EĞİTİM :

DURUM YÖNETİMİ VE TEMALAR

Bölüm :
Durum Yönetimi

Konu :
View State Management
(Görünüm Durumu Yönetimi)

Web sayfalarında, kullanıcı ile sayfa arasındaki bilgi alışverişi, HTTP protokolü ile gerçekleşmektedir. HTTP protokolü kullanılırken istemci ile sunucu bilgisayarları arasında gerçekleşen veri transferinde, kullanıcıdan alınarak sunucuya iletilen bilgiler, klasik yollar kullanıldığında kalıcı olamamakta ve tekrar erişilememektedir.

ASP.NET'te uygulama içerisindeki tüm sayfalar birer **sınıf (class)** olarak tutuldukları için, sayfaya gelen bir istekte sunucu belleğinde sayfanın bir **nesne (object)** örneği oluşturulur. Sayfanın sunucu tarafında işlenmesinin ve HTML kodlarına dönüştürülmesinin ardından sayfanın nesne örneği (ve içerisinde taşınan üyeler) sunucu belleğinden kaldırılmaktadır. Bu nedenle istemciye gönderilen verilere tekrar erişebilmek için, bu verilerin belirli nesneler içerisinde saklanmasını ve taşınmasını sağlayacak bazı yapılara ihtiyaç duyulmuştur. Bu noktada bazı durumlarda kolaylık sağlaması, bazı durumlarda güvenli bir şekilde verilerin taşınmasını sağlayan farklı yapılar geliştirilmiş ve ASP.NET uygulamaları içerisinde kullanılması sağlanmıştır. Web uygulamalarında veriler ile ilgili olarak durum yönetimi sağlayan, veriyi istemci veya sunucu tarafında saklanmasını sağlayan yöntemler aşağıda listelenmiştir.

- **ViewState**
- **QueryString**
- **Cookie**
- **Session**
- **Application**
- **Cross-Page PostBack**

Bu yöntemlerden **ViewState, Cookie, QueryString** verilerin istemci tarafında, **Cross-Page PostBack**, **Session**, ve **Application** ise verilerin sunucu tarafında saklanmasını sağlamaktadır.

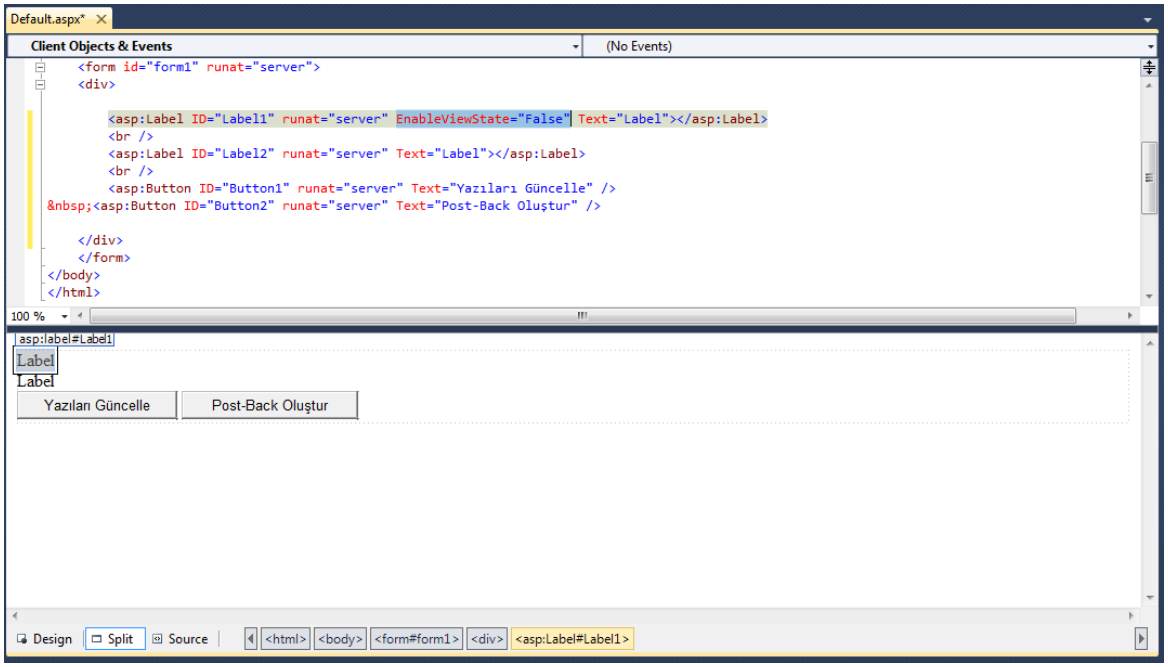
ViewState

ViewState nesnesi ASP.NET'te form düzeyinde verileri saklamak için kullanılır. **ViewState**, bir ASP.NET sayfası içerisinde bulunan kontrollerin özelliklerinin ve yazılımcının istediği verilerin saklanmasını sağlar. Yazılımcı belirtmese bile varsayılan olarak, sayfada Postback işlemi gerçekleştiğinde kontrollere ait bilgiler sunucu tarafında HTML kodları üretilirken şifrelenmiş bir şekilde View State içerisine yazılır. Sayfa tekrar yüklendiğinde ise kontrollerin özellikleri bu nesneden okunur. Böylece Postback işlemi sonucunda kontroller üzerinde yapılan değişiklikler sayfa tekrar yüklendiğinde kaybedilmeden elde edilebilir.

ASP veya PHP ile geliştirilen uygulamalarda bir sayfa içerisindeki kontrollerin değerleri, sayfa post edildiğinde kaybolurdu ve tekrar elde etmek için ekstra işlemler yapılması gerekirdi. ASP.NET'e geçiş yapan yazılımcıların ilk zamanlarda en çok dikkatini çeken durumlardan birisi de bir form post edildiğinde form üzerindeki verilerin kaybolmamasıydı. ASP.NET uygulamalarında form post olduğunda veri kaybolma sıkıntısından kurtaran ve post işleminden sonra da verilerin form üzerinde saklanmasını sağlayan nesne ViewState nesnesidir. Aşağıdaki örnek incelendiğinde ViewState nesnesinin uygulamalarda ne gibi bir kolaylık sağladığı daha net bir şekilde anlaşılacaktır.

Bir ASP.NET Web Formuna iki Label ve iki Button ekleyelim, eklenen Label'lardan ilkinin EnableViewState özelliği true olarak bırakılsın diğeri false olarak değiştirilsin. Örnek için tasarladığımız sayfa aşağıdaki resimdekine benzer şekilde olmalıdır. Gördüğün gibi alt alta iki tane Label ve onların altında da yan yana iki tane Button var. Tüm kontrollerin isimlerini Visual Studio'nun belirlediği şekilde varsayılan olarak bırakıyoruz.

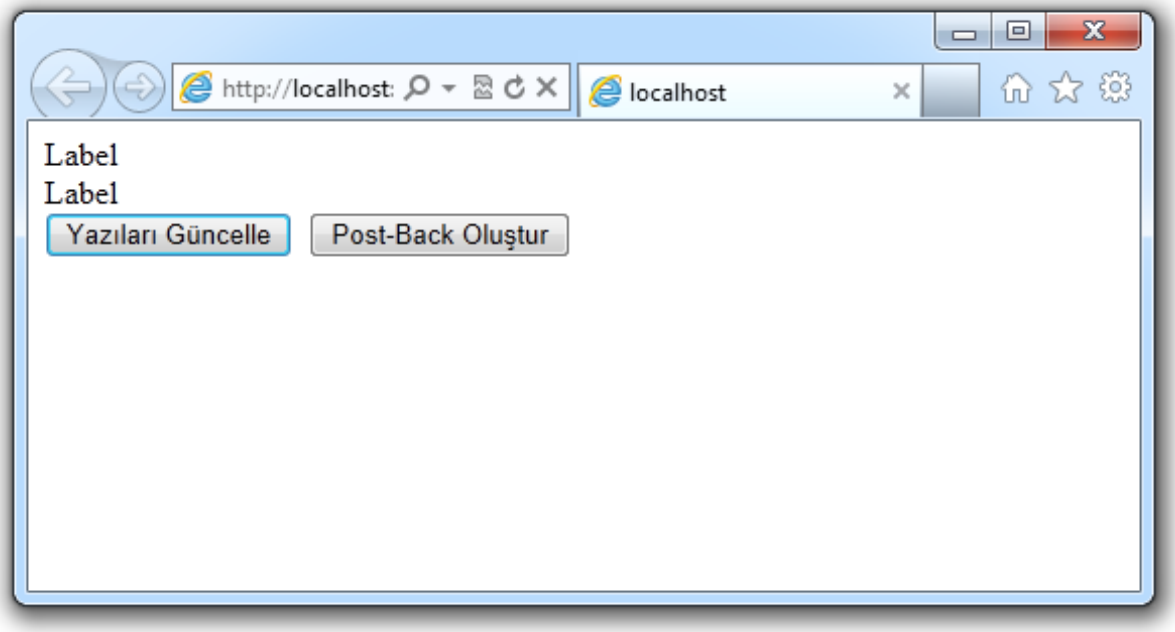
EnableViewState özelliği kontrolün özelliklerinin ViewState içerisinde taşınılıp taşınılmayacağını belirleyen özelliktir ve varsayılan değeri true olarak belirlenmiştir.



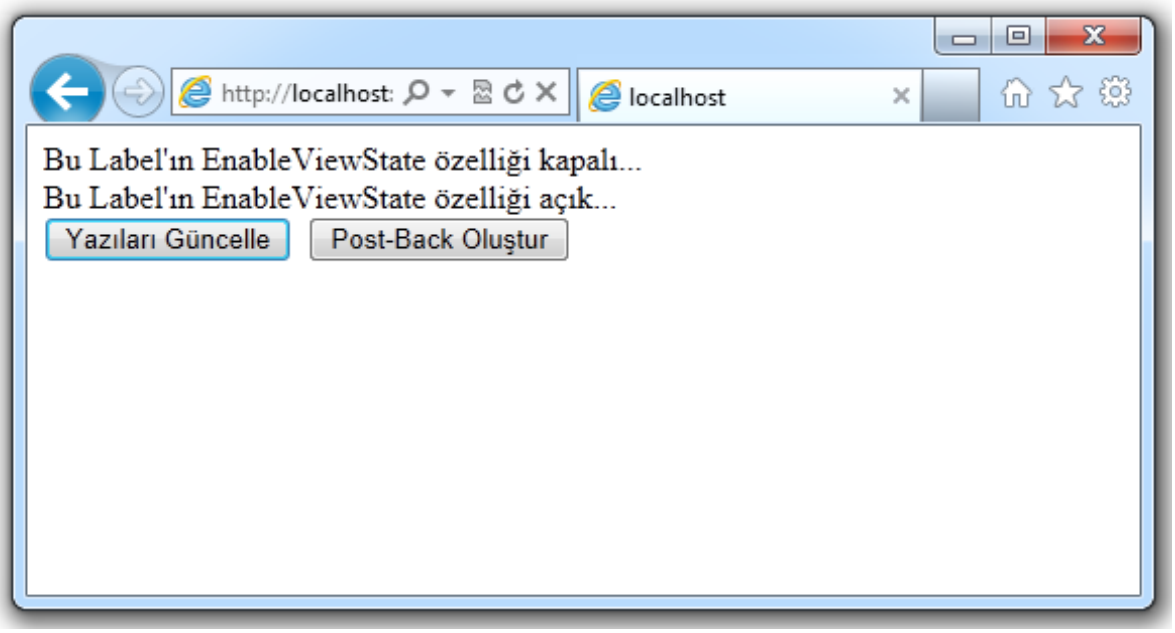
Soldaki, yani Text özelliği Yazıları Güncelle olarak ayarlanmış olan Button'un Click olayını ele alıp Label'ların Text özelliklerini çalışma zamanında güncelleyecek olan aşağıdaki kodları yazalım. Diğer Button'da herhangi bir değişiklik yapmıyoruz. Örneğimizde bu button'un görevi sadece Post-Back oluşturmak olacaktır.

```
protected void Button1_Click(object sender, EventArgs e)
 {
     Label1.Text = "Bu Label'ın EnableViewState özelliği kapalı...";
     Label2.Text = "Bu Label'ın EnableViewState özelliği açık...";
 }
```

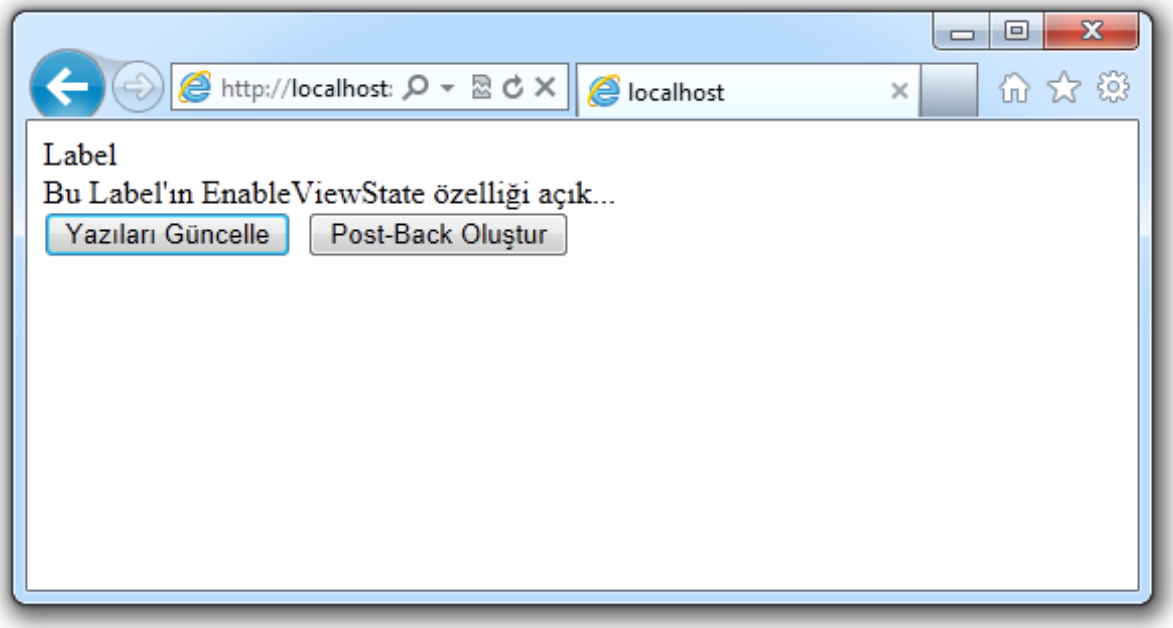
Kodları yazdıktan sonra sayfayı çalıştırıp uygulamayı test etmeye başlayalım. Uygulama ilk çalıştığında sayfa aşağıdaki resimde göründüğü gibi olacaktır.



Şimdi, Yazıları Güncelle Button'una tıklayalım. Ardından da her iki Label'daki yazının da değiştiğini gözlemleyelim. Bu işlemin ardından sayfa aşağıdaki görünümü alacaktır.



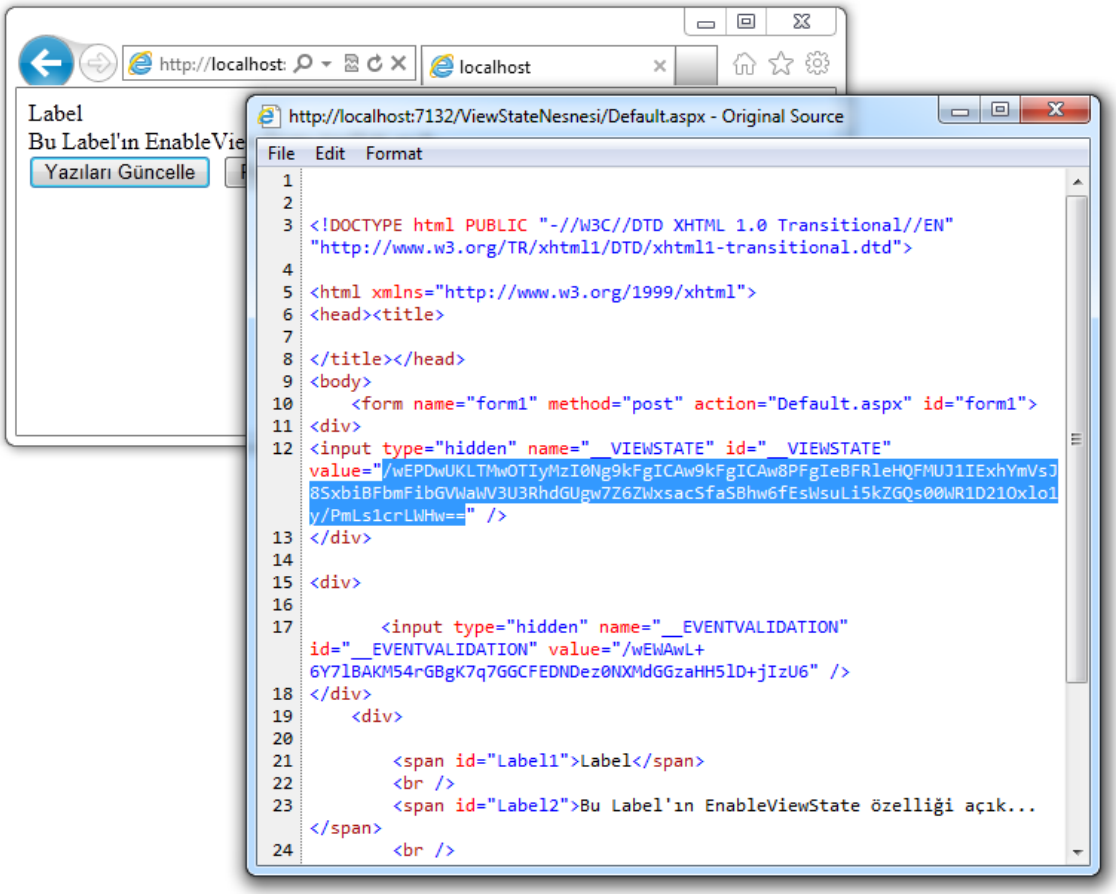
Şimdi EnableViewState özelliğinin ne işe yaradığını net bir şekilde görmek amacı ile Post-Back oluştur Button'una tıklayalım. Üstteki Label'ın yani EnableViewState özelliğini false olarak ayarladığımız Label'ın Text'i sayfa ilk çalıştırıldığı gibi varsayılan değeri ile karşımıza çıkacaktır.



Örnekten de görüldüğü gibi, ilk postback işlemi sonucunda, butonun Click olayı aracılığıyla her iki Label'a da değerler yazıldı. Fakat sayfa üzerinde farklı bir postback işlemi (labelların durumunu değiştirmeyen bir postback işlemi) gerçekleştiğinde EnableViewState özelliği false olan label kontrolü (Label1), bir önceki postbackten kazandığı değerini kaybetti. Diğer yandan, ViewState nesnesi içerisinde değeri saklanan label (Label2) farklı postback işlemlerinden geçmesine rağmen önceki sayfadaki değerini kaybetmedi.

View State içerisinde sadece formda bulunan input elementleri değil, GridView, DetailsView ve Calendar gibi diğer ASP.NET sunucu kontrollerinin özellikleri de saklanmaktadır. Bir uygulama adım adım incelendiğinde HTML kodları içerisinde taşınan ViewState nesnesi aşağıdaki gibi gözlemlenebilmektedir.

Az önce oluşturduğumuz uygulamayı yeniden çalıştırıp ViewState'e yazılan değerleri görmek için sayfanın üzerine sağ tuşla tıklayıp, menüden kaynağı görüntüle (View Source) seçeneğini seçelim.



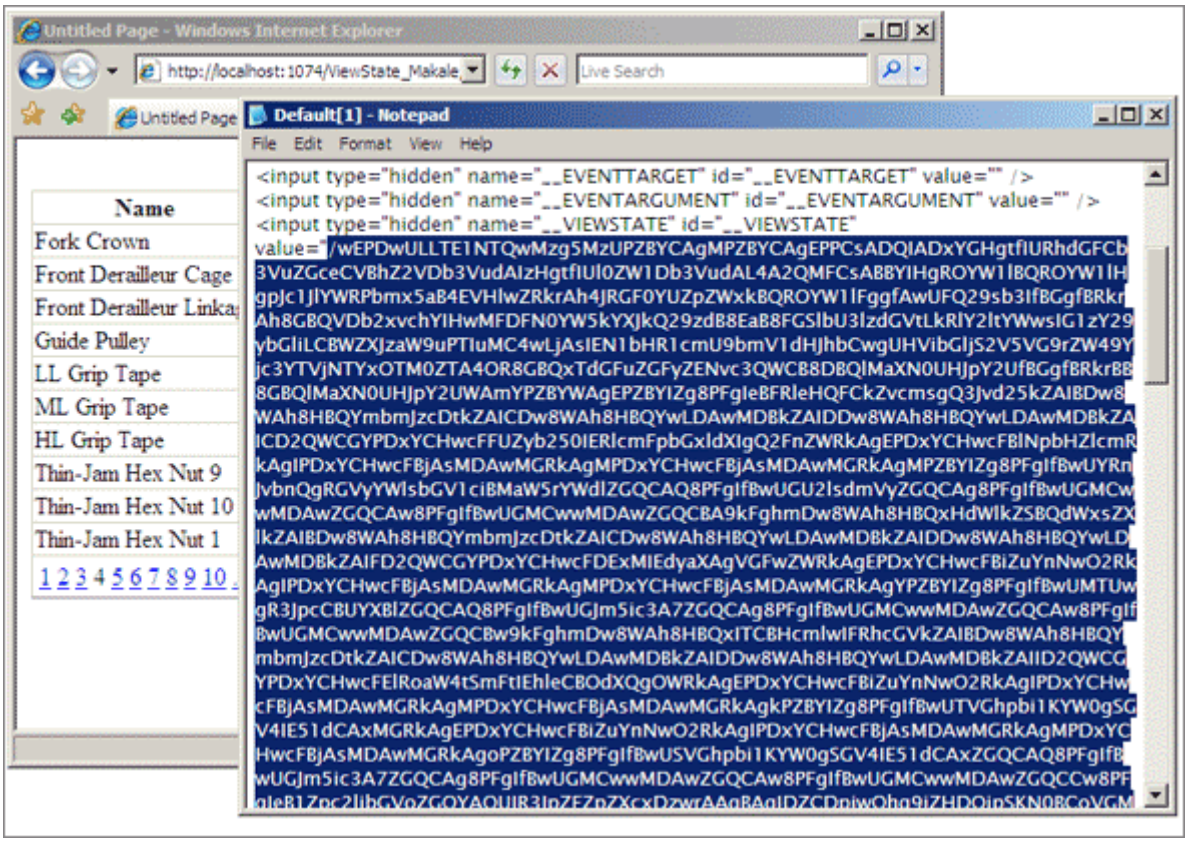
Görüldüğü gibi bilgiler HTML kodları içerisinde __VIEWSTATE adı verilen hidden tipinde bir input elementi ile bulunmaktadır. Bir ASP.NET sayfasında varsayılan olarak her zaman __VIEWSTATE değeri saklanmaktadır. Base64 formatı kullanarak şifrelenen veriler sayfa içerisinde saklanır ve sayfanın postback ile tekrar yüklenmesi durumunda gerekli bilgiler çözümlenerek sayfa içerisinde kullanılır. Yukarıdaki resimde seçili olan değer sayfanın şu andaki ViewState nesnesinde saklanılan değerin şifrelenmiş halini içermektedir.

__VIEWSTATE içerisinde formdaki kontrollerin dizilimi ve Postback işlemi sonucunda kontrollerin özelliklerinde yapılan değişiklikler tutulmaktadır. Kontrol üzerinde, sadece değişen Text özelliği değil, değişen diğer tüm özellikler de (BackColor, Width, Height, ... gibi diğer özellikler) ViewState içerisinde saklanır.

TextBox, CheckBox ve RadioButton gibi kontrollerin postback esnasında değişen özellikleri ViewState ile taşınmaz. Bu kontrollerin özellikleri HTML çıktısı olarak üretilir.

Yukarıdaki örneği çalıştırın ve herhangi bir Button'a tıklamadan kaynağı görüntüleyip _VIEWSTATE'e göz atın. Hemen ardından, "Yazıları Güncelle" Button'una tıklayıp, _VIEWSTATE'e yeniden göz atın. _VIEWSTATE içerisinde bulunan değerler karşılaştırıldığında, Label kontrolünün Text özelliğinde yapılan değişiklikten dolayı, boyutunun arttığı görülecektir. Label özelliğinin Text özelliğine yazılan değer _VIEWSTATE'e çok büyük miktarda

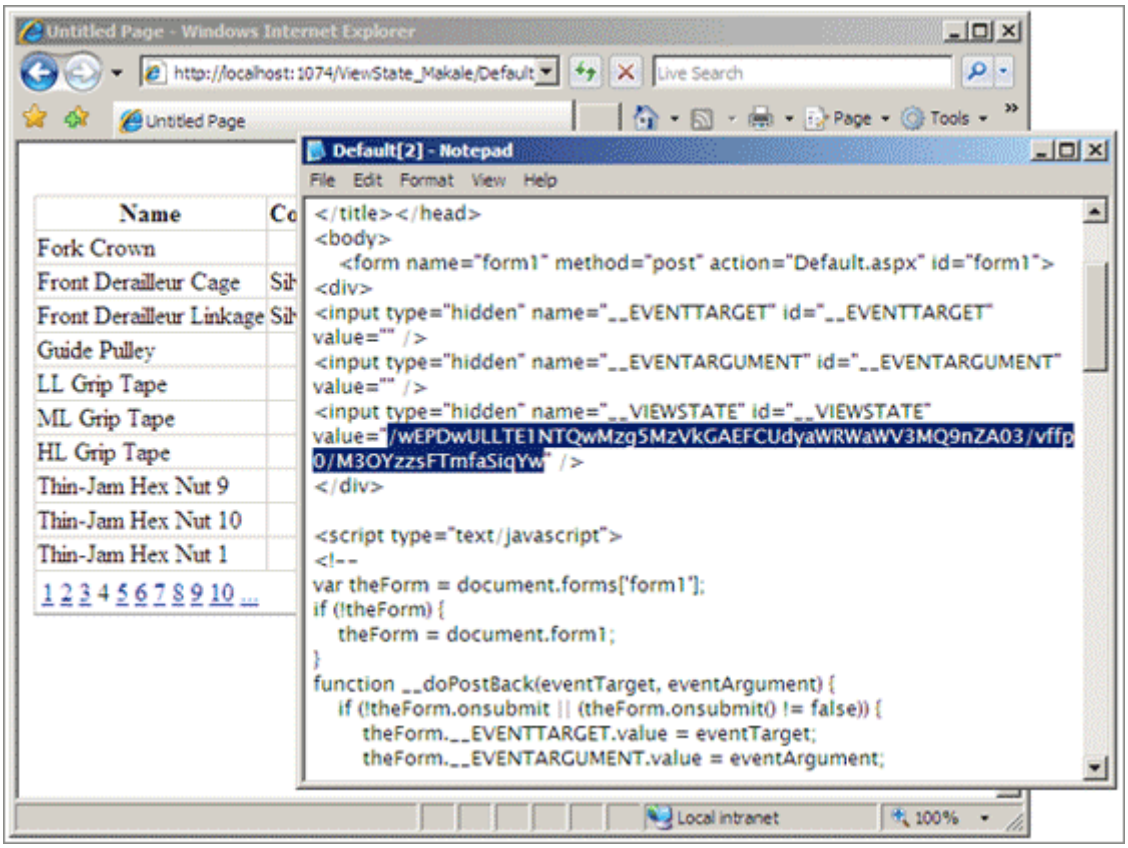
bir yük bindirmemiştir. Ancak, GridView gibi veri görüntüleyen kontroller göz önüne alındığında ve bu kontrollerin içeriği PostBack anında değiştirdiğinde durum ne olacak? Gelin bu duruma bir örnekle göz atalım. Bir Web Form'a bir tane GridView kontrolü sürükleyip bırakıp, içeriğini **AdventureWorks** veritabanındaki **Production.Product** tablosundaki veriler ile bir butonun click olayında, kod tarafında dolduralım. Bu işlem için Veriye Erişim bölümünde yer alan örneklerden birisini de kullanabilirsin. Bu durumdaki sayfayı çalıştırıp butona tıklanılıp, sayfanın kaynağı görüntülendiğinde karşılaşılacak olan görüntü aşağıdaki gibi olacaktır.



Yukarıdaki resimdeki _VIEWSTATE'in içeriği, diğer resimlerdeki içerikler ile karşılaştırıldığında oldukça büyük bir fark göze çarpmaktadır. _VIEWSTATE'in boyutu neredeyse sayfanın üçte birini kaplayacak kadar büyüdü.

Burada oluşturulan ViewState değeri her Postback işlemi gerçekleştiğinde sunucuya gönderileceği ve her sayfa oluşumunda da istemciye getirileceği için sürekli olarak sunucu-istemci arasında fazladan veri transferi söz konusu olacaktır. İşte bu noktada uygulamayı geliştiren kişiye önemli bir performans ayarlaması görevi düşmektedir. GridView vb. bir kontrolün ViewState içerisinde saklanması, sayfanın düzgün bir biçimde çalışması için gerekli midir, yoksa gereksiz midir? Bir başka söylemle; Sayfa, Postback olduğunda önceki halinde bulunan verileri hatırlamaya ihtiyacı var mıdır, yok mudur? Eğer cevap "Hayır, bu sayfanın tekrar oluşturulması esnasında, önceki halindeki GridView vb. bir nesnenin verisinin taşınmasına gerek yok" ise, bu noktada sayfanın sunucu-istemci arasında daha hızlı şekilde çalışabilmesi için bu değerlerin ViewState içerisinde saklanmasının engellenmesi gerekmektedir. ASP.NET sunucu kontrollerinin tamamında **EnableViewState** adında bir özellik bulunmaktadır. Bu özellik kontrolün postback işlemi sonucunda değişen değerlerinin ViewState içerisinde saklanıp saklanmayacağını belirler. bool tipinden bir değer alabilen EnableViewState özelliğinin varsayılan değeri true'dur. EnableViewState değerinin true olması, kontrolün özelliklerinden ViewState içerisinde saklanır, false olması durumunda ise saklanmaz. Eğer bir kontrolün özelliklerinin ViewState içerisinde taşınması istenmiyorsa, o kontrolün EnableViewState özelliğininin false olarak ayarlanması gerekecektir. GridView'in

EnableViewState özelliği false olarak ayarlayıp uygulamayı çalıştıralım. Uygulama çalıştıktan sonra kaynağı görüntüleyip ViewState'a göz attığımızda aşağıdaki gibi bir görünüm bizi bekliyor olacaktır.



Resimden de görüldüğü gibi GridView nesnesinin EnableViewState özelliğine false değeri atanarak kontrolün verilerini ViewState'te saklanması engellenmiş ve _VIEWSTATE'in boyutu azaltılmıştır.

Bir kontrolün ViewState özelliği kontrolün EnableViewState özelliği üzerinden gerçekleştirileceği gibi, bir sayfa içerisindeki tüm kontrollerin ViewState özelliği de sayfanın Page direktifi içerisinden veya Page nesnesi üzerinden yine EnableViewState özelliği ile kapatılabilir. Aşağıdaki kod blokları bu işlemin nasıl yapıldığını açıklamaktadır.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
    Inherits="_Default" EnableViewState="false" %>
```

```
protected void Page_Load(object sender, EventArgs e)
{
   Page.EnableViewState = false;
}
```

ViewState özelliğinin aktif olup olmayacağı sayfa veya kontrol bazında belirlenebildiği gibi aynı zamanda uygulama düzeyinde de belirlenebilmektedir. Uygulama düzeyinde ayar yapmak için uygulama ayar dosyasında ki (Web.config) <page>..</page> düğümünden ayar yapılabilmektedir. Bu işlemi açıklayan kodlar da aşağıda yer almaktadır.

```
<system.web>
.....
.....
```

```
<pages enableViewState="false"></pages>
.....

</system.web>
```

Sayfanın EnableViewState özelliği kapatıldığında artık kontrollerin postback sonrasında değişen özellikleri ViewState içerisinde tutulmayacaktır. Fakat böyle bir durumda oluşan HTML kodları içerisine bakılacak olursa, __VIEWSTATE alanında hala bazı bilgiler olduğu görülecektir. Her ne kadar kontrollerin değişen özellikleri artık bu nesnede saklanmasa da, sayfaya eklenen kontrollerin hiyerarşik yapısı burada tutulmaya devam edilecektir.

ViewState Nesnesi İçerisinde Özel Veri Taşımak

ViewState ile sayfa içerisindeki kontrollerin özellikleri dışında, yazılımcılar istedikleri verileri de taşıyabilmektedir. ViewState ile aynı adı taşıyan ViewState isimli nesne, aracılığıyla istenilen verileri saklayabilir ve aynı sayfa üzerinde gerçekleşen Postback işlemlerinden sonra bu veriler okunabilir. ViewState nesnesinin yapısını incelendiğinde geriye StateBag adında bir nesne örneği döndürdüğünü ve içerisindeki nesneleri IDictionary tipinde bir koleksiyonda sakladığı görülmektedir. Indeksleyiciler aracılığıyla içerisinde key-value çiftleri taşıyabilmektedir. Burada key değerimiz string, value değeri ise object tipinden olmalıdır. Yine saklanacak olan object tipindeki değer serileştirilebilir (serializable) olmak zorundadır. Yani .NET ortamında yazılımcılar kendi yazdıkları serileştirilebilir tipleri de ViewState içerisinde saklayabilmektedirler. Aşağıdaki kod bloğunda ViewState nesnesine, int ve DataTable tipinden değerlerin atanması gösterilmiştir.

```
ViewState["sayi"] = 1234;
ViewState["tablo"] = new DataTable("TestTablosu");
```

İndeksleyiciler aracılığıyla ViewState'e ekleme yapılabileceği gibi, Add metodunu kullanarak da **ViewState.Add("KeyAdı", value)** şeklinde bir nesne eklenebilir. ViewState'e eklenen nesneler Postback işlemi sonunda sayfanın HTML kodları içerisinde saklanacaktır. Postback işlemleri sonucunda hala aynı sayfanın üzerinde duruluyor ise ViewState'e atılan nesnelere aşağıdaki kod bloğunda görüldüğü gibi erişilebilir.

```
int tamsayi = (int)ViewState["sayi"];
DataTable dtTest = (DataTable)ViewState["tablo"];
```

ViewState içerisinde saklanılan değerler object tipinde saklanacağı için sayfa içerisinde bu verilere erişirken gerekli dönüşüm (cast) işlemlerini de gerçekleştirmek gerekecektir. ViewState içerisinden saklanan bir değeri çıkarmak için Remove metodu, programatik yollarla eklenen tüm verileri çıkarmak ise Clear metodu kullanılabilir. Aşağıda bu iki metodun kullanımı gösterilmiştir.

```
ViewState.Remove("tablo");  // tablo isimli nesne ViewState'den çıkarılır
ViewState.Clear();  // ViewState'e eklenen tüm kayıtlar silinir. (Kontrol bilgileri
silinmez!)
```

EĞİTİM :

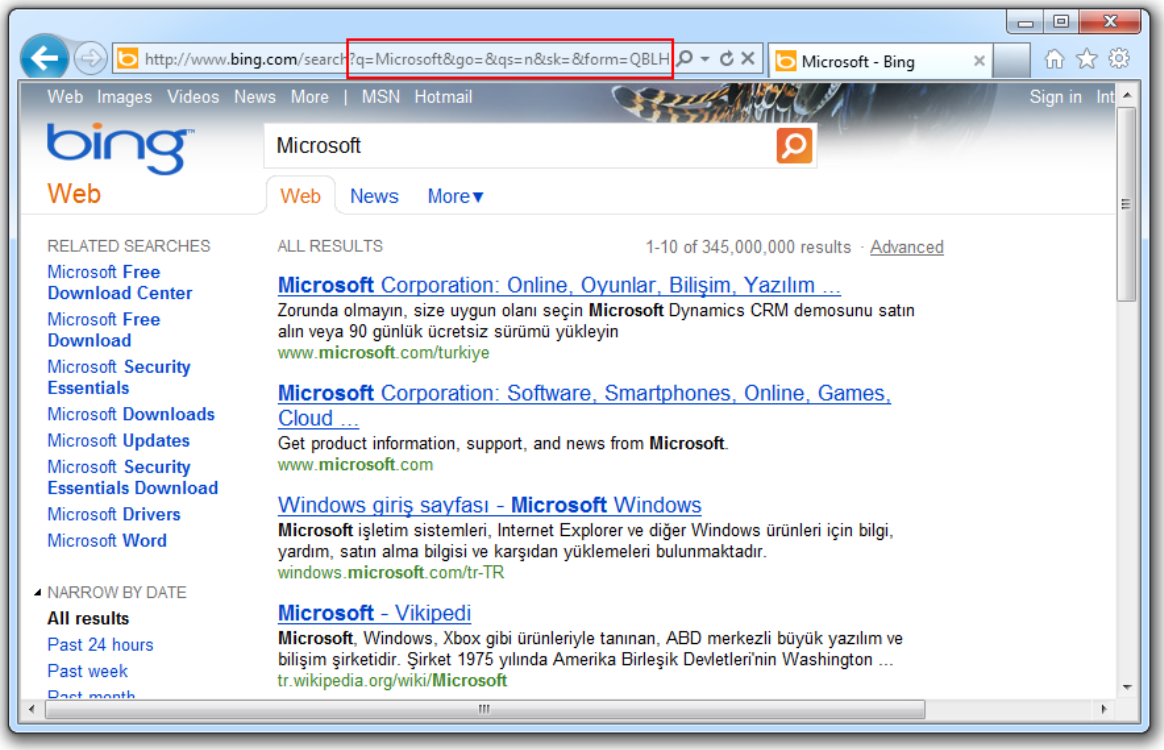
DURUM YÖNETİMİ VE TEMALAR

Bölüm :
Durum Yönetimi

Konu :
Query String(Sorgulama Cümlesi) ve Cookie (Çerez)

QueryString

QueryString nesnesi ASP.NET sayfaları arasında veri taşımakta kullanılan nesnelerden birisidir. QueryString ile taşınacak olan veri direkt URL aracılığı ile diğer sayfalara taşınabilmektedir. Taşınacak olan veriler sayfanın adı yazıldıktan sonra **?** işareti ile başlayan kısımda anahtar-değer (key-value) şeklinde taşınmaktadır. Bu yöntem kullanım kolaylığı ve sunucuya getirdiği yükün az olması nedeni ile pek çok yazılımcının tercihi olmaktadır ve hemen hemen her site bu yöntemi kullanmaktadır. Bing'de Microsoft sözcüğü aratıldığında sonuç sayfasının URL'ine dikkat edilecek olursa, sayfa adından sonra ? işareti ile başlayan kısımda sonuç sayfasına parametrelerin gönderildiği görülmektedir.



Resimdeki URL'i yakından incelediğimizde sözcük q=Microsoft şeklinde belirtilmiştir. Tabi sözcükle birlikte aramayı detaylandırabilmek için arada & işareti ile bir kaç tene daha anahtar değer çifti parametre olarak gönderilmiştir.

Resimden de görüldüğü gibi QueryString ile taşınan veriler kullanıcılar tarafından görüntülenebilmektedir ve dolayısıyla bir güvenlik açığı oluşturmaktadır. Kullanıcılar, bu verileri istedikleri şekilde değiştirip yönlendirilen forma farklı veriler gönderip farklı işlemler yapılmasını sağlayabilirler. Bu sorunu aşmak için veriler bir şekilde şifrelenerek gönderilse bile sonuçta verinin şifrelenmiş halini kullanıcılar görebileceklerdir. Bir takım yöntemlerle kullanılan şifreleme algoritması çözülüp yeniden şifrelenerek yine farklı bir veri gönderilebilir. Dolayısıyla, QueryString'e çok fazla güvenip çok önemli veriler taşınmamalıdır.

QueryString görüldüğü üzere çok kolay bir şekilde kullanılabilmektedir. Konuyu daha iyi anlatabilmek için bir örnekle ilerleyelim. Örneğimizde iki tane sayfa olsun ve bir sayfadan diğerine QueryString ile veri taşıyalım. Visual Studio'da yeni bir proje açıp iki tane sayfa ekleyelim. İlk sayfaya bir tane Button ekleyelim ve Button'un Click olayını ele aldıktan sonra aşağıdaki kodları ekleyelim.

```
protected void Button1_Click(object sender, EventArgs e)
```

```
    {
        Response.Redirect("Sayfa2.aspx?Goruntule=Microsoft");
    }
```

İkinci sayfada ise gönderilen Parametreyi ele alıp kullanmak gerekmektedir. Bunun için ikinci sayfanın Page_Load metoduna aşağıdaki kodları yazıp uygulamayı çalıştıralım. Uygulama çalıştırıldığında ilk sayfadan ikinci sayfaya Button aracılığı ile gittiğimizde ikinci sayfada göndermiş olduğumuz parametreyi görüyor olacağız.

```
protected void Page_Load(object sender, EventArgs e)
{
     string metin = Request.QueryString["YazilacakMetin"];
     Response.Write(metin);
}
```

Örnekten de anlaşılacağı gibi QueryString ile bir sayfadan başka bir sayfaya veri gönderirken ? işaretinden sonra **anahtar=deger** şeklinde veriler gönderilebilmektedir ve hedef sayfada da bu veriler **Request.QueryString[“anahtar”]** şeklinde ele alınıp kullanılabilmektedir.

QueryString ile gönderilen her anahtar-değer çiftine parametre denilmektedir. Bu parametre bir tane de olabilir birden fazla da olabilir. Gönderilecek olan parametrenin birden fazla olması durumunda parametreler **&** işareti ile birbirinden ayrılır. Aşağıdaki örnekte bu durum daha iyi anlaşılacaktır. Az önce yapılan örnek üzerinden devam ederek bu örneğe bir parametre daha ekleyip, diğer sayfada da eklenilen ikinci parametreyi görüntüleyelim. Aşağıdaki kod bloğunda, ilk sayfada bulunan Butonun Click olayının kodlarına **Goruntule2=www.microsoft.com** bölümü eklendikten sonraki son hali görünmektedir.

```
    protected void Button1_Click(object sender, EventArgs e)
    {

Response.Redirect("Sayfa2.aspx?Goruntule=Microsoft&Goruntule2=www.microsoft.com");
    }
```

İlk sayfanın hedef olarak gösterip parametreleri gönderdiği sayfa olan ikinci sayfanın Page_Load olayı da ikinci parametreyi alıp kullanmak için aşağıdaki gibi değiştirildi.

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

        string metin = Request.QueryString["Goruntule"];
        string metin2 = Request.QueryString["Goruntule2"];
        Response.Write(metin + " | " + metin2);
    }
```

QueryString nesnesinden veri alıp kullanacak olan sayfada parametrelerin taşıdığı değerler Request.QueryString[“Key”] şeklinde alınabileceği gibi parametrenin anahtar değeri yerine direkt index numarası da belirtilebilirdi. Bu işlem için Request.QueryString[0] şeklinde bir kod kullanılması gerekecekti.

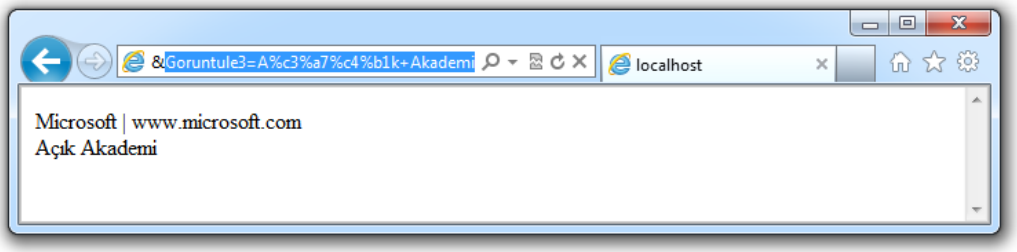
QueryString kullanılırken dikkat edilmesi gereken diğer bir nokta da QueryString ile taşınabilecek verilerin içereceği karakterlerin **ASCII** standartlarında olması gerekiyor. ASCII standartlarında olmayan karakterlerin taşınabilmesi için Server sınıfının **UrlEncode** ve **UrlDecode** metotları kullanılabilir. ASCII standartlarında olmayan karakterler **%** işareti ile başlayarak kodlanmaktadır. Bu durumu açıklamak için de az önceki örnekler üzerinden devam edilmektedir. İlk sayfaya bir parametre daha ekleyip içerisinde boşluk ve Türkçe karakter bulunan bir metin daha ikinci sayfaya gönderelim. Bu işlem için ilk sayfadaki Butonun Click olayındaki kodlar aşağıdaki hali alacaktır. Burada üçüncü parametrede kullanılan UrlEncode metoduna dikkat edelim.

```
   protected void Button1_Click(object sender, EventArgs e)
    {

Response.Redirect("Sayfa2.aspx?Goruntule=Microsoft&Goruntule2=www.microsoft.com&Gor
untule3="+Server.UrlEncode("Açık Akademi"));
    }
```

İlk sayfada Buton'a tıklanıldığında ikinci sayfaya yönlenirken URL'in hali aşağıda yer alan resimdeki gibi olacaktır. Üçüncü parametrenin URL'de taşınırken ASCII standartları içinde bulunmayan karakterlerin farklı birer hal aldıklarına dikkat edilmelidir. Üçüncü parametreyi görüntülemek için de ikinci sayfanın Page_Load olayındaki kodlar aşağıdaki hali alacaktır. Burada üçüncü parametre görüntülenirken daha önceki örneklerden faklı olan birkaç şey vardır. Bunlardan ilki Parametrenin içindeki değer alınırken anahtar isminin değil index numarasının kullanılması ve diğer fark da üçüncü parametre görüntülenirken UrlDecode metodunun kullanılıyor olmasıdır.

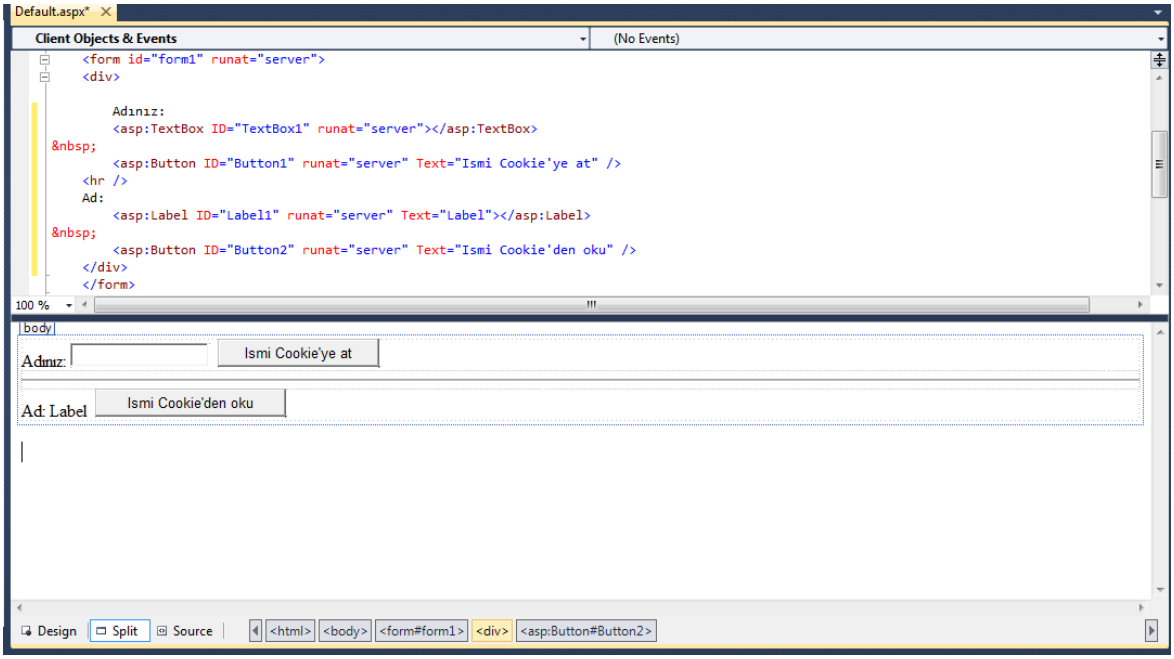
```
    protected void Page_Load(object sender, EventArgs e)
    {
        string metin = Request.QueryString["Goruntule"];
        string metin2 = Request.QueryString["Goruntule2"];
        string metin3 = Server.UrlDecode(Request.Params[2]);
        Response.Write(metin + " | " + metin2+"<br>"+metin3);
    }
```



Cookie

Web uygulamalarında bilgiler istemci tarafında HTML kodları içerisinden veya URL üzerinden taşınabileceği gibi, istemcinin bilgisayarında fiziksel olarak da saklanabilmektedir. Saklanan bu verilere sayfalar içerisinden erişilebilir ve bu bilgilere göre işlemler yapılması sağlanabilir. Cookie (çerez), istemci bilgisayarında disk üzerinde bu bilgilerin saklanmasını sağlayan nesnedir. Cookie nesnesinin en önemli avantajlarından biri uzun süre istemcinin bilgisayarında taşınabileceği için farklı zamanlarda erişilebilir olmasıdır. Fakat istemci kendi bilgisayarında Cookie'lerin saklanmasını engelleyebileceği için Cookie nesnesi ile işlemler yapmak bazen istenilmeyen sorunlara da yol açabilir.

ASP.NET uygulamalarında Cookie nesnesi ile ilgili işlemler yapabilmek için HttpCookies sınıfı kullanılmaktadır. HttpCookies sınıfından oluşturulacak nesne örneği üzerinden istemcinin bilgisayarına Cookie bilgilerinin yazılması, okunması ve bu bilgilerin ne kadar süre boyunca aktif tutulacağı gibi işlemlerin yapılması sağlanabilir. Cookie nesnesinin kullanımını görebilmek için basit bir form hazırlanıp, kullanıcıdan alınan isim ve şehir bilgileri istemcinin bilgisayarına kaydedilebilir. Yine benzer bir yol ile istemcinin bilgisayarındaki bu bilgiler okunup uygulama içerisinde ele alınabilir. Böyle bir işlem için aşağıdaki gibi bir form hazırlanabilir.



Formun üst kısmında bulunan TextBox'a girilen değerler **Ismi Cookie'ye At** butonuna tıklandığında istemcinin bilgisayarına yazılacaktır (Burada istemcinin bilgisayarında Cookie'lerin açık olması gerektiğinin unutulmaması gerekir). Daha sonra **Ismi Cookie'den Oku** butonuna tıklandığında, istemcinin bilgisayarından okunan değerler formun alt kısmında bulunan Label kontrolüne yazılacaktır. Aşağıdaki kod bloğunda metin kutularından alınan değerlerin Cookie nesnesine yazılaması ve bu nesneden okunması işlemleri sağlanmıştır.

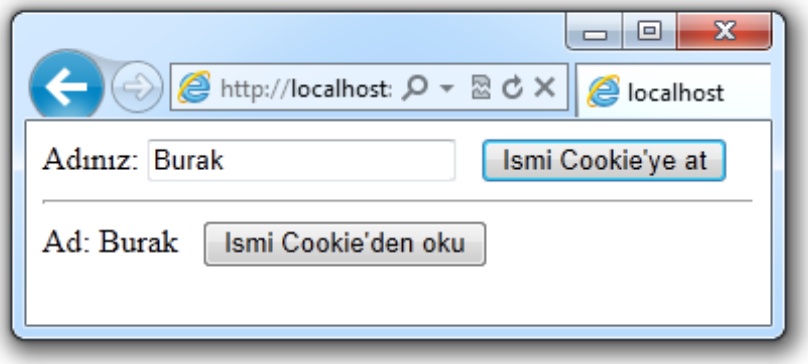
```
  protected void Button1_Click(object sender, EventArgs e)
    {
        // Cookie işlemlerini yapabilmek için HttpCookie sınıfınn nesne örneği
oluşturuluyor. Cookie nesnesine CookieTest ismi veriliyor.
        HttpCookie cerez = new HttpCookie("CookieTest");
        cerez["isim"] = TextBox1.Text;
        cerez.Expires = DateTime.Now.AddDays(10);
        Response.Cookies.Add(cerez);
    }

    protected void Button2_Click(object sender, EventArgs e)
    {
        HttpCookie cerezOku = Request.Cookies["CookieTest"];
        Label1.Text = cerezOku["isim"];
    }
```

Yukarıdaki ifadeleri incelediğimizde:

cerez["isim"] = TextBox1.Text; (isim adındaki Cookie anahtarına değer atanmasını sağlar)
cerez.Expires = DateTime.Now.AddDays(10); (Cookie nesnesinin 10 gün boyunca geçerli olacağını belirler)
Response.Cookies.Add(cerez); (cerez içerisinde taşınan değerlerin Cookie olarak istemcinin bilgisayarına eklenmesini sağlar)

Uygulama çalıştırıldığında metin kutularına bazı ifadeler girilip, Ismi Cookie'ye At butonuna tıklanırsa, girilen değerler istemcinin bilgisayarına yazılacaktır. Daha sonra Ismi Cookie'den Oku butonuna tıklandığında Cookie'den okunan değerler Label kontrollerine yazılacaktır.


http://localhost:
localhost
Adınız: Burak
Ismi Cookie'ye at
Ad: Burak
Ismi Cookie'den oku

EĞİTİM :

DURUM YÖNETİMİ VE TEMALAR

Bölüm :
Durum Yönetimi

Konu :
Session Management (Oturum Yönetimi)

Session

Web sunucusundan herhangi bir sayfa talep edildiğinde, web sunucusu her kullanıcı için bir oturum başlatır. Session nesneleri yardımıyla kullanıcıya ait olan bu oturum boyunca sayfalar arasında bilgi taşıma işlemi gerçekleştirilebilirsiniz. Bir alış veriş sitesini gözümüzün önüne getirdiğimizde, sepete atılan ürünlerin Session içerisinde taşındığını söyleyebiliriz. Tabi sadece bu değil pek çok bilgi de Session'da saklanabilmektedir. Session içerisinde taşınan veriler sitedeki tüm sayfalardan erişilebilir durumdadır. Burada, yine alış veriş sepetini gözümüzün önüne getirelim. Sepet sitenin bir yerinde sabit olarak durup aldığınız ürünleri size devamlı listeleyebilmektedir. Session'da saklanan veriler oturum açık olduğu sürece saklanır ve oturum kapatıldığında yok edilir. Durum yönetimi bölümünün başında da belirttiğimiz gibi Session sunucu taraflı bir nesnedir ve doğal olarak Session'da saklanan veriler sunucuda saklanır. Kullanıcılar siteyi terk ettiklerinde Session'daki bilgilerin temizlenmesi isteniyorsa muhakkak oturum sonlandırılmalıdır. Burada da Internet bankacılığı işlemlerini gözümüzün önüne getirelim, online banka şubesine girdiniz ve siz bilgisayardan kalktıktan sonra oturum oturum açık kalmaya devam etti. Bilgisayarınıza oturacak bir kişi hesabınızdaki bütün parayı dakikalar içerisinde kendi hesabına aktarabilir. Bu can alıcı örnekten de anlaşıldığı gibi Session içerisinde çok kritik veriler saklanabilmektedir ve bu sebeple Session mutlaka sonlandırılmalıdır. Session; oturum zaman aşımına uğradığında, kullanıcı pencereyi kapattığında, kullanıcı özellikle oturumu kapattığında ve sunucu yeniden başlatıldığında sonlanır.

ASP.NET'te her kullanıcı için oturum başlatılırken kullanıcıların farklı veya aynı bilgisayardan siteye erişip erişmedikleri önemli değildir. Aynı bilgisayarda bile olsa siteye erişen **her tarayıcı penceresi için bir oturum başlatılacaktır**.

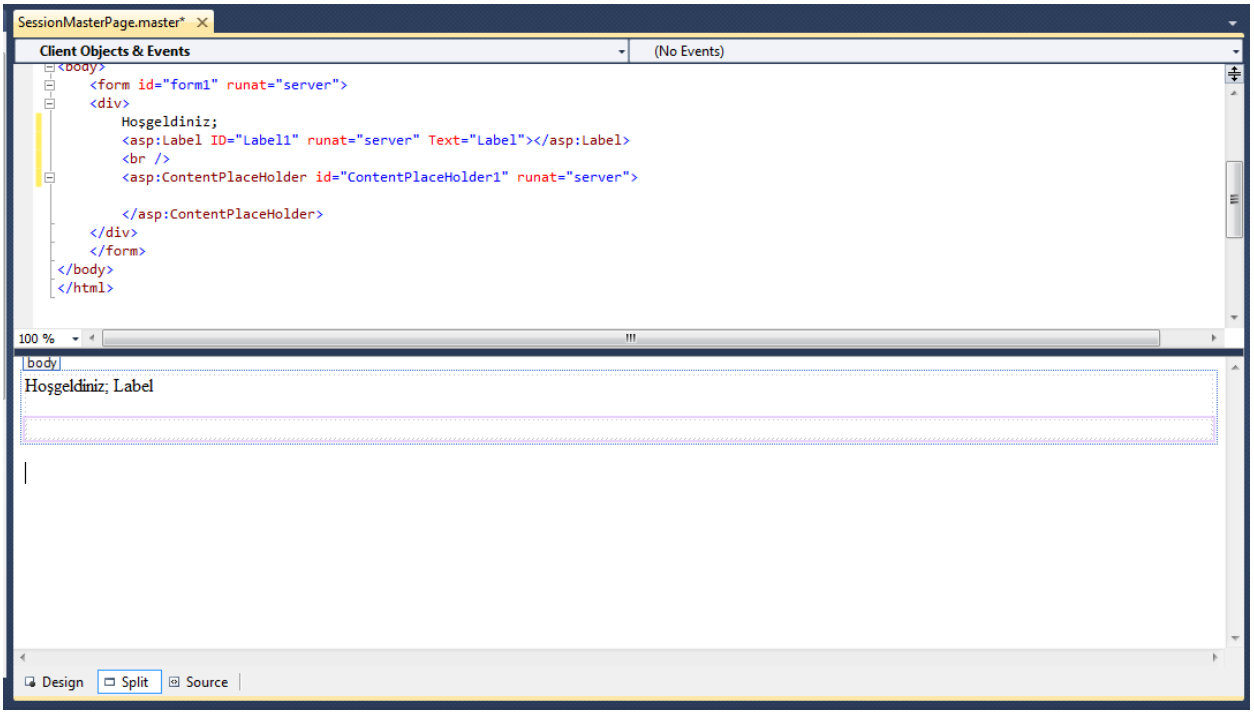
Sunucu tarafında oturum açan her kullanıcı için tane SessionID değeri üretilir. Bu değer hem sunucu tarafında hem de oturumun açıldığı bilgisayarda Cokkie'ler aracılığı ile saklanır. Burada ufak bir hatırlatma yapmakta fayda olacaktır. Session tekniğinde istemciye sadece SessinID değeri gönderilmektedir. Session içerisinde saklanan hiç bir değer kesinlikle istemciye gönderilmez ve sadece Sunucu tarafında saklanır. Bu sebeple Session nesnesi sayfalar arasında veri taşırken kullanılabilecek yöntemlerden en güvenli olanlarından biridir. Sunucu hangi çağrının hangi kullanıcıdan geldiğini buradaki Session değerlerini eşleştirerek bulabilir.

Cookie desteği aktif olmayan sunucularda SessionID değeri URL ile birlikte de saklanabilmektedir.

Session nesnesinde saklanan verilerin tamamı sunucu üzerinde saklandığı için Session'da gereğinden fazla veri saklıyor olmak sunucunun belleğini dolduracaktır. Çok kullanıcılı bir sistem tasarladığınızı var sayarsak Session üzerinde gereksiz verilerin saklanmasının önüne geçmeliyiz. Burada gerekli ya da gereksiz demek yerine kritik ya da kritik olmayan veriler şeklinde ikiye ayırmak daha anlamlı olacaktır. Çok daha güvenli olması nedeni ile kritik veriler Session'da, sayfa numarası gibi önemsiz veriler ise QueryString gibi yöntemlerle taşınmalıdır.

Session nesnesinin kullanımı ViewState nesnesi ile benzerlik göstermektedir. Session nesnesinde de veriler anahtar-değer çiftleri şeklinde taşınmaktadır. Session nesnesine veri atmak ve Session nesnesinde bulunan verileri okumak için yapılacak olan işlemlerin sözdizimi, ViewState nesnesi için yapılan işlemlerle aynıdır. Tabii ki Session nesnesinde, ViewState yerine Session sözcüğü kullanılacaktır. Session nesnesine bir veri atmak için kullanılacak olan sözdizimi **Session["keyAdi"]=deger** ya da **Session.Add("keyAdi",deger)** şeklinde olmaktadır. Veri okurken de kullanılacak olan sözdizimi **Session["keyAdi"]** şeklinde olacaktır. Şimdi birlikte bir örnek geliştirerek Session kullanımına göz atalım. Durum yönetiminde daha önceki örneklerimizde olduğu gibi bu örnekte de iki tane sayfamız olsun. Sayfalarımızın her ikisinin de üst tarafına bir bölüm ekleyip sayfada olan kullanıcımıza hoşgeldin diyebilceğimiz bir alan oluşturalım. Bu durumu daha az kod yazarak gerçekleştirmek için

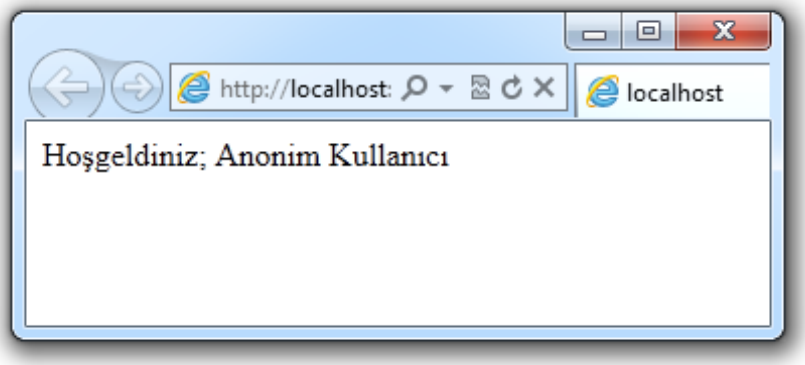
daha önceki bölümleri hatırlayalım. Bir tane MasterPage oluşturup diğer sayfalarımızı MasterPage'den türetelim ve hoşgeldiniz bölümünü aşağıdaki resimde de gördüğünüz gibi ContentPlaceHolder kontrolümüzün hemen üzerine ekleyelim.



MasterPage'i oluşturduktan sonra MasterPage çalıştırıldığında çalışmasını istediğimiz kodları PageLoad metoduna ekleyelim. Burada yapmayı hayal ettiğimiz şey, kullanıcı için Session nesnesinde bir değer saklanıyorsa ya da daha başka bir değilmle kullanıcı oturum açmış ise "**Hoşgeldiniz, Kullanıcı Adı**" yazması eğer kullanıcı oturum açmadıysa "**Hoşgeldiniz, Anonim Kullanıcı**" yazması. Burada yapılacak olan işlem oldukça basit. Session nesnesine değer atma ve değer okuma işlemininin teknik olarak ne şekilde yapabileceğimizi daha önce açıklamıştık. Burada Session'a KullaniciAdi anahtarı için bir değer atılmış mı onu kontrol edeceğiz eğer bu değer için herhangi bir değer atılmadıysa Hoşgeldinizi yazısından sonra eklemiş olduğumuz Label'a Kullanıcının belirteceği isim, eğer belirtilen anahtarın sorgulanmasında NULL değer dönüyor ise Anonim Kullanıcı sözcüklerini yazıyor olacağız. Bu işlemler için MasterPage'in PageLoad'ına yazdığımız kodları aşağıdaki bölümden inceleyebilirsiniz.

```
protected void Page_Load(object sender, EventArgs e)
    {
        if (Session["KullaniciAdi"]==null)
        {
            Label1.Text = "Anonim Kullanıcı";
        }
        else
        {
            Label1.Text = Session["KullaniciAdi"].ToString();
        }
    }
```

MasterPage'e gerekli kodu ekledikten sonra şimdi iki tane ContentPage oluşturalım. Bu sayfalardan bir tanesi Default.aspx diğeri ise OzelBilgiler.aspx isminde olsun. Sayfaları ekleyip projeyi çalıştırktan sonra aşağıdaki şekilde bir görünüm ile karşılaşmalısın.



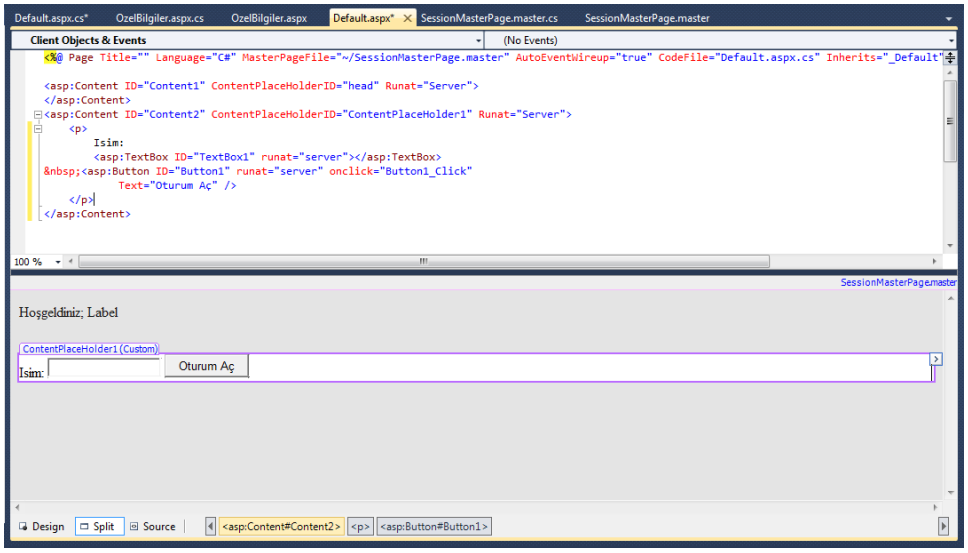
Eklemiş olduğumuz iki sayfadan OzelBilgiler.aspx'e oturum açmayan kullanıcıların erişmesini istemiyoruz. Normal şartlarda çok daha fazla kontrolden geçirilmesi gerektiği halde şu anda sadece Session'ı örneklemek adına kullanıcı için Session'da veri tutulup tutulmadığına bakıyor olacağız. Eğer Session'da data saklanıyorsa kullanının oturum açtığını varsayacağız, eğer Session'da data yoksa kullanıcıyı herhangi bir uyarı vermeden direkt default.aspx'e yönlendireceğiz. Bunun için OzelBilgiler.aspx'in PageLoad'ına aşağıdaki kodları yazıyor olalım.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Session["KullaniciAdi"]==null)
    {
        Response.Redirect("default.aspx");
    }
}
```

Kodlardan da anlayacağın gibi, ilk olarak kullanıcı için Session'da KullaniciAdi anahtarına bir değer atılıp atılmadığı kontrol ediliyor. Sonuç NULL döndüyse yani değer atılmadıysa kullanıcı direkt ana sayfaya yönlendiriliyor. Şimdi gelin kullanıcının oturum açmasını sağlayalım. Yukarıda da belirttiğim gibi normal şartlarda çok daha farklı teknikler kullanacağımız bu yöntem için şimdilik sadece bir tane TextBox ekleyelim ve girilen değerin Session'a aktarılmasını sağlayalım. Gerekli işlemi gerçekleştirmek için Default.aspx'e bir tane TextBox bir tane de Button ekleyelim. Button'un Text özelliğini Oturum Aç olarak ayarlayıp, Click olayını ele alalım. Button'un Click olayına aşağıdaki kodları yazıp Session'da data oluşmasını sağlayalım.

```
protected void Button1_Click(object sender, EventArgs e)
{
    Session.Add("KullaniciAdi", TextBox1.Text);
}
```

Gerçekleştirmiş olduğumuz sayfanın tasarımı aşağıdaki resimdeki gibi olmalıdır.



Şimdi sayfayı çalıştırıp Isim alanına birşeyler girip Oturum Aç Button'una tıkladıktan sonra, ikinci sayfaya geçtiğimizde artık ikinci sayfayı görüyor olmalıyız. İki sayfa arasında gezinirken şu anda URL'e elle müdahele edip geçiş yapmalıyız. Şimdi, tüm sayfalardan görünecek olan bir menü oluşturalım. Bunun için en anlamlı yer MasterPage olacaktır. MasterPage üzerine iki tane HyperLink bir tane de oturum kapatmak amacı ile LinkButton'u yan yana ekleyelim. HyperLink'ler tahmin edileceği üzere sayfaları işaret edecektir. LinkButton'un Text özelliğini de Oturumu Kapat olarak ayarlayıp Click olayını ele alalım. Bu işlemlerin ardından MasterPage'in body kısmının HTML kodları aşağıdaki hali alacaktır.

```
<body>
    <form id="form1" runat="server">
    <div>
        Hoşgeldiniz;
        <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
        <br />
        <asp:HyperLink ID="HyperLink1" runat="server"
NavigateUrl="~/Default.aspx">Ana Sayfa</asp:HyperLink>  
        <asp:HyperLink ID="HyperLink2" runat="server"
NavigateUrl="~/OzelBilgiler.aspx">Özel Bilgiler</asp:HyperLink>  
        <asp:LinkButton ID="LinkButton1" runat="server"
onclick="LinkButton1_Click">Oturumu Kapat</asp:LinkButton>
        <br />
        <asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server">

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

Şimdi sıra geldi oturumu kapatmaya. Bu işlem için LinkButton'ın Click olayını ele almıştık. Click olayı tetiklendiğinde çalışacak olan kodlar aşağıdakiler olmalıdır.

```
protected void LinkButton1_Click(object sender, EventArgs e)
    {
        Session.Abandon();
    }
```

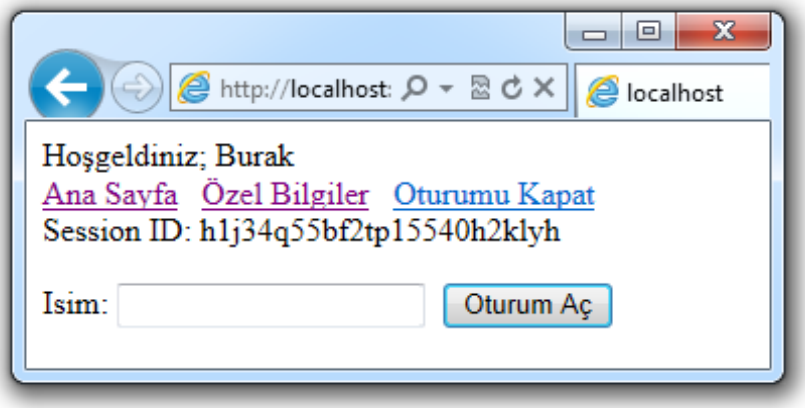
Session sınıfının **Abandon** metodu oturum kapatmak için kullanılmaktadır. Örneği çalıştırdığınızda ilk olarak TextBox'a herhangi bir şeyler giriip oturum açın ve ardından güvenli sayfamıza geçin. Ardından da oturumu kapattıktan sonra tekrar bu sayfaya geçmeyi deneyin. Şu ana kadar Session'ın kullanımı hakkında oldukça bilgi verdik. Yukarıda bahsettiğimiz özelliklerden SessionID'yi de görüntüleyip Session bölümünü

noktalıyor olalım. SessionID değerini görüntülemek için MasterPage'e bir tane Label atıp MasterPage'in PageLoad'ını aşağıdaki hale getirelim.

```
protected void Page_Load(object sender, EventArgs e)
 {
     if (Session["KullaniciAdi"]==null)
     {
         Label1.Text = "Anonim Kullanıcı";
     }
     else
     {
         Label1.Text = Session["KullaniciAdi"].ToString();
     }

     LabelSessionID.Text = Session.SessionID;
 }
```

Yukarıdaki kodlarda kalınla belirtmiş olduğumuz alanı yeni ekledik. Şimdi sayfayı çalıştırıp SessionID'yi görmeye çalışalım. İlk olarak Oturum açmadan ana sayfa linkine bir kaç defa tıklayalım ve her tıklamada yeni bir SessionID üretildiğini görelim. Bu durum son derece normaldir. Daha önceki açıklamalarımızı hatırlayacak olursak, Session ID oturum açılan bilgisayarlarlar ile sunucu tarafındaki oturumu eşleştirmek adına Cookie içerisinde saklanmaktaydı. Burada oturum açılmadığı için SessionID'nin Cookie'de saklanmasına gerek olmadığından her seferinde yeni bir ID üretiliyor. Bu durumu gözlemledikten sonra bir de TextBox'a bir şeyler yazıp Oturum Aç butonuna tıkladıktan sonra aynı şeyi yani defalarca ana sayfa linkine tıklamayı tekrarlayalım. Bu sefer SessionID'nin sabitlendiğini göreceksiniz. Çünkü Session'da bir değer saklanmaya başladı ve artık o kullanıcının oturum bilgileri anlamlı hale geldi. SessionID'nin de görüntülenmiş olduğu tasarımın çalışma zamanındaki halini aşağıda görebilirsiniz.



EĞİTİM :

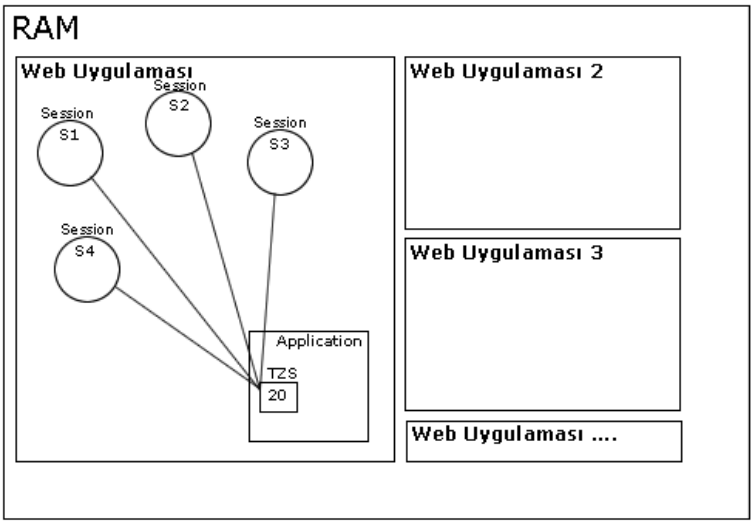
DURUM YÖNETİMİ VE TEMALAR

Bölüm :
Durum Yönetimi

Konu :
Application State Management (Web Uygulama Durum Yönetimi)

Application

Sitenin genelini ilgilendiren bilgilerin tutulabildiği nesne Application nesnesidir. Application nesnesi global bir değişken gibi düşünülebilir. Siteye erişen her kullanıcı aynı bilgileri görüp değiştirebilmektedir. Application nesnesinde de veri saklanırken, tıpkı Session ve ViewState nesnelerinde olduğu gibi **anahtar-değer** çiftleri kullanılır. Session ve ViewState nesnelerinde her anahtarda tutulan değer kullanıcıya özgü bir değerken, Application nesnesinde tutulan değer tüm kullanıcılar için ortak olmaktadır. Aşağıdaki resmi dikkatle incelediğimizde Application nesnesinin kullanım alanını daha iyi anlayabileceğiz.



Resmi yakından incelediğimizde RAM içinde her kullanıcı için farklı bir Session açıldığı görülmektedir. Bu kullanıcıların her birinin SessionID değerleri birbirinden farklıdır, her Session içinde kullanıcılara özel bilgiler birbirinden farklı olarak saklanabilmektedir. Session içinde saklanılan değerlerin anahtarları aynı olsa bile şekilden görüldüğü üzere her Session'ın farklı alanı olduğu için anahtarlar içinde tutulacak olan değerler birbirinden faklıdır. Application nesnesi için durum biraz farklıdır her uygulamanın kullandığı bir Application alanı vardır ve tüm kullanıcılar bu alan üzerinde işlem yaparlar. Resimde dikkat edilecek olursa tüm Session'lar bir tane Application alanına erişip orada işlem yapmaktadır. Application alanında tüm Session'ların erişip işlem yaptığı alan da TZS alanıdır. Application nesnesinde sitenin genelini ilgilendiren nesnelerin saklandığından bahsedilmişti gerçek hayata uyarlamak adına resimde görülen TZS alanını Toplam Ziyaretçi Sayısı olarak düşünüldüğünde yeni bir Session açıldığında bu sayı bir tane arttırılarak sitedeki toplam ziyaretçi sayısı hesaplanmış olmakta ve siteye bağlanan herkese de bu alandaki değer gösterilmektedir.

Application nesnesinin kullanırken kullanılacak olan sözdizimi de ViewState ve Session gibidir. Application nesnesine veri eklerken **Application.Add("key", "değer")** ya da **Application["key"] = "değer"** sözdizimi kullanılırken, Application nesnesinden veri okumak için **Application["key"]** sözdizimi kullanılmaktadır.

Application nesnesinin daha iyi anlaşılabilmesi için Resimde düşünülen senaryoyu gerçeğe dönüştürelim. Hatta bu senaryoya biraz daha ekleme yapılıp sitede o an kaç kişinin bulunduğu da belirlenebilsin. Bu işlemler için Global.asax dosyasında bulunan metotları kullanmalıyız.

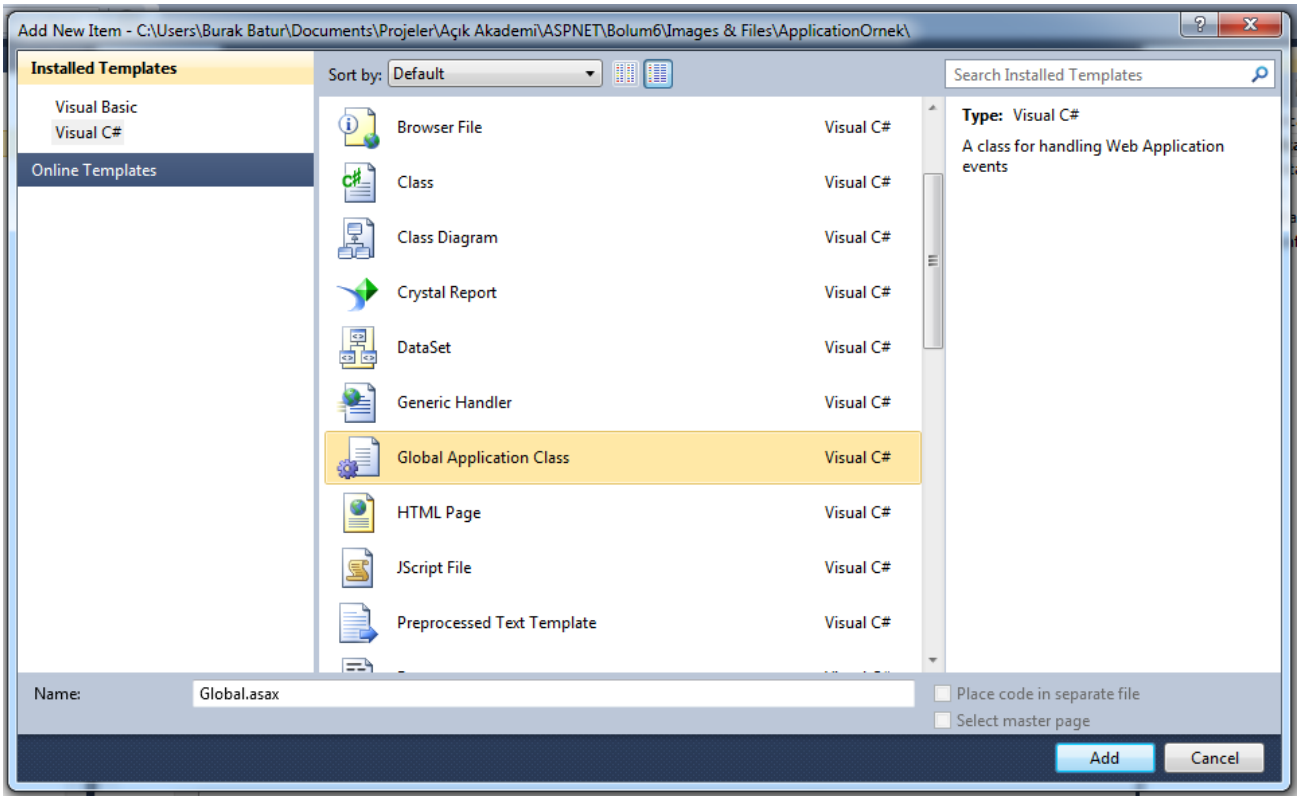
Global.asax dosyası ASP.NET Uygulama Dosyası (ASP.NET Application File) olarak da tanınan, uygulama düzeyinde (Application-level) ya da oturum düzeyinde (Session-level) olaylar

tetiklendiğinde çalıştırılmak üzere metotlar taşıyan kullanımı isteğe bağlı bir dosyadır. Global.asax dosyası uygulamanın en üstteki (root) klasöründe bulunur ve çalışma zamanında dinamik olarak derlenerek oradan çalışır. Uygulama dosyası URL'den talep edildiğinde bu talep reddedilir ve dosya kullanıcılara gösterilmez veya indirilemez böylece dosyanın içindeki kodlar korunmuş olur.

Global.asax dosyasında varsayılan olarak toplam beş tane olay yakalayıcı metot (Event Handler) bulunmaktadır. Bu metotlar, aşağıdaki tabloda bulunan olaylar tetiklendiğinde çalıştırılır.

Global.asax Dosyasında Bulunan Olay Yakalayıcıları	
Application_Start	Uygulama ilk defa çalıştırılırken tetiklenecek olan olay.
Application_End	Uygulama bilinçli bir şekilde kapatılırken tetiklenecek olan olay.
Application_Error	Uygulamada bir hata oluştuğunda tetiklenecek olan olay.
Session_Start	Web sitesinde yeni bir oturum açıldığında tetiklenecek olan olay.
Session_End	Web sitesinde bir oturum sonlandığında tetiklenecek olan olay.

Örnek geliştirilmeye başlanmadan önce yeni bir tane web sitesi projesi açılıp bir tane global.asax dosyası eklenmelidir. global.asax dosyasını projeye yeni öğe ekleme menüsünden ekleyebilirsiniz.



Site ilk defa çalıştırılırken toplam ziyaretçi sayısını ve o an sitede bulunan ziyaretçi sayısını tutmak üzere Application nesnesine iki adet faklı anahtar belirlenerek veri atılacaktır. Bu işlemi yapacak olan kodlar aşağıda listelenmektedir.

```
void Application_Start(object sender, EventArgs e)
    {
        // Code that runs on application startup
        Application.Add("TZS", 0);//Toplam ziyaretçi sayısı TZS isimli
             anahtarda tutulacaktır.
        Application["OZS"] = 0;//Online ziyaretçi sayısı OZS isimli
             anahtarda tutulacaktır.
```

```
    }
```

Yukarıdaki kod bloğu incelendiğinde iki farklı yöntemle Application nesnesine veri atıldığı görülmektedir. Her oturum açıldığında hem toplam ziyaretçi sayısı hem de online ziyaretçi sayısı bir arttırılmalıdır. Bu işlem için yazılan kodlar aşağıda yer almaktadır.

```
    void Session_Start(object sender, EventArgs e)
    {
        // Code that runs when a new session is started
        Application["TZS"] = ((int)Application["TZS"]) + 1;
        Application["OZS"] = ((int)Application["OZS"]) + 1;
    }
```

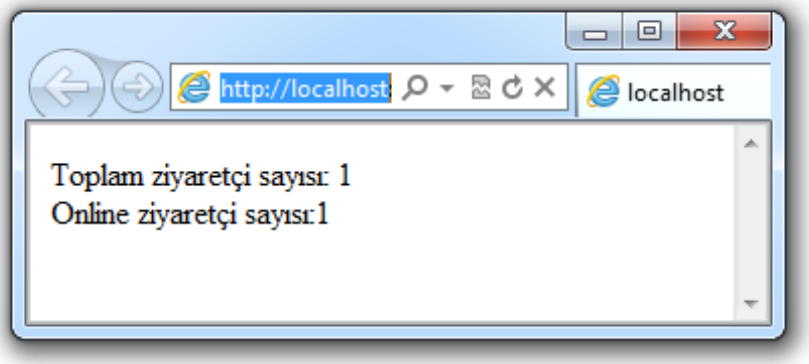
Yukarıdaki kodlar incelendiğinde Application nesnesinden alınan verinin int türüne dönüştürüldüğü görülmektedir. Application nesnesi içerisine **object** türünden veriler alır ve geriye **object** türünden veriler döndürür. Application nesnesinde yer alan veriler kullanılmak istenildiğinde kullanılmak istenilen türe dönüştürülmelidir. Şu ana kadarki kodlar Application nesnesine ilk değerleri tanımlamakta ve her oturum açıldığında da bu sayıları arttırmaktadır ancak online ziyaretçi sayısı hesaplanırken kullanıcıların oturumlarının sonlandığı durum ele alınmalı ve bu değer bir azaltılmalıdır. Bu işlem için yazılan kodlar aşağıda yer almaktadır.

```
void Session_End(object sender, EventArgs e)
    {
        // Code that runs when a session ends.
        Application["OZS"] = ((int)Application["OZS"]) - 1;
    }
```

Ana sayfanın page_load olayına da aşağıdaki kodlar eklenerek siteyi ziyaret eden kullanıcılara sitede o anda kaç kullanıcı olduğu ve siteyi uygulama çalıştırıldığından bu yana kaç kullanıcının ziyaret ettiği gösterilebilir. Burada senin de fark edeceğin gibi Response.Write ile direkt sayfaya içerik ekliyoruz, daha görsel bir tasarım için sunucu kontrollerinden faydalanabilirsin.

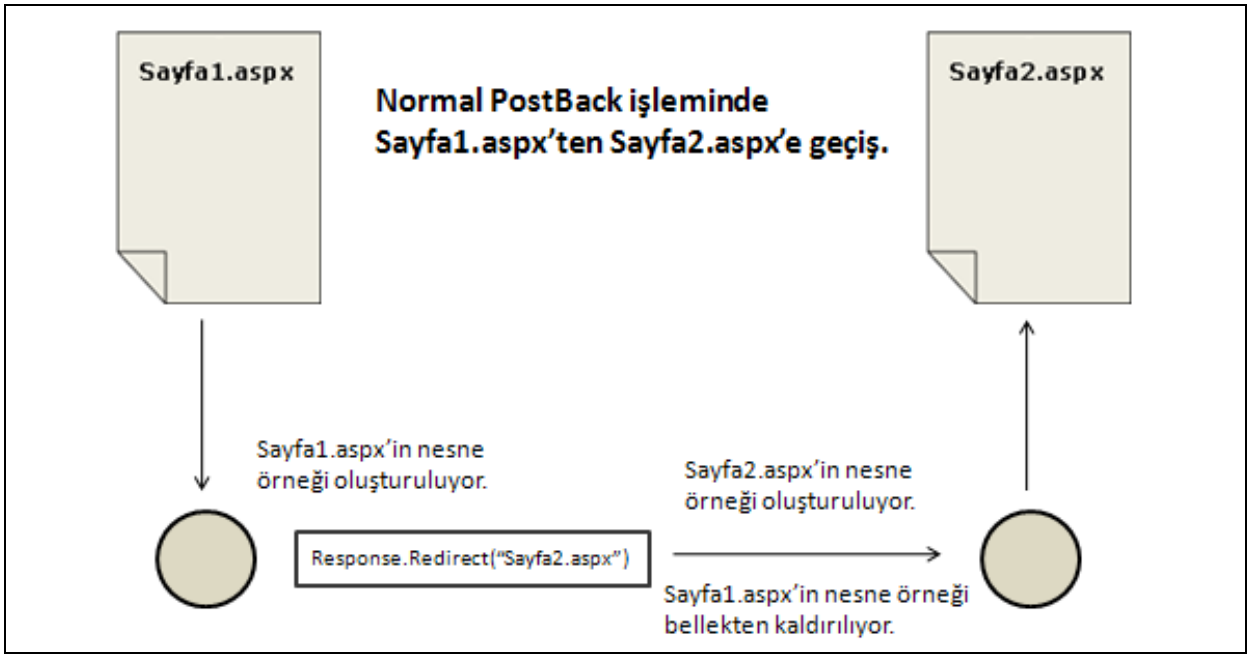
```
protected void Page_Load(object sender, EventArgs e)
    {
        Response.Write("Toplam ziyaretçi sayısı: "+Application["TZS"]);
        Response.Write("<br/>Online ziyaretçi sayısı:
"+Application["OZS"]);
    }
```

Uygulama ilk defa çalıştırıldığında aşağıda yer alan görüntü ile karşılaşılacaktır. Uygulamaya faklı tarayıcı pencerelerinden erişilerek bu değerler arttırılabilir. Uygulamayı çalıştıran web server servisi kapatılıp yeniden açılıp uygulama tekrar çalıştırıldığında Application_Start olayı yeniden çalışacak ve her şey yine sıfırdan başlayacaktır.

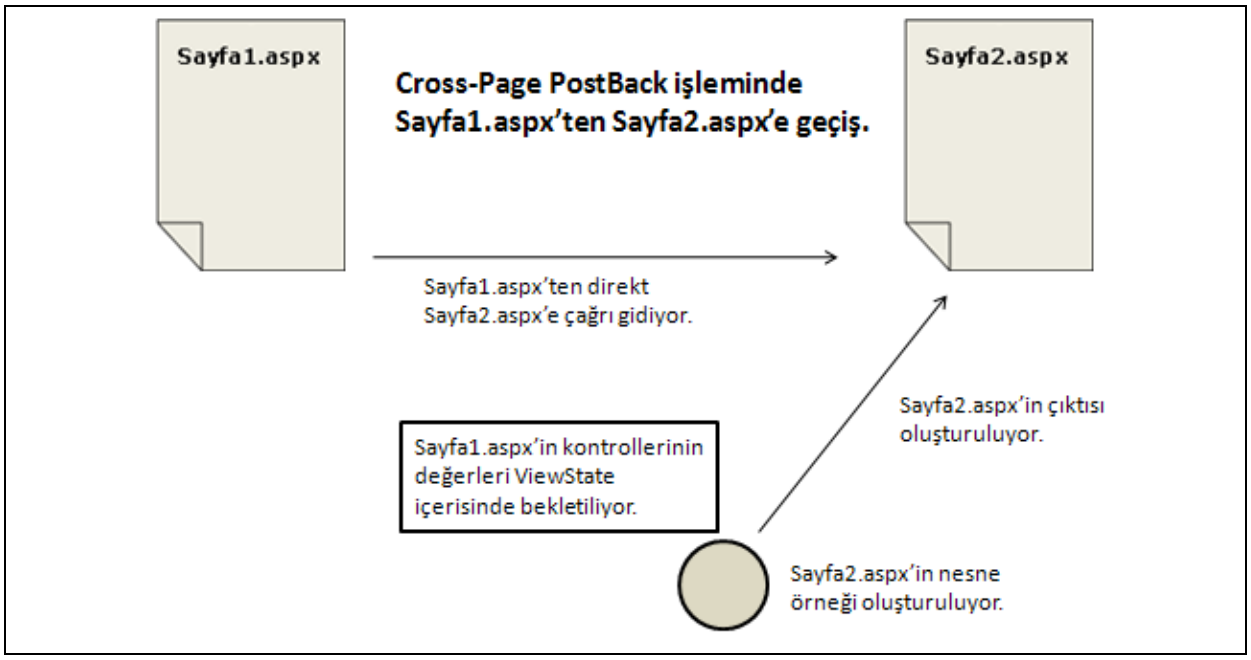


Cross-Page PostBack

Asp.Net 1.0 / 1.1 ile gelen yeniliklerden en önemlisi, sayfaların kendi kendilerine form verilerini postalayabilme kabiliyetleridir. Öyleki, Asp.Net ile geliştirilen web sayfaları aslında birer sınıf nesnesi olduklarından, sayfa üzerindeki form kontrollerine ve değerlerine kolayca erişilebilmektedir. Ancak bazı zamanlarda, sayfalarda yer alan form verilerinin başka sayfalara gönderilmesi istenilebilir. **Cross-Page PosBack** olarak adlandırılan bu işlemlerin, Asp.Net 2.0 ve sonraki sürümlerde gerçekleştirilmesi hem daha kolay hem de daha etkili hale getirilmiştir.



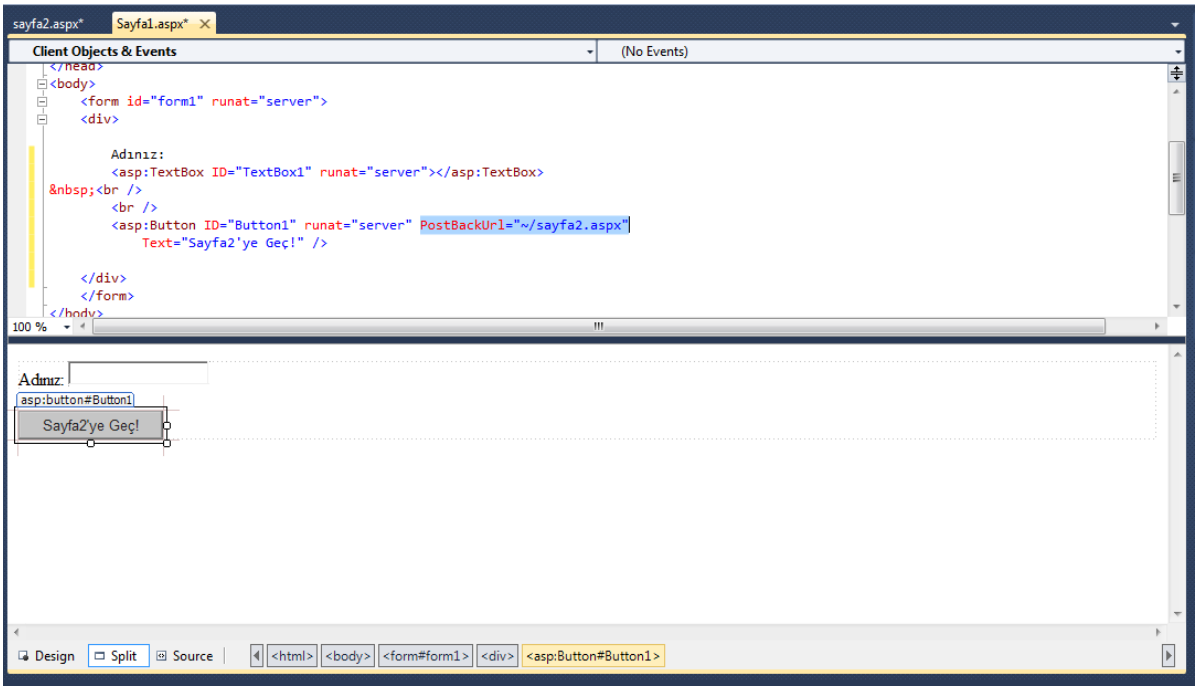
Yukarıdaki resimde Sayfa1.aspx'ten Sayfa2.aspx'e Response.Redirect() metodu ile geçiş gösterilmektedir. Bu döngüde bellekte ilk olarak Sayfa1'in nesne örneği oluşturuluyor daha sonra Response.Redirect("Sayfa2.aspx") komutu işletiliyor ve ilk sayfanın nesne örneği bellekten düşürülüp ikinci sayfanın nesne örneği oluşturulup çıktı gönderiliyor.



Cross-Page PostBack işleminde durum üstteki resimde görüldüğü gibidir. Sayfa1.aspx'den Sayfa2.aspx'e Cross-Page PostBack ile gidilmeye çalışıldığında bellekte Sayfa2.aspx'in nesne örneği oluşturulacak ve çıktısı üretilecektir ancak bu sırada Sayfa1.aspx'in kontrollerinin değerleri de ViewState içerisinde Sayfa2.aspx'in erişebileceği bir alanda saklanmaktadır. Sayfa2.aspx kendisini çağıran Sayfa1.aspx'e erişmeye çalıştığında Sayfa1.aspx'in de bir nesne örneği oluşturulup ViewState'den kontrollerin değerleri yüklenip Sayfa2.aspx'den erişilebiliyor olacaktır.

Cross-Page PostBack işlemini kullanmak için IButtonControl ara yüzünün (interface) uygulanmış olduğu kontrollerin (Button, LinkButton, ImageButton) PostBackUrl özelliği kullanılmalıdır. Üstteki resimde Sayfa1.aspx'den Sayfa2.aspx'e gitmek için Sayfa1.aspx'deki herhangi bir Button kontrolünün PostBackUrl özelliği Sayfa2.aspx olarak ayarlanmış ve istenilen durum sağlanmıştır. Cross-Page PostBack'te bir önceki sayfaya erişmek için **PreviousPage** sınıfı kullanılmalıdır. Bu durumun daha iyi anlaşılması için bir örnekle ilerleyelim.

Örneği geliştirmek için bir tane Visual Studio açıp web projesi oluşturalım. Projeye Sayfa1.aspx ve Sayfa2.aspx adında iki tane web sayfası ekleyelim. Sayfa1 aşağıdaki resimde görüntülendiği gibi tasarlanabilir.



Sayfa1.aspx'in HTML kodlarında kontrollerin özellikleri incelendiğinde Buttonun özelliği aşağıdaki kod parçasındaki gibi olacaktır. Burada dikkat edilmesi gereken özellik PostBackUrl özelliğinin Sayfa2.aspx olarak ayarlanmış olmasıdır.

```
<asp:Button ID="Button1" runat="server" PostBackUrl="~/sayfa2.aspx"
          Text="Sayfa2'ye Geç!" />
```

Asp.Net içerisinde yer alan önemli özelliklerden birisi de PostBackUrl özelliğidir. Bu Url, button kontrolüne basıldığında, Form üzerindeki verilerin belirtilen url' deki sayfaya gönderilmesini sağlamaktadır. İşleyiş şekli son derece etkilidir. Sayfa2'deki durumu gözlemlemek için Sayfa2'de herhangi bir yere bir tane Label ekleyelim. Sayfa2'nin daha hoş görünmesi için Label eklediğim yerin hemen önüne bir de "Hoşgeldin," metnini ekledim. Label eklendikten sonra ilk sayfadan gelen verilerin alınması için, senin de tahmin edeceğin gibi bu veri kullanıcının TextBox'a girecek olduğu veridir, gerekli kodları sayfanın PageLoad metoduna yazalım.

```
protected void Page_Load(object sender, EventArgs e)
  {
      TextBox tbAd = (TextBox)PreviousPage.FindControl("TextBox1");
      Label1.Text = tbAd.Text;
  }
```

Dikkat edilecek olursa burada son derece güçlü bir teknik kullanılmaktadır. Bir adet TextBox kontrolü tanımlanmış ve oluşturulmuştur. TextBox kontrolü oluştururken, **PreviousPage** sınıfının **FindControl** metodu ile, bu sayfaya Post işlemi ile form verisi gönderen sayfadaki TextBox1 kontrolü bulunmaktadır. Bu işlem, bulunan kontrolün TextBox kontrolü olarak cast edilmesi ile tamamlanır. Dolayısıyla, Sayfa2.aspx sayfasının Page_Load olay metodunda, kullanılabilecek bir adet TextBox kontrolü olacaktır. Bu TextBox kontrolü aslında, Sayfa1.aspx sayfasından gelen kontroldür. Böylece, Sayfa1.aspx sayfasındaki TextBox1 kontrolünün değerine ve özelliklerine bu sınıf içerisinden (Sayfa2.aspx sayfasından) kolayca erişilebilir. Asp.Net 2.0 ile birlikte, başka sayfalara form verisi gönderme işlemlerinde böylesine güçlü bir teknik ile gelmektedir. Uygulama çalıştırılıp test edildiğinde Sayfa1.aspx'teki verilerin Sayfa2.aspx'e sorunsuzca taşındığı görülecektir.

Burada ufak bir hatırlatma yapmakta fayda olacaktır. Konular ele alınırken muhtamel hata durumlarını engellemek için yazılması gereken kodlar, konunun anlaşılmasında konuya odaklanmayı kolaylaştırmak adına pas geçilmiştir. Yukarıdaki örneği sayfa1 yerine sayfa2'den çalıştırırsanız hata alacaksınız. Bu hatanın nedeni PreviousPage nesnesinin NULL dönüyor olmasıdır. Yani olmayan bir nesnenin olmayan bir kontrolünü elde etmeye çalışıyoruz. ViewState, QueryString, Session, Application ve Cookie'de de benzer durumlar söz konusudur. Eğer ilgili anahtardaki değer daha önce oluşturulmadı ise olmayan bir şeyi string'e dönüştürmeye kalktığınızda hata alacaksın. Bu gibi durumların önüne geçmek için basit bir NULL kontrolü yapabilirsin. Örneğin, sayfa2.aspx'in Page_Load'ını aşağıdaki hale getirerek uygulamnın sayfa2.aspx'den başlatılmasında dahi problem yaşanmayacaktır çünkü PreviousPage nesnesi var mı, yok mu diye kontrol ediyoruz.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (PreviousPage == null)
    {
        Response.Write("Lütfen önce ilk sayfayı ziyaret edin...");
    }
    else
    {
        TextBox tbAd = (TextBox)PreviousPage.FindControl("TextBox1");
        Label1.Text = tbAd.Text;
    }
}
```

EĞİTİM :

DURUM YÖNETİMİ VE TEMALAR

Bölüm :
Temalar

Konu :

Temalar

Temalar

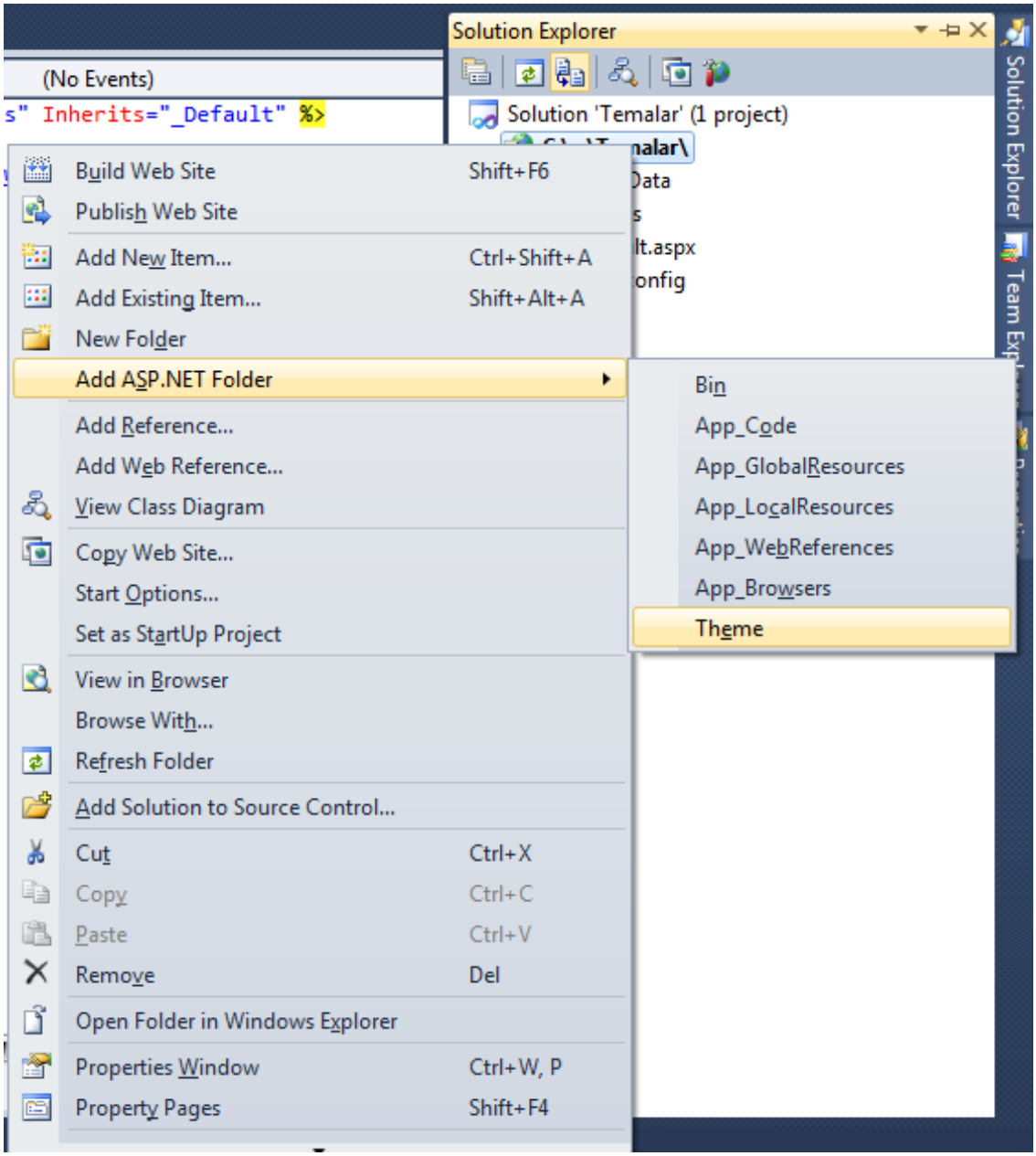
Temalar genel olarak bir uygulamanın nasıl görüntüleneceğini belirleyen yapılar olarak tanımlanabilir. ASP.NET'ten bağımsız olarak düşünüldüğünde işletim sistemi temaları ele alındığında işletim sisteminde yer alan pencerelerin, klasörlerin vb. diğer öğelerin nasıl görüntüleneceği işletim sisteminin teması tarafından belirlenmektedir. Windows 7'nin temasında şeffaf klasörler ile karşılaşılıyorken, Windows XP'de şeffaf olmayan ama oldukça güzel bir tema kullanıcıları karşılıyor, Windows'un daha eski versiyonarında ise keskin kenarlı ve daha az grafik içeren bir tema söz konusu. İşin güzel tarafı bir Windows 7 kullanıcısı bu temalardan istediğini seçip bilgisayarına uygulayabiliyor ve bilgisayarını nasıl görmekten hoşlanıyorsa o şekilde görebiliyor. Cep telefonlarında da durum bu şekildedir. Yan yana duran aynı marka ve aynı model cep telefonun menülerinin görünümü birbirinden faklı olabiliyor çünkü kullanıcılara pek çok tema sunuluyor ve kullanıcılar kendilerine en hoş geleni, bakmaktan en fazla zevk aldıkları görünümü telefonlarına uygulayıp kullanıyorlar.

Web sitelerinin görünümleri de en az işletim sistemleri, telefonlar, kişilerin dış görünümleri kadar önem arz etmektedir. Çünkü bir web sitesinin büyüklüğü siteyi ziyaret eden kullanıcı sayısı ile doğru orantılıdır. Siteyi ne kadar fazla kişi ziyaret ederse sitenin reklam gelirleri de o oranla artıyor olacaktır, hatta site online bir satış sitesi ise siteyi ziyaret eden kullanıcıların bir de ürün satın aldıkları düşünüldüğünde sitenin ziyaretçi sayısı direkt maddi kazanç olarak geriye dönecek bir kavram haline gelecektir. Bu sebeple web sitelerinin görünümü oldukça önemlidir, tasarım web tabanlı bir uygulamada oldukça önemli bir yer teşkil etmektedir. Sitenin kullanıcı sayısını arttırmak için sitenin tasarımının çok güzel olması da yeterli olmayacaktır çünkü kullanıcılar aynı görünüm ile karşılaşmaktan sıkılacaklar ve bir süre sonra sitenin ziyaretçi sayılarında düşüş yaşanmaya başlanacaktır. Bu durumun önüne geçmek için de sitenin görünümünü belli aralıklarla değiştirmek gerekmektedir.

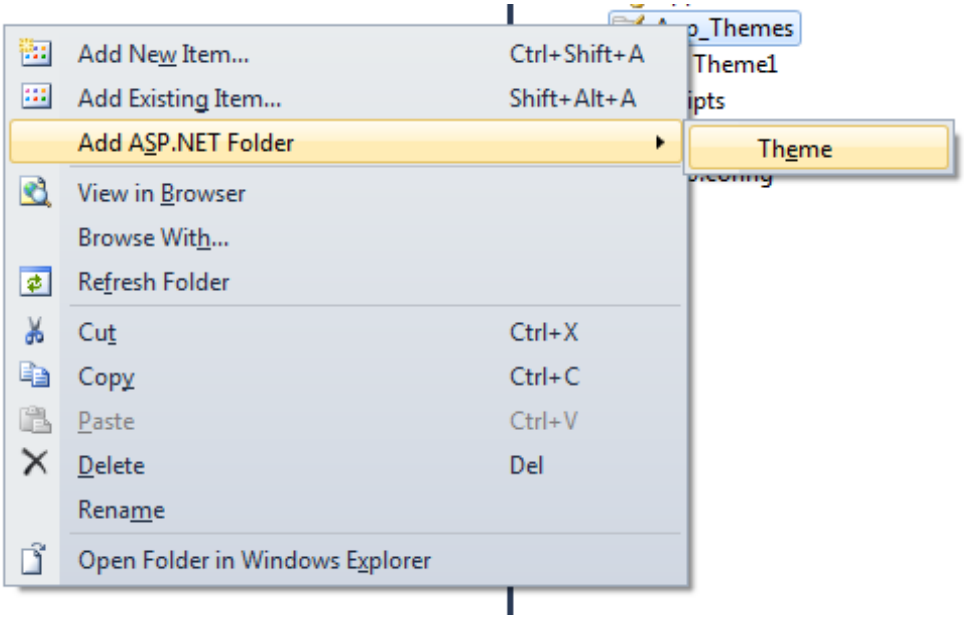
Beş on sayfadan oluşan bir site için sitenin görünümünü değiştirmek pek kolay olmamakla birlikte daha fazla örneğin yüz ya da beş yüz sayfadan oluşan bir sitenin görünümünü değiştirmek problem oluşturacaktır, bu sebeple tasarım anında gerekli planlama yapılıp oldukça esnek bir site tasarlanmalıdır. Sadece HTML'den oluşan siteler için sitenin sitillerini ayrı bir fiziksel dosyada barındırıp tüm sayfalara uygulamanın bir yolu mümkündür ve sitilleri barındıran dosyalara da **CSS** (Cascading Style Sheet) dosyaları denilmektedir ancak ASP.NET sunucu kontrolleri de işin içine girdiğinde tek başına CSS yeterli olmamakta ve sorun oluşmaktadır. **ASP.NET 2.0** ile birlikte bu sorun çözülerek **Tema** kavramı ASP.NET'te de yazılım geliştiricilerin hayatına girmiştir. Temalar ile birlikte ASP.NET sunucu kontrollerinin sitilleri de tek bir yerde depolanıp uygulamanın tüm sayfalarına uygulanabilmektedir, temaların sunduğu tek avantaj tabi ki bu değildir. Temalar tüm sitedeki sayfalara web.config dosyasında yazılacak tek satır kodla uygulanabilmektedir ve böylece her sayfaya temaların CSS'de olduğu gibi teker teker dahil edilmesine gerek kalmamaktadır. Temalar CSS dosyaları ile de sorunsuz bir şekilde çalışabilmekte ve oldukça güzel görünümler elde edilebilmektedir.

App_Themes Klasörü

ASP.NET 2.0 ve daha yeni versiyonlar ile hazırlanan bir sitede tema kullanmak için uygulamaya App_Themes klasörünün eklenmesi gerekmektedir. Daha önceki bölümlerden hatırlanacağı üzere App_Themes klasörü ASP.NET'in özel klasörlerinden birisidir ve içinde Temaları barındırır. Uygulamaya App_Themes klasörünü eklemek için Visual Studio ortamında proje adının üzerinde sağ tuşla tıklayıp **Add ASP.NET Folder** seçenği altında bulunan **Theme** seçilmelidir. Bahsedilen yol izlendiğinde App_Themes klasörü uygulamaya eklencek ve bu klasörün içinde de başka bir klasör yer alıyor olacaktır.



App_Themes klasörü altında yer alan klasörler de ASP.NET'in özel klasörleri arasındadır ve bunlara Theme adı verilir. App_Themes klasörü uygulamaya eklendikten sonra Theme1 isimli bir klasör yani **Theme1** isimli bir tema projeye eklenmiş durumdadır. Klasörün ismi değiştirilerek temanın ismi de değiştirilmiş olur ve sayfalarda klasörün ismi ile kullanılabilir. Projeye yeni tema eklemek için takip edilmesi gereken yol da App_Themes klasörü eklenirken takip edilen yolla benzerlik göstermektedir. Projeye yeni tema eklemek için App_Themes klasörü üzerinde sağ tıklanarak **Add ASP.NET Folder** seçeneğinden **Theme** seçilmelidir. Bu seçimden sonra App_Themes klasörü altına yeni bir klasör oluşturulacaktır ve ismi değiştirilip diğer işlemler gerçekleştirilebilir.



Projeye temalar eklendikten sonra temalar için gerekli ayarlamalar gerçekleştirilmeye başlanabilir. Tema klasörleri içlerine **.skin** uzantılı **Skin** doslayarı, **CSS** dosyaları ve resim dosyaları kabul edebilir. Skin dosyası ASP.NET sunucu kontrollerinin sitillendirmelerinin kaydedilebileceği dosyalardır, CSS dosyaları ise daha önce de bahsedildiği gibi HTML kontrolleri ve etiketleri için sitillerndirmelerin barındırabileceği dosyalardır. Temanın uygulandığı sayfalardaki görünümler Skin ve CSS dosyalarındaki verilere göre belirlenmektedir, dolayısıyla Skin ve CSS dosyalarının kullanımı bu alanda önem kazanmaktadır.

EĞİTİM :

DURUM YÖNETİMİ VE TEMALAR

Bölüm :
Temalar

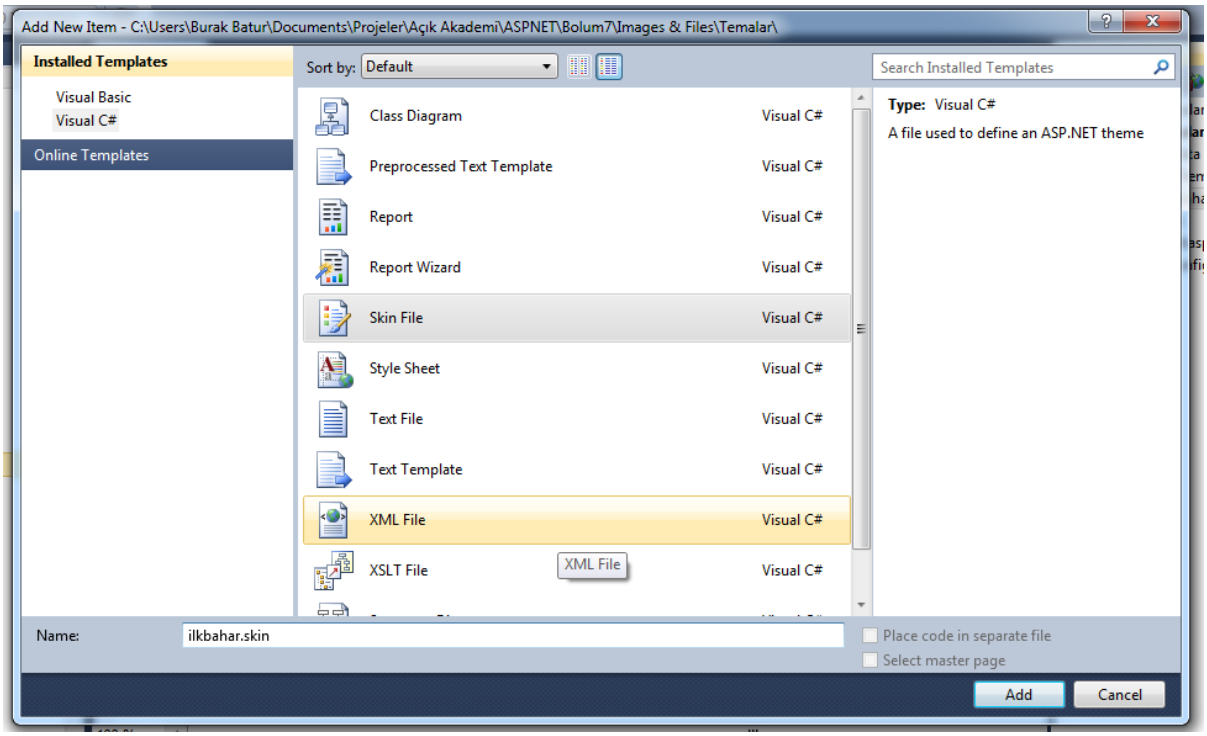
Konu :

Skin Dosyaları

Skin Dosyaları

ASP.NET sunucu kontrollerinin stillerini barındıracak olan özel dosyalardır. Skin dosyasında bulunan her bir sitile **Skin** adı verilmektedir, dosyanın adı da buradan gelmektedir. Bir sayfa için tema belirlendiği takdirte sayfa kullanıcıya gönderilirken kontroller Skin dosyasında tanımlanan tasarım uygulanarak kullanıcıya gönderilmektedir. Skin dosyaları içinde bir kontrol için birden fazla sitil tanımlanabilir ve sayfa içerisinde kontrollerde kullanılmak istenilen sitil belirtilerek kullanılabilir bu yöntem ilerleyen ekranlarda ele alınmaktadır.

Temaya yeni bir Skin dosyası eklemek için temanın klasörü üzerine sağ tuşla tıklanarak tema klasörü içerisinde kullanılabilecek dosyaların listelendiği pencere açılır ve açılan pencereden Skin File seçeneği seçilip Ok tuşu ile Skin dosyası temaya eklenir.



Temaya birden fazla Skin dosyası eklemekte mümkündür, tabi burada dikkat edilmesi gereken nokta daha önce oluşturulan dosya ile aynı ismi taşıyan bir dosya eklemeye çalışmamak olmalıdır. Temaya birden fazla Skin dosyası eklenmesi durumunda temanın uygulanmış olduğu sayfalardaki kontroller tüm Skin dosyalarındaki bilgilere göre oluşturuluyor olacaktır. Tek dosya eklemekten farkı yok gibi görünse de aslında kod karmaşıklığının önüne geçmek amacı ile Skin tanımlanacak olan her kontrol için yeni bir Skin dosyası ekleyip gerekli tanımlamaları her kontrolün kendi Skin dosyasında yapmak mantıklı olacaktır.

Temaya yeni bir Skin dosyası eklendiğinde örnek olarak iki tane Skin ile karşılaşılacaktır. Bunlardan ilki GridView diğeri ise Image kontrolü için tanımlanmış olan aşağıdaki kod bloğunda görüntülenen Skin'lerdir.

```
<%--

Default skin template. The following skins are provided as examples only.

1. Named control skin. The SkinId should be uniquely defined because
   duplicate SkinId's per control type are not allowed in the same theme.

<asp:GridView runat="server" SkinId="gridviewSkin" BackColor="White" >
   <AlternatingRowStyle BackColor="Blue" />
</asp:GridView>
```

```
2. Default skin. The SkinId is not defined. Only one default
   control skin per control type is allowed in the same theme.

<asp:Image runat="server" ImageUrl="~/images/image1.jpg" />
--%>
```

Yukarıdaki kodlar incelendiğinde Skin tanımlarken kullanılan kodların sayfaya kontrol eklerken kullanılan kodlar ile hemen hemen aynı olduğu fark edilmelidir. Ancak kodlar daha yakından incelendiğinde buradaki kodlarda **ID** özelliğinin yer almadığı dikkat çekiyor olmalıdır. Tanımlanan Skin'ler ID şeklinde bir özellik içermezler bunun yerine **SkinID** adında bir özellik içerebilirler ya da içermeyebilirler. Yeni Skin dosyası eklendiğinde örnek kodlarda yer alan açıklama da bu durum da yer almaktadır. Tanımlanan bir Skin'de SkinID yer alıyorsa bu Skin'e "**Named Skin**" (İsimlendirilmiş Skin), eğer SkinID özelliği yer almıyorsa buna da "**Default Skin**" (Varsayılan Skin) denilmektedir.

Varsayılan Skin (Default Skin)

Bir kontrole Skin tanımlanırken SkinID özelliği belirtilmiyorsa bu şekilde tanımlanan Skinler'e Varsayılan Skin denilmektedir. Bir kontrol için sadece bir tane Varsayılan Skin tanımlanabilir ve temanın uygulanmış olduğu sayfada aksi belirtilmedikçe aynı tipteki tüm kontrollere uygulanır. Bir TextBox kontrolü için Varsayılan Skin aşağıda görüldüğü şekilde tanımlanabilir.

```
<asp:TextBox  runat="server"
    BackColor="#3399FF"
    BorderColor="#333333"
    BorderStyle="Solid"
    ForeColor="White"
    Width="150px" />
```

Kodlardan anlaşılacağı üzere TextBox'ın arka plan, çerçeve ve yazı rengi belirlendi, çerçeve sitili "Solid" olarak belirlendi ve genişliği de "150px" olarak ayarlandı. Görüleceği üzere Skin tanımlamak sayfa üzerindeki herhangi bir kontrolü biçimlendirmekten çok farklı değil. Fark ID özelliğinin olmaması ve kontrol kapatılırken "/>" etiketi kullanılarak kapatılıyor olması. Eğer TextBox kontrolünün taşıdığı başka üyeler olsaydı </asp:TextBox> şeklinde kapatılacaktı ve <asp:TextBox> .... </ asp:TextBox> etiketleri arasında bu üyelerin sitilleri de ayrı ayrı belirtilebiliyor olacaktı. Bu durumu örneklemek için Skin dosyası sayfaya eklendiğinde Visual Studio tarafından oluşturulan örnek Skinler'deki GridView'e göz atılabilir. Yukarıdaki Skin'i içeren temanın uygulanmış olduğu tüm sayfalarda aksi belirtilmedikçe tüm TextBox'lar burada belirtilen şekilde görüntülenecektir.

Visual Studio ortamında Skin dosyasında kod yazarken intellisense (kod tamamlama) özelliği çalışmamaktadır. Intellisense özelliği olmadan kod yazma işlemi hem daha zahmetli hem de daha riskli bir hal almaktadır. Skin tanımlarken Visual Studio'nun özelliklerini kullanmak için herhangi bir sayfada Skin tanımlamak istenilen kontrolün sitil özellikleri belirlenir ve kopyala – yapıştır metodu ile Skin dosyasına eklenip gerekli düzenlemeler yapılarak Skin tanımlama işlemi işlemi daha kolay, hızlı ve güvenli bir şekilde gerçekleştirilebilir.

İsimlendirilmiş Skin (Named Skin)

Bir kontrol için birden fazla Skin söz konusu ise bunlardan sadece biri varsayılan skin olabilir. Diğer Skin'ler için mutlaka bir adet **SkinID** özelliği belirtilmelidir. SkinID özelliği sayfaya kontrol eklerken kullanılan ID özelliği gibi düşünülebilir, dolayısıyla ID özelliği için geçerli olan tüm kısıtlamalar SkinID özelliği için de geçerli olmaktadır. Yukarıda varsayılan Skin'i tanımlanmış olan TextBox'a bir tane İsimlendirilmiş Skin aşağıda görüldüğü gibi tanımlanabilir.

```
<asp:TextBox SkinID="Email"  runat="server"
    Font-Bold="True"
    BackColor="#3399FF"
    BorderColor="#333333"
    BorderStyle="Solid"
    ForeColor="White"
    Width="200px" />
```

Görüleceği üzere bu Skin bir öncekinden farklıdır ve bu Skin'in kullanılmak istenildiği TextBox'ların SkinID özelliği Email olarak ayarlanmalıdır. Yukarıdaki Skin tanımlamasında varsayılandan farklı olarak font kalınlaştırılmış ve TextBox'un genişliği biraz arttırılmış durumdadır.

EĞİTİM :

DURUM YÖNETİMİ VE TEMALAR

Bölüm :
Temalar

Konu :

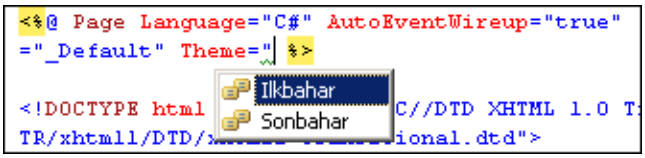
Temaların Siteye Uygulanması

Temaların Siteye Uygulanması

Oluşturulan bir temayı site ya da sayfa düzeyinde uygulamak mümkündür. Sayfa düzeyinde tema uygulamak için sayfanın Page direktifi kullanılabilir ve istenilen tema Page direktifinden ayarlanabilir. Site düzeyinde tema belirtmek içinse tahmin edileceği üzere uygulama ayarlama dosyası devreye girecek ve gerekli ayarlar web.config içerisinde gerçekleştiriliyor olacaktır.

Sayfa Düzeyinde Tema Belirlemek

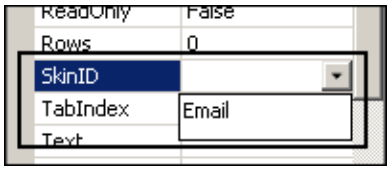
Sayfaya tema uygulamak için sayfanın Page direktifinde bulunan **Theme** özelliği kullanılabilir. Visual Studio ortamında sayfaya tema eklemek istenildiğinde Theme özelliği eklendikten sonra Visual Studio projede bulunan temaları listeliyor olacaktır, listelenen temalardan herhangi biri seçilip sayfaya uygulanabilir.



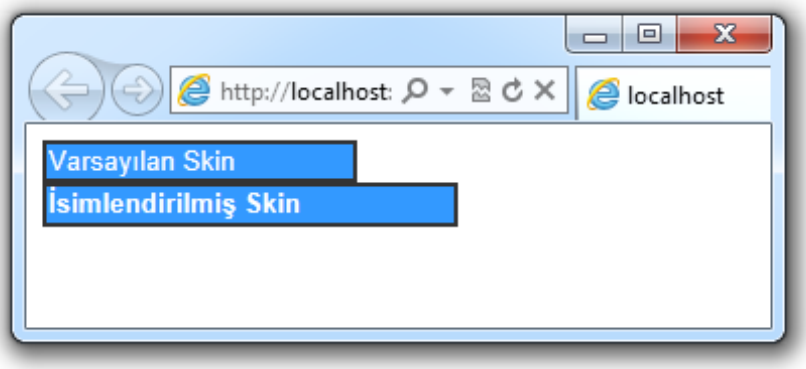
Sayfaya tema eklenmesi.

Sayfaya tema uygulandıktan sonra Skin tanımlanan tüm kontroller Skin dosyasında belitilen görünüme bürünecektir. Tema içerisinde Skin tanımlanan kontrollerin tamamı varsayılan Skin'de tanımlandığı gibi görülecektir. Eğer kontrol için isimlendirilmiş bir Skin tanımlanmış ise eğer istenilirse kontrolün özellikler penceresinden tanımlanmış olan Skin'lerden biri seçilebilir.

Bir kontrole İsimlendirilmiş Skin uygulamak için kontrolün **SkinID** özelliğinde kullanılmak istenilen Skin belirtilmelidir. Visual Studio tema eklerken projede bulunan temaları listeleyebildiği gibi bir kontrol üzerinde tanımlı olan Skin'leri de listeleyebilir ve bunlar arasından bir seçim yapılmasını sağlayabilir. Bu durumu örneklemek için ilk olarak bir tane Proje açın ve ardından bahsedilen işlemleri teker teker uygulayıp önce tema ardından skin ve css dosyalarını oluşturup içeriklerini önceki örnekler ile doldurun. Ardından hemen önceki konuda anlatıldığı gibi ilkbahar temasını bir sayfaya uygulayın. Ilkbahar temasının uygulandığı sayfaya iki adet TextBox sürükleyip bırakıp ikinci eklenenin özellikler penceresinde SkinID özelliği ayarlanmak istenildiğinde özellikler penceresinde aşağıdaki görünüm ile karşılaşılacaktır.



Sayfaya iki adet TextBox eklenip alttakinin SkinID özelliği Email olarak ayarlnamasının ardından dizay anında herhangi bir görünüm değişikliği olmadığı halde sayfa çalıştırıldıktan sonra sayfanın görünümü aşağıdaki hali alacaktır. Görüleceği üzere arka plan renkleri değişti ve alttaki TextBox'ın boyu biraz daha uzun hale geldi.



Yukarıdaki ekran görüntüsünde Ilkbahar adı verilen temanın uygulandığı sayfanın ekran görüntüsü yer almaktadır. Daha önceki sayfalardan hatırlanacağı üzere proje içerisinde bir de Sonbahar isimli tema yer almaktaydı ancak sonbahar isimli temanın içerisinde herhangi bir Skin tanımlaması yapmamıştık. Sonbahar isimli tema içerisinde TextBox kontrolü için Skin tanımlayalım. Bu işlem için bir adet Skin dosyası ekleyip dosyanın içerisine aşağıdaki kodları yazalım. Ilkbahar temasında yazı hatırlatacak olan renkler kullaılmışken sonbaharda ise kış mevsimini hatırlatan renkler kullanılıyor olacaktır. Aşağıda Skin dosyası içinde yer alan kodlar yer almaktadır.

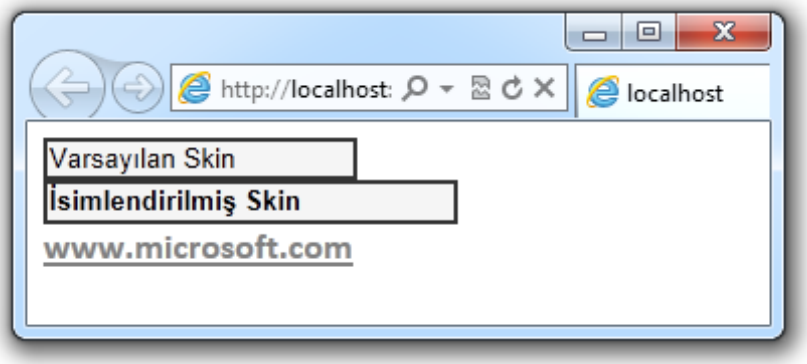
```
<asp:TextBox  runat="server"
    BackColor="WhiteSmoke"
    BorderColor="#333333"
    BorderStyle="Solid"
    ForeColor="Black"
    Width="150px" />


<asp:TextBox SkinID="Email"  runat="server"
    Font-Bold="True"
    BackColor="WhiteSmoke"
    BorderColor="#333333"
    BorderStyle="Solid"
    ForeColor="Black"
    Width="200px" />
```

Yukarıdaki kodlar ile Ilkbahar teması içinde buluanan kodlar karşılaştırıldığında aradaki farkın arka plan ve font renklerinde olduğu görülecektir. Hatırlanacağı üzere ilkbahar teması içerisine bir adet de CSS dosyası dahil edilmişti. Sonbahar temasına da bir adet CSS dosyası dahil edilip Ilkabahar temasındakine benzer ayarlar yapıldığında CSS dosyası aşağıdaki görünümü alacaktır.

```
a
{
    font-family: Calibri;
    font-size: large;
    font-weight: bold;
    font-style: normal;
    font-variant: normal;
    text-transform: none;
    color: Gray;
    text-decoration: underline;
}
.ArkaPlanRengi
{
    background-color: WhiteSmoke;
}
```

CSS dosyasında da Ilkbahar dosaysındakine göre ufak tefek değişiklikler yapılıp sadece .ArkaPlanRengi sitili ile a etiketinin renk özelliği değiştirilmiştir. CSS dosyasının kullanımı örneklemek için sayfa içerisine TextBox'ların altına bir adet link eklenip Page direktifinde Theme olarak **Sonbahar** ayarlandıktan sonra sayfanın çalışma zamanındaki görünümü aşağıdaki hali alacaktır. Şu andaki örnekte sadece TextBox'ların sitilleri üzerinde oynama yaptık ancak unutmamak gerekir ki ASP.NET sunucu kontrollerinin tamamı için sitillendirme ayarları yapılabilmektedir. Görüleceği üzere sayfayı yaz modundan kış moduna almak bu kadar kolaydır.



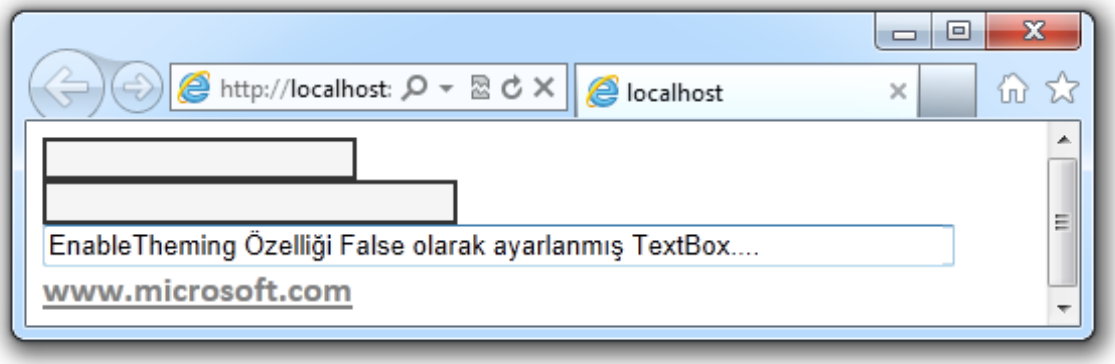
Görüleceği üzere ekstra bir işlem yapmadan CSS dosyasını tema içerisine eklemek CSS'in sayfaya uygulanaması için yeterli olmuştur. CSS dosyasını sayfaya direkt eklemek için HTML kodlarında HEAD bölümünde aşağıdaki söz dizimi kullanılabilir.

```
<head>
    ...
   <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
```

CSS dosyası Solution Explorer'dan <head>...</head> bölümü arasında sürüklenip bırakıldığında Visual Studio gerekli kodları oluşturacaktır.

Kontrole Uygulanan Skin'i İptal Etmek

Sayfaya tema uygulandığında sayfada bulunan ve Skin tanımlanan tüm kontroller eğer özel bir SkinID belirtilmemişse Varsayılan Skin'de tanımlanan şekilde görüntüleniyor olacaktır, ancak zaman zaman diğerlerinden daha farklı görünen kontroller tasarlanmasına ihtiyaç duyulabilir. Böyle bir durumda kontrolün üzerinde gerekli gerekli ayarlamalar yapıldığında Skin dosyasında tanımlı olan özellikler üzerinde yapılan değişikliklerin çalışma zamanında kaybolduğu görülecektir. Böyle bir durumun önüne geçmek için kontrolün temadan aldığı özelliklerin iptal edilmesi gerekmektedir. Kontrollerin **EnableTheming** özelliklerine **False** değeri atanarak temada tanımlanan Skin'in kontrole uygulanmasının önüne geçilmiş olunur. Aşağıdaki resimde görülen alttaki TextBox'ın EnableTheming özelliği False olarak ayarlanmış ve Skin dosyasında bulunan Varsayılan Skin'den daha faklı bir tasarım gerçekleştirilmiştir.



Sayfaya Çalışma Zamanında Tema Uygulamak

Temalar sayfanın Page direktifinden belirtilebileceği gibi çalışma zamanında da belirtilebilir. Çalışma zamanında tema değiştirilebilmesi, kullanıcıların sayfayı kendilerine özgü bir şekilde kişiselleştirilebilmeleri için seçilebilecek yollaradan biridir. Kullanıcıların sayfayı kişiselleştirmesi pek çok kurum tarafından etkin bir şekilde kullanılmaktadır, örneğin geliştirilen bir sistemde kullanıcılar sayfanın renklerini tuttukları takımın rengi şeklinde seçebildiklerinde o sistem içerisinde daha fazla ve eğlenceli vakit geçireceklerdir, bu da uygulamayı geliştiren veya kullanan şirkete ekstra müşteri memnuniyeti ve eksta gelir olarak geri dönebiliyor olacaktır.

Sayfanın temasını çalışma zamanında belirlemek için kontroller oluşturulmadan önce gerekli kodlar yazılmalıdır. Kontroller oluşturulmadan önce yapılmak istenilen işlemler sayfanın **PreInit** olayında yazılmalıdır. PreInit olayı ele alınıp tema bu alanda veya daha önce belirlenmelidir. Sayfanın temasının kod tarafında değiştirmek için **Page** nesnesinin **Theme** özelliği kullanılmalıdır. Theme özelliği ile sayfaya uygulanacak olan tema belirlenebileceği gibi uygulanmış olan tema da elde edilebilir. Sayfanın temasını kod tarafından değiştirmek için aşağıdaki kodlar kullanılabilir.

```
void Page_PreInit(object sender, EventArgs e)
{
    Page.Theme = "Ilkbahar";
}
```

Site Düzeyinde Tema Belirlemek

Site düzeyinde tema belirlemek için uygulama ayarlama dosyası (web.config) kullanılmalıdır. Web.config içerisinde sayfaların genel ayarlarının belirlenebileceği <system.web> ... </system.web> etiketleri arasında yer alan **<Pages>** etiketi vardır. <Pages> etiketi içerisinde sayfaların kullanacak olduğu tema da belirlebilmektedir. Tanımlanan tema tüm siteye uygulanmak isteniyorsa temanın teker teker sayfalara uygulanmasına gerek yoktur, bu işlem web.config aracılığı ile gerçekleştirilmelidir. Sayfanın teması web.config aracılığı ile belirtildiğinde temanın değiştirilmesi de oldukça kolay bir şekilde yapılabilir, bir yerdeki özellik değiştirilirek sitenin tüm görünümü değiştirilebilmektedir, böylece eğer istenilese haftanın günleri için dahi birer tema hazırlanıp her gün değiştirilebilir. Web.config içerisinde tema belirlemek için aşağıdaki kodlar kullanılmalıdır.

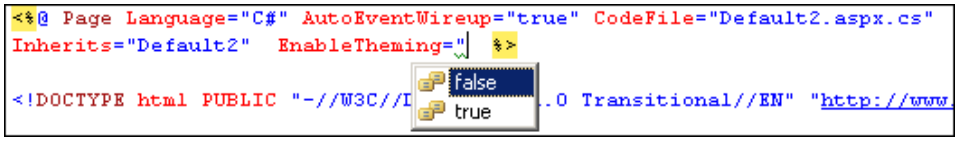
```
<system.web>
      <pages theme="Ilkbahar" />
      ...
</system.web>
```

Web.config'e yukarıdaki kodlar eklendikten sonra tüm sayfalara Ilkbahar teması uygulanıyor olacaktır, eğer istenilirse Ilkbahar değeri değiştirilerek tüm sitenin görünümü değiştirilebilir.

Site düzeyinde tema belirlendiği bir durumda sayfa düzeyinde de farklı bir tema uygulandığında sayfa düzeyinde uygulanan tema site düzeyinde uygulananı ezer ve sayfa düzeyinde Page direktifinde belirtilen tema sayfa için geçerli olur.

Sayfaya Uygulanan Temayı İptal Etmek

Hatırlanacağı üzere kontrole ugulanan Skin iptal edilebilmekteydi. Benzer bir durum sayfa için de söz konusudur. Bir sayfanın tasarımı olduğu gibi diğer sayfalardan farklı olacaksa teker teker içerisinde bulunan kontrollerine Skin uygulanmasını iptal ettmek yerine sayfaya uygulan temayı iptal etmek daha mentıklı bir durum olacaktır. Sayfaya uygulanan bir temayı iptal etmek için kontrol de olduğu gibi **EnableTheming** özelliği kullanılacaktır. Sayfanın Page direktifinde kullanılan EnableTheming özelliğine False değer atandığında web.config'de belirtilen tema sayfaya uygulanmayacak ve sayfa içerisinde sayfaya özgü bir sitillendirme belirlenebilecektir.



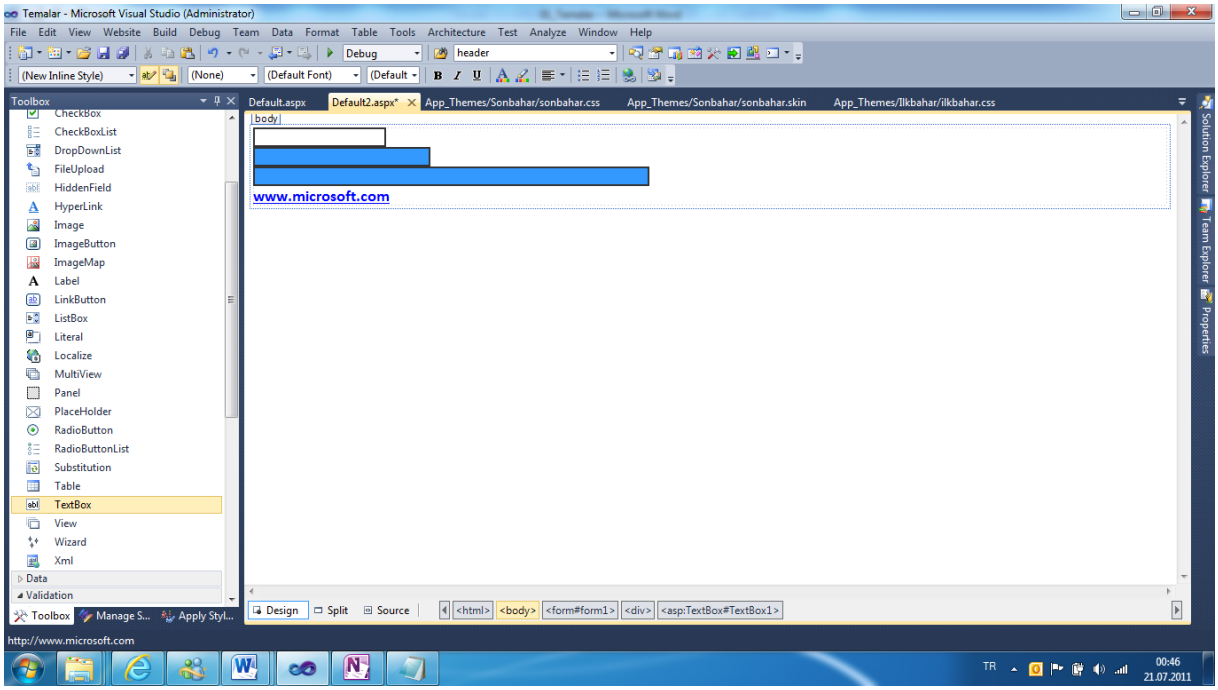
Sayfaya uygulanan temanın iptal edilmesi.

StyleSheetTheme Özelliği İle Tema Uygulamak

Sayfaya tema uygulamak için Page direktifi içerisinde **Theme** özelliği kullanılabileceği gibi **StyleSheetTheme** özelliği de kullanılabilir. Peki bu ikisi arasındaki fark nedir? Hatırlanacağı üzere tema sayfaya Theme özelliği ile uygulandığında kontrolde yapılan değişiklik eğer Skin dosyası içerisinde tanımlı ise temadan gelen sitil ayarı kontrolün üzerinde yapılan ayarı ezerek yapılan değişiklikleri geçersiz kılıyordu, bu durumu engellemek için ise kontrolün EnableTheming özelliğini false olarak ayarlamak gerekiyordu.

Temadan gelen sitili kontrolde ezmenin bir diğer yönetimi ise temayı sayfaya **StyleSheetTheme** özelliği ile uygulamaktır. StyleSheetTheme özelliği ile tema uygulanan kontroller üzerinde yapılacak olan değişiklikler, Skin dosyasındakini ezerek görüntüleniyor olacaktır. Durumu örneklemek için herhangi bir sayfaya daha önce tanımlanan Ilkbahar teması StyleSheetTheme özelliği ile uygulayalım ve daha önceki örneklerde yer alan sayfada bulunan üstteki TextBox'ın BackColor özelliği White olarak ayarlayalım.

Tema sayfaya uygandıktan sonra Theme özelliği ile uugylandığından farklı bir durum dikkat çekecektir. Theme özelliği ile sayfaya uygulanan tema tasarım zamanında görüntülenmezken StyleSheetTheme özelliği ile uygulanan temanın tasarım anında da görüntülendiği fark edilmelidir. Hatırlanacağı üzere CSS dosyaları da sayfaya uygulandıklarında tasarım anında görünüme uygulanmaktaydı. StyleSheetTheme özelliği ile uygulanan tema da CSS dosyası ile uygulanan sitillere benzer şekilde çalışır.



Sayfa çalıştırıldığında yukarıdaki resimde yer alan durum değişmeyecektir ve üstteki kontrolün BackColor özelliği Skin dosyasında tanımlanan BackColor özelliğini ezerek beyaz bir görünüm oluşturacaktır. Eğer tema Theme özelliği ile sayfaya uygulanmış olsaydı yukarıdaki durumun tam tersi bir durum ortaya çıkacaktı ve arka plan rengi Skin dosyasında tanımlandığı şekilde kalacaktı ezmek için ise EnableTheming özelliğine false değer atamak gerekecekti. Kontrole uygulanan tema iptal edildiğinde Skin dosyasından gelen tüm ayarlar iptal edilecektir oysaki yukarıdaki durumda sadece arka plan rengi değiştirilmiştir ve diğer ayarlar korunmaya devam edilmiştir, böyle bir senaryoda Theme yerine StyleSheetTheme özelliğini tercih etmek daha mantıklı bir durum olarak dikkat çekmektedir.

StyleSheetTheme hakkında bashedilecek bir diğer ilginç durumda sayfanın Page direktifinde hem Theme hem de StyleSheetTheme özelliği ile aynı anda tema belirlenebilecek olmasıdır. Böyle bir durumda Theme özelliğinde belirtilen Theme StyleSheetTheme özelliği ile belirtilen temadan gelen özellikleri ezecektir. Dolayısıyla belirtilen temalardan ortak olan özellikler kontrolde de değiştirilmiş olsa dahi Theme özelliği ile belirtilen tema içerisinden geliyor olacaktır ancak StyleSheetTheme özelliğinden gelen Theme özelliği içerisinde belirtilen tema ile ortak olmayan özellikler uygulanmaya devam edeceklerdir.

StyleSheetTheme özelliği site düzeyinde tema uygulanırken de tercih edilebilir.