T.C.
MİLLÎ EĞİTİM BAKANLIĞI

BİLİŞİM TEKNOLOJİLERİ

GELİŞMİŞ İNTERNET UYGULAMALARINDA DİNAMİK PROGRAMLAMA

Ankara, 2013

- Bu modül, mesleki ve teknik eğitim okul/kurumlarında uygulanan Çerçeve Öğretim Programlarında yer alan yeterlikleri kazandırmaya yönelik olarak öğrencilere rehberlik etmek amacıyla hazırlanmış bireysel öğrenme materyalidir.
- Millî Eğitim Bakanlığınca ücretsiz olarak verilmiştir.
- PARA İLE SATILMAZ.

İÇİNDEKİLER

AÇIKLAMALAR

ALAN	**Bilişim Teknolojileri**
DAL / MESLEK	**Veri Tabanı Programcılığı**
MODÜLÜN ADI	**Gelişmiş İnternet Uygulamalarında Dinamik Programlama**
MODÜLÜN TANIMI	Bu modül, XAML tabanlı geliştirime platformu ile İnternet uygulamaları geliştirmede, programlama dili kullanarak XAML ve web kontrollerinin yönetilmesinin kazandırıldığı bir öğrenme materyalidir.
SÜRE	40/32
ÖN KOŞUL	"Gelişmiş İnternet Uygulamalarında Tarayıcı İşlemleri ve Arayüz Oluşturma" modülünü tamamlamış olmak
YETERLİK	Etkileşimli web uygulamaları yapmak
MODÜLÜN AMACI	**Genel Amaç** Bu modül ile gerekli ortam sağlandığında; etkileşimli uygulamaları gerçekleştirebileceksiniz. **Amaçlar** **1.** Bir şablondan XAML oluşturabileceksiniz. **2.** İleri düzey etkileşimli uygulamaları yapabileceksiniz.
EĞİTİM ÖĞRETİM ORTAMLARI VE DONANİMLÂRI	**Ortam:** İnternete ve ağa bağlı bilgisayar laboratuvarı **Donanım:** Bilgisayar, tasarım ve programlama yazılımları
ÖLÇME VE DEĞERLENDİRME	Modül içinde yer alan her öğrenme faaliyetinden sonra verilen ölçme araçları ile kendinizi değerlendireceksiniz. Öğretmen, modül sonunda ölçme aracı (çoktan seçmeli test, doğru-yanlış testi, boşluk doldurma, eşleştirme vb.) kullanarak modül uygulamaları ile kazandığınız bilgi ve becerileri ölçerek sizi değerlendirecektir.

GİRİŞ

Sevgili Öğrenci,

Bilim ve teknolojinin baş döndürücü bir hızla ilerlediği günümüzde, web tabanlı uygulamaların önemi giderek artmaktadır. Bildiğiniz gibi *İnternet* uygulamaları geliştirmede, sayfa tasarımı, grafik ve animasyon çalışmaları, veritabanı yönetimi ve program geliştirme gibi farklı alanlarda çalışmalar yapmak gerekmektedir. Yazılım sektöründeki gelişmeyle orantılı olarak uygulama geliştirme yazılımlarında alternatifler bir hayli çoğalmıştır.

Son yıllarda yaygın olarak kullanılan XAML tabanlı uygulama geliştirme platformu teknolojisi, yazılım geliştiricilere, *İnternet* uygulamaları ve mobil cihazlar için uygulama geliştirmede, hayli kolay ve bir o kadar da güçlü yöntemler sunmuştur. Sayfa tasarımı, animasyon işlemleri, veri tabanı erişimi ve program geliştirme süreçlerini bir arada yürütmek gibi önemli bir imkânı beraberinde getirmiştir. XAML tabanlı uygulama geliştirme platformu, veritabanı alt yapısına sahip, etkileşimli *İnternet* uygulamaları geliştirmede etkili bir teknoloji olarak göze çarpmaktadır.

Bu modül içerisinde; XAML tabanlı uygulama geliştirme platformu uygulamalarında programlama dili kullanımı, çoklu ortam araçları, uygulamaların çalışma performanslarının artırılması için kullanılan yöntemler, etkileşimli şablonlar hazırlayıp kullanma, XAML tabanlı uygulama geliştirme platformu ile web sayfalarının etkileşimi, program kodlarında hata ayıklama yöntemleri ve XAML tabanlı uygulama geliştirme platformu uygulamaları ile web sunucularının etkileşimi gibi ileri düzey konularda gerekli bilgileri bulabilirsiniz.

XAML tabanlı uygulama geliştirme platformu uygulamalarında ileri düzeye ulaşmak için bir adım olacak bu modülü başarıyla bitirmeniz dileğiyle.

ÖĞRENME FAALİYETİ–1

AMAÇ

Bir şablondan XAML oluşturabileceksiniz.

ARAŞTIRMA

- Tasarım editörünün kullanımı hakkında araştırma yapınız.
- Uygulamalarda kullanılan çoklu işlemlerin uygulamalara sağlayabileceği katkılarını araştırınız.
- Kontrol şablonlarının hangi amaçlarla kullanıldığını araştırınız.

1. PROGRAMLAMA YETENEKLERİ

1.1. Dinamik XAML

XAML kullanan XAML tabanlı uygulama geliştirme platformu uygulamalarında, sayfa tasarımında kullanacağımız Button, TextBox v.s. gibi kontrolleri, uygulama geliştirme editörü veya tasarım editöründe bulunan araç kutuları yardımıyla sayfaya kolayca ekleyip, özelliklerini değiştirebiliriz. Ya da xaml sayfası içerisine gerekli XAML kodlarını yazarak, sayfa tasarımını yapabiliriz. Aynı zamanda uygulamanın çalışması esnasında da, sayfa üzerindeki XAML tabanlı uygulama geliştirme platformu kontrollerinin özelliklerini değiştirmek veya yeni kontroller oluşturmak mümkündür. Bu şekilde, dinamik olarak kontrol oluşturma, özelliklerini değiştirme gibi işlemler, C# programlama dili kodları kullanılarak yapılmaktadır.

1.1.1. XAML Kontrolleri ve Programlama Dili

Her hangi bir yöntemle xaml sayfası üzerine eklediğimiz ve isim verdiğimiz bir kontrole, C# kodları kullanarak ismi üzerinden erişebiliriz.

Örnek: Sayfaya yerleştirilmiş ve "buton1" ismi verilmiş olan düğmenin, üzerindeki metni, genişliğini ve yüksekliğini değiştiren aşağıdaki program kodunu inceleyelim.

```
public MainPage()
{
   InitializeComponent();
   buton1.Content = "TIKLA";
   buton1.Width = 80;
   buton1.Height = 25;
}
```

Programlama dilinde gerekli kodları yazarak uygulamanın çalışması (runtime) sırasında sayfada yeni kontroller oluşmasını sağlayabiliriz. Yapılacak şey, new operatörüyle bir sınıftan nesne oluşturmak ve nesne özelliklerini atamaktır.

Bir kontrol sınıfından yeni bir nesne oluşturmak ve bazı özelliklerini belirlemek için gerekli kod yazım kuralı, aşağıdaki gibidir.

```
Sınıf_Adı  yeni_nesne_adı = new Sınıf_Adı( );
yeni_nesne_adı.özellik_1 = değer_1;
yeni_nesne_adı.özellik_2 = değer_2;
.
.
yeni_nesne_adı.özellik_n = değer_n;
```

Örnek: TextBox nesnesinin dinamik teknikte oluşturulması

Aşağıda görülen kodlar, sayfa ilk açıldığı anda, "text1" isimli bir TextBox nesnesi oluşturmakta ve ayrıca genişlik ve yükseklik değerlerini belirlemektedir.

```
public MainPage()
{
   InitializeComponent();
   TextBox text1 = new TextBox();
   text1.Width = 75;
   text1.Height = 30;
   LayoutRoot.Children.Add(text1);
}
```

- **LayoutRoot.Children.Add(text1;** satırında, oluşturduğumuz "text1" isimli TextBox kontrolünün, "LayoutRoot" ismindeki Grid kontrolünün içinde yer alması sağlanıyor. Eğer bunu yapmazsak oluşturduğumuz TextBox kontrolünü sayfa üzerinde göremeyiz.

1.1.2. XamlReader Sınıfı Kullanarak Kontrol Oluşturma

Tasarımcının, tasarım editörü içerisinde yapmış olduğu tasarımı, C# koduna çevirmesi oldukça güç bir iştir. Bu durumda, yardımımıza XamlReader.Load() metodu yetişiyor. Bu metod, parametre değeri olarak aldığı string tipteki XAML kodundan XAML tabanlı uygulama geliştirme platformu nesneleri oluşturarak geri döndürüyor. XamlReader sınıfını kullanmak için programın baş kısmında **"using System.Windows.Markup;"** kodunu yazarak, sınıfın bulunduğu kütüphaneyi program koduna dâhil etmek gerekiyor.

- Bu teknikte ilk olarak, üretilecek nesnenin XAML kodları oluşturularak aşağıdaki yöntemle string tipinde bir değişkene atanır.

String xaml_kod = "…Nesneye ait xaml kodları…";

- İkinci olarak, string tipindeki değişken, XamlReader.Load() metoduna gönderilir. Gönderilen değişkendeki XAML koduna göre nesne üretilerek geri döndürülür.

var nesne_adı = XamlReader.Load(String XAML_kodu);

- Son olarak da, nesne üretildikten sonra LayoutRoot alanına eklenmesi gerekiyor. Bunu da aşağıdaki yöntemle yapıyoruz.

LayoutRoot.Children.Add((UIElement)nesne_adı);

Oluşturulan nesne, veri tipi olmayan bir değişkende tutulduğu için, önünde (UIElement) yazılarak tip dönüşümü sağlanır. Ve bu sayede, bir XAML tabanlı uygulama geliştirme platformu nesnesi haline gelir.

Örnek: XamlReader.Load ( ) metoduyla, XAML kodlarının dinamik olarak çalıştırılması.

Aşağıdaki program kodlarını inceleyelim.

```
public MainPage()
{
    InitializeComponent();
string adres =
"xmlns=\"http://schemas.microsoft.com/winfx/2006/xaml/presentation\"";

string xamlkod = "<Button " + adres +
" Width=\"50\" Content=\"TIKLA\" Height=\"30\" />";

var dugme = XamlReader.Load(xamlkod);
LayoutRoot.Children.Add((UIElement)dugme);
}
```

C# dilinde kullanılan " karakteri ile XAML kodunda kullanılan " karakterlerini ayırmak için XAML kodundaki " karakterlerinin önüne \ karakteri yazılarak bu tırnaklar escape karakter hâline getirilmelidir (Content=\"TIKLA\" kodunda olduğu gibi).

Aşağıdaki satırda, XAML kontrollerine ilişkin web bağlantı bilgisi, "adres" isimli string tipte bir değişkene atanıyor.

```
string adres = "xmlns=\"http://schemas.microsoft.com/winfx/2006/xaml/presentation\"";
```

Aşağıdaki satırda ise sabit XAML kodu iki parçaya ayrılmıştır. + operatörleri kullanarak, "adres" değişkeni iki parça halindeki XAML kodları arasında bulunması gereken konuma yerleştirilmiştir. Okların gösterdiği yerlere dikkat ediniz.

```
string xamlkod = "<Button " + adres + " Width=\"50\" Content=\"TIKLA\" Height=\"30\" />";
```

1.2. Çoklu Ortam Elemanları

XAML tabanlı uygulama geliştirme platformu, çoklu ortam (multimedia) işlemlerini, çok rahat bir şekilde yapmamıza imkân tanır. HD kalitede video ve canlı yayın servisi gibi hizmetleri çok kolay bir şekilde yapabiliyor olmamız, interaktif uygulamalar geliştirmemizi sağlıyor.

Tarayıcı ve platform bağımsız olarak istemci tarafında video ve ses dosyalarını oynatabiliyor olmak büyük avantaj sayılmaktadır. Tüm bunları yapabilmek için, XAML tabanlı uygulama geliştirme platformunun içerisindeki MediaElement kontrolünü kullanacağız.

1.2.1. MediaElement ve Özellikleri

MediaElement, en basit tanımıyla ses ve video içeriklerini görüntülememizi sağlayan nesnedir. MediaElement, çeşitli görüntü ve ses dosyası biçimlerini desteklemektedir. MediaElement kontrolüne ilişkin, Properties paneliyle veya program koduyla değiştirilebilen bazı önemli özellikler aşağıda belirtilmiştir.

Özellik	Görevi	Açıklama
Source	Video ya da ses dosyasını yüklemek için kullanılır.	String değer alır.
Volume	Ses seviyesi değerini ayarlar.	Sayısal değer alır.
IsMuted	Sesin açık olup olmayacağını ayarlamak için kullanılır.	True veya False değeri Alır.
Stretch	Video görselinin MediaElement içerisine nasıl yerleşeceğini belirlemek için kullanılır.	Fill, UniForm ya da UniFormToFill değerlerinden birini alır.
AutoPlay	İçeriğin otomatik olarak oynatılmaya başlaması ya da başlamamasını belirlemek için kullanılır.	True veya False değeri alır.
Balance	Sesin sağ ya da sol hoparlör dengesini ayarlar.	0, 1 veya -1 alır.
Position	İçeriğin yürütülmesi sırasında, zaman ilerlemesinin değerini alabilir veya belirli bir değere ayarlayabilir.	TimeSpan türünde değer alır.

Tablo 1.1: MediaElement Özellikleri

1.2.2. Programlama Dili ile MediaElement Kontrolü

Video dosyasını, projemizin "SilverlightApplication" klasörüne değil, "SilverLightApplication.Web" klasörü altındaki "ClientBin" klasörüne dahil etmemiz gerekmektedir. Eğer "SilverlightApplication" klasörüne dahil edersek, video verisini xap dosyasının içerisine entegre etmek zorunda kalırız. Bu durum, web ortamı için performansı olumsuz etkiler.

MediaElement nesnesinin bazı metod ve özellikleri sayesinde, video ya da ses dosyası yürütmesini kontrol edebiliriz. Bahsedilen metodlardan bazıları, aşağıda tablo halinde listelenmiştir.

Metod Adı	**Görevi**
MediaElement.Play( )	MediaElement kontrolüne yüklenmiş video ya da ses dosyasının yürütmesini başlatır.
MediaElement.Stop( )	MediaElement kontrolüne yüklenmiş video ya da ses dosyasının yürütmesini durdurur.
MediaElement.Pause( )	MediaElement kontrolüne yüklenmiş video ya da ses dosyasının yürütmesini duraklatır.

Tablo 1.2: MediaElement metodları

Örnek: XAML tabanlı uygulama geliştirme platformu ile basit bir "media player" uygulaması

- Uygulamada "mediaElement1" isminde bir MediaElement kontrolü, beş adet Button, ve "slider1" isminde bir Slider kontrolü kullanacağız.

Resim 1.1: Tasarlanan sayfanın görüntüsü

➢ Button kontrollerinin Click olaylarına gerekli kodları yazacağız.

```
void btnbaslat_Click(object sender, RoutedEventArgs e)
{
   mediaElement1.Play();  //Videoyu oynatmaya başlar.
}
void btndurdur_Click(object sender, RoutedEventArgs e)
{
   mediaElement1.Stop();  //Videoyu durdurur.
}
void btnduraklat_Click(object sender, RoutedEventArgs e)
{
   mediaElement1.Pause();//Videoyu duraklatır.
}
void btnileri_Click(object sender, RoutedEventArgs e)
{   //Video oynarken 3 saniye ilerletir.
    TimeSpan sure = new TimeSpan(0, 0, 3);
    mediaElement1.Position = mediaElement1.Position.Add(sure);
}
void btngeri_Click(object sender, RoutedEventArgs e)
{  //Videoyu 3 saniye geri sarar
   TimeSpan sure = new TimeSpan(0, 0, 3);
   mediaElement1.Position = mediaElement1.Position.Subtract(sure);
}
void slider1_ValueChanged(object sender,
            RoutedPropertyChangedEventArgs<double> e)
{   //ses değerini, slider nesnesine göre ayarlar.
    mediaElement1.Volume = slider1.Value;
}
```

➢ Uygulama çalıştırıldığında, aşağıdaki sayfa görüntüsü ortaya çıkmıştır.

Resim 1.2: Sayfanın çalışma anı görüntüsü

Ayrıca tasarım editörü ve XAML tabanlı uygulama geliştirme platformu editörlerindeki araç kutusundan eklenebilen MediaElement nesnesini, XAML kodlarıyla da oluşturma imkânımız vardır. Örneğin, aşağıda görülen XAML kodu bir MediaElement nesnesi için en temel özellikleri kullanır.

```
<MediaElement x:Name="media1" Source="dersvideo.wmv" Balance="0"
Volume="5.6" AutoPlay="True" IsMuted="False"/>
```

1.2.3. MediaElement Olayları

- **BufferingProgressChanged olayı**

MediaElement üzerinde yürütülen videonun, önbelleğe alınma (buffering) işlemi sırasında, önbelleğe alma miktarının her değişiminde tetiklenen olaydır. Bu olay içerisinde bazı yükleme animasyonlarının çalıştırılması sağlanabilir.

Bununla beraber, BufferingProgress özelliğinin de bilinmesi gerekiyor. Bu özellik, arabelleğe alma ilerlemesinin yüzde değerini belirten bir sayı içerir.

- **DownloadProgressChanged olayı**

MediaElement kontrolünde yürütülen videonun indirilmesi sırasında, indirme miktarının her değişiminde tetiklenerek çalışan olaydır. Ayrıca bu olay içerisinde kullanılabilen DownloadProgress özelliği ise, indirme yüzdesinin ilerlemesini belirten bir sayı içerir.

DownloadProgress özelliği, DownloadProgressChanged olayında kullanılarak indirme esnasında, yüzde değerini bildiren animasyonlar çalıştırılabilir.

1.3. Uygulama Olayları

Uygulamada ortaya çıkan, olağan ya da olağan dışı durumlara göre, bazı uygulama (application) olayları otomatik olarak tetiklenmektedir. Bahsedilen olaylara ait metodlar, "App.xaml.cs" isimli kod dosyasında mevcuttur. Gerekli program kodları, "App.xaml.cs" içindeki bu metodlar içine yazılabilir.

Uygulama olayları ise;

- Application.Startup olayı,
- Application.UnhandledException olayı,
- Application.Exit olayıdır.

1.3.1. Application.Startup Olayı

Application.Startup(), uygulama çalışmaya başladığı anda tetiklenerek çalışan olaydır. "App.xaml.cs" dosyası içinde Startup olayını temsil eden "Application_Startup" metodu mevcuttur. Uygulamanın başlangıçta çalıştırmasını istediğimiz program kodlarını, aşağıda görüldüğü gibi, "Application_Startup" metodu içerisine yazmalıyız.

```
private void Application_Startup(object sender, StartupEventArgs e)
{
    this.RootVisual = new MainPage();
    MessageBox.Show("HOŞGELDİNİZ...");
}
```

1.3.2. Application.UnhandledException Olayı

Uygulamaların çalışması esnasında, sıfıra bölme hatası ya da taşma hataları gibi hatalar meydana gelebilir. Bu tip hatalar, runtime hataları olarak da bilinir. Uygulama içinde try-catch bloklarıyla yönetilmeyen ve runtime hatalarına sebep olan program kodları, aynı zamanda Application.UnhandledException olayının tetiklenmesine sebep olurlar. UnhandledException olayını temsil eden "Application_UnhandledException()" metodu içerisine yazılacak kodlar sayesinde çalışma esnasında oluşabilecek olağandışı durumlara karşı tedbir almak mümkündür.

Bu olayın kullanımına dair aşağıdaki kod örneğini inceleyiniz.

```
private void Application_UnhandledException(object sender,
ApplicationUnhandledExceptionEventArgs e)
   {
      if (!System.Diagnostics.Debugger.IsAttached)
      {

         e.Handled = true;
         Deployment.Current.Dispatcher.BeginInvoke(delegate {
   ReportErrorToDOM(e); });
      }
      MessageBox.Show("TANIMLANMAYAN BİR HATA MEYDANA GELDİ");
   }
```

1.3.3. Application.Exit Olayı

Application.Exit olayı, olağan ya da olağan dışı bir sebeple uygulamanın kapanması esnasında tetiklenir. Uygulamanın kapanması esnasında çalıştırılacak program kodlarını, bu olaya bağlı olarak çalışan "Application_Exit( )" metodu içine yazarız.

Bu olayın kullanımına dair kod örneği arka sayfada gösterilmiştir.

```
private void Application_Exit(object sender, EventArgs e)
{
  MessageBox.Show("GÜLE GÜLE...");
}
```

1.4. XAML Tabanlı Uygulama Geliştirme Platformu ile Çoklu İşlemler

Çoklu işlemler (multithreading), uygulamaların iş parçacıklarına bölünerek birden fazla iş parçacıklarının bir arada yürütülebilmesi ve bu görevlerin birbirine engel olmaması esasına dayanır. Program geliştirmede çoklu işlemleri kullanmak, uygulama performansını artırıcı yönde etki yapar. XAML tabanlı uygulama geliştirme platformu birkaç farklı yöntemle, programcıya çoklu işlemler oluşturup kullanma imkânları sunmaktadır.

Bir uygulamada çalıştırılan iş parçacıklarını "thread" olarak adlandırabiliriz. İş parçacığı ise, bir grup program kodudur. XAML tabanlı uygulama geliştirme platformu uygulamaları çalışmaya başladığında, arka planda bir "thread" meydana gelip çalışmaya başlar. "Main thread" olarak adlandırdığımız bu işlem, sayfadaki nesneleri ve eylemleri yönetir. Kritik öneme sahip olan "main thread"ın çalışması bir sebeple aksadığında, uygulama yanıt vermez hale gelir. Aksamaya sebep olan işlem bitene kadar pencere tepkisiz halde bekler. Çalıştırılan bir kod parçasının uygulamanın çalışmasını yavaşlatmasını istemiyorsak yapacağımız şey bir "thread" oluşturup yavaşlatmaya sebep olabilecek program kodlarını bu "thread" üzerinden çalıştırmaktır.

1.4.1. Thread Oluşturmak ve Kullanmak

Bir thread oluşturmak için, new bildirisini kullanarak Thread sınıfından bir nesne oluşturmamız gerekir. Thread sınıfını program içinde kullanabilmek için, kod sayfasının yukarı kısmına, "using System.Threading;" yazmalıyız. Daha sonra Thread oluşturmak için aşağıdaki yöntemi kullanmamız gerekir.

Thread thread_ismi = new T**hread**(metod_ismi);

"metod_ismi" ile belirtilen parametre, thread üzerinde çalışacak metodun ismini temsil eder.

Örnek: Thread gorev1 = new Thread(dongu1);

Yukarıdaki kod satırı, "gorev1" isminde bir thread oluşturacak ve bu işlem başlatıldığında "dongu1" isimli metodu arka planda yürütecektir.

Thread nesnesini oluşturduktan sonra, nesneye ait bazı üye metodlarını kullanarak, başlatma, durdurma, duraklatma gibi yönetim işlemlerini yapabiliriz. Aşağıdaki tabloda, bazı üye metodların görevleri mevcuttur.

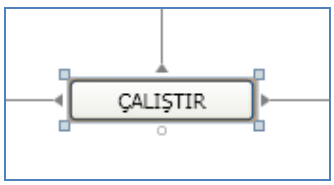
Metod	Görevi	Açıklama
Start()	Bu metod kullanıldığında, tanımlanmış bir thread çalışmaya başlar.	
Abort()	Çalışır durumda bulunan bir thread, sonlan-dırılmış olur.	
Suspend()	Çalışır durumda bulunan bir thread, durak-latılır.	
Resume()	Duraklatılmış durumda bulunan bir thread, çalışmasına, kaldığı yerden devam eder.	
Sleep()	Bir thread, bir süre duraklatılıp bekletilir. Süre tamamlandığında, thread çalışmaya kaldığı yerden devam eder.	Milisaniye cinsinden süre değeri alır.

Tablo 1.3: Thread sınıfının metodları

Örnek: Onar saniye sürecek olan iki döngüyü, aynı anda iki ayrı thread üzerinden çalıştıralım.

Uygulamada iki adet metod yazacağız. Her iki metod içinde, onar saniye süreyle işlem yapacak birer döngü yapısı kuracağız. Daha sonra, her iki metodu iki ayrı thread üzerinden çalıştıracağız. Aşağıdaki işlem adımlarını takip edelim.

- Aşağıda görüldüğü gibi uygulamanın "MainPage" sayfası üzerine bir buton yerleştirilmiştir.



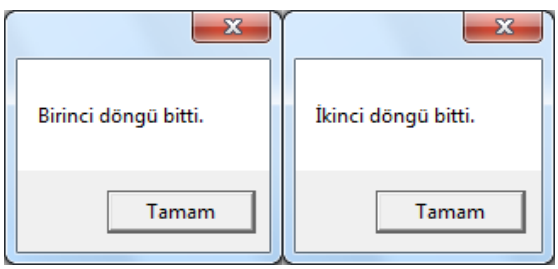
Resim 1.3: Sayfa görüntüsü

- Daha sonra, "dongu1" ve "dongu2" isimlerinde iki ayrı metod oluşturulmuştur.
- Sayfa üzerindeki düğmenin Click olayı içerisinde, her iki döngü için, iki ayrı thread oluşturulup, başlatılmıştır.

```
private void dongu1()
{
   for (int i = 0; i < 10; i++)
   {
      Thread.Sleep(1000);
   }
   MessageBox.Show("Birinci döngü tamamlandı!");
}

private void dongu2()
{
   for (int i = 0; i < 10; i++)
   {
      Thread.Sleep(1000);
   }
   MessageBox.Show("İkinci döngü tamamlandı!");
}

void btnbaslat_Click(object sender, RoutedEventArgs e)
{
   Thread islem1 = new Thread(dongu1);
   Thread islem2 = new Thread(dongu2);
   islem1.Start();
   islem2.Start();
}
```

Th **Thread oluşturan program**

Oluşturulan her iki thread, "start" metoduyla çalıştırıldıktan sonra "dongu1()" ve "dongu2()" metodlarını çağırıyor. Döngüler, arka arkaya değil, beraber çalışıyorlar. Çünkü bu döngüler, birbirinden bağımsız iki thread üzerinde çalışıyor. Ne birbirilerine, ne de uygulamanın işleyişine etkileri olmuyor. Bu yüzden, arka planda döngüler çalışırken, uygulama penceresinin işleyişi devam edecektir (**Eğer thread kullanmadan, klasik yöntemle, dongu1() ve dongu2() metodlarını çağırsaydık, birinci döngü bitmeden, ikinci döngü başlamayacaktı. Ve ayrıca döngülerin ikisi de tamamlanıncaya kadar, uygulama penceresi tepkisiz halde kalacaktı.**). İki döngü aynı anda biteceği için, iki mesaj penceresi, aşağıdaki gibi aynı anda görüntülenecektir.



Resim 1.4: Ekran görüntüsü

1.4.2. Dispatcher Sınıfının Kullanılması

Sonradan oluşturulmuş bir iş parçası(thread) üzerinde çalışan bir program kodu, sayfa üzerinde bulunan, Button, TextBox, ListBox gibi kontrollerin özelliklerini değiştiremez. Örneğin, kendi oluşturduğumuz bir thread üzerinden, bir düğmenin üzerindeki Content metnini, direkt olarak değiştirmek mümkün değildir. Çünkü sayfa üzerinde kullandığımız nesneler, "main thread" tarafından oluşturulur ve yönetilir. Harici bir thread üzerinden, bu nesnelere erişim imkânı yoktur.

İşte tam burada Dispatcher sınıfı devreye giriyor. Sayfa üzerine yerleştirdiğimiz Button, TextBox, ListBox gibi her bir kontrol, kendi içinde bir Dispatcher nesnesine sahiptir. Bir kontrolün sahip olduğu Dispatcher sayesinde, o kontrol harici bir thread tarafından erişilir hale gelmiş olur. Bu ise Dispatcher nesnesinin BeginInvoke() metodu kullanılarak gerçekleştirilir.

Dispatcher sınıfının üye metodu olan BeginInvoke() metodu sayesinde, harici bir thread üzerinde çalışan kod parçası, "main thread" üzerinde çalıştırılır. Bu metod, dışarıdan bir delege parametresi almaktadır. Ve bu delege parametresi, "main thread" üzerinde çalıştırılacak metodun adını temsil eder.

Örnek: Dispatcher sınıfının işlevlerini daha iyi anlayabilmek için, sayfa görüntüsü aşağıdaki gibi olan bir program örneğini ele alalım.

Resim 1.5: Sayfa tasarımı

- Sayfa üzerinde bulunan düğmeye tıklandığında, bir ile yirmi arasındaki ardışık sayıları, birer saniye arayla liste kutusuna ekleyen bir uygulamayı, dispatcher yöntemini kullanarak yapacağız. Bunun için, aşağıdaki işlemleri takip edelim.

  - Öncelikle delege metod başlığı, aşağıdaki gibi, delagete bildirisiyle tanımlanır.
    ```
    public delegate void delegeekle();
    ```

  - Daha sonra, "sayi" isminde, global bir değişken tanımlarız.
    ```
    int sayi;
    ```

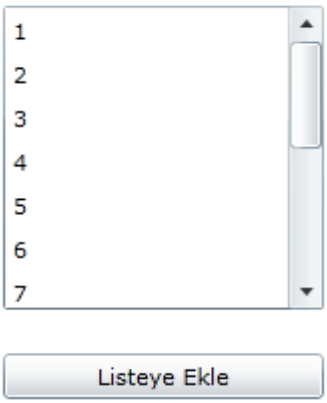
- Üçüncü olarak, “ekle( )” isminde bir metod oluşturuyoruz. Bu metod, delege metod vasıtası ile “main thread” üzerinde çalıştırılacak olan ve liste kutusuna eleman ekleyen iş parçasıdır.
- Dördüncü adımda, “listeyeekle( )” ismindeki metodu oluşturacağız.
- Son olarak, düğmenin Click olayı için gerekli kodları yazacağız.

Yazılan program kodları ise aşağıda görüldüğü gibidir.

```
public delegate void delegeekle();
int sayi=0;

public void ekle()
{
  listBox1.Items.Add(sayi);
}

private void listeyeekle()
{
   for (sayi = 1; sayi <= 20; sayi++)
   {
     listBox1.Dispatcher.BeginInvoke(new delegeekle(ekle));
     Thread.Sleep(1000);
   }
}

void buton1_Click(object sender, RoutedEventArgs e)
{
    Thread islem = new Thread(listeyeekle);
    islem.Start();
}
```

Boyalı olan program satırına dikkat edelim. Bu satırdaki program kodunda, “listBox1” nesnesine ait dispatcher nesnesinin BeginInvoke() metodu çağrılmıştır. Fonksiyon parametresinde new operatörü kullanılarak, “ekle()” metodunu temsil eden bir delege oluşturulmuş ve oluşan bu iş parçacığı, BeginInvoke() fonksiyonu sayesinde, “main thread” üzerinden çalıştırılmıştır. Bu sayede “islem” isimli thread üzerinden, “listBox1” içine sayı eklenebilmiştir.

- Bu durumda butona basıldığında, Resim 1.6’da görüldüğü gibi, birden yirmiye kadar birer artan sayılar, her saniyede bir, liste kutusuna eklenecek ve bu durumdan uygulama penceresi hiçbir şekilde olumsuz etkilenmeyecektir.



Resim 1.6: Ekran çıktısı

1.4.3. BackGroundWorker ile Çoklu İşlemler

BackGroundWorker sınıfı, thread yapılarına benzer şekilde, birbirinden bağımsız birden fazla işlemin aynı anda yürütülmesini sağlar. Ayrıca bu sınıf sayesinde, işlem hakkında durum bilgisi alabilir ve işlemin tamamlanmasına göre, bazı program kodlarının çalıştırılmasını sağlayabiliriz. BackgroundWorker kullanımını bir örnek üzerinden anlamaya çalışalım.

Örnek: Sayfa üzerinde bulunan düğmeye basıldığında, arka planda on saniye sürecek bir döngüyü çalıştıracak olan uygulamayı, BackgroundWorker kullanarak yapmaya çalışalım.

- Uygulama sayfasında sadece bir buton vardır.
- Her şeyden önce, BackgroundWorker kullanabilmek için, bu sınıfın tanımlı olduğu kütüphaneyi, aşağıda görüldüğü gibi, uygulama kodlarımıza eklememiz gerekiyor.
  using System.ComponentModel;
- Daha sonra aşağıdaki gibi, butonun Click olayı içinde, BackgroundWorker sınıfından bir nesne değişkeni oluşturmamız gerekiyor.

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    BackgroundWorker bg1 = new BackgroundWorker();
}
```

- Önceki adımda oluşturulan BackGroundWorker nesnesinin DoWork olayına, arka planda çalıştıracağı metodun ismini atamamız gerekiyor.

```
private void button1_Click(object sender, RoutedEventArgs e)
{
  BackgroundWorker bg1 = new BackgroundWorker();
  bg1.DoWork += new DoWorkEventHandler(bg1_DoWork);
}
```

DoWork olayı, arka planda, asenkron olarak yürütülecek olan "bg1_DoWork()" metodu ile bağlanır. "bg1_DoWork( )" ise on saniye bekleyen döngü kodlarının bulunacağı metodun ismidir.

- Daha sonra, BackGroundWorker nesnesi, "bg1_DoWork()" metodunda kodlanmış işlemi yürütüp bitirdiğinde, hangi metodun tetiklenerek çalışmasını istiyorsak, bu metodun ismini, RunWorkerCompleted olayına atamamız gerekiyor.

```
private void button1_Click(object sender, RoutedEventArgs e)
{
   BackgroundWorker bg1 = new BackgroundWorker();
   bg1.DoWork += new DoWorkEventHandler(bg1_DoWork);
   bg1.RunWorkerCompleted +=
   new RunWorkerCompletedEventHandler(bg1_RunWorkerCompleted);
 }
```

RunWorkerCompleted olayı, yürütülen iş parçası sonlandığında tetiklenir.

- Bunun ardından, kodlara bg1 isimli BackgroundWorker nesnesinin asenkron olarak arka planda çalışmaya başlamasını sağlayan, RunWorkerAsync( ) metodunu ilave ederiz.

```
private void button1_Click(object sender, RoutedEventArgs e)
{
   BackgroundWorker bg1 = new BackgroundWorker();
   bg1.DoWork += new DoWorkEventHandler(bg1_DoWork);
   bg1.RunWorkerCompleted += new
RunWorkerCompletedEventHandler(bg1_RunWorkerCompleted);
   bg1.RunWorkerAsync();
 }
```

- Daha sonra DoWork olayına ait olan ve arka planda çalıştırılacak olan "bg1_DoWork( )" metoduna ait kodları yazmamız gerekiyor.

```
public void bg1_DoWork(object sender, DoWorkEventArgs e)
  {
      for (int i = 1; i <= 10; i++)
      {
         Thread.Sleep(1000);
      }
  }
```

Bu döngü on saniye bekledikten sonra tamamlanacaktır.

- Son olarak, RunWorkerCompleted olayına bağlı olarak çalışacak olan, "bg1_RunWorkerCompleted( )" metoduna ait kodlar yazılır.

```
public void bg1_RunWorkerCompleted(object sender,
 RunWorkerCompletedEventArgs e)
{
   MessageBox.Show("İşlem tamamlandı...");
}
```

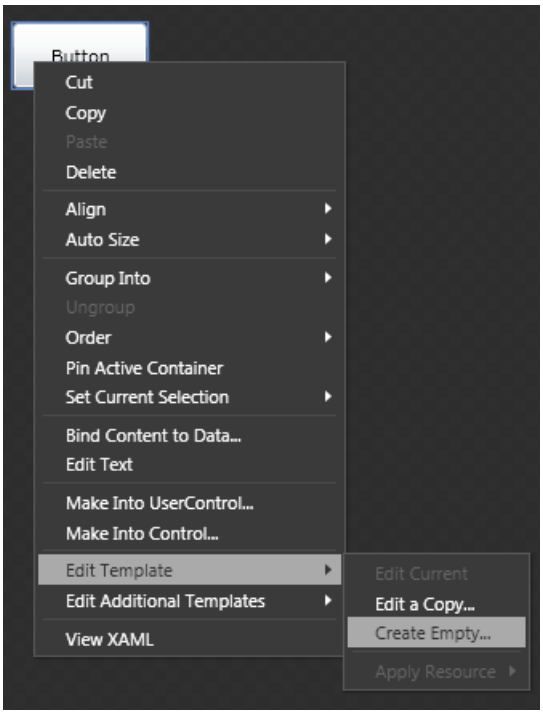
1.5. Gelişmiş Kontrol Yapıları

XAML tabanlı uygulama geliştirme platformu ile kullandığımız, Button, TextBox v.s. bileşenler, normalde klasik görünüme sahiptir. Ancak XAML tabanlı uygulama geliştirme platformu ile beraber, sayfaya yerleştirdiğimiz kontrollerin temalarında değişikliler yaparak, özel kontroller tasarlama imkânı sunulmuştur. Mesela, tasarımcılar butonların görünümlerini değiştirebilir. Ve bu gibi stil ve özellik değişikliklerini, sonradan kullanmak üzere, bir şablon hâline getirebilir. Bu teknik bir bakıma, web tasarımında kullandığımız CSS yapılarına benzer.

Özellikle animasyon uygulamaları hazırlarken, alışageldiğimiz klasik kontrol görünümleri dışında, farklı görünümlü veya etkileşimli nesneler kullanmayı tercih ederiz. XAML tabanlı uygulama geliştirme platformu, kontrol görünümlerini, istenilen biçimde tasarlama imkânını sunmuştur.

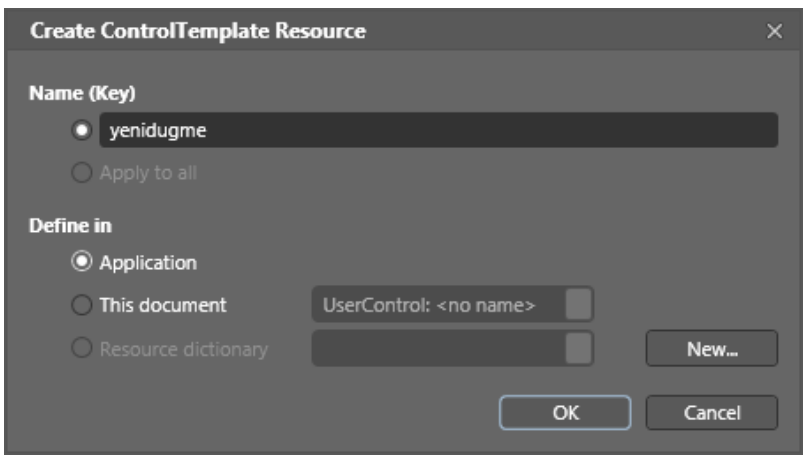
1.5.1. Editör ile Kontrol Şablonlarını Oluşturma ve Kullanma

- Tasarım editöründe, standart bir Button kontrolünden yola çıkarak, kendi tasarımımız olan bir buton şablonu oluşturmaya çalışacağız. Önce, "LayoutGrid" içine bir Button kontrolü yerleştiriyoruz. Ardından Resim 1.7'de görüldüğü gibi, düğme üzerinde sağ tuş menüsünü açıp "Edit Template" altındaki "Create Empty" seçeneğini tıklayacağız.



Resim 1.7: Yeni şablon oluşturma

- "Create Empty" tıklandığında, aşağıda görülen "Create Control Template Resource" başlıklı bir pencere açılacaktır.



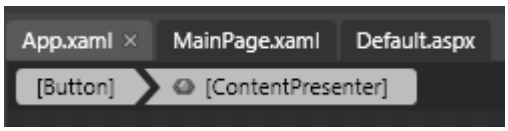
Resim 1.8: Şablon özelliklerini belirleme

Pencerede, Name kısmında, düğme şablonumuza bir isim vereceğiz. Alt kısımda bulunan "Define in" bölümünde ise üç seçenek mevcuttur.

Bu seçeneklerden, "Application" işaretlenirse, oluşacak olan şablon, üzerinde çalıştığımız projenin bütün xaml sayfalarında geçerli hâle gelir.

"This document" seçeneği işaretlenirse sadece üzerinde çalıştığımız xaml sayfasında geçerli olacaktır. "Resource dictionary" seçeneği ise, şablonumuzu harici bir xaml dosyası içinde oluşturup, hem üzerinde çalıştığımız proje, hem de dâhil ettiğimiz başka projelerde kullanılabilmemizi sağlar.

"Application" seçeneğini işaretleyip OK düğmesine tıklayacağız. Bunun ardından editör, üzerinde çalıştığımız "MainPage.xaml" sayfasından, buton tasarımı yapacağımız "App.xaml" sayfasına geçer. Ardından, buton tasarımına başlayabiliriz.



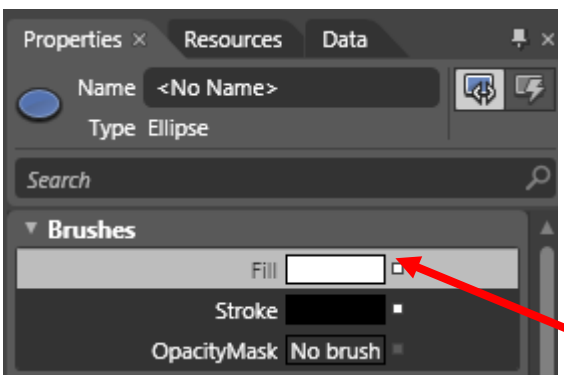
Resim 1.9: Ekran görüntüsü

- Önce, bir elips çizeceğiz. Daha sonra elipsin iç dolgu rengini belirlememiz gerekiyor.

Resim 1.10: Elips çizimi

Elipsin Fill özelliğini, Template Binding altındaki BackGround özelliğine bağlamamız gerekiyor. Bu sayede, elipsin, dolgu rengini, şablonu kullanacak olan Button nesnesinin BackGround özelliğinden almasını sağlıyoruz. Bu da, "her bir Button için farklı bir renk", anlamına gelir. Bu işlemin yapılışını aşağıdaki resimlerde görebiliriz.

Önce, elips seçili hale getirilir. Daha sonra, Properties panelinden, Fill özelliğinin yanındaki küçük kare kullanılır.



Resim 1.11: Elipsin dolgu rengi

Bu kareye tıklandığında açılan menüden, "Template Binding" altındaki "BackGround" seçilir.

- Buton üzerindeki Content metnini oluşturmak için, ContentPresenter ismindeki kontrolü, Assets paneli içerisinden ekleyeceğiz. ContentPresenter kontrolü, düğme metnini, şablonu kullanacak olan butonun Content özelliğinden alır.

- Bunları yaptıktan sonra, düğme şablonumuzun tasarımı bitmiş olacaktır. File menüsünden Save seçeneğini kullanarak değişiklikleri kaydedelim. Ve daha sonra "MainPage.xaml" sayfasına geri dönelim.

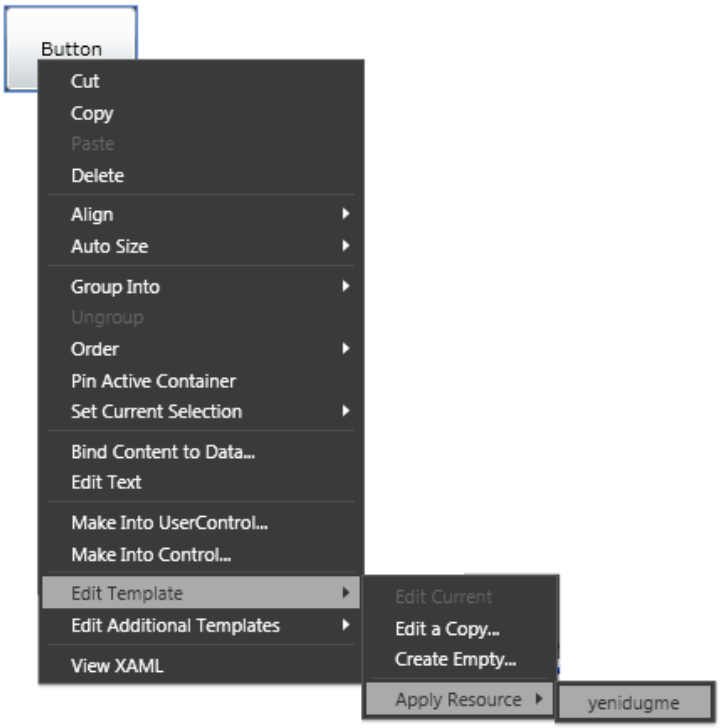
Resim 1.12: Button

Kontrol şablonlarının oluşturulmasının ardından, bu şablonların kontrollere nasıl uygulanacağı sorusu akla gelir. Oluşturduğumuz düğme şablonunu kullanmak için aşağıdaki adımlara dikkat edelim.

- Düğme şablonunu oluşturup, "MainPage.xaml" sayfasına geri dönmüştük. En başta bir Button üzerinden yola çıkarak kontrol şablonunu oluşturduğumuz için, oluşturduğumuz şablonun bu butona uygulanmış durumda olduğunu göreceğiz. Bunun yanı sıra, "MainPage" üzerine birkaç Button daha ekleyeceğiz.

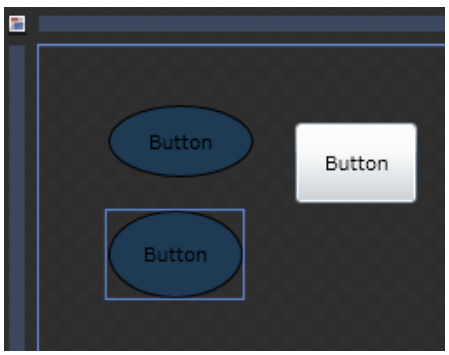
- Yeni eklediğimiz Button kontrolleri, standart görünüme sahiptir. Oluşturduğumuz kontrol şablonunu bu düğmelere uygulamak için, düğme üzerinde sağ tuş menüsü açılmalıdır.

Bu menüde, fareyi, "Edit Template" alt menüsü altındaki "Apply Resource" üzerine getirdiğimizde, bir alt menü daha açılacak ve oluşturduğumuz kontrol şablonunun ismini burada göreceğiz. Kontrol şablonumuzu temsil eden seçeneğe, Resim 1.13'te görüldüğü gibi tıklarsak, şablon düğmeye uygulanmış olur.



Resim 1.13: Şablonun uygulanması

Burada "yenidugme" isimli kontrol şablonu, düğmeye uygulanmış olur. Şablonu, diğer düğmelere de aynı şekilde uygulayabiliriz.

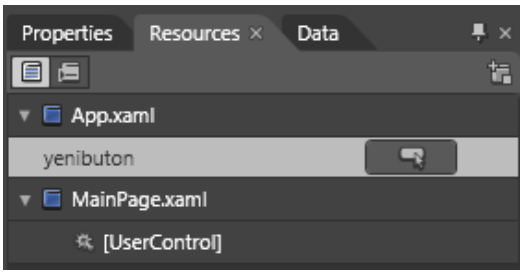


Resim 1.14: Sayfa görüntüsü

- Ve her üç düğme için Properties paneli üzerindeki Background ve Content özelliklerini değiştirebiliriz. File menüsünden "Save All" seçeneği ile yaptıklarımızı kaydedip uygulamayı uygulama geliştirme editöründe çalıştıralım.

Resim 1.15: Çalışma durumu

Ayrıca tasarım editöründe, kontrol şablonlarına özel, Resources isminde bir panel bulunmaktadır. Tasarladığımız kontrol şablonu, bu panele dahil edilmiş durumdadır. Düğmemizi, sürükle bırak yöntemiyle, bu panelden sayfa üzerine ekleyebiliriz.



Resim 1.16: Resources paneli

1.5.2. XAML Kodları ile Şablon Oluşturma ve Kullanma

Kontrol şablonlarını, bulunduğu sayfada geçerli olanlar, projedeki bütün sayfalarda geçerli olanlar ve harici dosyada tanımlanıp istenilen her projede geçerli olabilenler olarak üç gruba ayırmıştık. Bu konu içerisinde her üç türdeki şablon tasarımını, XAML kodları ile gerçekleştireceğiz.

İlk olarak, sadece bir sayfanın bütününde geçerli olacak bir kontrol şablonunun oluşturulmasını ele alacağız. Bu tür şablonlarda, XAML kodları o sayfanın kodları arasına yazılırlar. Arka sayfadaki örnek kodları inceleyerek, hem kontrol şablonunun tanımlanması hem de bu şablonun istenilen bir kontrole uygulaması prensiplerine değineceğiz.

Kodlar içinde, sarı renk ile vurgulanmış olan üstteki kısım, kontrol şablonu tanımlamasına ilişkin XAML kodlarıdır. Alttaki açık mavi renk ile vurgulanmış olan kod satırı ise, tanımlanan şablonun düğmeye uygulandığı satırdır. Kodların alt tarafında açıklanan, şablon tanımlaması ve kullanılmasına ilişkin prensipleri bilmek gereklidir.

```
     <UserControl x:Class="SilverlightApplication6.MainPage"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      Width="400" Height="300">
     <UserControl.Resources>
          <ControlTemplate x:Key="kirmizidugme" TargetType="Button">
            <Grid>
            <Rectangle Fill="#FFF25454" Stroke="Black"
Margin="7,4,9,4"/>
             <ContentPresenter HorizontalAlignment="Center"
VerticalAlignment="Center"/>
            </Grid>
           </ControlTemplate>
        </UserControl.Resources>

     <Grid x:Name="LayoutRoot" Background="White">
 <Button Content="TIKLA" Template="{StaticResource kirmizidugme}"/>
     </Grid>
     </UserControl>
```

- Tek bir sayfada geçerli bütün kontrol şablonu tanımlamaları, <UserControl.Resources> </UserControl.Resources> etiketleri arasına yapılır.

- <UserControl.Resources></UserControl.Resources> etiketleri arasında, her bir kontrol şablonu tanımlaması için ayrı ayrı <ControlTemplate> </ControlTemplate> etiketleri kullanılır.

- Kontrol şablonlarına isim vermek için, <ControlTemplate> etiketindeki x:Key özelliği kullanılır. Örneğin: x:Key="kirmizidugme"

- Şablonun hangi tür kontrollerde kullanılacağını belirtmek için, <ControlTemplate> etiketindeki TargetType özelliği kullanılır. Örneğin: TargetType="Button"

- <ControlTemplate> </ControlTemplate> etiketleri arasında, şablonda olması gereken alt kontroller ve özellikleri, XAML kodlarıyla oluşturulur.

- Hangi türde olursa olsun, oluşturulan bir şablonu bir kontrole uygulamak için, o kontrolün Template özelliği kullanılmalıdır. Template özelliği, şablonun ismini değer olarak alır. Örnek:

```
<Button Content="TIKLA" Template="{StaticResource kirmizidugme}"/>
```

Uygulamadaki bütün sayfalarda geçerli olan şablon tanımlamaları yapmak için, öncekinden farklı olarak, şablona ait XAML kodlarının, "App.xaml" sayfası içinde yazılması gerekmektedir.

```
<Application
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             x:Class="SilverlightApplication6.App">
   <Application.Resources>
       <ControlTemplate x:Key="yenidugme" TargetType="Button">
   <Grid>
           <Button Content="TAMAM" Width="70" Height="25"/>
         </Grid>
</ControlTemplate>
   </Application.Resources>
</Application>
```

"App.xaml" içinde şablon tanımlamaları yaparken oluşturacağımız şablonların bütün sayfalarda geçerli olabilmesi için tanımlamaları, **<Application.Resources>** **</Application.Resources>** etiketleri arasına yapmamız gerekmektedir.

Kontrol şablonlarını, Resource Dictionary denilen harici xaml dosyalarında, XAML kodları yardımıyla tanımlamak için; öncelikle üzerinde çalıştığımız proje içerisine, uygulama geliştirme editörü yada tasarım editörü yardımıyla bir "Resource Dictionary" dosyası eklememiz gerekiyor. Aslında bu dosya da bir xaml dosyasıdır.

```
<ResourceDictionary
     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d">
     <!-- Resource dictionary entries should be defined here. -->
     <ControlTemplate x:Key="iptaldugme" TargetType="Button">
          <Grid Width=”100” Height=”40”>
               <Rectangle Fill="#FFEC653F" />
               <TextBlock Margin="29,11,33,8" Text="İPTAL" />
          </Grid>
     </ControlTemplate>
</ResourceDictionary>
```

Yukarıda görüldüğü gibi "Resource Dictionary" dosyası içerisinde hazır bulunan, **<ResourceDictionary></ResourceDictionary>** etiketleri arasına, kontrol şablonuna ilişkin xaml kodları yazılmalıdır.

Bunun ardından, oluşturulan “Resource Dictionary” dosyasının, “App.xaml” sayfasına bağlanması gerekmektedir. Aksi hâlde, bu dosyada bulunan şablon tanımlamalarını kullanamayız. Bağlama işlemi için, “App.xaml” dosyası içerisine gerekli kodları yazmalıyız. “App.xaml” dosyası içine yazılan xaml kodları şöyledir.

```
<Application
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             x:Class="SilverlightApplication6.App"
             >
     <Application.Resources>
<ResourceDictionary>
   <ResourceDictionary.MergedDictionaries>
      <ResourceDictionary Source="sablonlarim.xaml"/>
   </ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
     </Application.Resources>
</Application>
```

Yukarıda vurgulanmış kodlar, “sablonlarim.xaml” dosyasını “App.xaml” dosyasına bağlamıştır. Bağlama işleminden sonra, bu şablon, projede bulunan bütün sayfalardaki butonlara uygulanabilir.

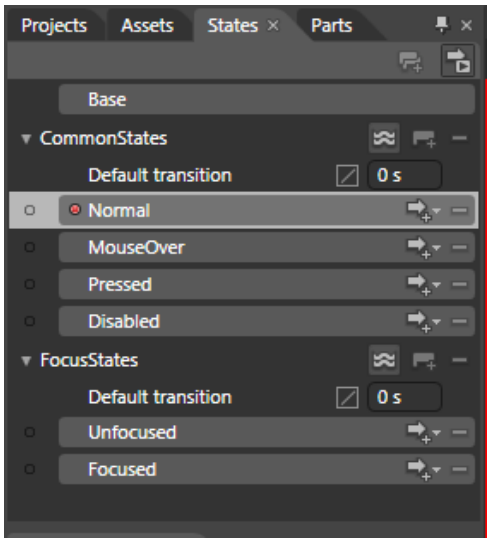
1.5.3. Kontrol Şablonlarında Durum Yönetimi

Oluşturduğumuz kontrol şablonlarının, uygulamanın çalışması sırasında, tıklama, üzerinde farenin hareket ettirilmesi gibi eylemlere maruz kaldıklarında biçimsel açıdan durum değiştirmeleri için, şablonlara ait durum tasarımlarını yapmamız gerekir. Şablonları etkileşimli hâle getirmek, uygulamalarımıza görsel açıdan zenginlik kazandıracaktır. XAML tabanlı uygulama geliştirme platformu bünyesinde bulunan VisualStateManager sayesinde, kontrol şablonlarının farklı durumlarına ilişkin, farklı tasarımlar yapabiliriz. Ve bu sayede, kullanıcı ile etkileşimli kontrol şablonları tasarlayabiliriz. XAML tabanlı uygulama geliştirme platformu ile durum yönetimi işlemlerini yapmak için, iki teknik karşımıza çıkmaktadır. Bunlardan birisi XAML kodları kullanarak VisualStateManager yapısı oluşturmak, diğeri de tasarım editörü kullanarak durumları tasarlamaktır.

Durum tasarımları yapmak için, ilk olarak tasarım editörünü ele alacağız. “MainPage.xaml” sayfasına yerleştirilen bir Button kontrolünden yola çıkarak, States panelini kullanıp, durumlara ilişkin tasarımlar yapacağız.

- Önce sayfa üzerine bir Button kontrolü yerleştirip, daha önceki konulardan da hatırlayacağımız gibi, sağ tuş menüsünden, “Edit Template→Create Empty” seçilmelidir. Karşımıza gelecek pencereden, yeni şablonumuza bir isim vererek şablonun tanımlanacağı konumu seçmeliyiz.

Bu sayede, şablonu tasarlayacağımız sayfa açılmış olacaktır. İçerisi boş bir grid kontrolüyle karşılaşacağız. Bu sırada, States panelini açık hâle getirdiğimizde, bu panel üzerinde bir buton için kullanılabileceğimiz durumlar görüntülenecektir. States panelinde gördüğümüz durumlar, her kontrol için aynı değildir. Örneğin tasarladığımız şablon bir Slider kontrolüne ait olsaydı, panelde görüntülenen durumlar farklı olacaktı.



Resim 1.17: States Paneli

Paneli incelediğimizde, durumların CommonStates ve FocusStates olarak iki gruba ayrıldığını görürüz. CommonStates altında bulunan durumlar aşağıda belirtilmiştir.

- **Normal:** Kontrolün herhangi bir eyleme maruz kalmadığı durumdur.
- **MouseOver:** Kontrol üzerinde fare işaretçisinin bulunduğu durumdur.
- **Pressed:** Kontrol üzerinde farenin sol tuşu tıklatıldığı durumdur.
- **Disabled:** Kontrol'e ait IsEnabled özelliğinin False yapılmasıyla, kontrolün pasif hâle geldiği durumdur.

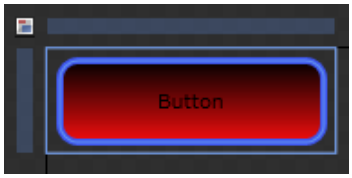
FocusStates bölümünde ise iki durum vardır.

- **Focused:** İmlecin bir kontrol üzerine odaklandığı durumdur.
- **Unfocused:** İmlecin bir kontrolden ayrıldığı durumdur.

Bir duruma ilişkin tasarım yapmak için, o durum üzerinde tıklanmalıdır. Örneğin, yukarıdaki resimde Normal durum seçilidir. Ayrıca bir durum seçildiğinde, sayfalar sekmesinin altında, hangi durum üzerinde çalıştığımızı görebiliriz (bk. Resim 1.18).

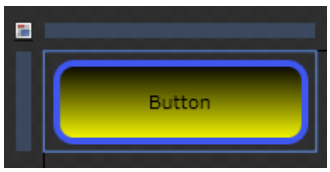
Resim 1.18: Aktif durum

Normal durumunu seçerek butonun Normal durumuna ilişkin tasarımı yapacağız.



Resim 1.19: Normal durum tasarımı

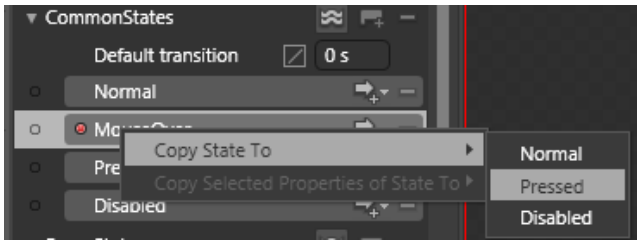
Normal durumun ardından MouseOver durumunu tasarlayacağız.



Resim 1.20: MouseOver durumu tasarımı

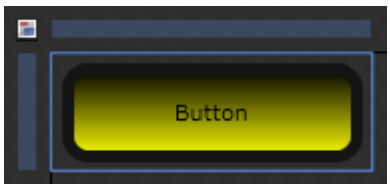
Pressed durumunun tasarımına geçmeden önce, hazırladığımız MouseOver durumunu, Pressed durumu içerisine kopyalayacağız. Çünkü ilk başta, Pressed içinde Normal durumun bir kopyası mevcuttur. Pressed durumunun, özelliklerini Normal durumdan değil, MouseOver durumundan alması daha uygun olacaktır.

Bir durum içindeki tasarım, istenilen başka bir duruma kopyalanabilir. Bunu anlamak için, aşağıdaki resmi inceleyelim.



Resim 1.21: Durum kopyalama

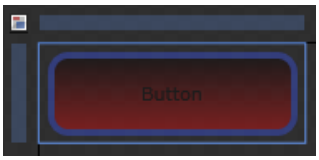
Resimde görüldüğü gibi, MouseOver durumu üzerinde sağ tuş menüsü açılıp "Copy State To" altında görünen durum isimleri içinden Pressed seçilmiştir. Kopyaladıktan sonra MouseOver durumu ile aynı tasarıma sahip olan Pressed durumunda değişiklik yapacağız.



Resim 1.22: Pressed durumu tasarımı

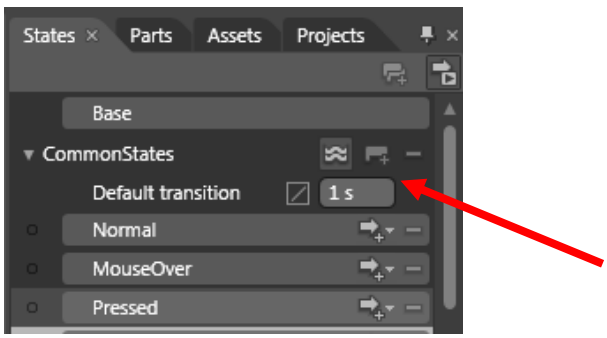
Pressed durumunu bitirdikten sonra Disabled durumunun tasarım özelliklerini değiştireceğiz. Zaten Disabled durumu içerisinde Normal durumun bir kopyası bulunmaktadır. Yapacağımız şey, sadece tasarım alanındaki Grid kontrolünü seçip Opacity

(Şeffaflık) özelliğinin değerini %40 yapmaktır. Bu sayede, eğer düğme uygulama tarafından pasif hâle getirilirse, yarı şeffaf görünecektir.



Resim 1.23: Disabled durumu tasarımı

Bütün durumları tasarladıktan sonra durumlar arası geçişleri süreli yaparak daha esnek durum değişiklikleri sağlayabiliriz. Bunun için States panelinin üst kısmındaki "Default transition" bölümüne, saniye cinsinden süre değeri belirtmemiz gerekiyor. Aşağıdaki resimde bu işlemin yapılması gösterilmiştir.



Resim 1.24: Geçiş süresini ayarlama

Daha sonra "MainPage.xaml" sayfasına birkaç düğme yerleştirip oluşturduğumuz "renklibuton" isimli kontrol şablonunu düğmelerde kullanacağız (birisi pasif hâlde).

Resim 1.25: Çalışma anı görüntüsü

XAML kodları yardımıyla şablonlarda durum yönetimi için ise, VisualStateManager yapısı kullanılmaktadır. VisualStateManager yapısını, herhangi bir xaml sayfasında, "App.xaml" sayfasında yada bir "Resource Dictionary" sayfasında oluşturabiliriz. Bilindiği

gibi şablonun oluşturulduğu sayfa, o şablonun nerelerde geçerli olacağını belirlemesi açısından önemlidir. VisualStateManager yapısı, aşağıda görüldüğü gibi oluşturulur.

```
<VisualStateManager.VisualStateGroups>
        <VisualStateGroup x:Name="FocusStates">
            <VisualState x:Name="Focused"/>
            <VisualState x:Name="Unfocused"/>
        </VisualStateGroup>
        <VisualStateGroup x:Name="CommonStates">
            <VisualState x:Name="Normal"/>
            <VisualState x:Name="MouseOver" />
            <VisualState x:Name="Pressed"/>
            <VisualState x:Name="Disabled"/>
        </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
```

Bu yapının oluşturulmasına ilişkin bazı kurallar mevcuttur.

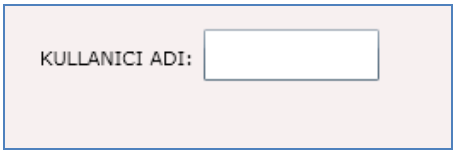
- Görüldüğü gibi bütün tanımlamalar; <VisualStateManager.VisualStateGroups> </VisualStateManager.VisualStateGroups> etiketleri içinde yapılır.
- Ayrıca her bir durum grubu, <VisualStateGroup x:Name="..."> </VisualStateGroup> etiketleri arasında tanımlanır.
- Durum grupları içindeki her bir durum, <VisualState x:Name="..."/> biçiminde tanımlanır.
- Eğer bir durum için biçimsel bir değişiklik söz konusu ise, gerekli biçimsel özellikler o durum altında <Storyboard></Storyboard> bloğunda animasyon tipinde tanımlanır.

1.5.4. Kullanıcı Tanımlı Durumlar

VisualStateManager yapısının bir büyük avantajı da, kontrollerden bağımsız durumlar oluşturmamızı ve programlama dili kodlarıyla uygulama çalışırken bu durumlar arasında geçişleri sağlamasıdır. Yani projemiz çalıştırıldığında, program akışında oluşan bazı şartlara göre, nesnelerin dinamik olarak durum değiştirip farklı görünümlere sahip olmasını sağlayabilmesidir. Bu konuyu da bir örnek uygulama üzerinde inceleyelim.

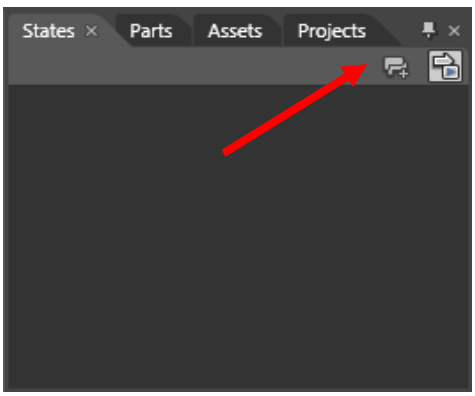
Örnek: Sayfa üzerinde bulunan bir TextBox kontrolüne kullanıcı adı girişi yapılacaktır. TextBox nesnesi, veri girişi sırasında, kendisine girilen harf sayısını kontrol edecek ve eğer harf sayısı üçten az ise kırmızı renk, harf sayısı üç ile yedi arasında ise turuncu renk, harf sayısı sekiz ve daha fazla olduğunda sarı renk alacaktır. Bu uygulamayı, adım adım geliştireceğiz.

- İlk olarak MainPage sayfası üzerine, "textBox1" ismini vereceğimiz bir adet TextBox ve bir adet TextBlock yerleştirip, aşağıda görülen sayfa tasarımını oluşturacağız.

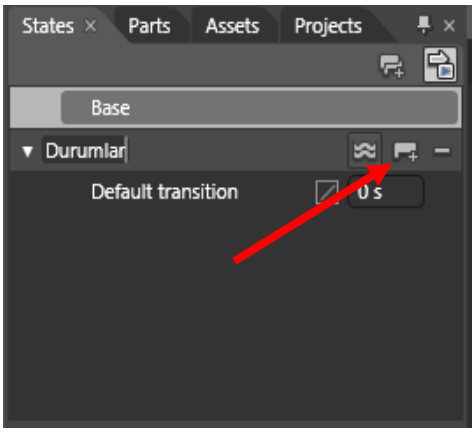


Resim 1.26: Sayfa tasarımı

Daha sonra sayfa üzerinde seçili nesne olmamasına dikkat edeceğiz. States panelini açık hâle getireceğiz. Panel ilk başta boş olarak görünecektir. Buradan bir durum grubu(State Group) ekleyerek, ismini Durumlar olarak belirleyeceğiz.

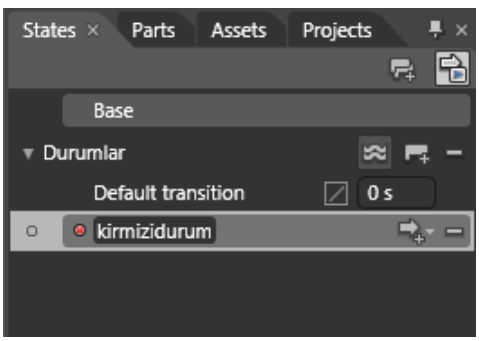


Resim 1.27: Durum grubu ekleme



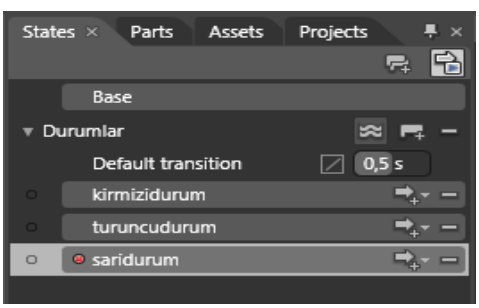
Resim 1.28: Durum ekleme

Durumlar grubu oluştuktan sonra yukarıdaki resimde okla gösterilen "Add State" ikonuna tıklayarak, grup içine yeni bir durum oluşturup daha sonra bu duruma isim vereceğiz.



Resim 1.29: Duruma isim verme

Oluşturduğumuz duruma "kirmizidurum" ismini veriyoruz. Bu durumu oluşturduğumuz aynı yöntemle, "turuncudurum" ve "saridurum" isimli iki durum daha ilave edeceğiz. Durumlar arası geçiş süresi 0,5 sn. olarak belirlenmelidir. Sonuçta States paneli aşağıda gösterildiği gibi olacaktır.



Resim 1.30: Bütün durumları oluşturma

Daha sonra sırayla her bir duruma tıklayıp, "MainPage" üzerinde bulunan TextBox nesnesinin arka plan rengini, her bir durum için durum ismine uygun renkler yapmalıyız.

Daha sonra, “MainPage.xaml” sayfası üzerinde bulunan TextBox nesnesinin TextChanged olayına, aşağıdaki kodları yazacağız.

```
void textBox_TextChanged(object sender, TextChangedEventArgs e)
{
      if (textBox1.Text.Length < 3)
      {
         VisualStateManager.GoToState(this, "kirmizidurum", true);
      }
      else if (textBox1.Text.Length < 8)
      {
         VisualStateManager.GoToState(this, "turuncudurum", true);
      }
      else
      {
          VisualStateManager.GoToState(this, "saridurum", true);
      }
}
```

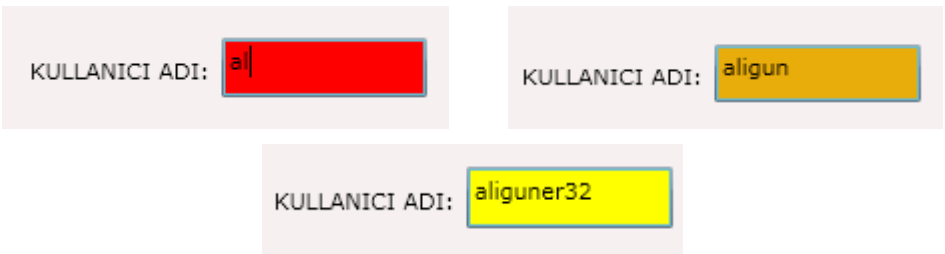
textBox1.Text.Length özelliği, “textBox1” içine yazılan metnin uzunluğunu verir.

Aşağıdaki kod satırında “kirmizidurum” isimli durum çağrılmıştır.

```
VisualStateManager.GoToState(this, "kirmizidurum", true);
```

VisualStateManager.GoToState() metodu, üç parametre almaktadır. This gönderilmiş olan ilk parametre, durum geçişlerinin uygulanacağı kontrolün ismini alır. İkinci parametre string olup çağrılacak olan durum ismini temsil eder. Üçüncü parametrenin true olması, geçiş animasyonuna izin verilmesi anlamına gelir.

Uygulama çalıştırıldığında, text kutusuna veri girişi esnasında her üç durum arasında geçiş olduğu gözlemlenecektir.



Resim 1.31: TextBox içerisine metin yazılırken renk değiştirmesi

UYGULAMA FAALİYETİ

Bu uygulama faaliyetinde, şablon oluşturup kullanmayı öğrenerek uygulayacaksınız.

İşlem Basamakları	Öneriler
➢ Uygulama geliştirme editöründe oluşturduğunuz bir projeyi, tasarım editöründe açarak sayfa üzerine bir adet ListBox, bir adet Button ve bir adet TextBox yerleştiriniz.	➢ ListBox için listBox1 ➢ TextBox için textBox1 ➢ Button için button1 isimlerini kullanınız.
➢ listBox1'i seçiniz.	➢ listBox1 üzerinde tıklayınız.
➢ States panelinde, listBox1'in birinci durumunu ekleyiniz.	➢ Durum adını "yesil" olarak belirleyiniz. ➢ listBox1'in arka planını, yeşil renk yapınız.
➢ States panelinde listBox1'in ikinci durumunu ekleyiniz.	➢ Durum adını "kirmizi" olarak belirleyiniz. ➢ listBox1'in arka planını, kırmızı renk yapınız.
➢ Projeyi kaydedip, uygulama geliştirme editörünü açık hâle getiriniz.	➢
➢ Buton1'in Click metoduna giriniz.	➢ Buton1 üzerinde çift tıklayınız.

➢ Click olayı içerisine gerekli kodları yazınız.	void buton1_Click(object sender, RoutedEventArgs e)
	{
	listBox.Items.Add(text1.Text);
	if (listBox.Items.Count < 10)
	{ VisualStateManager.GoToState(this, "yesil", true);
	}
	else
	{ VisualStateManager.GoToState(this, "kirmizi", true);
	}
	➢ }

➢ Projeyi çalıştırınız.	➢ F5 tuşunu kullanınız.
➢ Text kutusuna girdiğiniz değerleri düğmeye tıklayarak Liste kutusuna ekleyiniz.	➢ Eleman sayısı ondan az olmamalıdır.
➢ Eleman sayısına göre liste kutusunun renk değiştirdiğini gözlemleyiniz.	

ÖLÇME VE DEĞERLENDİRME

Aşağıdaki soruları dikkatlice okuyunuz ve doğru seçeneği işaretleyiniz.

1. MediaElement özelliklerinden hangisi, oynatılan ses dosyası için hoparlör dengesini ayarlamayı sağlar?

A) Volume
B) Stretch
C) Balance
D) Source

2. Bir thread çalışır durumda iken bir süre çalışmasını duraklatıp süre tamamlandığında çalışmasına devam etmesini sağlayan, Thread sınıfının hangi metodudur?

A) Suspend()
B) B) Sleep()
C) Abort()
D) Start()

3. Başka projelerde de geçerli olabilecek şablonlar oluşturmak için aşağıdakilerden hangisini yapmalıyız?

A) Resource Dictionary sayfası oluşturmalıyız.
B) MainPage.xaml içerisinde tanımlamalıyız.
C) App.xaml sayfası içinde tanımlamalıyız.
D) Üzerinde çalıştığımız sayfa içerisinde tanımlamalıyız.

4. Tasarım editöründe durum yönetimi işlemleri hangi panel içinden yapılır?

A) Assets
B) Projects
C) States
D) Parts

5. Tek bir sayfada geçerli şablon tanımlamaları hangi etiket bloğunda yapılır?

A) <UserControl.Resources>
B) <Application.Resources>
C) <ResourceDictionary>
D) <VisualStateManager.VisualStateGroups>

6. Bir kontrolün IsEnabled özelliğinin False yapılmasıyla pasif olduğu durum hangisidir?

A) Normal
B) MouseOver
C) Pressed
D) Disabled

Aşağıdaki cümleleri dikkatlice okuyarak boş bırakılan yerlere doğru sözcüğü yazınız.

7. XAML kodları içerisinde, şablonların durum yönetimi için kullanılan etiket…….dir.

8. XAML kodları içerisinde, kontrol şablonlarının durumlarında biçimsel tasarım yapmak için…….etiketi kullanılır.

9. C# dilinde, kullanıcı tarafından tanımlanmış durumlar arasında geçiş sağlamak için…….. metodunu kullanırız.

10. Tasarım editöründeki States panelinde, durumlar arasındaki geçiş süresini belirlemek için…….bölümü kullanılır.

DEĞERLENDİRME

Cevaplarınızı cevap anahtarıyla karşılaştırınız. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt ettiğiniz sorularla ilgili konuları faaliyete geri dönerek tekrarlayınız. Cevaplarınızın tümü doğru ise bir sonraki öğrenme faaliyetine geçiniz.

ÖĞRENME FAALİYETİ–2

AMAÇ

İleri düzey etkileşimli uygulamaları yapabileceksiniz.

ARAŞTIRMA

- Temel html etiketlerini araştırınız.
- Web sayfalarında kullanılan CSS stilleri hakkında araştırma yapınız.
- JavaScript dilinde metodların oluşturulması konusunu araştırınız.

2. ETKİLEŞİMLİ WEB UYGULAMALARI

2.1. Uygulama Geliştirme ve IIS

Sunucu bilgisayarda çalışan IIS(Internet Information Services), çeşitli sebeplerden dolayı bazen sunucuda yüklü XAML tabanlı uygulama geliştirme platformu uygulamalarına ait olan XAML dosyalarını, istemciye gönderemeyebilir. Bu durumu düzeltmek için, sitemize ait, IIS MIME Type ayarının yapılması gerekiyor.

Bunun işlemi yapmak için "Internet Information Services Yöneticisi" (IIS) kullanılmalıdır. IIS üzerinde tanımlanmış durumda bulunan web sitemizin MIME türleri içine, aşağıdaki iki özellik eklenmelidir (Bu işlemler, ancak web sitemizdeki xaml sayfaları açılmıyorsa yapılmalıdır.).

Dosya uzantısı: .xaml
MIME type: application/xaml+xml

Dosya uzantısı: .xap
MIME type: application/x-silverlight-app

2.2. HTML Erişimi

XAML tabanlı uygulama geliştirme platformu uygulamaları çalıştırıldığında, tarayıcı (browser) üzerinde açılan html ya da asp.net sayfasının bir bölümüne yerleştirilmiş olarak

görünür. Yani açılan web sayfasında XAML tabanlı uygulama geliştirme platformu uygulamasının yanı sıra başka html ya da asp.net kontrolleri bulunabilir.

XAML tabanlı uygulama geliştirme platformu, içerdiği bazı sınıf yapıları sayesinde, uygulamanın, aynı sayfa üzerinde bulunan HTML elemanları ile etkileşimine izin verir.

Aşağıdaki tabloda, uygulamalar içinden HTML erişimi için kullanılan sınıflar gösterilmiştir.

Sınıf Adı	**Açıklama**
HtmlPage	XAML tabanlı uygulama geliştirme platformu uygulamasının üzerinde çalıştığı HTML sayfasını temsil eder.
BrowserInformation	Tarayıcı ile ilgili bazı bilgilere ulaşmak için kullanılır.
HtmlDocument	HtmlPage içindeki Document nesnesini temsil eder. Document ise sayfadaki bütün nesneleri içerir.
HtmlElement	Sayfa içerisindeki HTML elemanlarının tamamını temsil eder.

Tablo 2.1: HTML erişim sınıfları

HTMLPage sınıfı, HtmlDocument sınıfını içerir. HtmlDocument sınıfı ise sayfadaki HTML elemanlarına erişebilen HtmlElement sınıfını içerir.

Yukarıda bahsedilen sınıfların bazı metodları sayesinde, html elemanlarının çeşitli özellikleri değiştirilebilir. Ya da sayfa üzerine yeni html elemanları yerleştirilebilir. Kısacası html kodlarıyla statik olarak yaptırılan her şey, uygulamanın çalışması esnasında dinamik olarak yaptırılabilir.

Not: HTML erişimine ilişkin sınıfları içeren kütüphaneyi, kod yazdığımız dosyanın üst tarafına, aşağıdaki gibi dahil etmemiz gerekiyor.

```
using System.Windows.Browser;
```

HTML erişimi için kullandığımız bazı metodları inceleyelim.

- **HtmlDocument.GetElementById( ) metodu**

Html sayfasından istenilen bir html nesnesini elde eder ve HtmlElement türünde geriye döndürür. String türde bir parametre alır. Elde etmek istediğimiz nesnenin id değerini, bu metoda string türünde bir parametre olarak göndermeliyiz. Aşağıdaki örneği inceleyiniz.

Örnek:

```
HtmlDocument sayfa = HtmlPage.Document;
```

```
HtmlElement paragraf = sayfa.GetElementById("paragraf1");
```

Kodlarda görüldüğü gibi, sayfada bulunan ve id özelliği "paragraf1" olan html elemanını yakalayıp HtmlElement türünde bir değişkene atar.

- **HtmlElement.SetAttribute() metodu**

Bir html elemanının, herhangi bir özelliğini değiştirmek amacıyla kullanılır. Dışarıdan string türünde iki parametre değeri alarak çalışmaktadır. Aldığı parametre değerlerinden ilki, html elemanının, değiştirilecek olan özelliğinin ismidir. Diğer parametre ise, özelliğe atanacak olan değerdir. Aşağıdaki örneği inceleyiniz.

Örnek:

```
HtmlDocument sayfa = HtmlPage.Document;
HtmlElement metinkutu = sayfa.GetElementById("text1");
metinkutu.SetAttribute("value", "MERHABA");
```

Çalıştırıldığında "text1" isimli metin kutusunun value değeri(içeriği), "MERHABA" olacaktır.

- **HtmlDocument.CreateElement() metodu**

Sayfa üzerinde bir html nesnesi oluşturmak için kullanılır. Dışarıdan string türünde bir parametre alır. Aldığı değer, oluşturacağı nesnenin etiket ismidir. Örneğin, bir paragraf oluşturmak için "p" değeri, bir bağlantı oluşturmak için "a" değeri gönderilmelidir.

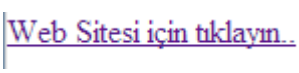
- **HtmlDocument.Body.AppendChild() metodu**

CreateElement() metoduyla oluşturulan bir html elemanı, bu metod sayesinde sayfanın body bölümüne eklenir ve görünür hâle gelir.

Örnek:

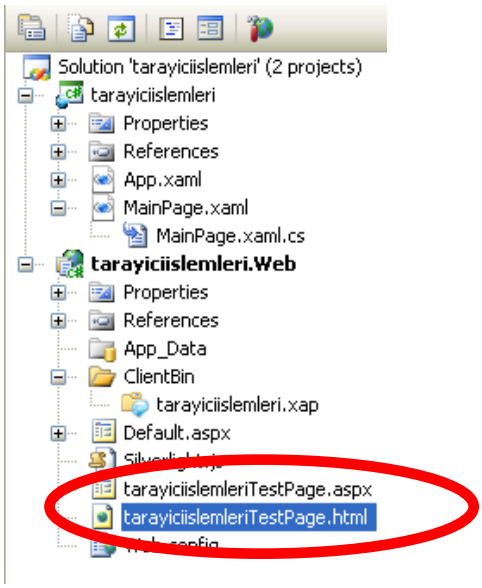
```
HtmlDocument sayfa = HtmlPage.Document;
HtmlElement baglanti = sayfa.CreateElement("a");
baglanti.SetAttribute("href", "http://www.google.com");
baglanti.SetAttribute("innerHTML", "Google Sayfası için Tıklayın.");
sayfa.Body.AppendChild(baglanti);
```

Çalıştırıldığında sayfa üzerinde, aşağıdaki gibi bir bağlantı oluşturacaktır.



Html erişimine ilişkin sınıf ve metodların bazılarını, örnek bir uygulama üzerinde kullanacağız.

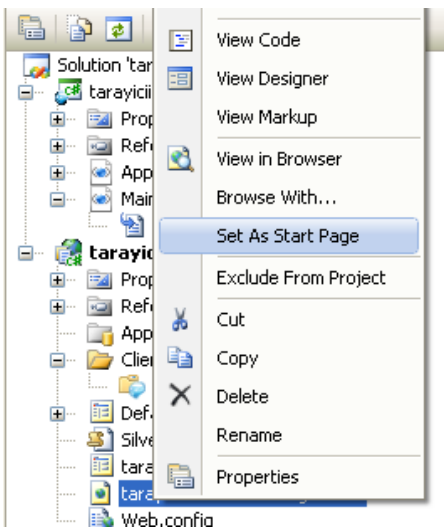
- İlk olarak uygulama geliştirme editöründe yeni oluşturulan projemizin, Solution Explorer panelini inceleyeceğiz. Projemizin Web kısmında bulunan ve Resim 2.1'de işaretlenmiş iki dosyaya dikkat edelim.



Resim 2.1: Tarayıcıda açılan web sayfaları

Bu iki dosyadan birisi asp.net sayfası, diğeri ise html sayfasıdır. Eğer istersek bu iki dosyadan hangisinin başlangıç sayfası olduğunu belirleyebiliriz. Html etiketleri ile ilgili işlemler yapacağımızdan dolayı, "tarayiciislemleriTestPage.html" dosyasını başlangıç sayfası yapacağız.

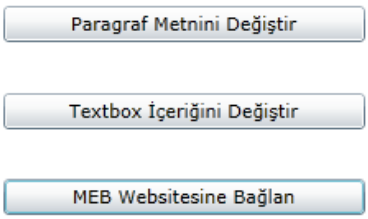
Sayfa ismi üstünde sağ tuş menüsü açıldıktan sonra "Set as Start Page" seçeneği tıklanırsa o sayfa açılış sayfası yapılmış olur.



Resim 2.2: Bir sayfanın açılış sayfası yapılması

- Uygulamanın XAML tabanlı uygulama geliştirme platformu kısmına ait tasarım, "MainPage.xaml" sayfası üzerinde yapılacaktır. HTML kısmına ait düzenlemeler ise "tarayiciislemleriTestPage.html" sayfası üzerinde yapılacaktır.

Tasarım editöründe aşağıdakine benzer bir tasarım yapacağız.



Resim 2.3: Sayfanın tasarım görünümü

- Daha sonra "tarayiciislemleriTestPage.html" sayfasını açarak, bu sayfadaki html kodlarını inceleyeceğiz. Bu sayfada epeyce uzun CSS, JavaScript ve HTML kodları var. Sadece HTML kodlarındaki <BODY> bloğuna dikkat edeceğiz. Ve daha sonra arka sayfadaki HTML kodlarında belirttiğimiz yere, bazı HTML kodlarını yazacağız.

```
<body>
    <form id="form1" runat="server" style="height:100%">
    <div id="silverlightControlHost">

        <object data="data:application/x-silverlight-2," type="application/x-
 silverlight-2" width="100%" height="100%">
              <param name="source" value="ClientBin/tarayiciislemleri.xap"/>
              <param name="onError" value="onSilverlightError" />
              <param name="background" value="white" />
              <param name="minRuntimeVersion" value="3.0.40818.0" />
              <param name="autoUpgrade" value="true" />
              <a href="http://meb.gov.tr " style="text-decoration:none">
                   <img src="silver.jpeg" alt="tarayici işlemleri"
 style="border-style:none"/>
              </a>
          </object>


         !!!!YENİ HTML KODLARINI BURAYA YAZACAĞIZ!!!!!

         <iframe id="_sl_historyFrame"
style="visibility:hidden;height:0px;width:0px;border:0px"></iframe></div>
    </form>

</body>
```

Yukarıdaki HTML kodları içinde, <object></object> bloğuna dikkat edelim. Bu kodlar, XAML tabanlı uygulama geliştirme platformu uygulamamızı, sayfa üzerine yerleştiren kodlardır. Yukarıdaki kodlar içinde, <object></object> bloğunun alt tarafında

renkli olarak belirttiğimiz yere, yeni html kodlarını yazacağız. Yazacağımız kodlar aşağıda gösterilmiştir.

```
<br />
<br />
<p> HTML ELEMANLARI</p>
<p id="paragraf1"> Benim içeriğimi değiştirecek!</p>
<input id="text1" type="text" value="Benimkini de değiştirecek!" />
```

Bilindiği gibi,
 etiketleri alt satıra geçmeyi sağlar. Ayrıca <p> etiketleri ile iki adet paragraf ve <input> etiketiyle bir adet metin kutusu oluşturulmuştur. Paragrafların birisinin id özelliğine, "paragraf1" ismi atanmıştır. Çünkü XAML tabanlı uygulama geliştirme platformu program kodları, HTML nesnelerine erişirken nesnelerin id özelliklerini kullanır. Aynı şekilde, metin kutusunun içeriği de XAML tabanlı uygulama geliştirme platformu tarafından değiştirileceği için, onun id özelliğine de "text1" ismi atanmıştır.

Html sayfasına bu kodları ekledikten sonra uygulamanın tasarım görüntüsü aşağıdaki gibi olacaktır.

Resim 2.4: Tasarım görünümü

- Ardından "MainPage.xaml" sayfasındaki düğmelerin Click olayları için gerekli C# kodlarını yazacağız. Paragraf metnini değiştirecek olan "buton1" düğmesinin Click olayına, aşağıdaki kodlar yazılmıştır.

```
void buton1_Click(object sender, RoutedEventArgs e)
{
    HtmlDocument sayfa = HtmlPage.Document;
    HtmlElement paragraf = sayfa.GetElementById("paragraf1");
    paragraf.SetProperty("innerHTML", "BEN DEĞİŞTİM!");
}
```

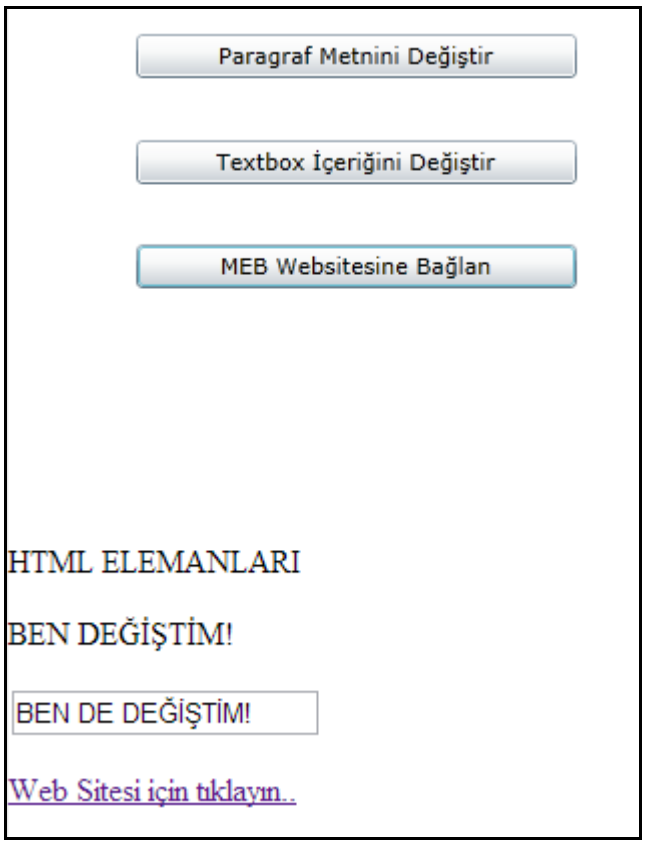
- Metin kutusunun içeriğini değiştirecek olan "buton2" düğmesinin Click olayı içerisine ise, arka sayfadaki kodlar yazılmıştır.

```
void buton2_Click(object sender, RoutedEventArgs e)
{
     HtmlDocument sayfa = HtmlPage.Document;
    HtmlElement metinkutu = sayfa.GetElementById("text1");
 metinkutu.SetAttribute("value", "BEN DE DEĞİŞTİM!");}
```

- Google sitesi bağlantısını oluşturacak olan "buton3" düğmesinin, Click olayına aşağıdaki kodlar yazılmıştır.

```
void buton3_Click(object sender, RoutedEventArgs e)
{
  HtmlElement baglanti = HtmlPage.Document.CreateElement("A");
            baglanti.SetAttribute("HREF", "http://www.meb.gov.tr");
            baglanti.SetProperty("innerHTML", "Web Sitesi icin
tıklayınız..");

            HtmlPage.Document.Body.AppendChild(baglanti);
}
```

- Uygulama çalıştırılıp her üç düğmeye de tıklandığında, sayfa aşağıdaki gibi görünecektir.



Resim 2.5: Çalışma anındaki sayfa görüntüsü

2.3. CSS Erişimi

XAML tabanlı uygulama geliştirme platformu uygulamalarının web tarayıcısında çalışması sırasında, uygulamanın bulunduğu sayfada yer alan html elemanlarının CSS(stil) özelliklerini değiştirmemiz mümkündür. Bunun için, HtmlElement.SetStyleAttribute( ) metodu kullanılmalıdır.

Bu metod, dışarıdan string türde iki parametre değeri almaktadır. Aldığı parametre değerlerinden ilki, değiştirilecek olan CSS özelliğinin ismidir. Diğer parametre değeri ise CSS özelliğine atanacak olan değerdir.

Örnek:

```
HtmlDocument sayfa = HtmlPage.Document;
HtmlElement paragraf = sayfa.GetElementById("paragraf1");
paragraf.SetStyleAttribute("background", "red");
```

Şimdi, SetStyleAttribute metodunu bir örnek üzerinde kullanalım.

- Öncelikle, aşağıdaki form tasarımını "MainPage.xaml" sayfası üzerinde yapacağız.

Resim 2.6: Sayfa tasarımı

- Projenin html sayfasını açılış sayfası yaptıktan sonra, bu sayfanın html kodlarında <object> </object> bloğunun bittiği yerin altına aşağıdaki paragraf kodunu yazacağız.

```
<p id="paragraf1"> HTML kodlarıyla oluşturduğumuz paragraf.</p>
```

- "MainPage.xaml" sayfasında bulunan düğmelerin Click olaylarına, arka sayfadaki kodlar yazılmalıdır.

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    HtmlDocument sayfa = HtmlPage.Document;
    HtmlElement paragraf = sayfa.GetElementById("paragraf1");
    paragraf.SetStyleAttribute("color", "green");
}

private void button3_Click(object sender, RoutedEventArgs e)
{
     HtmlDocument sayfa = HtmlPage.Document;
     HtmlElement paragraf = sayfa.GetElementById("paragraf1");
     paragraf.SetStyleAttribute("background", "red");
}

private void button2_Click(object sender, RoutedEventArgs e)
{
    HtmlDocument sayfa = HtmlPage.Document;
    HtmlElement paragraf = sayfa.GetElementById("paragraf1");
    paragraf.SetStyleAttribute("fontSize", "24px");
}
```

- Sayfanın çalışma görünümü aşağıdaki gibi olacaktır.

Resim 2.7: Çalışma anındaki sayfa görüntüsü

2.4. JavaScript Erişimi

XAML tabanlı uygulama geliştirme platformu uygulaması içinden, uygulamanın bulunduğu sayfanın kodları içine yazılmış olan bir JavaScript metodunu kolaylıkla çağırabiliriz. Bunun için ScriptObject sınıfını kullanmamız gerekiyor. Bu sınıf, web sayfalarındaki script unsurlarını temsil eder.

JavaScript erişimi için kullandığımız metodları inceleyelim.

- **HtmlPage.Window.GetProperty() metodu**

Bu metod, string türde bir parametre değeri alarak çalışır. Bu metoda, çağırmak istediğimiz JavaScript metodunun adını göndermemiz gereklidir. Görevi ise çağrılmak istenen JavaScript metodunu elde etmektir. Elde ettiği metodu içeren, Object türünde bir nesneyi geri döndürür. Bunun için geri döndürdüğü nesnenin ScriptObject sınıfına çevrilmesi gereklidir.

- **ScriptObject.InvokeSelf() metodu**

InvokeSelf() metodu, bir script unsurunun, temsil ettiği JavaScript metodunu çalıştırmasını sağlar. Parametre değeri olarak, bir ya da birden fazla string türde değer alabilir. Çalıştıracağı Javascript metoduna, dışarıdan aldığı değerleri parametre olarak gönderir. Aşağıdaki örneği inceleyiniz.

Örnek:

```
HtmlDocument sayfa = HtmlPage.Document;
ScriptObject jsmetod = (ScriptObject)HtmlPage.Window.GetProperty("mesajgoster");
jsmetod.InvokeSelf("Javascript'ten Merhaba");
```

Yukarıdaki örnekte, ikinci ve üçüncü satırlardaki kodlarda, "mesajgoster" isimli javascript metodu elde edilip "jsmetod" isimli bir ScriptObject nesnesine atanmıştır. Son satırda ise, "jsmetod" isimli nesne, kendi kendini bir parametreyle çağırmıştır.

ScriptObject sınıfı ve InvokeSelf( ) metodunu bir örnek üzerinde kullanalım.

- Bir XAML tabanlı uygulama geliştirme platformu projesi oluşturarak "MainPage.xaml" sayfasında bir düğme yerleştireceğiz.

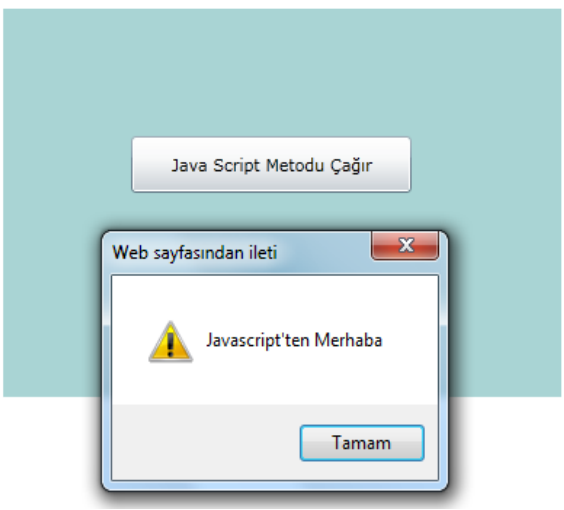
- Projenin html sayfasını, açılış sayfası yaptıktan sonra bu sayfanın baş tarafındaki <script></script> bölümü içerisinde aşağıdaki JavaScript metodunu yazacağız.

```
function mesajgoster(mesaj)
{
  alert(mesaj);
}
```

- "MainPage.xaml" sayfasında bulunan düğmenin Click olayına aşağıdaki kodlar yazılmalıdır.

```
private void button1_Click(object sender, RoutedEventArgs e)
{
 HtmlDocument sayfa = HtmlPage.Document;
 ScriptObject jsmetod =
(ScriptObject)HtmlPage.Window.GetProperty("mesajgoster");
  jsmetod.InvokeSelf("Javascript'ten Merhaba");
}
```

Sayfanın çalışma görüntüsü ise aşağıda verilmiştir.



Resim 2.8: Sayfanın çalışma durumu

2.5. Optimizasyon

XAML tabanlı uygulama geliştirme platformu uygulamaları hazırlarken, alabileceğimiz bazı tedbirler sayesinde, uygulamamızın sistem kaynaklarını gereksiz kullanmasının önüne geçerek çalışma hızını artırabiliriz. Bu tedbirleri birkaç madde hâlinde inceleyelim.

- Şeffaf arka plan renkleri performansı düşürebilir.

Gerekli olmadığı hâlde, şeffaf arka plan rengi kullanmak, performans kaybına neden olacaktır.

- Animasyon tasarımında StoryBoard kullanmak, daha iyi performans sağlar.

StoryBoard yerine JavaScript kullanarak animasyonlar oluşturmak, ciddi performans kaybı anlamına gelir.

- Metin animasyonlarını, vektörel metinlerle yapmalıyız.

Metinlerle animasyonlar yaparken, klasik metinler kullanmak yerine, metinleri vektörel çizimler hâlinde kullanmak, performansın iyileşmesine katkı sağlayacaktır.

- Görünmezlik özelliği için, Opacity değeri yerine Visibility özelliği kullanmalıyız.

Bir nesneyi görünmez yapmak için Opacity değerini sıfır yapmak yerine söz konusu nesnenin Visibility özelliğini Collapsed olarak ayarlamak daha uygundur.

- Videolarımızı kullanmadan önce yeniden encode etmek gerekiyor.

Bir videoyu *İnternet*te yayınlamak istiyorsak video dosyamızı küçük bir MediaElement içerisine yerleştirerek küçültmek yerine, videoyu söz konusu boyutlarda tekrar encode işlemine tabi tutarak, uygulamanızda yeni boyutlarıyla göstermek daha iyi performans sağlar.

- Yoğun JavaScript işlemleri kullanmaktan kaçınmak gerekiyor.

JavaScript arka planda çalışırken, XAML tabanlı uygulama geliştirme platformu animasyonlarının devam etmesi zor olacaktır ve performans düşüşü yaşanacaktır.

- Program kodları içinde Thread sınıfları kullanmak, performansı artırmaya yardımcı olur.

Thread nesneleri, uygulamadaki işlemlerin sıra beklemeden paralel yürümesini sağladığı için, çalışma hızını artırır.

2.6. Hata Ayıklama

Uygulama geliştirme editöründe uygulama geliştirme sırasında, program kodu içindeki mantıksal hatalarının tespit edilmesi için Debug menüsü seçenekleri kullanılır.

Uygulamaların bazen istediğimiz sonuçları vermediklerini ya da hatalı işlemler yaptıklarını görürüz. Bunun sebebi, programcının algoritmayı kurarken yaptığı bir takım fikir hatalarıdır. Birçok programlama dili editörü, "debugger" olarak bilinen hata ayıklama araçları sayesinde, programın çalışması sırasında program kodları üstünde analizler yapmaya imkân vermektedir.

Uygulama geliştirme editöründe hata ayıklama işlemleri için, Debug menüsü seçenekleri kullanılmaktadır. Bu menüde bulunan bazı seçenekleri açıklayacağız.

- Step Into komutuyla adım adım çalıştırma

Debug menüsündeki "Step Into" seçeneği (F11) uygulamamızı adım adım çalıştırmak için kullanılır. F11 tuşuna ilk basıldığında ilk satır çalıştırılır. Ve daha sonra bu tuşa her basıldığında, program satırları sırayla çalışmaya devam eder. Bu sırada, editörün alt kısmında görüntülenen Locals panelinde, değişken değerlerinin her adımda hangi değerlere ulaştıklarını görebiliriz. Adım adım ilerlerken, metod çağrılarına rast geldiğinde, metod içerisine dallanarak buradaki kodları da sırayla çalıştırır.

- "Step Over" komutuyla adım adım çalıştırma

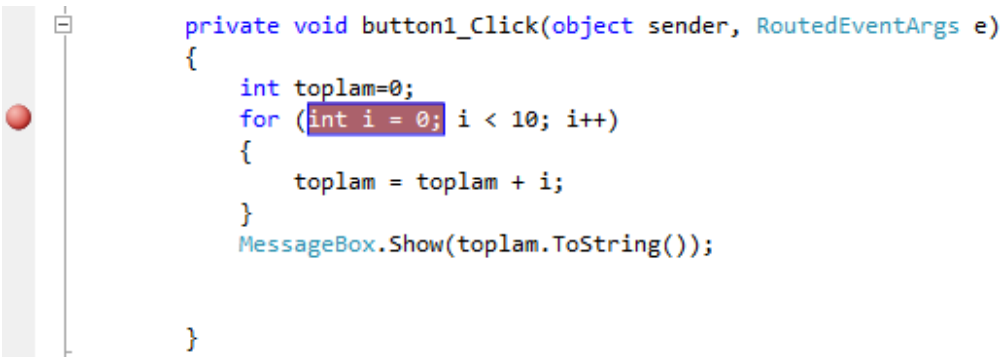
Bu seçenek de "Step Into" seçeneği gibi kodları arka arkaya çalıştırır. "Step Over" seçeneğinin farkı, kodlarda ilerlerken metod çağrısına rastgeldiğinde, metod içindeki kodlara dallanmadan metodu icra ederek, diğer satıra geçmesidir. "Step Over" komutunun kısayol tuşu ise F10 tuşudur.

- "Toogle Breakpoint" ile durma noktaları oluşturma

"Step Into" veya "Step Over" komutları kullanıldığında program satırları baştan başlayarak sırayla çalışır. Fakat bu yöntem, çok fazla kod satırı içeren uygulamalarda büyük zaman kaybına sebep olur. Çünkü, yüzlerce satırı tek tek çalıştırmak, pek de hoş bir durum değildir.

"Toggle Breakpoint" komutu sayesinde, mantık hatasının oluştuğunu düşündüğümüz kod satırlarının başladığı satıra işaret koyarak, durma noktaları oluşturabiliyoruz. Bunun için, imleci o satıra konumlandırıp Debug menüsünden "Toggle Breakpoint" seçeneğini tıklamak veya F9 tuşuna basmak yeterlidir. İşaretlenmiş bir satırın işaretini kaldırmak için yine aynı yöntemi kullanmalıyız. Kırılma noktasını oluşturduktan sonra programı normal yolla (yani F5 tuşuyla) çalıştırmalıyız. Bu sayede işaretli satırdan önceki kod satırlarının seri olarak çalıştırılmasını sağlamış oluruz. Program akışı kırılma noktasının bulunduğu satıra geldiğinde, program çalışması durur. Artık bu aşamadan sonra, F10 (Step Over) veya F11 (Step Into) seçeneklerini kullanarak satırları sırayla çalıştırmaya devam edebiliriz.

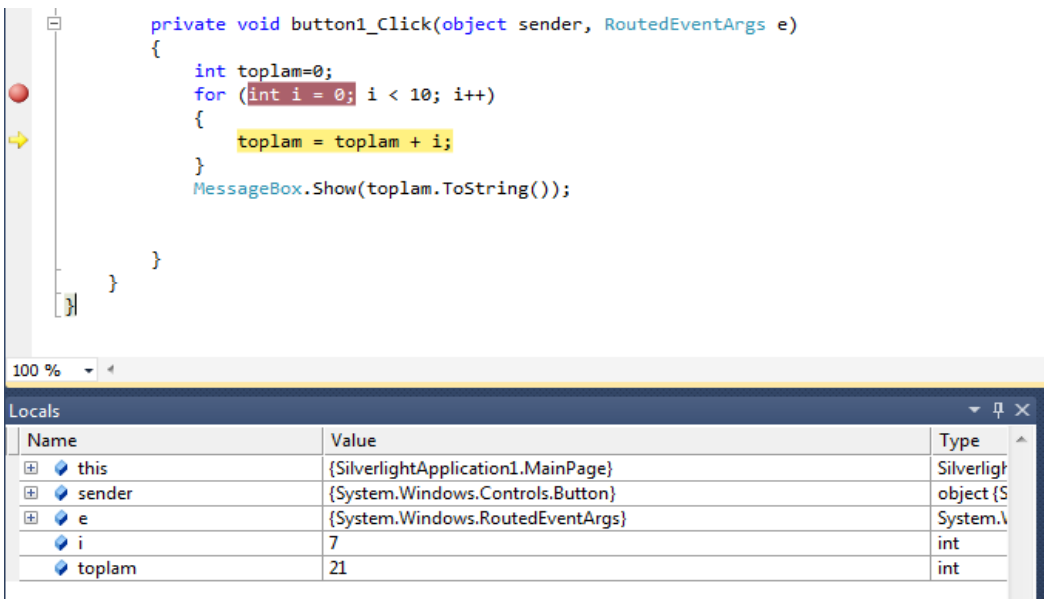
Arka sayfadaki resmi inceleyelim. Döngünün başladığı satırda "break point" noktası oluşturulmuştur. Daha sonra F5 tuşuyla program çalıştırılacaktır.



Resim 2.9: Break Point noktasının oluşturulması

Çalışan program üzerinde düğmenin tıklanmasından sonra "break point" satırında duracaktır. Ve bu satırdan sonra F11 tuşu kullanarak kodlar sıra ile çalıştırılmıştır.

Aşağıdaki resimden anlaşılacağı gibi, döngünün sekizinci tekrarında değişkenlerin aldığı değerler Locals panelinde gösterilmiştir.



Resim 2.10: Değişken değerlerinin izlenmesi

2.7. Sunucu Taraflı XAML

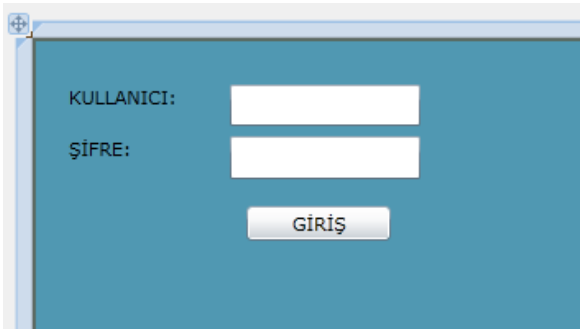
XAML tabanlı uygulama geliştirme platformu uygulamaları sunucu (server) üzerinde çalışmaz. İstemci (client) bilgisayarda çalışır. Bu yüzden, sunucudan veri transfer etmek ya da veri göndermek istediğimizde "Web service" denilen hizmet sınıfları oluşturarak XAML tabanlı uygulama geliştirme platformu uygulamalarının sunucu ile etkileşimini sağlarız.

Örnek bir uygulama üzerinde, bir projeye "Web Service" ekleyip kullanmayı öğreneceğiz. Örnekteki işlem adımlarında bir takım sınıf, metod ve olaylar açıklanmıştır.

Örnek uygulama:

Bir uygulama üzerinden, kullanıcı adı ve şifre girişi yapılması sağlanarak bu bilgilerin doğruluğunun sunucu üzerinde yaptırılması sağlanacaktır. Girilen kullanıcı adı ve şifre bilgilerinin doğruluğu, sunucu üzerinde kontrol edilerek, doğruluğu istemci bilgisayara, sunucu tarafından bildirilecektir. İstemci bilgisayar, sunucudan gelen değere göre, giriş işleminin başarılı ya da başarısız olduğu hakkında mesaj penceresi görüntüleyecektir. Aşağıdaki işlem adımlarını dikkatle takip ediniz.

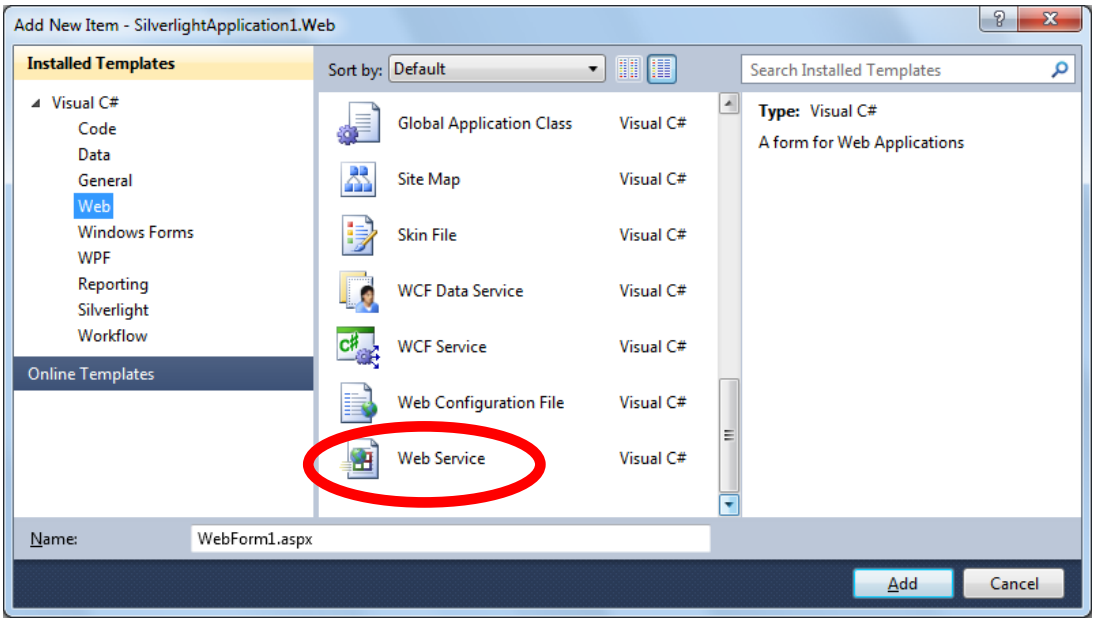
- Öncelikle uygulama geliştirme editöründe yeni bir XAML tabanlı uygulama geliştirme platformu projesi oluşturacağız. Ve aşağıda gördüğümüz sayfa tasarımını yapacağız (Şifre için PasswordBox kullanılacaktır.).



Resim 2.11: Sayfanın tasarım görüntüsü

- Daha sonra "Solution Explorer" paneli üzerinde, projemizin web tarafına, bir web servisi ekleyeceğiz. Proje içerisindeki web içeriğinin bulunduğu (**".Web"** ile biten) klasör üzerinde sağ tuşa tıklayıp, "Add→New Item" seçeneğine girilir.

- Açılan "Add New Item" penceresinde, "Web Service" seçilerek "Add" düğmesine tıklanır.



Resim 2.12: Proje içerisine Web Service eklenmesi

- "WebService1.asmx" isminde bir web service dosyası, ve buna bağlı olan "webservice1.asmx.cs" isimli bir C# kod dosyası eklenecektir.

- Daha sonra "WebService1.asmx.cs" dosyasının kodlarını editörde görüntüleyeceğiz. "WebService1.asmx.cs" kod dosyasında "WebService1" sınıf yapısını oluşturan kodlar, hazır olarak gelecektir. Bu kodlar içine metodumuzu ilave edeceğiz. Farklı renkle vurgulanmış kodlara dikkat edelim. Bu satırlar, kullanıcı adı ve şifre kontrolünü yapan program kodlarıdır.

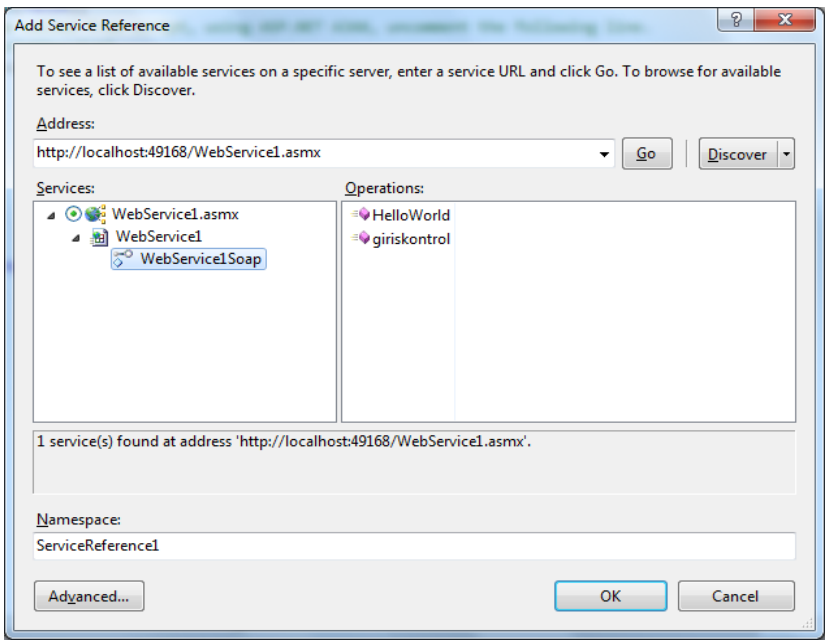
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;

namespace SilverlightApplication1.Web
{
    /// <summary>
    /// Summary description for WebService1
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
    // To allow this Web Service to be called from script, using ASP.NET
AJAX, uncomment the following line.
    // [System.Web.Script.Services.ScriptService]
    public class WebService1 : System.Web.Services.WebService
    {

        [WebMethod]
        public string HelloWorld()
        {
            return "Hello World";
        }

         [WebMethod]
         public bool giriskontrol(string kullanici,string sifre)
         {
             if (kullanici == "admin32" && sifre == "1234")
             {
                 return true;
             }
             else
             {
                 return false;
             }
         }


    }
}
```

Metodun başına [WebMethod] bildirisini eklememiz gerekiyor. [WebMethod] bildirisi, yazdığımız metodların uzak web istemcileri tarafından çağrılabilmesini sağlar. Metod, parametre değeri olarak aldığı kullanıcı adı ve şifre doğru ise, true sonucunu geriye döndürecek, aksi halde false sonucunu geriye döndürecektir.

- WebService1 class yapısı içinde "giriskontrol()" metodunu yazdıktan sonra projeyi, "Build" menüsünden "Build Solution"(ya da F6) komutuyla derleyeceğiz.

- Projeyi derledikten sonra servis referansı oluşturarak XAML tabanlı uygulama geliştirme platformu uygulaması ile oluşturduğumuz web servisi arasında bağlantı kurmamız gerekiyor. Bunun için "Solution Explorer" panelinde bulunan "SilverlightApplication1" üzerinde sağ tuş menüsünü görüntüleyerek, "Add Service Reference" komutu seçilir.

Açılan pencerede "Discover" düğmesini tıklayarak oluşturduğumuz web servisinin algılanmasını sağlayacağız. Oluşacak servis referansına, "ServiceReference1" ismi otomatik olarak verilmektedir.



Resim 2.13: Servis referansı oluşturma

- Artık, "MainPage.xaml.cs" kodları içinde sunucuda bulunan "giriskontrol()" metodunu çağırmak için gerekli kodları yazacağız. Ama bunun öncesinde yapılması gereken birkaç işlem vardır.

- Web servisine, program kodları ile erişmek için "MainPage.xaml.cs" kodları içinde "webservis" isminde bir sınıf örneği tanımlaması yapacağız. Arka sayfada "MainPage1.xaml.cs" içindeki kodlardan bir kısmı gösterilmiştir. İşaretlenmiş kod satırlarına dikkat ederek web servisinin nerede ve nasıl tanımlandığını görebiliriz.

```
namespace SilverlightApplication1
{
    public partial class MainPage : UserControl
    {
       ServiceReference1.WebService1SoapClient webservis = new
       ServiceReference1.WebService1SoapClient();
        public MainPage()
        {
          InitializeComponent();
          webservis.giriskontrolCompleted += new
EventHandler<ServiceReference1.giriskontrolCompletedEventArgs>(webservis_g
iriskontrolCompleted);
        }
```

- İstemci üzerinde çalışan bir XAML tabanlı uygulama geliştirme platformu uygulaması, sunucudaki web servisi üzerinde tanımlanmış bir metodu çağırdığında, söz konusu metod, sunucu üzerinde çalışıp bittiği anda, istemci tarafında bu metoda bağlı olarak Completed olayı tetiklenir. Ayrıca, sunucu üzerinde çalıştırılan metod, geriye bir değer döndürmüşse, bu değeri çağrıldığı yere değil istemci üzerinde tanımlanmış bulunan Completed olayına parametre olarak gönderir. Completed olayına bağlı olarak çalışacak olan metodu, programcının oluşturması gereklidir.

İşte bu adımda, "MainPage.xaml.cs" içerisinde, sunucudaki "giriskontrol()" metodunun Completed olayına bağlı olarak çalışacak olan metodu yazacağız. Bunun için "MainPage.xaml.cs" dosyasında, "MainPage()" kurucu metodu içerisinde, Completed olayına bir metod ismini atayacağız. Bu işlem, bir butonun Click olayına bir metodu bağlamaya benzer. Aşağıda gösterilen ve "MainPage.xaml.cs" dosyasındaki kodların bir kısmını içeren program kodlarında, işaretlenmiş satırlara dikkat edelim.

```
namespace SilverlightApplication1
{
    public partial class MainPage : UserControl
    {
ServiceReference1.WebService1SoapClient webservis = new
ServiceReference1.WebService1SoapClient();
        public MainPage()
        {
          InitializeComponent();
          webservis.giriskontrolCompleted += new
EventHandler<ServiceReference1.giriskontrolCompletedEventArgs>
(webservis_giriskontrolCompleted);
        }
```

"webservis" nesnesi, web servisi üzerindeki "giriskontrol()" metoduna ait "giriskontrolCompleted" olayını içermektedir. İşte yukarıda işaretli olan kodlar, bu olaya "webservis_giriskontrolCompleted()" metodunu bağlamaktadır.

- "webservis_giriskontrolCompleted" metodunu oluşturacağız.

Aşağıda işaretlenmiş durumda bulunan kodlara dikkat edelim.

```
namespace SilverlightApplication1
{
    public partial class MainPage : UserControl
    {
        ServiceReference1.WebService1SoapClient webservis = new
ServiceReference1.WebService1SoapClient();
        public MainPage()
        {
            InitializeComponent();
            webservis.giriskontrolCompleted += new
EventHandler<ServiceReference1.giriskontrolCompletedEventArgs>(webservis_
giriskontrolCompleted);
        }

        void webservis_giriskontrolCompleted(object sender,
ServiceReference1.giriskontrolCompletedEventArgs e)
        {
            bool sonuc = e.Result;
            if (sonuc == true)
            {
                MessageBox.Show("Giriş başarılı.");
            }
            else
            {
                MessageBox.Show("Giriş başarısız.");
            }
        }
```

Metod başlığında gözümüze çarpan "e" parametresi, sunucudaki "giriskontrol()" metodunun, bu metoda döndürdüğü değeri tutar.

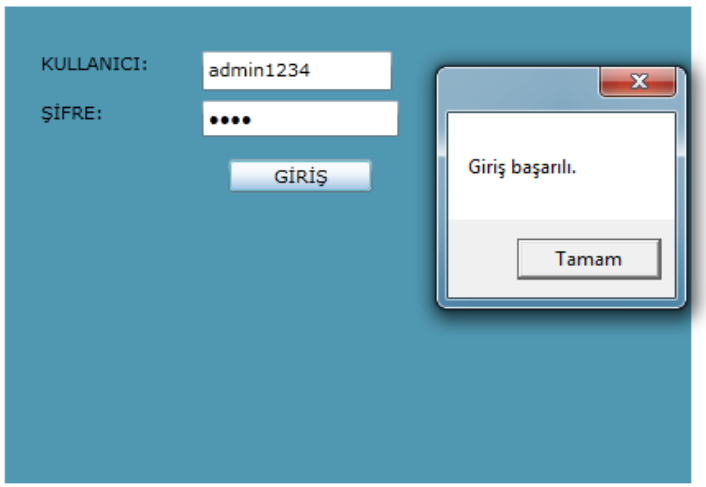
Metod içine yazılan kodlar, "e" değişkeninin içerdiği değeri kontrol etmektedir. Bu değişkenin değerine göre mesaj penceresi görüntülenmektedir.

- Son olarak, sayfa tasarımında kullandığımız "GİRİŞ" düğmesinin Click metodu içerisinde, web servisi üzerindeki "giriskontrol()" metodunu çağıran kodları yazacağız. Arka sayfada, farklı renkle vurgulanmış kodlara dikkat edelim.

```
namespace SilverlightApplication1
{
    public partial class MainPage : UserControl
    {
        ServiceReference1.WebService1SoapClient webservis = new
ServiceReference1.WebService1SoapClient();
        public MainPage()
        {
            InitializeComponent();
            webservis.giriskontrolCompleted += new
EventHandler<ServiceReference1.giriskontrolCompletedEventArgs>(webservis_g
iriskontrolCompleted);
        }

        void webservis_giriskontrolCompleted(object sender,
ServiceReference1.giriskontrolCompletedEventArgs e)
        {
            bool sonuc = e.Result;
            if (sonuc == true)
            {
                MessageBox.Show("Giriş başarılı.");
            }
            else
            {
                MessageBox.Show("Giriş başarısız.");
            }
        }

        private void button1_Click(object sender, RoutedEventArgs e)
        {
            string user = textBox1.Text;
            string pass = passwordBox1.Password;
            webservis.giriskontrolAsync(user, pass);

        }


    }
}
```

Görüldüğü gibi, kodların yukarı kısmında tanımlanan “webservis” nesnesinin bir üyesi olan “giriskontrolAsync()” metodu çağrılarak “user” ve “pass” değişkenleri bu metoda parametre değeri olarak gönderilmiştir. Bu sayede, sunucu üzerindeki “giriskontrol()” metodu, istemci üzerinden çağırılmış olur. Daha öncede belirttiğimiz gibi sunucudaki “giriskontrol( )” metodu, true ya da false değeri gönderecektir.

Uygulama çalıştırıldığında oluşan ekran görüntüsü ise aşağıdaki gibi olacaktır.



Resim 2.14: Sayfanın çalışma anındaki görüntüsü

UYGULAMA FAALİYETİ

Bu uygulama faaliyetinde, XAML tabanlı uygulama geliştirme platformu uygulamaları üzerinden HTML elemanları üzerinde değişiklikler yapmayı öğreneceksiniz.

İşlem Basamakları	Öneriler
➢ Uygulama geliştirme editöründe yeni bir XAML tabanlı uygulama geliştirme platformu projesi oluşturunuz.	➢ File→New Project→Silverlight Application seçilmelidir.
➢ HTML erişimine ilişkin sınıfları içeren kütüphaneyi, kod dosyasına dâhil ediniz.	➢ HTML erişimine ilişkin sınıfları içeren kütüphaneyi, kod dosyasına üst tarafına, aşağıdaki gibi dâhil etmemiz gerekiyor. ➢ using System.Windows.Browser;
➢ MainPage sayfası üzerine bir düğme yerleştiriniz.	➢ ToolBox ta bulunan Button kontrolünü kullanınız.
➢ Projenin html sayfasını başlangıç sayfası yapınız.	➢ Html sayfası üzerinde sağ tuş menüsünü açarak, "Set As Start Page" seçeneğini tıklayınız.
➢ Projenin html sayfasının kodlarını görüntüleyiniz.	➢ "Solution Explorer" üzerinde dosyaya çift tıklayınız.
➢ <object></object> bloğunun altında bir div etiketi oluşturunuz.	➢ ** DÜĞMEYE TIKLAYINIZ ** yazınız.
➢ MainPage üzerindeki düğmenin Click olayına giriniz.	➢ Düğme üzerinde çift tıklayınız.
➢ Yan tarafta bulunan kodları yazınız.	HtmlDocument sayfa = HtmlPage.Document; HtmlElement katman = sayfa.GetElementById("kutu"); ➢ katman.SetProperty("innerHTML", "KATMAN İÇERİĞİ DEĞİŞTİ");
➢ Projeyi derleyip, çalıştırınız.	➢ F5 tuşunu kullanabilirsiniz.
➢ Sayfa üzerinde bulunan düğmeye tıklayarak sonucu gözlemleyiniz.	

ÖLÇME VE DEĞERLENDİRME

Aşağıdaki soruları dikkatlice okuyarak doğru seçeneği işaretleyiniz.

1. HtmlDocument sınıfı metodlarından hangisi, html sayfasından istenilen bir html nesnesini elde eder?

A) SetAttribute()
B) getElementById()
C) CreateElement()
D) AppendChild()

2. HtmlDocument sınıfı metodlarından hangisi, sayfa üzerinde yeni bir html elemanı oluşturmak için kullanılır?

A) SetAttribute()
B) getElementById()
C) CreateElement()
D) AppendChild()

3. HtmlElement sınıfı metodlarından hangisi, sayfa üzerinde bulunan bir html nesnesinin CSS özelliklerini değiştirmeye yarar?

A) SetAttribute()
B) SetStyleAttribute()
C) CreateElement()
D) AppendChild()

4. Aşağıda verilenlerden hangisi uygulama performansını artırıcı yönde etki **yapmaz**?

A) Animasyon tasarımlarında StoryBoard kullanmak
B) Metin animasyonlarını yaparken vektörel metinler kullanmak
C) Uygulamalarda Thread kullanmak
D) Şeffaf arka plan renkleri kullanmak

5. Program kodları içerisinde durak noktaları oluşturmak için Debug menüsündeki seçeneklerden hangisi kullanılır?

A) Toggle Break Point
B) Step Into
C) Step Over
D) Attach Process

Aşağıdaki cümleleri dikkatlice okuyarak boş bırakılan yerlere doğru sözcüğü yazınız.

6. Debug menüsü içerisindeki Step Into komutunun kısayol tuşu……..dir.

7. Debug menüsü içindeki Step Over komutunun kısayol tuşu…….dur.

8. Uygulama geliştirme editöründe, hata ayıklama işlemleri esnasında, değişkenlerin aldıkları değerler……panelinde görüntülenir.

DEĞERLENDİRME

Cevaplarınızı cevap anahtarıyla karşılaştırınız. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt ettiğiniz sorularla ilgili konuları faaliyete geri dönerek tekrarlayınız. Cevaplarınızın tümü doğru ise "Modül Değerlendirme"ye geçiniz.

MODÜL DEĞERLENDİRME

Aşağıdaki cümleleri dikkatlice okuyarak boş bırakılan yerlere doğru sözcüğü yazınız.

1. MeidaElement kontrolünün ses seviyesi değerini ayarlayan özelliği……dir.

2. MediaElement kontrolünde, sesin açık olup olmayacağını ayarlamak için……özelliği kullanılır.

3. ……uygulama çalıştığı anda tetiklenerek çalışan olaydır.

4. Bir thread oluşturmak için……..bildirisini kullanarak Thread sınıfından bir nesne oluşturmamız gerekir.

5. BackgroundWorker nesnesinde yürütülen iş parçası sonlandığında…….....olayı tetiklenir.

6. Proje içerisindeki bütün sayfalarda geçerli olacak olan kontrol şablonlarını tasarlamak için, "Create ControlTemplate Resource" penceresinde bulunan "Define in" bölümündeki……..seçeneği kullanılır.

7. Bir şablonu, bir kontrole uygulamak için o kontrolün…….özelliği kullanılmalıdır.

8. Uygulamadaki bütün sayfalarda geçerli olan şablon tanımlamaları yapmak için

9. şablona ait XAML kodlarının……. sayfası içinde yazılması gerekmektedir.

10. Sayfa içerisindeki HTML kontrollerinin tamamını temsil eden sınıf……..sınıfıdır.

11. XAML tabanlı uygulama geliştirme platformu uygulamalarının sunucu ile etkileşimini sağlamak için, sunucuda hizmet verecek olan……..denilen hizmet sınıfları oluşturmamız gerekir.

12. ……..bildirisi, yazdığımız metodların uzak web istemcileri tarafından çağrılabilmesini sağlar.

13. İstemci üzerinde bulunan bir XAML tabanlı uygulama geliştirme platformu uygulaması, sunucuda bulunan web servisi üzerindeki bir metodu çağırdığında, söz konusu metod sunucu üzerinde çalışıp bittiği anda, istemci tarafında, bu metoda bağlı olarak olayı tetiklenir.

DEĞERLENDİRME

Cevaplarınızı cevap anahtarıyla karşılaştırınız. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt ettiğiniz sorularla ilgili konuları faaliyete geri dönerek tekrarlayınız. Cevaplarınızın tümü doğru ise bir sonraki modüle geçmek için öğretmeninize başvurunuz.

CEVAP ANAHTARLARI

ÖĞRENME FAALİYETİ-1'İN CEVAP ANAHTARI

1	**C**
2	**B**
3	**A**
4	**C**
5	**A**
6	**D**
7	**VisualStateManager**
8	**StoryBoard**
9	**GoToState**
10	**Default Transition**

ÖĞRENME FAALİYETİ-2'NİN CEVAP ANAHTARI

1	**B**
2	**C**
3	**B**
4	**D**
5	**A**
6	**F11**
7	**F10**
8	**Locals**

MODÜL DEĞERLENDİRME'NİN SORULARI CEVAP ANAHTARI

1	**Volume**
2	**IsMuted**
3	**Application.Start**
4	**new**
5	**RunWorkerCompleted**
6	**Application**
7	**Template**
8	**App.xaml**
9	**HtmlElement**
10	**Web service**
11	**[WebMethod]**
12	**Completed**

KAYNAKÇA

- MSDN Yardım Dokümanları
- http:// www.daron.yondem.com/tr/ (03.19.2013)