başka bir sayfaya yönlendirip, o sayfada Session'dan ismi çekebilmek.

Ornek1.aspx

```
<body>
  <form id="form1" runat="server">
    <div>
```

```
<asp:TextBox ID="txtAd" placeholder="Adınızı Yazınız" runat="server" />
<asp:Button Text="Değer Ata" ID="btnAta" OnClick="btnAta_Click" runat="server" />
</div>
</form>
</body>
```

“isim” adında bir Session oluşturup, içerisine kullanıcının girdiği değeri atıyoruz.

Ornek1.aspx.cs

```
protected void btnAta_Click(object sender, EventArgs e)
{
    Session["isim"] = txtAd.Text;
    Response.Redirect("Ornek2.aspx");
}
```

Ornek2.aspx sayfası oluşturup, Response.Redirect ile Ornek2.aspx sayfasına yönlendirme yapıyoruz. Bu sayfada hiç code behind'a ihtiyaç duymadan Session üzerindeki değeri yazdırmak istiyoruz. <% %> arasına yazılan =, Response.Write ile aynı anlamı taşıyor. Böylece ekrana direk kullanıcının Ornek1.aspx'te girdiği ismi yazdırıyoruz.

Ornek2.aspx

```
<body>
    <form id="form1" runat="server">
        <div>
            Hoşgeldiniz <%= Session["isim"] %>
        </div>
    </form>
</body>
```

<input type="text" value="Adınızı Yazınız"/>	<input type="button" value="Değer Ata"/>
--	--

Hoşgeldiniz Bilge Adam

Application State

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 ya da sunucu kapatılana kadar (IIS restart 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 gösterilebilir veya değiştirmelerine olanak tanınabilir.

Application State için aşağıdaki gibi bir kullanım gerçekleştirebilirsiniz;

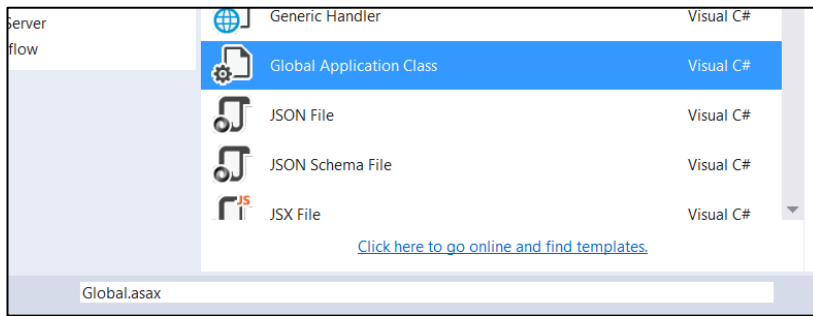
```
Application["EtiketAdi"] = "EtiketinDegeri";
```

Application başlangıcından uygulama durdurulana yada kapatılana kadar gerçekleşen süreçler Global.asax (ASAX => Active Server Application X) içerisindeki eventlere yazılmaktadır.

Bazılarını inceleyecek olursak;

- **Application_Start:** Uygulamanızın başlangıcı esnasında çalışacak olan metottur. Bu metod 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:** Uygulamadaki herhangi bir dosyaya talep(request) geldiğinde tetiklenir.
- **Application_AuthenticateRequest:** FormsAuthentication kullanıldığında kullanıcının sisteme başarılı şekilde giriş yapması durumunda tetiklenecek olan olaydır.
- **Application_Error:** Uygulamada bir hata ile karşılaşılması durumunda tetiklenecek event'dir. Bu event uygulamaya erişim kesildiğinde yani sunucu kapandığında çalışmayacaktır.
- **Session_End:** Kullanıcı siteyi terk ettiği veya session timeout yaşandığı esnada tetiklenen event'dir.
- **Application_End:** Uygulama kapatıldığında çalışan event'dir. Sunucuda oluşan özel hatalar ya da elektrik kesintisi gibi durumlarda çalışmayacaktır.

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 event'ler üzerinden online kullanıcılarımızın bilgileri için ilk olarak Application_Start eventimiz içerisine web sayfamız yayına başladığı anda oluşturulacak olan bir Application nesnesi oluşturuyoruz.

Global.asax

```
protected void Application_Start(object sender, EventArgs e)
{
    Application.Add("OnlineKullanicilar", 0);
}
```

Oluşturulan OnlineKullanicilar etiketine sahip olan Application nesnemizin varsayılan değerini 0 olarak belirledik. Bu işlemi aşağıdaki yöntemle de yapabirdik;

Örnek

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

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 eventi içerisinde gerçekleştirmeliyiz.

Global.asax

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

Yukarıdaki örnek kodda, Application["OnlineKullanicilar"]'dan dönen değeri int değerine cast ettik ve değeri bir artırarak Application nesneminin içerisine tekrar teslim ettik. Bu şekilde online kullanıcı sayımızı + 1 yaparak güncel online kullanıcı sayımızı application nesnesi içerisinde tutabiliriz.

Her ne kadar birden fazla kullanıcının aynı anda (milisaniye cinsinden dahi olsa) girmesi çok mümkün gözükmesine de bizler her ihtimali göze alarak bu duruma da çözüm üretmeliyiz. Aynı anda kullanıcıların giriş yapması durumunda, önce koleksiyonu kilitler, daha sonra içerideki değeri değiştirir ve koleksiyonu tekrar kullanıma açarız. Böylece o esnada Application nesnesinin başka bir Session tarafından kullanılması engellenmiş olacaktır. Bir nevi turnike sistemi de denilebilir.

Kullanıcı sayımızdaki artışı tamamladık şimdi geldi sıra oturum (Session) sonlandırıldıysa bu kullanıcıyı online kullanıcı sayısından düşürmeye. Bunun için oturum bittiğinde, Session_End eventini kullanabiliriz.

Global.asax

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

Sayfamızın html kısmına bir label ekledik;

Default.aspx

```
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label ID="lblOnlineUsers" Font-Bold="true" runat="server" />
        </div>
    </form>
</body>
```

```
</div>
</form>
</body>
```

Sayfamızın code behind bölümüne geliyoruz ve label içerisine online kullanıcı değerimizi veriyoruz.

Default.aspx.cs

```
protected void Page_Load(object sender, EventArgs e)
{
    lblOnlineUsers.Text = string.Format("Şu anda sitemizde toplam {0} kişi online durumdadır!",
    Application["OnlineKullanici"]);
}
```

Online kullanıcılarımızı takip edebildiğimiz ve kullanıcılarımıza gösterebildiğimiz uygulamamız bu şekilde son buluyor.

ViewState

Sayfanın yaşam döngüsünün ilk basamağı olan initalization aşamasında, tüm sayfa ve sayfa üzerindeki kontroller yeniden oluşturulur. Bu oluşturulma süreci sayfa ilk isteğinde de ("GET" metodunda) sayfanın postback'le dönmesinde de ("POST" metodunda) gerçekleşmektedir. Ancak kontroller yeniden oluşturulmasına rağmen, bazı kontrollerin üzerlerindeki değerleri muhafaza ettiği görülür. Bu işin arkasındaki gizli kahraman "ViewState" dediğimiz nesnedir.

ViewState sayfanın üzerinde muhafaza edilmesi gereken değerleri, aslı bir HTML input kontrol olan "hidden"da saklı tutar.

Varsayılan olarak tüm kontrollerin ViewState değerleri açık durumdadır. Yani ViewState'den yararlanmak için herhangi özel bir işlem yapmamıza gerek yoktur. Ancak bazı senaryolar gereği sayfa üzerindeki bir kontrolün taşıdığı değerin saklanmaması istenebilir. Örneğin GridView'de amacımız sadece veri göstermekse her defasında yeniden doldurulmasını tercih edebiliriz. Bunun için yapılması gereken şey ViewState özelliğinin kapatılmasıdır. Bu işi de 3 şekilde yapabiliriz:

Kontrol Bazlı ViewState Kapatma: TextBox, Label, DropDownList, Button gibi kontrollerin ViewState'ini kapatmak için yapılması gereken tek şey "EnableViewState" özelliğini "False" olarak atamaktır.

EnableViewState Kullanımı

```
<asp:Label ID="Label1" EnableViewState="false" runat="server" />
```

Sayfa Bazlı ViewState Kapatma: Sayfa bazlı olarak ViewState'i kapatmak içinse yapılması gereken şey, sayfanın en tepesinde bulunan PageDirective kısmına gelerek "EnableViewState="false" ibaresini eklemektir. Sayfa en nihayetinde bir kontrol olduğundan, sayfanın ViewState'ini kapatırsanız o sayfadaki bir kontrolün ViewState özelliği açık olsa bile artık işlem görmeyecektir!

Uygulama Bazlı ViewState Kapatma: Bunun için gidilmesi gereken referans yol, web.config yoludur. Web.config dosyanızda pages elementinin "enableViewState" attribute'üne "false" değeri atarsanız uygulama bazlı olarak da ViewState'i kapatabilirsiniz. Buradaki ayar bir "zorlama" değildir. Yani, sayfanızda ViewState açıksa ve sayfa üzerindeki kontrolde de açıksa o kontrol ViewState değeri taşır.

ÖNEMLİ NOT : Bazı internet hurafeleri TextBox, DropDownList, CheckBoxLayout gibi kontrollerin "ViewState" özelliğinin kapatılmadığını ifade etmektedir. Bu bilinen çok ciddi bir yanıltır! Bu kontrollerin ViewState özellikleri kapatılabilir.

Peki, TextBox üzerindeki yazıyı, DropDownList seçili elemanı, CheckBoxLayout check edilmiş elemanı yeniden oluşturulmasına rağmen (ViewState ile taşınamamasına rağmen) nasıl korur? Bunun tek sebebi "IPostbackDataHandler" interface'idir. Bu interface'in uygulandığı kontrol class'ları bazı özelliklerini ViewState kapatılmış olsa bile muhafaza eder ve asla kaybetmezsiniz.

Query String

Birçok web sayfasında sayfanın uzantısından sonra soru işaretiyle başlayan bir alan görmüşüzdür. Bu alana Query String adı verilmektedir. QueryString, sayfalar arasında URL aracılığı ile metinsel bir değerin taşınmasına olanak sağlayan durum yönetim nesnedir. Bu metinsel veri, istekte bulunan istemcinin tarayıcısı aracılığı ile taşınır. Dolayısıyla client-side bir yöntemdir.

QueryString, sayfamızın uzantısının ardından soru işareti ile başlayarak, string değişken isimlerimizi verebildiğimiz alanlardan oluşmaktadır. Genel şablonu şu şekildedir.

QueryString Örneği

Adresimiz.com/OrnekSayfa.aspx?degiskenIsmi=degiskeninDegeri

Yukarıdaki gibi bir OrnekSayfa.aspx sayfamızın code behind bölümünden degiskenIsmi isimli string tipindeki değişkenimiz içerisinde bulunan değeri rahatlıkla aşağıdaki gibi yakalayabiliriz.

QueryString Yakalanması

```
string yeniDegisken = Request.QueryString["degiskenIsmi"];
```

Querystring üzerinden birden fazla değişken yani bir diğer adıyla parametre gönderebilmek için & (and) karakteri kullanılır. Birden fazla parametre gönderilmiş Query String örneği aşağıdaki gibidir:

www.adresimiz.com/OrnekSayfa.aspx?Degisken=deger&DegiskenIki=deger2

Projemize bir Default.aspx sayfası ekleyelim ve aşağıdaki tasarımı oluşturalım.

Default.aspx

```
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label Text="Kullanıcı Adı:" ID="lblKAdi" runat="server" />
            <asp:TextBox ID="txtKAdi" runat="server" />
            <br />
            <asp:Label Text="Şifre:" ID="lblSifre" runat="server" />
            <asp:TextBox ID="txtSifre" runat="server" />
            <asp:Button Text="Göster" ID="btnGoster" OnClick="btnGoster_Click" runat="server" />
        </div>
    </form>
</body>
```

Daha sonra code behind tarafına gelerek Query String ile TextBox'lara girilen verileri başka bir sayfaya yollayalım. Bunun için öncelikle Bilgi.aspx adlı bir sayfa ekleyelim.

Default.aspx.cs

```
protected void btnGoster_Click(object sender, EventArgs e)
{
    Response.Redirect("Bilgi.aspx?KullaniciAdi=" + txtKAdi.Text + "&Sifre=" + txtSifre.Text);
}
```

Bilgi.aspx sayfasına KullaniciAdi ve Sifre isminde 2 ayrı parametre gönderdik. Bunları Bilgi.aspx'e label ekleyerek gösterelim.

Bilgi.aspx

```
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label ID="lblKAdi" runat="server" />
            <asp:Label ID="lblSifre" runat="server" />
        </div>
    </form>
</body>
```

Code behind tarafında Request nesnesi yardımıyla Query String'lerimize isimlerini vererek ulaşabiliriz. İsterseniz indexlerine göre de Query String'lere ulaşabilirsiniz.

Bilgi.aspx.cs

```
protected void Page_Load(object sender, EventArgs e)
{
    //lblKAdi.Text = Request.QueryString["KullaniciAdi"];
    lblKAdi.Text = Request.QueryString[0];
    lblSifre.Text = Request.QueryString["Sifre"];
}
```

localhost:38792/Default.aspx

Kullanıcı Adı:

Şifre:

localhost:38792/Bilgi.aspx?KullaniciAdi=BilgeAdam&Sifre=123

BilgeAdam 123

Cookies

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 tutmak, gerektiği zaman yine istemci tarafından o bilgiye erişebilmek için kullanılır.

Avantajları:

- Kullanımı oldukça kolay, sunucu tarafını yormayan, veri saklama esnasında geliştiricinin sorumluluk almak zorunda olmadığı bir yöntemdir.
- Cookie ile ilgili işlemlerin tamamından client bilgisayarın tarayıcısı sorumludur ve neredeyse tüm tarayıcılar çerez yönetimini desteklemektedir.
- Bir istemci bilgisayarında birden fazla cookie bulundurulabilir ya da bir cookie içerisinde birden fazla metinsel değer taşıyabilirsiniz.

Dezavantajları:

- Cookie'lerin boyutları genellikle 4KB ile sınırlı olduğundan büyük verileri içerisinde barındıramazsınız. (Tarayıcıdan tarayıcıya değişen bir değerdir.)
- Her web uygulamasının, istemci başına sınırlı sayıda cookie hakkı vardır. (Tarayıcılar genellikle bir uygulama başına maksimum 20 cookie sağlamaktadır).
- Kullanıcılar diledikleri zaman cookie'leri silebilir ya da tarayıcı ayarlarıyla oynayarak hiç cookie tutulmamasını da isteyebilir.

Cookie nesnesi en çok web sayfalarında üyelik girişlerinde bulunan beni hatırla özelliğinde kullanılır. Biz de kullanıcının giriş bilgilerini saklayan bir örnek yapalım.

Bunun için Default.aspx sayfası açarak aşağıdaki tasarımı yapıyoruz.

Default.aspx

```
<body>
    <form id="form1" runat="server">
        <div>
            Kullanıcı Adı: <asp:TextBox ID="txtKAdi" runat="server" />
            Şifre: <asp:TextBox ID="txtSifre" runat="server" />
            <asp:Button Text="Oluştur" ID="btnOluştur" OnClick="btnOluştur_Click" runat="server" />
            <asp:Label Text="text" ID="lblAd" runat="server" />
            <asp:Label Text="text" ID="lblSifre" runat="server" />
        </div>
    </form>
</body>
```

Default.aspx.cs

```
protected void btnOluştur_Click(object sender, EventArgs e)
{
}
```

```
//Bir cookie oluşturabilmek için HttpCookie nesnesinden bir instance almamız gerekmektedir.  
HttpCookie cerez = new HttpCookie("KullaniciBilgileri");  
cerez["KullaniciAdi"] = txtKAdi.Text;  
cerez["Sifre"] = txtSifre.Text;  
  
//cerez.Expires => Cookie'nizin istemci bilgisayarda ne kadar süre tutulacağını belirtir.  
Genellikle net tarih vermektense DateTime'ın Add... metotlarını kullanmayı tercih ederiz.  
  
cerez.Expires = DateTime.Now.AddDays(30);  
Response.Cookies.Add(cerez);  
}
```

Default.aspx

```
protected void Page_Load(object sender, EventArgs e)  
{  
    if (Request.Cookies["KullaniciBilgileri"] != null)  
    {  
        HttpCookie cerez = Request.Cookies["KullaniciBilgileri"];  
        lblAd.Text = cerez["KullaniciAdi"];  
        lblSifre.Text = cerez["Sifre"];  
    }  
}
```

Yukarıda öncelikle “KullaniciBilgileri” isminde bir cookie var mı bunun kontrolünü sağladık, var ise yani içeriği null değil ise hemen HttpCookie nesnesi oluşturup oradan gelen cookie nesnesinin atamasını yaptık. Label'ların içerisine gelen cookie nesnemiz içerisinde saklamış olduğumuz verileri yazdırdık.

Projeyi çalıştırıp button'a bastığımızda, page bir kez daha load olduğunda artık label'ların cookie'den dolduğunu görebiliriz. Tarayıcımızın çerezlerini açarak bu çereze ulaşabilir, silebilir, başlangıç-bitiş tarihlerini ve değerlerini görebiliriz.

Çerezler ve site verileri

Site
Yerel olarak depolanan veriler
Tüm gösterilenleri kaldır
local

localhost
2 çerez
ASP.NET_SessionId
KullaniciBilgileri

Adı: KullaniciBilgileri
İçerik: KullaniciAdi=Bilge Adam&Sifre=123
Etki Alanı: localhost
Yol: /
Gönder: Herhangi bir bağlantı türü
Komut dosyasına erişilebilir
Oluşturma tarihi: 1 Mart 2017 Çarşamba 02:58:04
Son Geçerlilik Tarihi: 31 Mart 2017 Cuma 02:58:05
Kaldır

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 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 yöntem veritabanınızı yormayacağı gibi aynı zamanda hız anlamında da ekstra olarak bir performans sağlayacaktır.

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

Data Caching (Veri önbellekleme) : Herhangi bir veri kaynağından gelen verilerinizi önbellekleyerek, yukarıdaki örnekte bahsetmiş olduğumuz gibi veriler için performans kazanmanıza olanak tanımaktadır.

Output Caching(Çıktı Önbellekleme): Sayfanızın tüm Html ve Css çıktısını önbelleğe alarak kullanıcılarınıza bir sonraki önbellekleme zamanına kadar aynı görüntünün gösterilmesini sağlayan önbellekleme mekanizmasıdır.

Data Caching (Veri Önbellekleme)

Northwind veritabanından Categories tablosunu Ado.Net sorgusu ile bir DataTable nesnesi içerisinde dolduralım. Daha sonra bu DataTable'ı önbelleğe alalım.

Verilerimizi gösterebilmek için bir repeater oluşturduk ve içerisine CategoryName verilerini bastıracağımızı söyledik. Ancak bu sefer verilerimizi Cache üzerinden çekeceğiz.

Default.aspx

```
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Repeater ID="rptKategoriler" runat="server">
```

```
<ItemTemplate>
    <asp:Label Text='<%#Eval("CategoryName") %>' ID="lblKategoriAdi" runat="server"
/>

    <br />
</ItemTemplate>
</asp:Repeater>
<asp:Button Text="Postback Yap" ID="btnPostbackYap" runat="server" />
</div>
</form>
</body>
```

Cache Parametreleri

Absolute Expiration: Kesin zamanlı olarak bir Cache'in önbellekte ne zaman bozulacağını belirtirsiniz. Genellikle `new DateTime()` diyip yeni bir tarih üretmektense, o anki tarihe ekleme yapan `DateTime`'in `Add` metotlarını kullanırsınız.

Sliding Expiration: Önbelleğe atılan nesnelerin ötelemeli olarak yaşamasını istiyorsanız bu tipi seçmelisiniz. `TimeSpan` sizden öteleme miktarını isteyecektir.

Northwind üzerinden kategorileri çekip Cache'e ekleyen bir metot yazalım.

Default.aspx.cs

```
void KategorileriGetir()
{
    SqlDataAdapter da = new SqlDataAdapter("Select CategoryName from Categories",
"Server=.;Database=NORTHWND;Integrated Security=true");
    DataTable dt = new DataTable();
    da.Fill(dt);

    //Absolute Expiration
    //Cache.Insert("dtKat", dt, null, DateTime.Now.AddSeconds(30),
System.Web.Caching.Cache.NoSlidingExpiration);

    //Sliding Expiration
    Cache.Insert("dtKat", dt, null, System.Web.Caching.Cache.NoAbsoluteExpiration,
TimeSpan.FromMinutes(1));
}
```

Burada "dtKat" isminde bir Cache oluşturarak içerisine `DataTable`'ımızı verdik. Basit bir şekilde `Cache.Insert("dtKat",dt);` komutunu da kullanabilirdik ancak burada Cache'imize bir zaman belirledik.

Sıra verilerimizi Cache'ten çekmeye geldi. Her page load olduğunda öncelikle `dtKat` adında bir Cache var mı kontrol ettik. Eğer böyle bir Cache yoksa metodu çağırarak Cache'in oluşturulmasını sağladık. Ancak böyle bir Cache varsa database'den verileri çekmeye gerek kalmadan `Repeater`'ın `DataSource`'una Cache'ten gelen veriyi atadık.

Default.aspx.cs

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Cache["dtKat"] == null)
        KategorileriGetir();

    rptKategoriler.DataSource = Cache["dtKat"];
    rptKategoriler.DataBind();
}
```

Output Caching (Çıktı Önbellekleme)

Çıktı önbellekleme yöntemi sayfanızın tekrar derlenip, sunucuya gidip gelmesi ile kullanıcıya gösterilen içeriğin tekrar tekrar oluşturulmasının önüne geçerek ekstra bir performans kazandıracaktır. Aşağıdaki kod örneğini sayfanızın page directive bölümünün hemen altına eklerseniz output caching işlemini gerçekleştirmiş olacaksınız.

Output Caching

```
<%@ OutputCache Duration="20" VaryByParam="none" %>
```

Duration: Sayfanızın kaç saniye boyunca önbellekte kalacağını belirttiğiniz özelliktir.

VaryByParam: Bu özelliğin none olarak bırakılması sayfanızın tamamının önbelleğe alınacağını ifade eder.