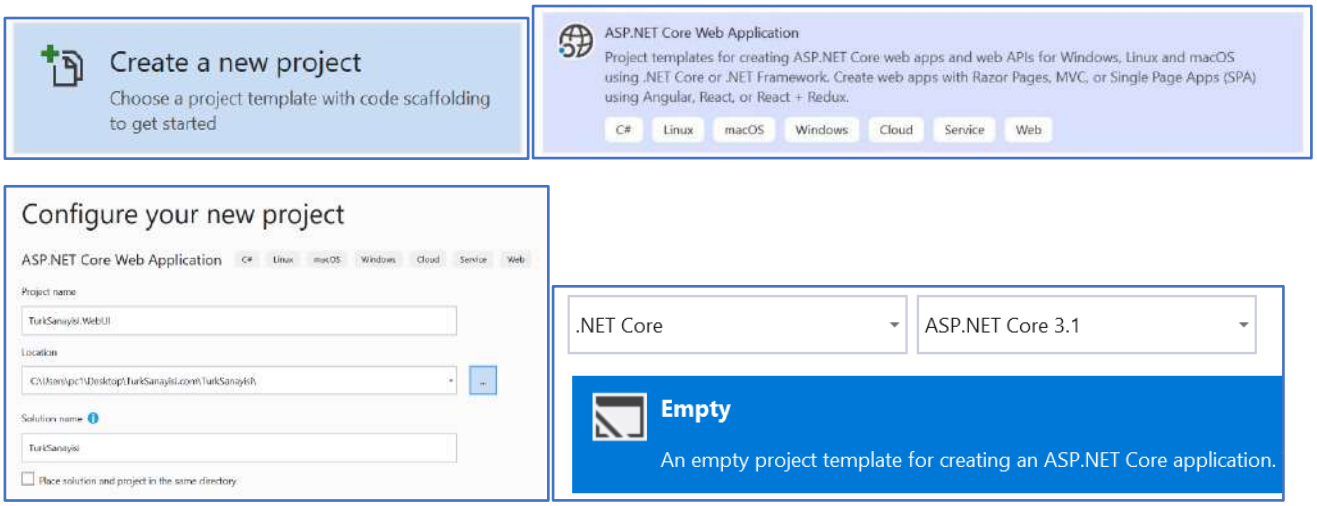
ASP.NET CORE-MVC

İçindekiler

ASP.NET CORE TEMEL ÖZELLİKLER

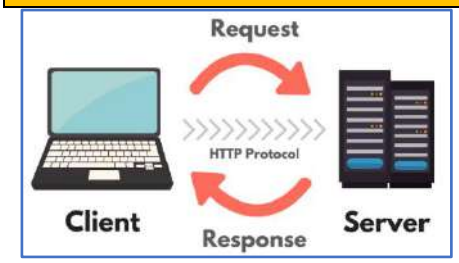
Proje Oluşturma

Normal Visual Studio yu açıyoruz. (VS-Code olanı değil). Aşağıdaki isimdeki proje tipini seçerek oluşturuyoruz.



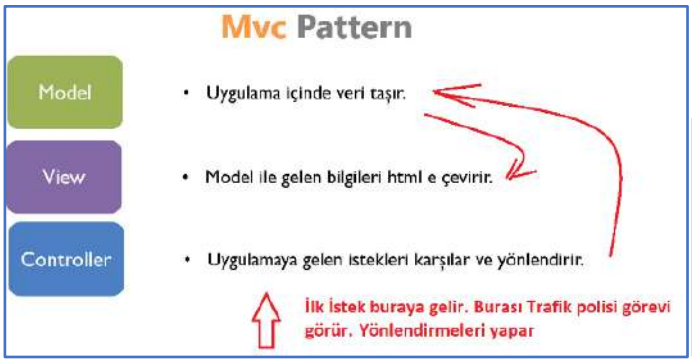
Not: Solution projeleri içinde barındıran üst klasördür. İçerisinde çok sayıda proje oluşturulabilir. Aşağıdaki kodlar belli bir yere kadar Solution klasörü oluşturulmadan yapılmıştır. Belli bir yerden sonra solution üst klasörüne ihtiyaç oldu. Solution>Proje yapısı şekilde olacak. TS>TS.WebUI

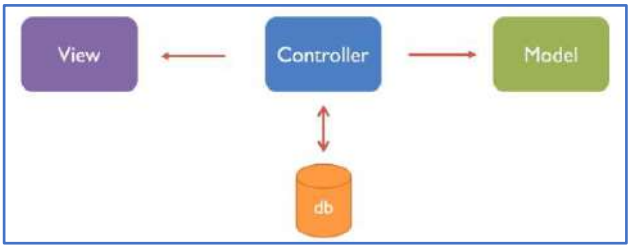
Projenin Çalıştırılması



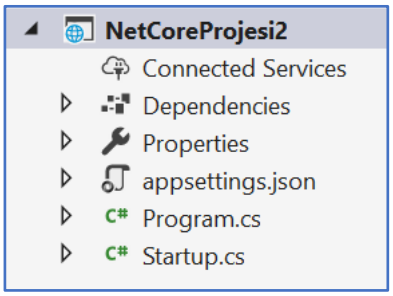
539. Mvc Pattern

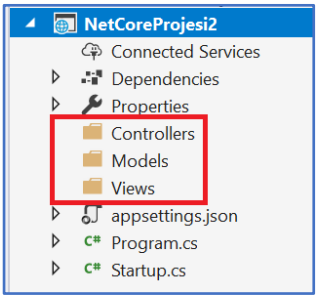
Mvc dediğimiz kavram aslında web sitesini oluştururken kullandığımız tasarım alt yapısının desen yapısını ifade etmektedir. Bir NetFramework yada Core gibi bir platform vs değildir. Bu MVC yapısı Php, Python ve başka diller içinde de web tasarım deseni olarak kullanılmaktadır.





Visual Studio (VS) da Projemizin üzerine sağ tuşa tıklayarak bu üç tane desen için gerekli olan Klasörlerimizi ekleyelim.





(Controllers)-Link ve Kontroller sayfalarının oluşturulması

Klasörlerimizi oluşturduktan sonra öncelikle **Startup.cs** dosyası içinde url şemamızı oluşturmamız gerekiyor. Bunun için önce ConfigureServices içerisine aşağıdaki gibi "Services.AddContrllersWithViews()" metodunu ekleyelim. {Bu metod Core platformu içinde bulunan Mvc deseni içinde vardır. RazorPages (diğer bir alternatif Core desen türü) içinde bulunmamaktadır. RazorPages ler desenini kullanan yapıda Controller kavramı yok.}

Startup.cs

```
public void ConfigureServices(IServiceCollection services)
{
      services.AddControllersWithViews();
}
```

Startup.cs dosyasını içerisinde aşağıdaki gibi bir kod bulunmaktadır. Bu kod ile gelen istek adresinde alt bir desen yoksa, yani sitenin sadece kendi domain adı varsa ("/" burayı ifade etmiş oluyor) bu durumda bekleyen içerik içerisine "Hello World" yazılacağı anlatılmış oluyor. Eğer F5 tıklayıp projeyi çalıştırırsak Tarayıcıda bu yazıyı görebiliriz.

Startup.cs

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapGet("/", async context =>
    {
        await context.Response.WriteAsync("Hello World!");
    });
});
```

Bir web adresinin devamında aşağıdaki gibi alt link şemaları bulunabilir. Bu şemaları nasıl oluşturacağımıza şimdi bakalım.

localhost:5001/home/index

localhost:5001/product/list

Öncelikle yukarıda ismini verdiğimiz UseEndpoints motodunun içini boşaltalım. Ve buraya aşağıdaki gibi bir link şemasının formatını yazalım.

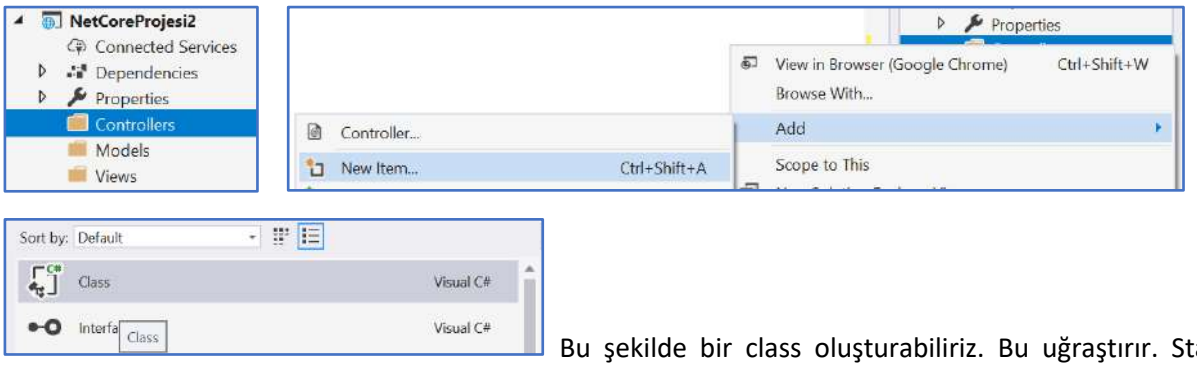
Startup.cs

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name:"default",
        pattern: "{controller}/{action}/"
        );
});
```

Buraya yazdığımız adres deseni aslında şu şekilde bir adresin desenini gösterir.

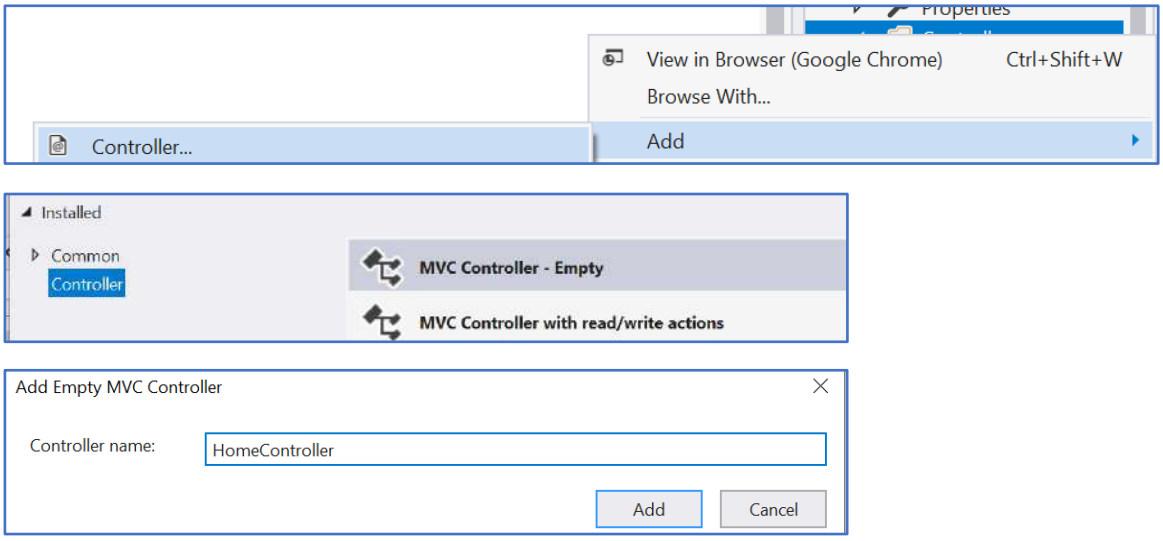
//localhost:5000/product/list

Ardından yukarıda oluşturduğumu "Controllers" klasörü içerisine buradaki desene uygun bir class şeklinde bir Controller sayfası ekleyelim. Controller sayfasını normal class şeklinde eklersek formatı düzenlemek için uğraşırız. Örneğin aşağıdaki gibi yaparsak.



Bu şekilde bir class oluşturabiliriz. Bu uğraştırır. Standart class ekleme yöntemidir. Böyle oluşturursak içeriğin aşağıdaki gelecek şekilde düzenlememiz gerekecektir.

Onun yerine daha controller yapısına uygun hazır bir class formatı eklemek için açılan penceredeki "Controler.." seçeneğini kullanalım. Çıkan ekranda Class ismi verilirken "HomeController" adını verelim.




```
using System;
```

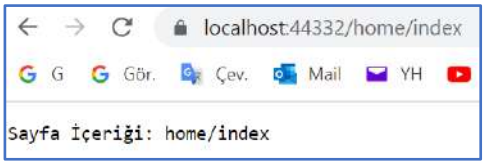
```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;

namespace NetCoreProjesi2.Controllers
{
    public class HomeController : Controller //bizim HomeController classımız, Core geliştiricilerin yazdığı Controller özelliklerine sahip olacaktır. Fakat bu nesnenin çalışabilmesi için yukarıda sarı ile gösterilen .MvC şeklinde olan kütüphanenin eklenmiş olmamız gerekir. Buradaki uygulamada VS bu satırı zaten ekledi.
    {

        //Aşağıdaki metod Views klasörü içinde sayfalar varken çalışır. Oraya yönlendirme yapmak içindir.
        public IActionResult Index()
        {
            return View();
        }
    }
}
```

Yukarıda bir controller için class ı nasıl ekleyeceğimizi gördük. Her controller classı aslında link adres şemasındaki ilk ifadeyi gösterir (örneğin: localhost:5001/home) . Bir alt ifade daha içerir ise bu controller içerisine Action metodu yazmamız gerekir. Bu metodlar ile linklerimiz aslında iki şemadan oluşur. Örneğin:

localhost:5001/home/index yada

localhost:5001/home/about gibi.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;

namespace NetCoreProjesi2.Controllers
{
    //Aşağıdaki kontroller kısmı adres satırına yazılan
    //localhost: 44345/home
    //kısmına kadar olan yer için yazıldı. Eğer iç kısmında daha bir alt adres varsa alt metodlarda yazılmalıdır.
    public class HomeController : Controller
    {
        //Burada yazılan metod ise adresin alt aksiyonu içindir. Yani
        //localhost: 44345/home/index
        //gibi bir adres için yazılmış oldu. Controller altına yazılan metodlara mcv de action metodu denir.

        //Dikkat aşağıdaki metod daha önceki kodlarda geçen IactionResult metodundan farklı. Sebebi bu metodlar View
        //içindeki sayfalara yönlendirme yapmıyor. Sadece linke ait sayfayı açıyor ve içinde aşağıdaki gibi bir yazı
        //çıkarıyor.
        public string Index()
        {
            return "Sayfa İçeriği: home/index";
        }

        //Bir metod daha ekleyelim. Örneğin Hakkımızda sayfası için about linki oluşturalım.
        public string about()
        {
            return " Sayfa İçeriği: home/about";
        }

    }
}
```

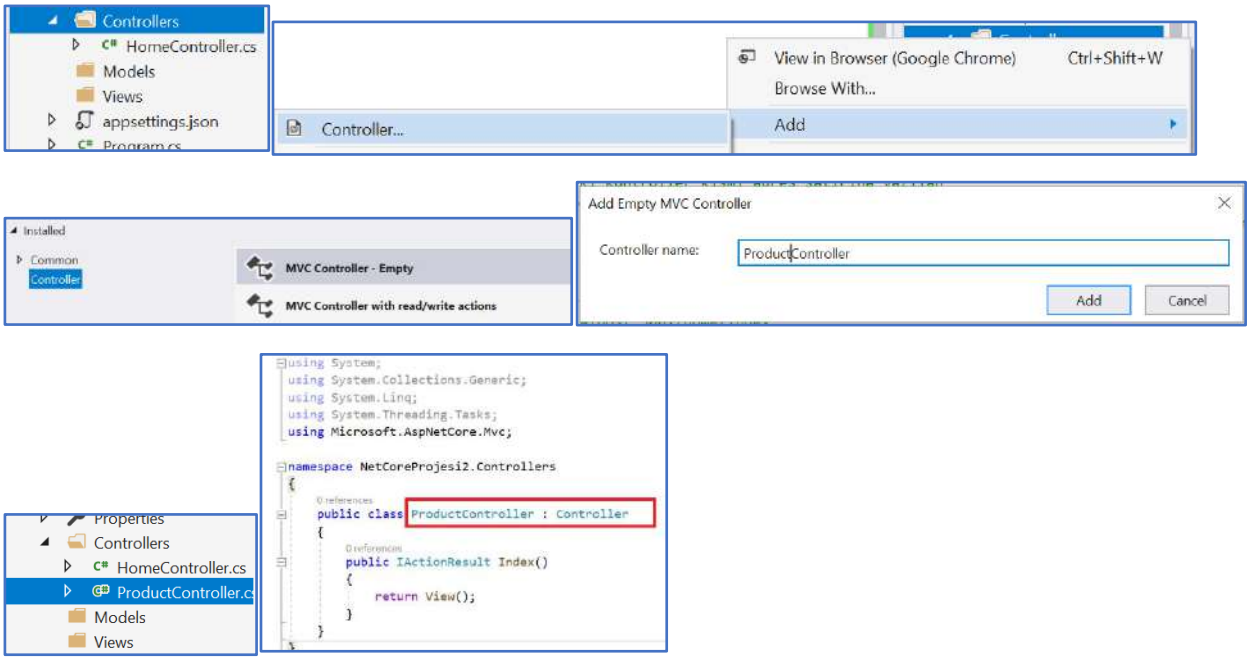


Bir kontroller daha ekleyelim. Bu seferki controller ımız “Product” olsun. Altında iki adet Action linkimiz olsun. Yani çalışacak linklerimiz şu şekilde olsun.

localhost:5001/product/list

localhost:5001/product/details

ProductController eklemek için Controllers klasörü üzerine sağ tuşa tıklayalım ve aşağıdaki ekranlardan adımları takip edelim.



ProductController mız oluştu. İçerisi boş şimdi içerisini dolduralım.

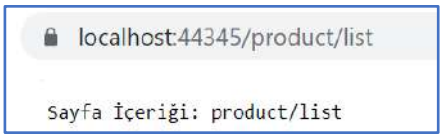
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;

namespace NetCoreProjesi2.Controllers
{
    public class ProductController : Controller
    {
        //localhost:44345/product/list linki için metod.
        public string list()
        {
            return "Sayfa Adresi: product/list";
        }

        //localhost:44345/product/details linki için metod.
        public string details()
        {
            return "Sayfa Adresi: product/details";
        }

    }
}
```

Şimdi linkler için hazırladığımız controller ları test edelim. F5 tıklayıp siteyi çalıştıralım.

Direk link çalışmadı. Çünkü Controller ı yoktur.

Bu linkte çalışmadı.

Bu link çalıştı. Controller ı vardı.



Bu link de çalıştı. Controller ı var.

Not: Linkin devamında bir üçüncü bir ek daha olsaydı, örneğin listelenecek ürünün bir Id si gibi sayı. Bu durumda yine çalışmazdı. Eğer bu şekilde bir linki çalışır hale getirmek istiyorsak "Startup.cs" dosyasının içerisinde linklerin şemasını ona uygun hale getirmemiz gerekir.


```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name:"default",
        pattern: "{controller}/{action}/{id}"
        );
});
```

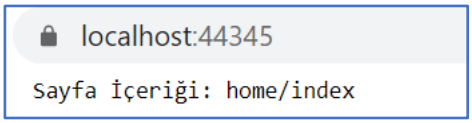
Bu yeni durumda bu link çalışmadı. Çünkü beklediği üçüncü alt adres yoktur.

Üçüncü adres bilgisini de ekleyince çalıştı.

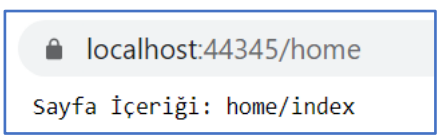
Default Routing-Varsayılan Link Desenini Düzenleme

Yukarıdaki adresleri denerken …/home ve sadece …/product yazdığımızda adresler çalışmamıştı. Böyle durumlarda yani hem controller adres kısmı yazılmaza (1. Seviye link), yada Action adres kısmı yazılmazsa (2. Seviye link) varsayılan Controllers yada Action hangisinin çalıştırılacağını "Startup.cs" içinde yazdığımız adres şemasının içerisini aşağıdaki gibi düzenlemeliyiz.

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=home}/{action=index}"
        );
});
```



Önceden çalışmayan bu linkimiz şuanda çalıştı.

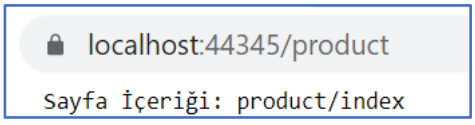


Bu linkimizde çalıştı.

Bu linkimiz çalışmadı. Sebebi Product controller altında varsayılan olarak index (action=2 seviye link) motodunu aramaktadır. Bunu bulamadığı için çalışmadı. O yüzden farklı kontroller altında çalışmasını istiyorsak aynı isimde action (2 seviye link) olarak "index" metodunu eklemeliyiz.

Product controller ın son hali aşağıdaki gibi olur.

```
public class ProductController : Controller
{
    //localhost:5001/product/list linki için metod.
    public string list()
    {
        return "Sayfa İçeriği: product/list";
    }

    //localhost:5001/product/details linki için metod.
    public string details()
    {
        return "Sayfa İçeriği: product/details";
    }
    //sonradan eklendi.
    public string Index()
    {
        return "Sayfa İçeriği: product/index";
    }
}
```



Bu sefer bu link çalıştı. Dikkat edilirse home altındaki index değildir burası. Aynı isimde product altına koydumuz index tir. Eğer sadece product/ şeklinde link yazılırsa hata sayfası çıkmasın diye onunda altına index metodu koymak (action seviyesi) gereklidir.

Not: Özetlersek adres satırını boş bulursa "/" gibi o zaman default olarak "/home" controller ı çalıştıracak. "/???/ " herhangi bir kontroller yazıldı fakat sonrasında Action metod kısmı boş bırakılmışsa varsayılan olarak hangi controller çalıştırılıyorsa altında "index" metodunu arayacaktır. Bu durumda bu metod tüm controller altında bu isimde olması gerekecektir. Tabi ki link boş iken de çalıştırılması isteniyorsa.

Linke Id bilgisi ekleme

Şimdi yapacağımız uygulamada ise adres satırında ikinci seviye olan action dan sonra gelen üçüncü seviye Id numarasının denemesini yapalım. Bunun için öncelikle "startup.cs" dosyasının içindeki link şemasını düzenlemeliyiz. Buraya "/{id?}" ifadesini ekleyelim. Soru işareti "istenirse konulabilir" anlamındadır. Koymazsak 3.seviye bütün linklerde arayacaktır.

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=home}/{action=index}/{id?}"
        );
});
```

Bu eklemeyi yaptıktan sonra hangi Action metodu içinde Id kullanacaksak orayı düzenleyelim. Örneğin ürünler bilgisi altında detay bilgisi getirilirken 5 nolu ürünün bilgisini getirmek istiyorsak adres satırını "products/details/5" bu şekilde yazarız. O zaman Id bilgisini details metodu içinde kullanalım. Bu metodu şu şekilde düzenleyelim.

```
public string details(int Id)
```

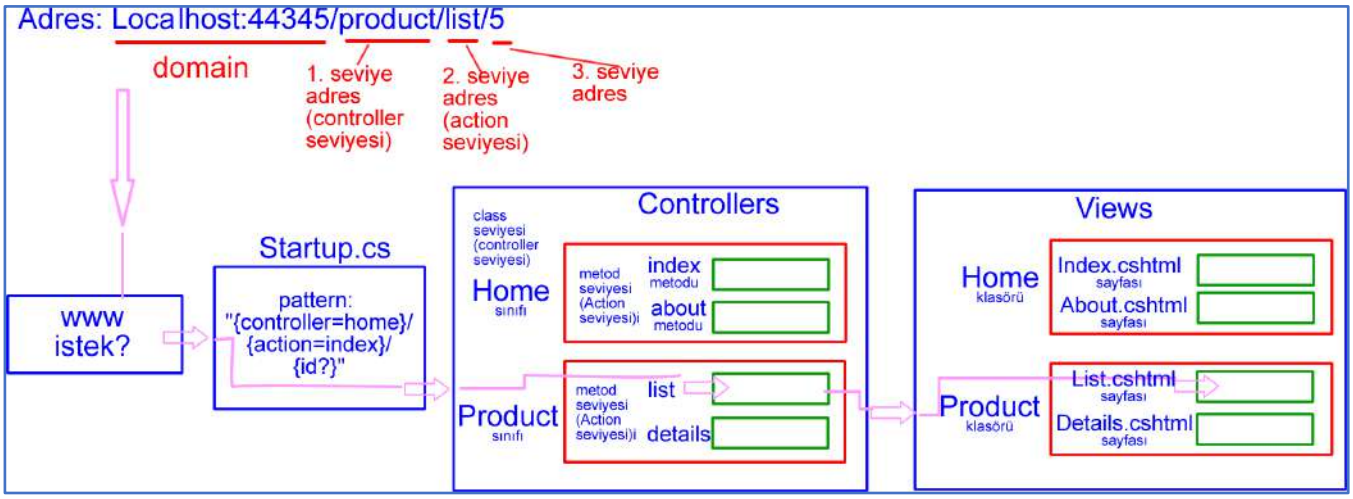
```
{
    return "Sayfa İçeriği: product/details/" + Id;
}
```

bu link çalıştı.

(Views)- Views Klasörü İçerisinde İçerik Sayfalarını Oluşturma

Şu ana kadar yaptığımız, Controller yapısı ile yani siteye gelen isteklerin içeriğine göre hangi sayfalara yönlendirileceğine dair kodlamayı gördük. Aslında yaptığımız iş sayfalara yönlendirmeden daha çok sayfa yerine bir string ifadeyi görüntülemiştik. Oysa esas yapmamız gereken bir html sayfasına yönlendirip orada görüntüleme yapmaktır.

Açılacak olan sayfalarımız Views klasörü içerisinde bulunması gerekiyor. Bu klasörün içerisinde oluşturulan klasörler , Controller katmanına karşılık gelir (adres linkinde 1 seviyeyi gösterir). Views/klasör yapısının içerisinde html sayfalarımız olmalıdır. Bunlar Action katmanına karşılık gelir (yani adres satırında 2. Seviyeyi gösterir). Bununda ötesinde ise 3. Seviyede sayfalara gönderilen parametreler olacaktır. Bu yapıyı şematik olarak şu şekilde oluşturabiliriz.



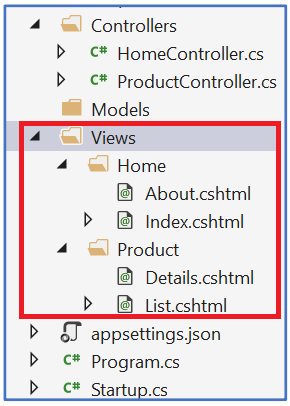
Bu şemayı gerçekleştirmek için Controllers class larımız şu şekilde olacaktır. Burada action metodları View() döndüğü için metodun tipi IActionResult olarak değişti. Artık yukarıdaki denemelerde yapıldığı şekilde string döndermiyor sayfa dönderiyor.

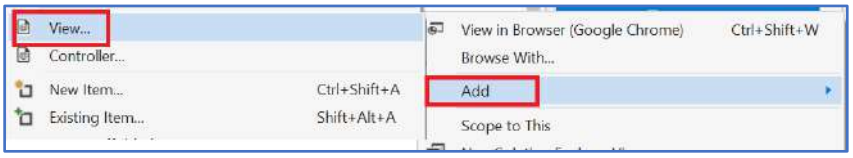
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;

namespace NetCoreProjesi2.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }

        public IActionResult about()
        {
            return View();
        }

    }
```

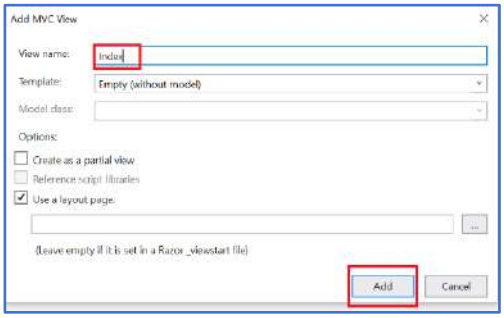
```
}
```

Burada dönderilen View ler için View klasörünün içerisine bakacaktır. Aynen bu klasörün içerisinde de Home ve Product yapıları olması gerekir. Bunlar klasör olarak gözükmelidir. Bunların içerisinde ise Metodlara (action) karşılık gelen sayfalarımız bulunmalıdır. Yapısı aşağıda şekilde görüldüğü gibidir.

Burada Views içinde .cshtml sayfalarını oluşturmak için ekranlar aşağıdaki şekildedir. Direk herhangi bir html dosyası oluşturulup uzantısını değiştirmek çözüm olmaz. Bu şekilde proje içerisine kendi formatında eklenmelidir. Aşağıdaki örnekte Home klasörü üzerine sağ tuşa tıklandığında oluşturulan Index.cshtml dosyasının yolu gösterilmiştir.







Burada oluşturulan Index.cshtml sayfalarının içeriği de varsayılan olarak aşağıdaki şekilde oluşturuldu. Bu sayfa istendiğinde aşağıdaki gibi bir görüntü vermiş oldu.

Farklı İsimde Bir Sayfaya Yönlendirme

Buraya kadar olan kodları incelediğimizde Home altında Index, About sayfalarını ve benzer şekilde Product altında List ve Details gibi sayfaları hep sınıf ve metodların isimlerinden otomatik olarak gördü. Örneğin controller (`public class HomeController : Controller`) ifadesinden Views içindeki Home klasörüne gideceğini ve (`public IActionResult Index()`) ifadesinden ise oradaki Index.cshtml sayfasını açacağını anladı.

Peki böyle bir durumda biz başka bir sayfayı açmasını isteseydik nasıl yapacaktık? Örneğin aşağıdaki Home controller içerisinde Contact metodunu kullandığımızda normalde Contact.cshtml sayfasını arayacaktır fakat bunun yerine Türkçe Iletisim.cshtml gibi bir sayfayı açmasını isteyelim. Bu durumda Home kontroller içindeki Contact metodunu içerisi şöyle yazılır.

HomeController

```
public IActionResult contact()
```

```
{
    return View("Iletisim");
}
```

Bilgi Taşıma Yöntemleri

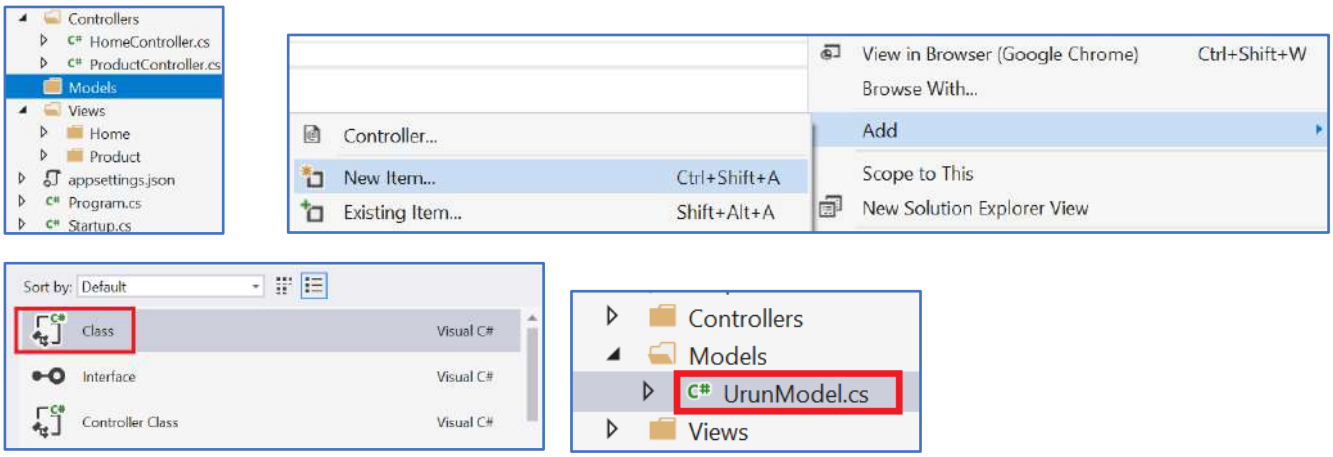
Controller dan View bilgi aktarırken dört tane yöntemden bahsedebiliriz. Bunlar arasındaki farkları ve kullanıldığı yerleri detaylandıralım.

a) Model
b) ViewData
c) ViewBag
d) ViewModels

A) Model Yapısı İçerisinde Bilgi Taşıma

Örnek 1: Model yapısını oluşturma ve tek bir ürün için bilgi gönderme

Önce bir Model yapısı oluşturalım. Bu amaçla Models klasörü üzerinde sağ tuşa tıkayıp yeni bir C# Class oluşturalım. Bunun içeriğini aşağıdaki şekilde düzenleyelim.

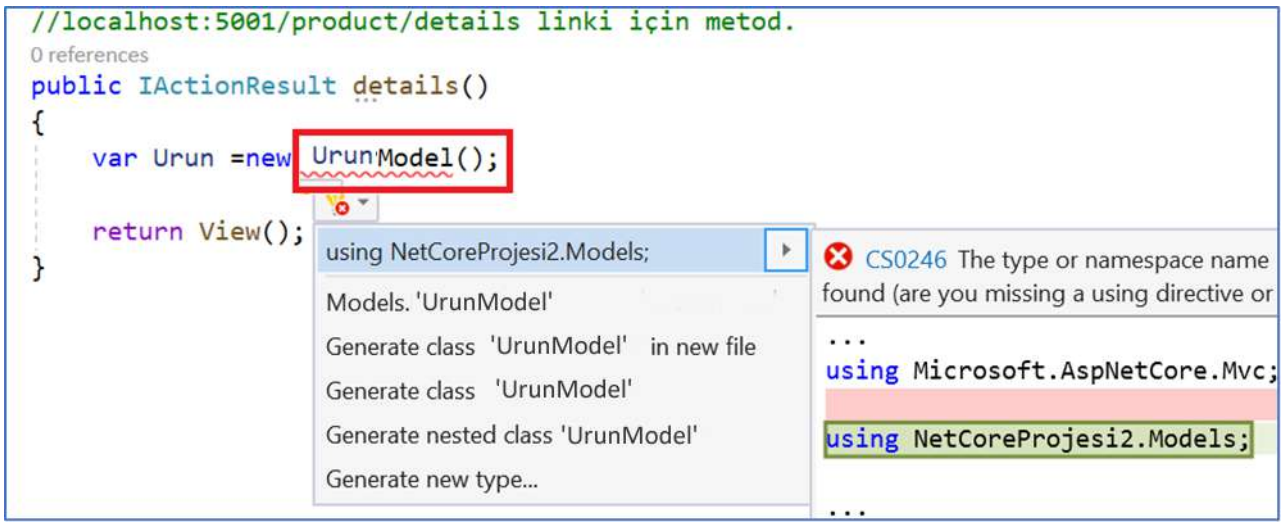



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace NetCoreProjesi2.Models //Dikkat namespace adını bulunduğu klasörden aldı.
{
    public class UrunModel
    {
        public string Ad { get; set; } //Bu yazıyı kısayol şeklinde çıkarmak için prop yazıp iki defa tab tuşuna
tıklayın.
        public double Fiyat { get; set; }
        public string Tanim { get; set; }
    }
}
```

Not: Class ları yazarken şu şekilde görünen (`public int MyProperty { get; set; }`) özellik tanımlama satırlarını otomatik oluşturmak için **prop** yazıp iki kere **tab** tuşuna tıklarsak otomatik olarak formatı kodlar arasına koyacaktır.

Buraya kadar ürünlerimiz için bir model (şablon) oluşturduk. Bu modeli nasıl kullanacağız onu görelim. Bu amaçla Product Controller içerisinde yeni bir Action (ikinci seviye link, yada metod seviyesi diyebiliriz) kodu oluşturalım.

Bu amaçla aşağıdaki gibi details motodunu yazıp içerinde ProductModel() nesnesini oluşturmaya çalıştığımızda görmemektedir. Çünkü bu nesne başka bir Namespace içerisinde bulunmaktadır. Bu namespace adını bu class içerisinde yukarıya eklememiz gerekir. Eklemek için (Ctrl + .) tıklarsak yardım ekranı çıkacaktır ve buradaki using ile başlayan oluşturulduğu namespace ekleyebiliriz. .




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using NetCoreProjesi2.Models;

namespace NetCoreProjesi2.Controllers
{
    public class ProductController : Controller
    {
        //localhost:5001/product/details linki için metod.
        public IActionResult details()
        {
            var Urun =new UrunModel();

            Urun.Ad = "Telefon";
            Urun.Fiyat = 3000;
            Urun.Tanim = "En İyi Yerli Telefon";

            return View(Urun); //Views giderken artık Ürüne ait bütün bilgiler tek seferde taşınmış olacaktır.
        }

    }
}
```

Artık controller içerisinden Views klasörünün ürünümüze ait Model (şablon) şeklinde bilgilerimizi gönderiyoruz. Views içerisinde gideceğimiz sayfa bizi karşılayıp bu bilgileri alması gerekir. Aldığı bu bilgileri ise sayfada görüntüleyebiliriz.

```
@*Burası Details.cshtml sayfasıdır.*@
@using NetCoreProjesi2.Models


<h1>Detay Sayfası</h1>

<p>Ürün Adı: @Model.Ad </p>
```

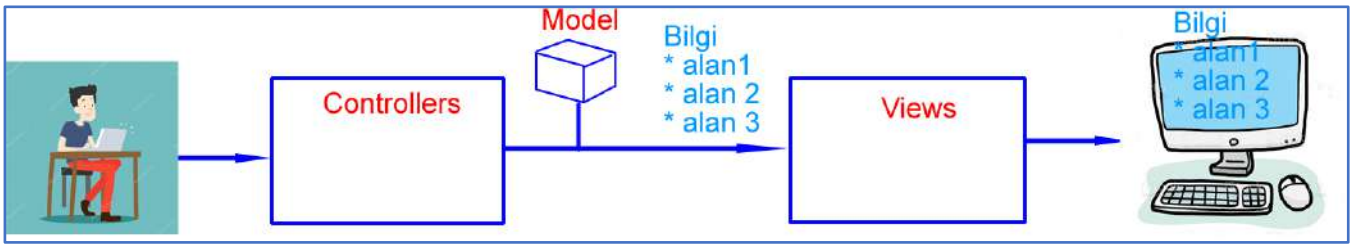
```
<p>Ürün Adı: @Model.Fiyat </p>
<p>Ürün Adı: @Model.Tanim </p>
```

Sayfayı çalıştırdıktan sonra üzerine sağ tuşa tıklayıp kaynak kodlarını inceleyelim. Sayfa kod yapısında model ile alakalı herhangi bir bilginin olmadığını göreceğiz. Kodlar tamamen html koduna çevrilerek görüntülenmiş olmaktadır. Böylelikle bize ait C# kodlamalarına ait gizli kalmasını istediğimiz bilgiler kullanıcıya gönderilmemiş olmaktadır. Dinamik olarak hazırladığımız web sayfası, statik html sayfası olarak kullanıcıya gönderilmiş (gösterilmiş) oluyor. Bu kaynağı inceleyen birisi hangi dil ile geliştirdiğimizi anlayamaz. Burası Php ile de geliştirilmiş olsa, yada Jango (python) platformlarında da üretilmiş olsa göremeyecektir.

Satır kaydır

```
1
2  <h1>Detay Sayfası</h1>
3
4  <p>Ürün Adı: Telefon </p>
5  <p>Ürün Adı: 2500 </p>
6  <p>Ürün Adı: En &#x130;yi Yerli Telefon </p>
7
8
```

Buraya kadar olan kısmın işleyişini şematik olarak gösterecek olursak yukarıdaki şemamız şuna dönüşmüş olmaktadır.



Yukarıda Model class yapısı ile bilgi taşımayı uyguladık. Model in kendisi zaten bir nesne olduğundan bu nesneyi Views e direk gönderebiliriz ve orada alt özelliklerini de nokta (.) koyarak kullanabiliriz. Html sayfasında nesne kullanılırken "@Model.AltOzellik" şeklinde kullanılır.

Burada Model gönderirken kendi içinde bir bütün olduğundan Ayrı bir bilgi olan KategoriAdi nı bunlarla birlikte gönderemeyiz. Bu durumda ViewBag yada ViewData nesnesini kullanabiliriz.

ProductController.cs

```
public class ProductController : Controller
{
    public IActionResult Index()
    {
        var UrunNesnesi = new UrunModel { Ad = "Bilgisayar", Fiyat = 6000, Tanim = "Diz Üstü Bilgisayar" };

        ViewBag.KategoriAdi = "Elektronik Eşyalar";

        return View(UrunNesnesi);
    }
}
```

Product/Index Sayfası

```
@using NetCoreProjesi2.Models

@model UrunModel

<h1>Product/Index Sayfası</h1>

<h2>@ViewBag.KategoriAdi</h2>

<p>@Model.Ad</p>
<p>@Model.Fiyat</p>
<p>@Model.Tanim</p>*@
```

Not: View sayfalarında @model sınıfı tanımlanırken **küçük harf**le başladığına ve bu aşağılarda bu sınıf kullanılırken **büyük harf**le başladığına dikkat edin.

Örnek 2: Model Bilgisini Liste şeklinde oluşturma (çoklu bilgi halinde) ve Sayfaya Taşıma

Şimdi View e bilgiyi gönderirken daha önce oluşturduğumuz Model formatında (içerisinde Ad, Fiyat ve Tanım vardı) çok sayıda bilgiyi gönderelim. Yani bu formatta tek bir ürünü değil, çok sayıda ürünü göndermek istersek ne yapacağız?

Böyle bir durumda ürünleri içinde tutan bir List nesnesi (dizi şeklinde bir yapıdır) kullanabiliriz. List içerisindeki kullanılan yapı ise ürünler için oluşturduğumuz model yapısı olabilir.

Product/List altında bilgileri çıkarmak istersek ProductController içerisinde aşağıdaki kodları yazalım.

ProductController.cs Sayfası

```
public class ProductController : Controller
{
    public IActionResult list()
    {
        var UrunlerNesnesi = new List<UrunModel>()
        {
            new UrunModel{Ad= "GM", Fiyat = 2000,Tanim = "İyi Telefon"},
            new UrunModel{Ad= "Vestel", Fiyat = 1500,Tanim = "Yerli Telefon"}
        };

        return View(UrunlerNesnesi);
    }
}
```

Product/List Sayfası

```
@using NetCoreProjesi2.Models

@model List<UrunModel>;

<h1>List Sayfası</h1>

<div style="background-color:azure;">
    <p>Ürün Adı:@Model[0].Ad</p>
    <p>Ürün Fiyatı: @Model[0].Fiyat </p>
    <p>Ürün Açıklaması:@Model[0].Tanim </p>

</div>

<div style="background-color:bisque;">
    <p>Ürün Adı: @Model[1].Ad</p>
    <p>Ürün Fiyatı: @Model[1].Fiyat</p>
    <p>Ürün Açıklaması: @Model[1].Tanim</p>

</div>
```

B) ViewData İçerisinde Bilgi Taşıma

Öncelikle daha önce oluşturduğumuz UrunModel sınıfından bir nesne türetelim. Bu nesnenin alt üç tane alanı vardı. Bu üç alana bilgileri nesneyi oluştururken atayalım. Bu işlemleri controller içinde yapıyorduk. Bilgiler Product/index sayfası içinde gözüksün. Bunun için nesneyi ve bilgileri ProductController içinde oluşturalım.

ProductController.cs

```
    public class ProductController : Controller
    {
        public IActionResult Index()
        {
            var UrunNesnesi = new UrunModel { Ad = "Bilgisayar", Fiyat = 6000, Tanim = "Diz Üstü Bilgisayar" };

            ViewData["UrunBilgisi"] = UrunNesnesi;

            ViewData["KategoriAdi"] = "Elektronik Eşyalar";

            return View();
        }
    }
```

Product/Index Sayfası

```
<h1>Product/Index Sayfası</h1>

<h2>@ViewData["KategoriAdi"]</h2>
<p>@(((NetCoreProjesi2.Models.UrunModel)ViewData["UrunBilgisi"]).Ad)</p>
<p>@(((NetCoreProjesi2.Models.UrunModel)ViewData["UrunBilgisi"]).Fiyat)</p>
<p>@(((NetCoreProjesi2.Models.UrunModel)ViewData["UrunBilgisi"]).Tanim)</p>
```

Yukarıdaki kodları aşağıdaki şekilde de kullanabiliriz. Namespace.Class yolu sayfanın en üst kısmında tanımlandığında her satırda belirtmeye gerek kalmaz.

Product/Index Sayfası

```
@using NetCoreProjesi2.Models

<h1>Product/Index Sayfası</h1>

<h2>@ViewData["KategoriAdi"]</h2>
<p>@(((UrunModel)ViewData["UrunBilgisi"]).Ad)</p>
<p>@(((UrunModel)ViewData["UrunBilgisi"]).Fiyat)</p>
<p>@(((UrunModel)ViewData["UrunBilgisi"]).Tanim)</p>
```

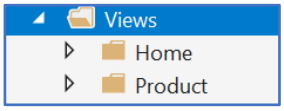
Benzer olarak şu şekilde bir kod gösterim şekli de kullanılabilir (bazı yerleri değiştirmelisiniz!).

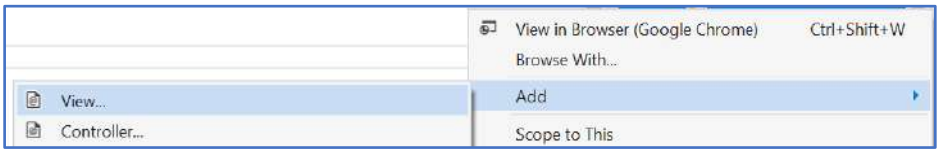
```
@using NetCoreProjesi2.Models

@model ViewData["Urunler"]

@{int i = 0;}

<p> ViewData[]</p>

@foreach (var Urun in Model)
{
    i++;
    <p><b>Ürün No:@i</b></p>

    <p>Ürün Adı:@Urun.Name</p>
    <p>Ürün Fiyatı:@Urun.Price</p>
    <p>Ürün Açıklama:@Urun.Definition</p>
```

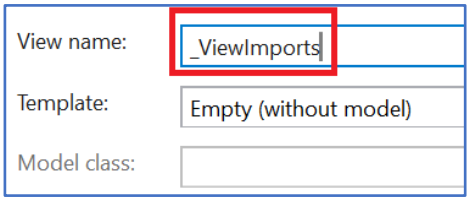
```
}
```

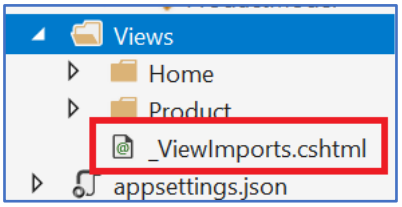
Not: Class Bildirimlerini Tüm Dosyalar için Tek bir yerden atama

Namespace.Class bildirimini eğer Views klasörü içerisindeki tüm dosyalarda tanınmasını istiyorsak Views ilk seviye içerisine _ViewImports.cshtml dosyası olarak ekleyebiliriz (Bu isimde olması zorunlu değil. Önemli olan içerideki klasörlerin üzerinde bir seviyede .cshtml dosyası içinde yazılmış olması. Yinede uluslararası alışılmış ifadeleri kullanmak faydalıdır. Başındaki alt çizgi tam bir sayfa olmadığını, bir sayfa parçası olduğunu gösteren anlamdadır).









_ViewImports.cshtml Dosyası

```
@using NetCoreProjesi2.Models
```

Böylece Views içindeki tüm alt klasörler içinde bulunan sayfalarda Namespace.class bildirimini yazmamıza gerek kalmaz.

C) ViewBag içerisinde bilgi taşıma

Örnek 1: ViewBag içerisinde tek bilgi taşıma

Sayfamızda dinamik bir veriyi göstermek için bir örnek yapalım. Bur örnekte Controller kısmında sistem saatini alalım. Ayrıca yanında Herhangi bir kullanıcı adını oradan alalım. Bu bilgileri View içindeki bir sayfada gösterelim. Bunun için Controller ve View içinde yazacağımız kodlar aşağıdaki şekilde olacaktır.

Home Controller içeriğini aşağıdaki şekilde oluşturabiliriz. Burada oluşturduğumuz değişkenleri View içindeki sayfaya **ViewBag** nesnesi ile taşıdığımıza dikkat ediniz.

```
public class HomeController : Controller
    {
        public IActionResult Index()
        {
            int Saat = DateTime.Now.Hour;

            string Mesaj = null;
            if (Saat < 12)
                Mesaj = "Günaydın";
            else
                Mesaj = "İyi Günler";

            string Kullanici = "Ali Su";

            //Bilgileri Views altına taşımak için ViewBag nesnesini kullanabiliriz.
            ViewBag.SelamlamaIfadesi = Mesaj;
            ViewBag.ZiyaretciAdi = Kullanici;
```

```
            return View();
        }
}
```

Index.cshtml sayfamızın içeriği ise aşağıdaki şekilde yazılmıştır. Burası aslında bir html sayfasıdır ama formatı bu şekildedir. Bu sayfada ViewBag nesnesi içinden gelen değişken bilgilerinin nasıl kullanıldığına dikkat ediniz. Başlarında @ kullanılmıştır.

```
<h1>Index Sayfasi</h1>

<p>@ViewBag.SelamlamaIfadesi @ViewBag.ZiyaretciAdi </p>

<p>Web Sayfama Hoş Geldiniz</p>
```

Örnek 2: Model bilgisini ViewBag içerisinde taşıma

Nesne yapısı içerisindeki aynı bilgileri bu sefer ViewBag içerisinde taşıyalım.

ProductController.cs

```
    public class ProductController : Controller
    {
        public IActionResult Index()
        {
            var UrunNesnesi = new UrunModel { Ad = "Bilgisayar", Fiyat = 6000, Tanim = "Diz Üstü Bilgisayar" };


            ViewBag.UrunBilgisi = UrunNesnesi;
            ViewBag.KategoriAdi = "Elektronik Eşyalar";

            return View();
        }
    }
}
```

Product/Index Sayfası

```
<h1>Product/Index Sayfası</h1>

<h2>@ViewBag.KategoriAdi</h2>


<p>@(((UrunModel)ViewBag.UrunBilgisi).Ad)</p>
<p>@(((UrunModel)ViewBag.UrunBilgisi).Fiyat)</p>
<p>@(((UrunModel)ViewBag.UrunBilgisi).Tanim)</p>
```

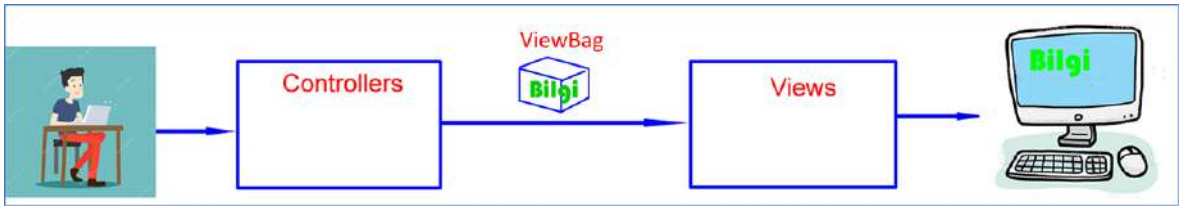
Dikkat edilirse ViewData ile bilgi taşınırken nesne adı özellik şeklinde (tırnaklar içinde) belirtilmekte, ViewBag ile bilgi taşırken ise alt fonksiyon şeklinde (.altfonksiyon gibi) tanımlanmaktadır. Her ikisinde de nesnenin Namespace.Class bildirimi sayfa içerisinde yada üst klasörde bir dosya içinde tanımlanmış olmalıdır.

Ayrıca ViewBag kullanılırken yukarıda olduğu gibi nesne tanımlaması yapılarak kullanılsa bile bu yapılmadan da kullanılabilir. Hatta nesnenin alt özellikleri de daha basit olarak gösterilebilir. Aynı dosyayı şu şekilde oluşturursak yine çalışacaktır.

```
<h1>Product/Index Sayfası</h1>
```

```
@*<h2>@ViewBag.KategoriAdi</h2>

    <p>@(((UrunModel)ViewBag.UrunBilgisi).Ad)</p>
    <p>@(((UrunModel)ViewBag.UrunBilgisi).Fiyat)</p>
    <p>@(((UrunModel)ViewBag.UrunBilgisi).Tanim)</p>*@

<h2>@ViewBag.KategoriAdi</h2>

<p>@ViewBag.UrunBilgisi.Ad</p>
<p>@ViewBag.UrunBilgisi.Fiyat</p>
<p>@ViewBag.UrunBilgisi.Tanim</p>
```

Kullanıcıdan gelen ilk istekleri Controller sınıfları karşılamaktadır. Link içerisinde istenen bilgiye göre Controller kişiyi View içerisindeki klasörlere yönlendirmektedir. Bu esnada eğer Controller, Views içindeki sayfaya giderken yanından bazı bilgileri götürüp oradaki sayfada göstermek isterse bu yanında götürdüğü bilgileri **@ViewBag** nesnesi içerisinde taşımıştı. Bu yapıyı şu şekilde sembolize edebiliriz.



Bu yapıda her bilgi ayrı bir paket halinde (handbag içinde ayrı bir değişken halinde) taşınmıştı. Oysa bilgilerin daha düzgün bir formatta kategorize edilmiş bir şekilde taşınması daha uygun olur. Örneğin bir ürün bilgisinin, adı, fiyatı, özellikleri vs formatlı bir şekilde taşınabilir. Böyle bir durumda Viewbag kullanarak değilde models sınıfı oluşturularak bu sınıf içerisinde taşınması sağlanacaktır.

Örnek 3: List<Model> Bilgisini ViewBag İçerisinde Sayfaya Taşıma

HomeController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc; //Controller alt
yapının çalışması için gerekli.
using WebApplication7.Models;

namespace WebApplication7.Controllers
{
    public class HomeController : Controller
//HomeController sınıfı Controller alt yapısını
kullanacak. Çalışabilmesi için yukarıya using
Microsoft.AspNetCore.Mvc; kütüphanesinin eklenmesi
gerekir.
    {
        public IActionResult Index()
        {
            var Urun = new ProductModel {Name="Kalem",
Price =10, Definition="Kırmızı Kalem" };
            var Urunler = new List<ProductModel>() {
                new ProductModel {Name="Kalem", Price
=10, Definition="Kırmızı Kalem" },
                new ProductModel {Name="Defter", Price
=5, Definition="Kareli Defter" },
                new ProductModel {Name="Kitap", Price
=8, Definition="Necip Fazıl Kısakürek" }
            };

            ViewBag.Urunler = Urunler;
            return View(ViewBag.Urunler);
        }
```

Index.cshtml

```
@using WebApplication7.Models

@model List<ProductModel>

@foreach(var Urun in Model)
{
    <p> @Urun.Name / @Urun.Price / @Urun.Definition
</p>
}
```

```
    }
}
```

Kalem / 10 / Kırmızı Kalem

Defter / 5 / Kareli Defter

Kitap / 8 / Necip Fazıl Kısakürek

Örnek 4: Model Bilgisini ViewBag içerisinde sayfaya taşıma

Bilgilerimizi sayfaya ViewBag içerisinde taşırken daha önceden sadece string bir ifadeyi taşımıştık. Örneğin sayfaya Kategori bilgisini string içerisinde taşımak istiyorsak, controller sayfasında

`ViewBag.KategoriAdi = "TELEFONLAR";` Şeklinde kullanabiliriz. Sayfa içerisinde ise bilgileri göstermek için

`@ViewBag.KategoriAdi` Şeklinde kullanabiliriz.

Controller Sayfası

```
ViewBag.KategoriAdi = "TELEFONLAR";
ViewBag.KategoriAciklama = "Yerli Telefonlar";
```

Listeleme Sayfası

```
<h2>List Sayfası</h2>

<p>@ViewBag.KategoriAdi</p>
<p>@ViewBag.KategoriAciklama</p>
```

Peki burada göndereceğimiz Kategori bilgilerini sınıf şeklinde bir Model nesnesi altında gönderirsek ve aynı gönderme işlemi içinde yukarıda yaptığımız ürünlere ait bilgileri de göndermek istersek, her iki Model tanımlamasını **return View();** içerisinde nasıl göndereceğiz buna bakalım.

Öncelikle Kategori için bir Model sınıfı oluşturalım. Model klasörü üzerinde sağ tuşa tıklayıp yeni bir Class dosyası ekleyelim. Ve içerisine aşağıdaki kodları yazalım.

KategoriModel.cs dosyası

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace NetCoreProjesi2.Models
{
    public class KategoriModel
    {
        public string Ad { get; set; }
        public string Aciklama { get; set; }
    }
}
```

Controller sayfasındaki bilgileri

```
var KategoriNesnesi = new KategoriModel()
{
    Ad= "TELEFONLAR",
    Aciklama = "Yerli Telefonlar"
};

//Yukarıdaki yazım şeklini şu şekilde de kullanabilirdik.

//var KategoriNesnesi = new KategoriModel()
//KategoriNesnesi.Ad = "TELEFONLAR";
//KategoriNesnesi.Aciklama = "Yerli Telefonlar";
```

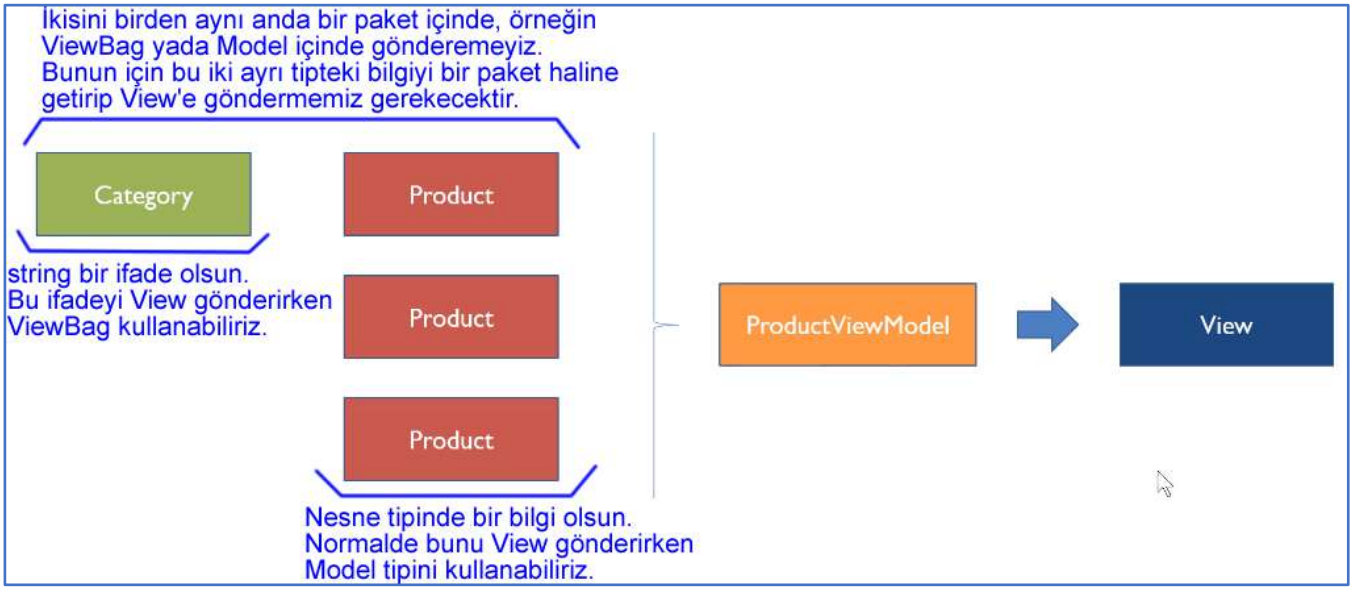
```
ViewBag.Kategori = KategoriNesnesi;
```

View sayfasındaki bilgiler

```
<p>@ViewBag.Kategori.Ad</p>
<p>@ViewBag.Kategori.Aciklama</p>
```

D) ViewModels İçerisinde İki Ayrı Model Yapısını Tek bir Paket Halinde Taşıma

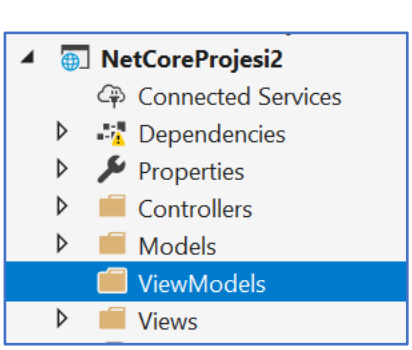
Önceki bölümlerde string bir ifadeyi View gönderirken ViewBag içinde göndermiştik. Sınıf yapısında bir ifadeyi de Model yöntemiyle göndermiştik. Fakat bunların her ikisini tek bir sınıf içinde gönderememiştik. İşte bu şekilde iki farklı Model sınıfını tek bir sınıf içinde göndermek için ViewModel yapısını kullanacağız. Bu sınıf içinde diğer iki model sınıfını tek bir paket haline getireceğiz. O şekilde View sayfasına göndereceğiz.

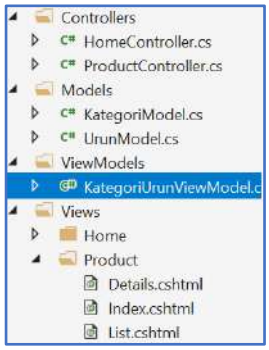


İki Adet Model Bilgisini tek bir seferde Sayfaya nasıl taşırız. Esas konumuz burasıydı. İki adet Model bilgisini sayfaya taşımak için şu aşamaları takip edelim

Örnek 1:

a) Ana klasör içerisine **ViewModels** isminde bir klasör oluşturalım.





b) Bu klasör içerisine KategoriUrunViewModel.cs isminde bir sınıf dosyası oluşturalım. Sınıf ismi de aynı isimde olsun. Aslında yaptığımız işlem bu sınıf içerisinde daha önceden oluşturduğumuz iki adet Model sınıfını kullanarak tek bir sınıf haline getirmektir. Sınıf içindeki tanımlamalarımız aşağıdaki şekilde olacaktır.

UrunModel.cs içerisi	**KategoriModel.cs içerisi**
`public class UrunModel`	`public class KategoriModel`

```
    {
        public string Ad { get; set; }
        public double Fiyat { get; set; }

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

```
    {
        public string Ad { get; set; }
        public string Aciklama { get; set; }
    }
```

KategoriUrunViewModel.cs içerisi

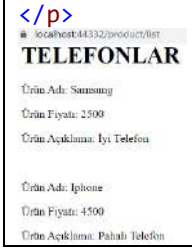
```
public class KategoriUrunViewModel
{
    public KategoriModel Kategoriler { get; set; }
    public List<UrunModel> Urunler { get; set; }
}
```

ProductController.cs içerisi

```
public IActionResult List()
{
var UrunListesi = new List<UrunModel>()
    {
        new UrunModel { Ad = "Samsung", Fiyat = 2500, Aciklama = "İyi Telefon" },
        new UrunModel { Ad = "Iphone", Fiyat = 4500, Aciklama = "Pahalı Telefon" },
        new UrunModel { Ad = "Vestel", Fiyat = 1500, Aciklama = "Yerli Telefon" }
    };

var TelefonKategorisi = new KategoriModel()
{
    Ad = "TELEFONLAR",
    Aciklama = "TELEFONLAR KATEGORİSİ"
};

var KategoriveUrunler = new KategoriUrunViewModel()
{
    Kategoriler = TelefonKategorisi,
    Urunler = UrunListesi
};

return View(KategoriveUrunler);
```

List.cshtml içeriği

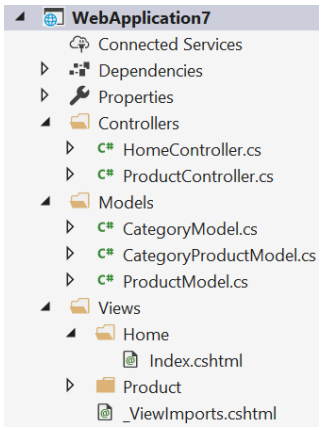
```
@using WebApplication1.ViewModels

@model KategoriUrunViewModel;

<h1> @Model.Kategoriler.Ad </h1>

<p>Ürün Adı: @Model.Urunler[0].Ad </p>
<p>Ürün Fiyatı: @Model.Urunler[0].Fiyat </p>
<p>Ürün Açıklama: @Model.Urunler[0].Aciklama
</p>

<br />

<p>Ürün Adı: @Model.Urunler[1].Ad </p>
<p>Ürün Fiyatı: @Model.Urunler[1].Fiyat </p>
<p>Ürün  Açıklama:  @Model.Urunler[1].Aciklama
</p>
```



(Not: Buradaki uygulamanın bir ileri versiyonu hem Kategoriler List<sınıf> şeklinde hemde Urunler List<sınıf> şeklinde oluşturulup ViewModel içinde tek paket haline getirme aşağıda gelecektir. O konu ile ilgili uygulamayı oradan inceleyin)

Örnek 2:

Başka bir örnek ile aynı konuyu tekrar edelim. Bu sefer ViewModel klasörü yerine bütün sınıfları Models klasörü içinde toplayalım (dikkat: deneme başka bir proje içinde yapılmıştır). Sonuçta tüm model yapıları tek bir klasörde toplamak daha anlaşılır olur. ViewModels diye ayrı bir klasörde toplarsak başka bir yapı gibi izlenim oluşuyor.



CategoryModel

```
namespace WebApplication7.Models
{
    public class CategoryModel
    {
        public string Ad { get; set; }
        public string Aciklama { get; set; }

    }
}
```

ProductModel

```
namespace WebApplication7.Models
{
    public class ProductModel
    {
        public string Name { get; set; }
        public int Price { get; set; }
        public string Definition { get; set; }

    }
}
```

CategoryProductModel

```
namespace WebApplication7.Models
{
    public class CategoryProductModel
    {
        public CategoryModel Kategori { get; set; }
        public List<ProductModel> Urunler { get; set;
}

    }
}
```

HomeController

```
public IActionResult Index()
{

    var KategoriAdi = new CategoryModel()
    {
        Ad = "Kırtasiye",
        Aciklama = "Kırtasiye Ürünleri Kategorisi"
    };

    var UrunlerAdi = new List<ProductModel>()
    {
        new ProductModel{Name="Kalem", Price=10,
Definition="Kırmızı renkli kalem" },
        new ProductModel{Name="Defter", Price=15,
Definition="Karali Defter, 100 sayfa"},
        new ProductModel{Name="Silgi", Price=3,
Definition="Temiz silgi"}
    };

    //------- iki modeli tek bir model içinde
paketleme işlemi ------------ 1. Yazım şekli
    //var KategoriVeUrunler = new
CategoryProductModel();

    //KategoriVeUrunler.Kategori = KategoriAdi;
    //KategoriVeUrunler.Urunler = UrunlerAdi;

    //------- iki modeli tek bir model içinde
paketleme işlemi ------------2. Yazım şekli
    var KategoriVeUrunler = new CategoryProductModel()
    {
        Kategori = KategoriAdi,
        Urunler = UrunlerAdi
    };

    return View(KategoriVeUrunler);
}
```

Index.cshtml

Çıktısı

```
@using WebApplication7.Models

@model CategoryProductModel

<p>@Model.Kategori.Ad (@Model.Kategori.Aciklama)</p>

@foreach(var Urun in Model.Urunler)
{
    <p> @Urun.Name / @Urun.Price / @Urun.Definition
</p>
}
```

(Razor Sayfa Yapısı) Asp sayfaları içinde C# kodlarının kullanımı

Şu ana kadar kullandığımız sayfalarda bilgileri görüntülerken her bilgi için html nesnelerimizi ayrı ayrı oluşturuyorduk. Örneğin 2 tane ürün bilgimiz varsa iki defa etiketlerini kullanarak bilgileri görüntüledik. Oysa yüzlerce ürün bilgimiz olabilirdi. Böyle bir durumda .cshtml sayfaları içerisinde C# kodlarını döngü şeklinde kullanmamız gerekir. Şimdi bu amaçla örnek bir kod yazalım.

Örnek 1: Örneğimizde 4 tane telefon bilgimiz olsun. Bu telefonları listelerken <ul> etiketi kullanarak bir liste şeklinde görüntülüyelim. Ayrıca list içinde bir ürün bilgisi gelmez ise sayfaya uyarı mesajı versin.

List.cshtml sayfası

```
@model NetCoreProjesi2.ViewModels.UrunViewModel

<h2>List Sayfası</h2>

<p>@Model.KategoriViewModel.Ad</p>

@if (Model.UrunlerViewModel.Count > 0)
{
    <ul>
        @foreach (var Eleman in Model.UrunlerViewModel)
        {
            <li>
                <div>
                    <p style="background-color:lightcyan;">@Eleman.Ad </p>
                    <p>@Eleman.Fiyat </p>
                    <p>@Eleman.Tanim </p>
                </div>
            </li>
        }

    </ul>
}
else
{
    <p style="background-color:orange;">Ürün Bulunamadı</p>
}
```

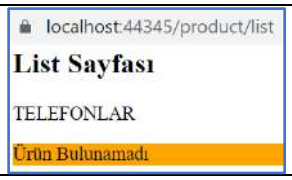
ProductController.cs Controller dosyası

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using NetCoreProjesi2.Models; //Sayfaya eklendi
using NetCoreProjesi2.ViewModels; //Sayfaya eklendi

namespace NetCoreProjesi2.Controllers
{
    public class ProductController : Controller
    {
        //localhost:5001/product/list linki için metod.
```

```
        public IActionResult list()
        {
            var KategoriNesnesi = new KategoriModel()
            {
                Ad= "TELEFONLAR",
                Aciklama = "Yerli Telefonlar"
            };

            var UrunlerlerNesnesi = new List<UrunModel>()
            {
                new UrunModel{Ad= "GM", Fiyat = 2000,Tanim = "İyi Telefon"},
                new UrunModel{Ad= "Vestel", Fiyat = 1500,Tanim = "Yerli Telefon"}
            };

            var UrunViewModelNesnesi = new UrunViewModel()
            {
                KategoriViewModel = KategoriNesnesi,
                UrunlerViewModel = UrunlerlerNesnesi
            };

            return View(UrunViewModelNesnesi);
        }

    }
}
```

Eğer bilgilerin gönderildiği Controller sayfasındaki ürünlere ait şu bilgileri gizlersek uyarı çıktısını da alırız.

```
var UrunlerlerNesnesi = new List<UrunModel>()
{
  //new UrunModel{Ad= "GM", Fiyat = 2000,Tanim = "İyi Telefon"},
  //new UrunModel{Ad= "Vestel", Fiyat = 1500,Tanim = "Yerli Telefon"}
};
```



Örnek 2: Aynı örnek üzerinde devam edersek, sitil dosyalarımızıda C# ile nasıl kontrol ederiz onu görelim. Örneğin bir Kategori içerisindeki ürün sayısı 2 den fazla ise bu ürünleri listelerken Kategori başlığının zemin rengini Turuncu, 2 den az ise Mavi şeklinde yapalım.

```
@model NetCoreProjesi2.ViewModels.UrunViewModel

@{
    var PopularDegiskeni = "";

    if (Model.UrunlerViewModel.Count > 2)
        PopularDegiskeni = "popular";
    else
        PopularDegiskeni = "popularDegil";
}

<style>
    .popular{
        background-color:orange;
        font-weight:800;
    }

    .popularDegil {
        background-color: lightblue;
        font-weight: 300;
    }
</style>

<h2>List Sayfası</h2>

<p class="@PopularDegiskeni">@Model.KategoriViewModel.Ad</p>

@if (Model.UrunlerViewModel.Count > 0)
{
    <ul>
        @foreach (var Eleman in Model.UrunlerViewModel)
        {
```

2 adet Ürün varken görüntü

3 Adet Ürün Varken

```
            <li>
                <div>
                    <p style="background-color:lightcyan;">@Eleman.Ad </p>
                    <p>@Eleman.Fiyat </p>
                    <p>@Eleman.Tanim </p>
                </div>
            </li>
        }

    </ul>
}
else
{
    <p style="background-color:orange;">Ürün Bulunamadı</p>
}
```

Not: Dikkat edilirse bu sayfada kullandığımız Model nesnesi içerisinde aslında iki tane sınıf bilgisi vardır (Kategori ve Ürünlere ait bilgiler şeklinde). Tüm sınıfın adını kullanırken, bilgiler biraz uzun gözükmektedir. Bu sınıflara ait bilgileri daha kısa değişkenlere (bunlarda yine sınıftır) aktarırsak program daha sade gözükecektir. Örneğin bazı satırları şu şekilde değiştirebiliriz.

```
@model NetCoreProjesi2.ViewModels.UrunViewModel

@{
    var Kategori = Model.KategoriViewModel;
    var Urunler = Model.UrunlerViewModel;


    var PopularDegiskeni = "";

    if (Urunler.Count > 2)
        PopularDegiskeni = "popular";
    else
        PopularDegiskeni = "populardegil";
}

<style>
    .popular{
        background-color:orange;
        font-weight:800;
    }

    .populardegil {
        background-color: lightblue;
        font-weight: 300;
    }
</style>

<h2>List Sayfası</h2>

<p class="@PopularDegiskeni">@Kategori.Ad</p>

@if (Urunler.Count> 0)
{
    <ul>
        @foreach (var Eleman in Urunler)
        {
            <li>
                <div>
```

```
                    <p style="background-color:lightcyan;">@Eleman.Ad </p>
                    <p>@Eleman.Fiyat </p>
                    <p>@Eleman.Tanim </p>
                </div>
            </li>
        }

    </ul>
}
else
{
    <p style="background-color:orange;">Ürün Bulunamadı</p>
}
```

Örnek 3: Ürünlerimizi listelerken, Stokda olanların adlarını yeşil bir zeminle, stokta olmayanların zemilerini ise sarı bir zeminle görüntüleyelim.

Bunun UrunModel sınıfının içerisine bir alan daha ekleyelim. "StokDurumu" isminde ve tipi Boolean olsun. Tüm kodları bir daha verelim..

UrunModel.cs dosyası

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace NetCoreProjesi2.Models
{
public class UrunModel
{
public string Ad { get; set; }
public double Fiyat { get; set; }
public string Tanim { get; set; }

public bool StokDurumu { get; set; }
}
```

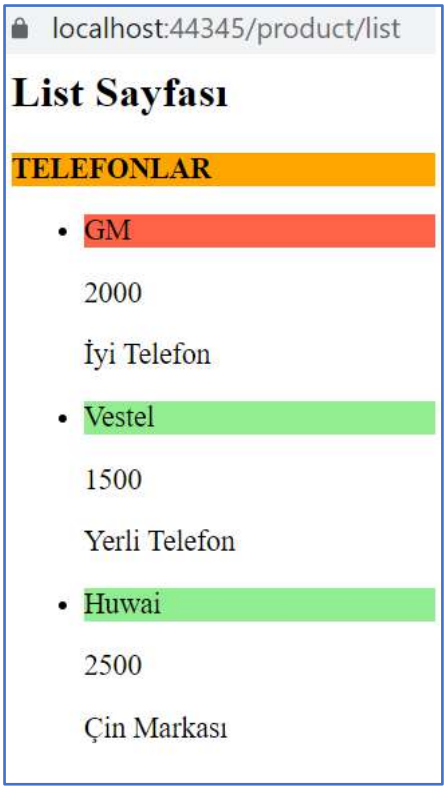
List.cshtml dosyası

```
@model NetCoreProjesi2.ViewModels.UrunViewModel

@{
var Kategori = Model.KategoriViewModel;
var Urunler = Model.UrunlerViewModel;


var PopularDegiskeni = "";

if (Urunler.Count > 2)
PopularDegiskeni = "popular";
else
PopularDegiskeni = "populardegil";
}

<style>
.popular{
background-color:orange;
font-weight:800;
}

.populardegil {
background-color: lightblue;
font-weight: 300;
}
```

ProductController.cs dosyası

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using NetCoreProjesi2.Models; //Sayfaya eklendi
using NetCoreProjesi2.ViewModels; //Sayfaya
eklendi

namespace NetCoreProjesi2.Controllers
{
  public class ProductController : Controller
  {
public IActionResult list()
{
var KategoriNesnesi = new KategoriModel()
{
Ad= "TELEFONLAR",
Aciklama = "Ucuz Telefonlar"
};

var UrunlerlerNesnesi = new List<UrunModel>()
{
new UrunModel{Ad= "GM", Fiyat = 2000,Tanim =
"İyi Telefon", StokDurumu=false },
new UrunModel{Ad= "Vestel", Fiyat = 1500,Tanim
= "Yerli Telefon", StokDurumu=true },
new UrunModel{Ad= "Huwai", Fiyat = 2500,Tanim =
"Çin Markası", StokDurumu=true }
};

var UrunViewModelNesnesi = new UrunViewModel()
{
KategoriViewModel = KategoriNesnesi,
UrunlerViewModel = UrunlerlerNesnesi
};

return View(UrunViewModelNesnesi);
}
}
}
```

```
</style>

<h2>List Sayfası</h2>

<p class="@PopularDegiskeni">@Kategori.Ad</p>

@if (Urunler.Count> 0)
{
<ul>
@foreach (var Eleman in Urunler)
{
<li>
<div>
@if (Eleman.StokDurumu == true)
{
<p style="background-color:lightgreen;">@Eleman.Ad </p>
}
else
{
<p style="background-color:tomato;">@Eleman.Ad </p>
}

<p>@Eleman.Fiyat </p>
<p>@Eleman.Tanim </p>
</div>
</li>
}

</ul>
}
else
{
<p style="background-color:yellow;">Ürün Bulunamadı</p>
}
```



Bootsrap Tasarımın Hazırlanması

Sonraki uygulamalarımızda ihtiyaç olacağından sayfalarımızda hızlıca Css tabanlı tasarım alt yapısı olan Bootsrap kütüphanesini kullanarak bir tasarım gerçekleştirelim. Bootsrap sitesine girip “Get started” kısmından linkimizi alalım.


```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta2/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-BmbxuPwQa2lc/FVzBcNJ7UAyJxM6wuqIj61tLrc4wSX0szH/Ev+nYRRuWlolflfl"
crossorigin="anonymous">
```

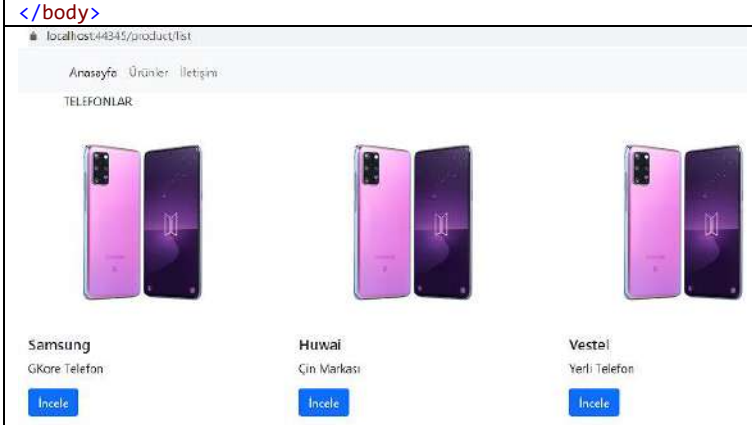
Bu linki List.cshtml sayfası içerisinde <head> içerisine yapıştıralım. İki tane sayfa içerisinde bir tasarımız olsun. Urunler (product/list) sayfasında hem bir NavBar (gezgin çubuğu) olsun hemde ürünleri gösteren bir kart tasarımımız olsun.

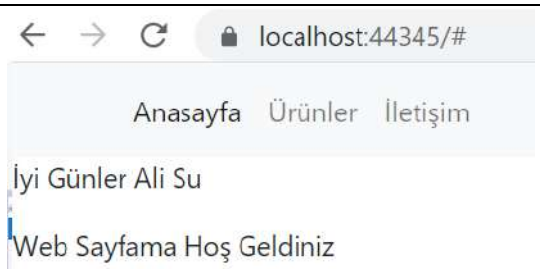
Product/list sayfası	**home/index sayfası**

```
@model NetCoreProjesi2.ViewModels.UrunViewModel

<head>
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta2/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-
BmbxuPwQa2lc/FVzBcNJ7UAyJxM6wuqIj61tLrc4wSX0szH/Ev+nYRRuWlolf
lfl" crossorigin="anonymous">
</head>

<body>
<header>
@*********NAVBAR*************@
<nav class="navbar navbar-expand-lg navbar-light bg-light">
<div class="container-fluid">

<div class="collapse navbar-collapse"
id="navbarSupportedContent">
<div class="container">

<ul class="navbar-nav me-auto mb-2 mb-lg-0">
<li class="nav-item">
<a class="nav-link active" aria-current="page"
href="#">Anasayfa</a>
</li>

<li class="nav-item">
<a class="nav-link" href="#">Ürünler</a>
</li>

<li class="nav-item">
<a class="nav-link" href="#">İletişim</a>
</li>
</ul>
</div>
</div>
</div>
</nav>

</header>

<main>
@{
var Kategori = Model.KategoriViewModel;
var Urunler = Model.UrunlerViewModel;
}
@*********KART GÖRÜNÜMÜ*************@
<div class="container">

<div class="row">
<div class="col-md-12">
<p>@Kategori.Ad</p>
</div>
</div>
</div>

@if (Urunler.Count > 0)
{
<div class="row">

@foreach (var Eleman in Urunler)
{
<div class="col-md-3">
<div class="cart">
<img class="card-img-top"
src="https://images.samsung.com/is/image/samsung/tr-galaxy-
s20-plus-bts-sm-g985fzpdtur-frontbpurple-
263911073?$684_547_PNG$" alt="" />
<div class="card-body">
<h5 class="card-title">@Eleman.Ad</h5>
<p class="card-text">@Eleman.Tanim </p>

<a class="btn btn-primary">İncele</a>
</div>

</div>
</div>
```

```
<head>
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.
0.0-beta2/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
BmbxuPwQa2lc/FVzBcNJ7UAyJxM6wuqIj61tLrc4wSX0szH
/Ev+nYRRuWlolflfl" crossorigin="anonymous">
</head>
<body>
<header>
@*********NAVBAR*************@
<nav class="navbar navbar-expand-lg navbar-
light bg-light">
<div class="container-fluid">

<div class="collapse navbar-collapse"
id="navbarSupportedContent">
<div class="container">

<ul class="navbar-nav me-auto mb-2 mb-lg-0">
<li class="nav-item">
    <a class="nav-link active" aria-
current="page" href="#">Anasayfa</a>
</li>

<li class="nav-item">
    <a class="nav-link" href="#">Ürünler</a>
</li>

<li class="nav-item">
    <a class="nav-link" href="#">İletişim</a>
</li>
</ul>
</div>
</div>
</div>
</nav>

</header>

<main>

<p>@ViewBag.SelamlamaIfadesi
@ViewBag.ZiyaretciAdi </p>

<p>Web Sayfama Hoş Geldiniz</p>
</main>
</body>
```

```
}

</div>

}
else
{
<div class="row">
<div class="col-md-12">
<div class="alert alert-danger">
Ürün Bulunamadı
</div>
</div>
</div>
}
</main>
</body>
```





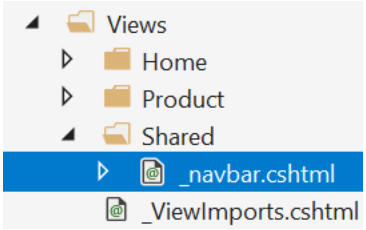
(Partial Views) – Sayfayı Alt Kısımlara Bölme

Sayfalarımızın içerisinde kullandığımız tasarımların bazı bölümleri bütün sayfalarda yada belli sayfalarda kullanılıyor olabilir. Örneğin bir Gezgin Çubuğunun (NavBar) bütün sayfalarda üstte gözükmesini isteyebiliriz. Böyle bir durumda tüm sayfalara bu gezgin çubuğunun etiketlerini eklememiz gerekir. Üzerinde yapacağımız en ufak bir değişiklikte tüm sayfalarda bu değişikliği yapmamız gerekecektir. Madem tüm sayfalarda tasarımın belli bir kısmında aynı etiketleri kullanacağız, o zaman bunları tek bir yerde oluşturup, tüm sayfaların içeriği bu yerden alması, hem daha pratik olur, hemde yönetmesi kolay olur.

Örnek 1: Her sayfada gözükecek Gezgin Çubuğunu Tek Bir Yerden Kontrol etme

Bunun için View altında "**Shared**" klasörünü kullanalım. Bildiğimiz gibi View altında Controllerların yönettiği sayfalar vardır (Views>Home. Views>Product gibi). İşte Shared klasörünüde bütün Controllerlar ortak kullanmaktadır.

Oluşturduğumuz bu Shared klasörü içerisine **_navbar.cshtml** isminde normal bildiğimiz bir html sayfası oluşturalım. İsmine dikkat edersek alt çizgi ile başlamaktadır. Bu çizgi zorunlu olmasa da adet üzere tam bir sayfa değil, bir sayfanın parçası anlamında kullanılmaktadır.



Bu kısmi sayfa içerisine daha önce oluşturduğumuz NavBar kopyalayalım.

_navbar.cshtml dosyası

```
@*********NAVBAR*************@
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="container-fluid">

        <div class="collapse navbar-collapse" id="navbarSupportedContent">
            <div class="container">

                <ul class="navbar-nav me-auto mb-2 mb-lg-0">
                    <li class="nav-item">
                        <a class="nav-link active" aria-current="page" href="#">Anasayfa</a>
                    </li>

                    <li class="nav-item">
                        <a class="nav-link" href="#">Ürünler</a>
                    </li>

                    <li class="nav-item">
                        <a class="nav-link" href="#">İletişim</a>
                    </li>
                </ul>
            </div>
        </div>
    </div>
</nav>
```

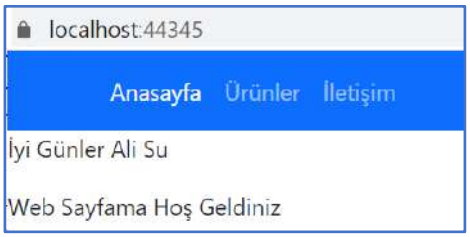
Gezgin çubuğunu yukarıdaki iki sayfada kullanmıştık. Bu sayfalardan navbar etiketlerini kaldırıp yerlerine aşağıdaki kodları yazalım. Bu kod sayesinde Shared klasörü altında bu isimdeki dosya direk eklenecektir. Dosyanın uzantısını yazmaya gerek yoktur. Aynı merkezden değiştiğini görmek içinde navbar görünümü üzerinde ufak bir değişiklikte yapabiliriz. Bunun için <nav> etiketinin olduğu yeri aşağıdaki satırla değiştirelim. Görünümü mavi renk alsın.

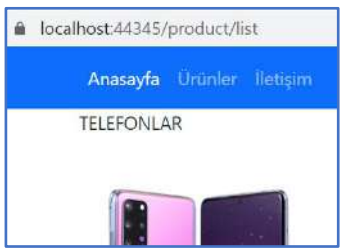
```
<nav class="navbar navbar-expand-lg navbar-dark bg-primary">
```

Daha sonra sayfalarımız içindeki <nav></nav> kısımlarıda kaldırıp şu kodları oraya yazalım.

```
@await Html.PartialAsync("_navbar")
```

Ardından çalıştırırsak navbar eklediğimiz tüm sayfalarda gezgin çubuğunun değiştiğini görürüz.

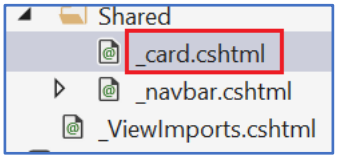




Örnek 2. Dinamik olarak yönetilmesi gereken Kart görünümlerini tek bir yerden yönetme

Yukarıda yaptığımız örnekde kullanılan Gezgin çubuğu tüm sayfalarda aynı görünüme sahiptir ve shared klasörü içinden bir değişiklik yaptığımızda tüm sayfalar değişecektir. Ama dinamik olarak kontrol edilmesi gereken yerlerde olabilir. Örneğin sayfada ürünleri gösterdiğimiz kart görünümlerinde sayfada kaç tane kart varsa o kadar kod yazmamız gerekiyordu. Oysa tüm kartlar aynı görünüme sahipse bunlarda tek bir yerden yönetebiliriz.

Bunun için shared klasörü altında **_cart.cshtml** dosyasını oluşturalım. Bu dosya içerisine list sayfasından tek bir kart için oluşturulan divleri buraya kopyalayalım. Kart görünümü dışında bulunan container ve row divleri durmalı.



List.cshmtl sayfası öncesi

```
<div class="row">

@foreach (var Eleman in Urunler)
{
<div class="col-md-3">
<div class="cart">
<img class="card-img-top"
src="resimadresi.jpg " alt="" />
<div class="card-body">
    <h5 class="card-
title">@Eleman.Ad</h5>
    <p class="card-
text">@Eleman.Tanim </p>

    <a class="btn btn-
primary">İncele</a>
</div>

</div>
</div>
}

</div>
```

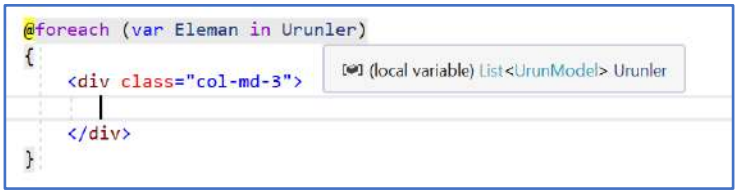
List.cshmtl sayfası sonrası

```
<div class="row">

@foreach (var Eleman in
Urunler)
{
<div class="col-md-3">
@CART görünümü @
@await
Html.PartialAsync("_Cart",
Eleman)
</div>
}

</div>
```

_cart.cshtml sayfas

```
<div class="cart">
    <img class="card-img-top"
src="resimadresi.jpg" alt="" />
    <div class="card-body">
        <h5 class="card-
title">@Eleman.Ad</h5>
        <p class="card-text">@Eleman.Tanim
</p>
        <a class="btn btn-
primary">İncele</a>
    </div>
</div>
```

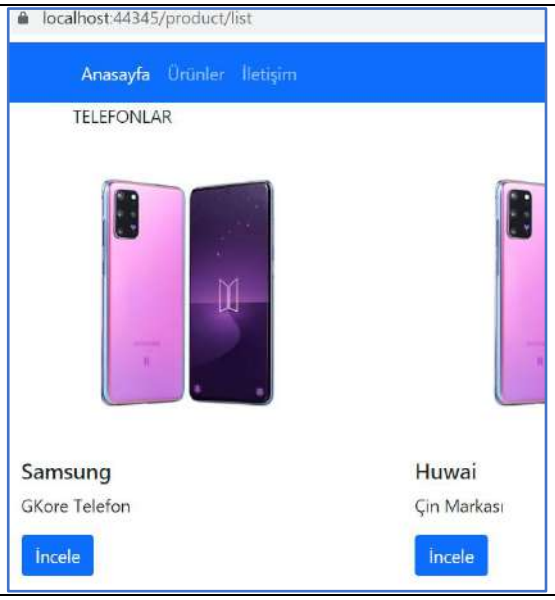
Kart görünümüne ait div ler içerisinde bilgileri foreach döngüsü ile C# üzerinden dinamik olarak alıyorduk. Oysa divlerin taşındığı sayfada bilgiler için kullandığımız değişkenler çalışmayacaktır. Foreach döngüsü içerisinde kullandığımız "Urunler" bir List sınıfına aittir. İşte bu sınıfı _card.cshtml sayfası içerisinde en üstte Model sınıfının içerisine atmamız gerekir. Böylece buradaki bilgiler orayada taşınmış olacaktır. Orada ise artık model sınıfını kullanacağımızdan sayfa içerisinde değişken isimlerinde @model.Ad şeklinde kullanım yapmalıyız. (*Dikkat edilirse tanımlarken küçük harf "model" kullanırken büyük harf "Model" şeklindedir*)




```
@model UrunModel

<div class="cart">
    <img class="card-img-top" src="https://images.samsung.com/is/image/samsung/tr-galaxy-s20-plus-bts-sm-
g985fzpdtur-frontbpurple-263911073?$684_547_PNG$" alt="" />
    <div class="card-body">
        <h5 class="card-title">@Model.Ad</h5>
        <p class="card-text">@Model.Tanim </p>
        <a class="btn btn-primary">İncele</a>
    </div>
</div>
```

Son olarak list sayfamıza kart bilgilerini getirmek üzere gideceği kodları yazalım. Dikkat edilirse burada Urunler aslında <list> şeklinde bir sınıftır. Yani içerisinde onlarca ürün bilgisi olabilir ve bunlara index numarası ile erişebiliriz. Eleman ise yine bu sınıfın tipinde tek bir ürün bilgisini tutan değişkendir. _cart.cshtml sayfasına bu bilgi götürülmektedir fakat orada Model sınıfını oluşturmak için bu hem Urunler değişkenininin hemde Eleman değişkenin türetildiği UrunModel sınıfı karşılamış olmaktadır. Bu sınıfla aynı tipi sayfa içerisinde kullanacağımız model değişkeni oluşmaktadır. Artık sayfa içerisinde değişken olarak sadece Model değişkeni ve alt bilgilerini kullanmaktayız.

```
<div class="row">

    @foreach (var Eleman in Urunler)
    {
    <div class="col-md-3">
        @await Html.PartialAsync("_card", Eleman)
    </div>
    }

</div>
```

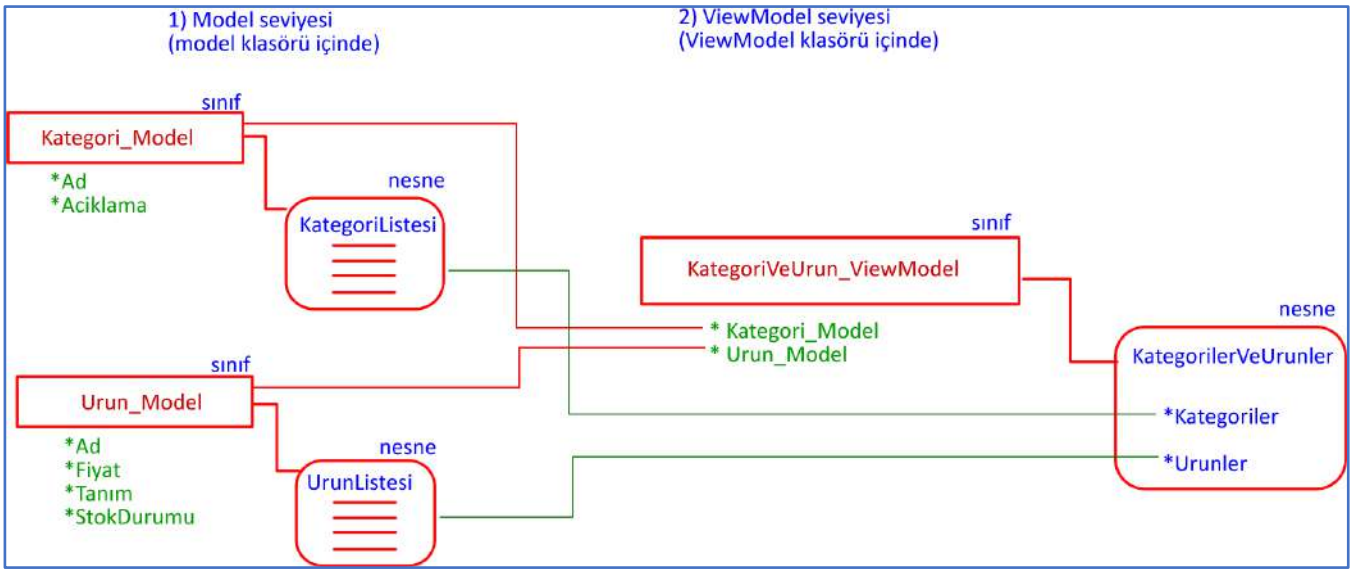


Böylece artık nerede bu kart görünümünü kullanıyorsak hepsini tek bir merkezden yönetmiş oluruz.

Kategorileri Tüm Sayfalarda "Shared" olarak Görüntüleme

Yukarıda kodlarda kullanılan sınıfların isimleri çok karıştığından biraz daha anlaşılması için Model ve ViewModel sınıflarının isimleri şu şekilde değiştirildi. Bunlardan türetilen nesnelerinde isimleri gösterildi. Buradaki yapıyı incelediğimizde iki tane alt Model vardır (Kategori_Model, Urun_Model). Bunların birleşiminden bir de üst model olarak KategoriVeUrun_ViewModeli oluşturulmuştur. Yani iki tane Model sınıfları birleştirilerek ViewModel sınıfını oluşturmuştur. Model ler içinde <List> şeklinde çoklu bilgiler tutulmuştur.

Örnek 1: List sayfası içinde Kategori ve Ürünleri Görüntüleme-(List<KategoriModel>, List<UrunModel> şeklinde bilgi taşıma)



Kategori_Model.cs	Urun_Model.cs	KategoriVeUrun_ViewModel.cs
using System; using System.Collections.Generic;	using System; using System.Collections.Generic;	using NetCoreProjesi2.Models; //Sayfaya Eklenmeli.

```
using System.Linq;
using System.Threading.Tasks;

namespace NetCoreProjesi2.Models
{
    public class Kategori_Model
    {
        public string Ad { get;
set; }
        public string Aciklama {
get; set; }
    }
}
```

```
using System.Linq;
using System.Threading.Tasks;

namespace NetCoreProjesi2.Models
{
    public class Urun_Model
    {
        public string Ad { get;
set; } //Bu satırı hızlı yazmak
için prog yazıp iki defa tab tuşuna
tıklayın.
        public double Fiyat { get;
set; }
        public string Tanim { get;
set; }

        public bool StokDurumu {
get; set; }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace
NetCoreProjesi2.ViewModels
{
    public class
KategoriVeUrun_ViewModel
    {
        public List<Kategori_Model>
Kategoriler { get; set; }
        public List<Urun_Model>
Urunler { get; set; }

    }
}
```

ProductController.cs

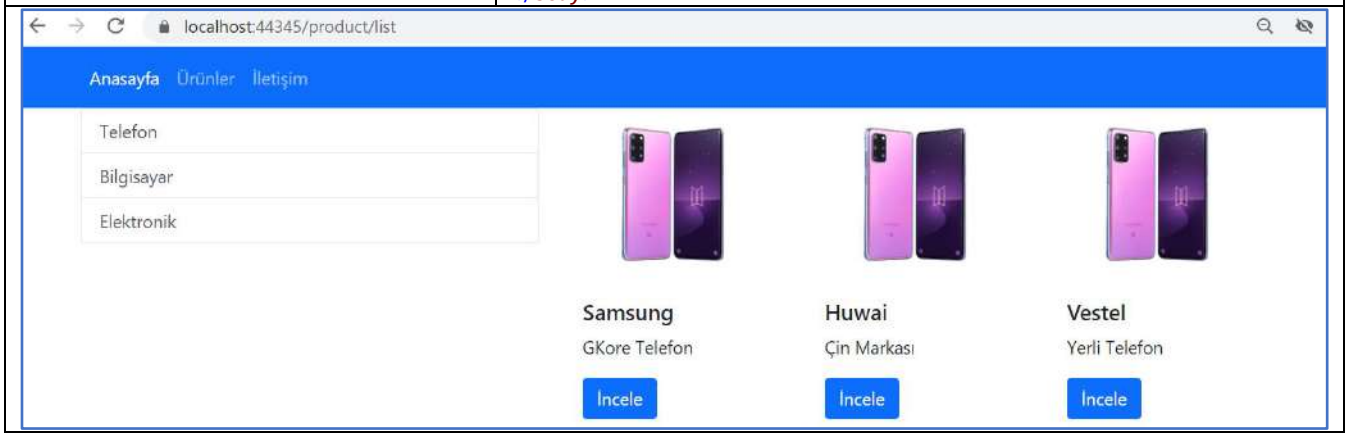
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using NetCoreProjesi2.Models; //Sayfaya
eklendi
using NetCoreProjesi2.ViewModels;
//Sayfaya eklendi

namespace NetCoreProjesi2.Controllers
{
    public class ProductController :
Controller
    {
        //localhost:5001/product/list
linki için metod.
        public IActionResult list()
        {

            var KategoriListesi = new
List<Kategori_Model>()
            {
                    new
Kategori_Model{Ad= "Telefon",
Aciklama="Akıllı Telefonlar"},
                    new
Kategori_Model{Ad= "Bilgisayar",
Aciklama ="Diz Üstü Bilgisayarlar"},
                    new
Kategori_Model{Ad= "Elektronik",
Aciklama ="Tüketici Elektroniği
Ürünleri"}
            };

            var UrunListesi = new
List<Urun_Model>()
            {
                    new Urun_Model{Ad=
"Samsung", Fiyat = 2000,Tanim = "GKore
Telefon", StokDurumu=false },
                    new Urun_Model{Ad=
"Huwai", Fiyat = 2500,Tanim = "Çin
Markası", StokDurumu=true },
                    new Urun_Model{Ad=
"Vestel", Fiyat = 1500,Tanim = "Yerli
Telefon", StokDurumu=true }
            };

            var KategorilerVeUrunler =
new KategoriVeUrun_ViewModel()
            {
```

List.cshtml

```
@model NetCoreProjesi2.ViewModels.KategoriVeUrun_ViewModel

<head>
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-
BmbxuPwQa2lc/FVzBcNJ7UAyJxM6wuqIj61tLrc4wSX0szH/Ev+nYRRuWlolflfl"
crossorigin="anonymous">
</head>

<body>
     <header>
        @await Html.PartialAsync("_navbar")
    </header>
    <main>
        @{
            var Kategoriler = Model.Kategoriler;
            var Urunler = Model.Urunler;
        }

        <div class="container">

            <div class="row">
        @*********KATEGORİ GÖRÜNÜMÜ**********@

                <div class="col-md-3">
                    <div class="list-group">
                        @foreach (var Kategori in Kategoriler)
                        {
                            <a href="#" class="list-group-item list-
group-item-action">@Kategori.Ad</a>
                        }
                    </div>
                </div>

        @*********KART GÖRÜNÜMÜ**********@

                <div class="col-md-9">

                    @if (Urunler.Count > 0)
                    {
                        <div class="row">
                            @foreach (var Urun in Urunler)
                            {
                            <div class="col-md-3">
                                @*Dikkat kart görünümü share klasörü
altından alıyor*@

                                @await Html.PartialAsync("_card",
Urun)

                            </div>
                            }
                        </div>
                    }
                    else
```

```
                Kategoriler =
KategoriListesi,
                Urunler = UrunListesi
            };

            return
View(KategorilerVeUrunler);
```

```
                    {
                        <div class="row">
                            <div class="col-md-12">
                                <div class="alert alert-danger">
                                    Ürün Bulunamadı
                                </div>
                            </div>
                        </div>
                    }
                </div>
            </div>
        </div>
    </main>
</body>
```



Bu yaptığımız uygulamada List sayfasında Kategoriler ve Ürünleri görüntüledik. Ama böyle bir sitede Kategoriler kısmı her sayfada bulunuyor olabilir. Böyle bir uygulama için Kategorilerin görüntülendiği listeyi Shared klasörünün içine taşıyabiliriz. Bu durumda tüm istediğimiz sayfalarda Kategorileri gösterebiliriz. Şimdi buna uygun satırları görüntüleyelim.

Örnek 2: Home/Index sayfasında Kategorileri Ortak paylaşılan alandan çekerek (Shared) görüntüleme.

Üstteki uygulamalar listeleme sayfasına giderken hem kategorileri hemde ürünleri yanımızda birlikte götürdük. Bu esnada KategoriVeUrun_ViewModel sınıf yapısını kullandık. Şimdi ise Giriş Index sayfasında sadece Kategorileri görüntüleyelim. Fakat hem listeleme sayfası için hemde , hemde Index sayfası için Kategorileri ortak bir alandan çekelim. Bunun için Shared klasörü içerisinde **_Kategoriler.cshtml** ortak dosyasını oluşturalım. Bu sayfaya giderken kategorileri <List> şeklinde götüreceğimizden bu kismi sayfaya erişen her View sayfası verileri bu tipte göndermelidir.

HomeController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using NetCoreProjesi2.Models; //Sayfaya
eklendi
using NetCoreProjesi2.ViewModels; //Sayfaya
eklendi

namespace NetCoreProjesi2.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
```

Index.cshtml

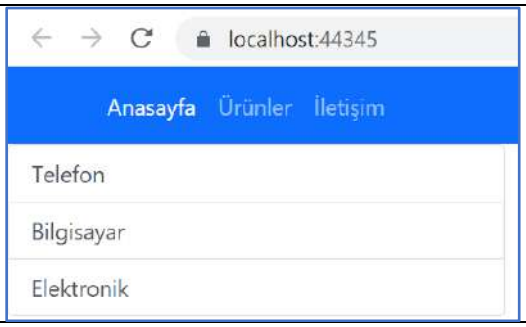
```
@model List<Kategori_Model>

<head>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta2/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-
BmbxuPwQa2lc/FVzBcNJ7UAyJxM6wuqIj61tLrc4wSX0szH/Ev+nYRRuWlolflfl"
crossorigin="anonymous">
</head>

<body>
    <header>
        @await Html.PartialAsync("_navbar")
    </header>
    <main>
        <div class="col-md-3">
            @await Html.PartialAsync("_Kategoriler", Model)
        </div>
```

```
            var KategoriListesi = new
List<Kategori_Model>()
            {
                    new Kategori_Model{Ad=
"Telefon", Aciklama="Akıllı Telefonlar"},
                    new Kategori_Model{Ad=
"Bilgisayar", Aciklama ="Diz Üstü
Bilgisayarlar"},
                    new Kategori_Model{Ad=
"Elektronik", Aciklama ="Tüketici
Elektroniği Ürünleri"}
            };
            return View(KategoriListesi);
        }
    }
}
```

```
    </main>
</body>
```

_Kategoriler.cshtml

```
@model List<Kategori_Model>

<div class="list-group">
    @foreach (var Kategori in Model)
    {
        <a href="#" class="list-group-item list-group-
item-action">@Kategori.Ad</a>
    }
</div>
```

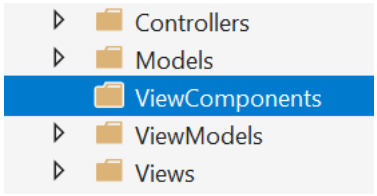


(ViewComponent) Kategori Listesini, yanımızda götürmeden, ortak bir kaynaktan çekme

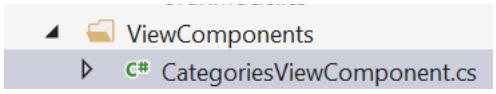
Biz istediğimiz sayfalarda Kategorileri göstermek isterken, her o sayfaya gidişimizde Controller içerisinden Kategorileride götürmemiz gerekiyordu. Onun yerine kategoriler sonuçta sabit bir paket ise, biz hangi sayfaya gidersek gidelim bunları yanımızda götürmek yerine, açtığımız her sayfada bunları sabit bir yerden alabiliriz.

Bu amaçla **ViewComponent** klasörü içinde oluşturduğumuz nesneleri sayfalarımızda kullanabileceğiz. Bu klasör içinde oluşturacağımız sınıfların sonunda ViewComponent yazısını da eklememiz gerekiyor. Bu işlem için şu adımları takip edelim.

1) Ana klasör içine **ViewComponents** isminde bir klasör oluşturalım.



2) Bu klasör içerisinde boş bir **Class** oluşturalım. İsminide **CategoriesViewComponent** şeklinde atayalım. Dikkat edersek Categories ifadesinin sonuna ViewComponent ifadesini ekledik. Tıpkı Product sonuna Controller eklediğimiz gibi bir uygulama yapmış olduk. Bu ifadenin eklenmesi zorunludur.



3) Bu Class özel bir Class tır. Dolayısı ile ViewComponent sınıfından türetilmesi için Class ın adının sonuna `:ViewComponent` sınıfını eklemeliyiz (bundan miras alacak). Bu sınıfı görebilmesi için üstede `using`

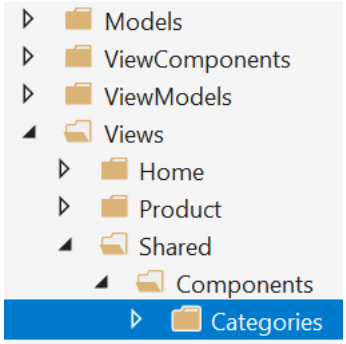
`Microsoft.AspNetCore.Mvc;` kütüphanesini eklemeliyiz. Ayrıca Class adının devamı "ViewComponent" yazısı ile bitmelidir. (Dikkat yanlışlıkla çoğul kullanılmasın: CategoriesViewComponent)

```
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace WebApplication1.ViewComponents
{
    0 references
    public class CategoriesViewComponent:ViewComponent
    {
    }
}
```

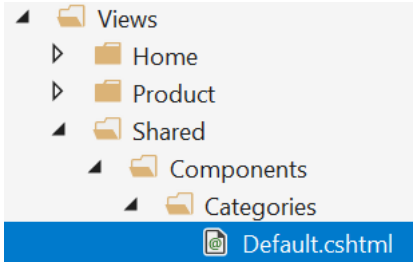
4) Class ın içerisinde Kategorileri tutacağımız ortak fonksiyonu oluşturalım.

```
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using WebApplication1.Models;

namespace WebApplication1.ViewComponents
{
    public class CategoriesViewComponent:ViewComponent
    {
        public IViewComponentResult Invoke()
        {
            var KategoriListesi = new List<KategoriModel>()
            {
                new KategoriModel { Ad = "BİLGİSAYARLAR", Aciklama = "Bilgisayar Kategorisi"},
                new KategoriModel { Ad = "TELEFONLAR", Aciklama = "Telefon Kategorisi"},
                new KategoriModel { Ad = "ELEKTRONİK", Aciklama = "Elektronik Ürünler Kategorisi"}
            };

            return View(KategoriListesi);
        }
    }
}
```

5) Bu oluşturduğumuz Class içindeki bilgileri View klasörü içindeki Shared klasörü içinde oluşturacağımız yine aynı isimdeki **Components** klasörü içinde oluştacağımız komponentimizin adı ile oluşturacağımız **Categories** e yönlendireceğiz. Önce bu bahsettiğimiz klasörleri oluşturalım.



6) Şimdi bu **Categories** klasörümüz içinde cshtml sayfamızı oluşturalım. Sayfamızın adı **Default.cshtml** ismi ile olsun.



7) Bu dosyanın içerisine (**Default.cshtml**) daha önce kullandığımız _Categories.cshtml içindeki partial olarak hazırladığımız Kategorilerin görüntüsünü oluşturan kodları alıp yapıştıralım.

```
@model List<KategoriModel>

<div class="list-group">
    @foreach (var Kategori in Model)
    {
        <a href="#" class="list-group-item list-group-item-action">@Kategori.Ad</a>
    }
</div>
```

İleride Default sayfanın içinde oluşturduğumuz Kategori görünümlerinin farklı versiyonlarını oluşturabiliriz. Burada Default dosyası için varsayılan görünüm gösterilecektir.

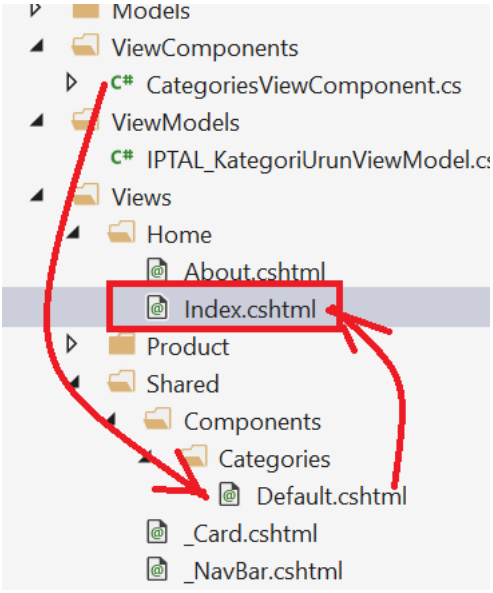
8) Artık Kategorileri göstereceğimiz **Index** ve **List** sayfaları içinde kodlarımızı aşağıdaki şekilde kullanabiliriz.

```
@*********KATEGORİ GÖRÜNÜMÜ*************@
<div class="col-md-3">
   @await Component.InvokeAsync("Categories")
</div>
```

9) Madem Kategorileri sayfada görüntülerken Components içinden okutacağız. O zaman ViewComponent içinde oluşturduğumuz Kategoriler ve Urunler sınıf paketine ihtiyacımız kalmaz. Yani ViewModeli iptal edebiliriz. Urunlerimizi UrunModel içinde taşırsak, Kategorilerimizide Components klasöründen okutursak yeterli olacaktır.
Kategorilerimizin listesini oluşturduğumuz HomeController ve ProductController içindeki ifadeleride kaldıralım.
Ayrıca partial olarak oluşturduğumuz _Categories.cshmtl sayfasını da iptal edebiliriz.

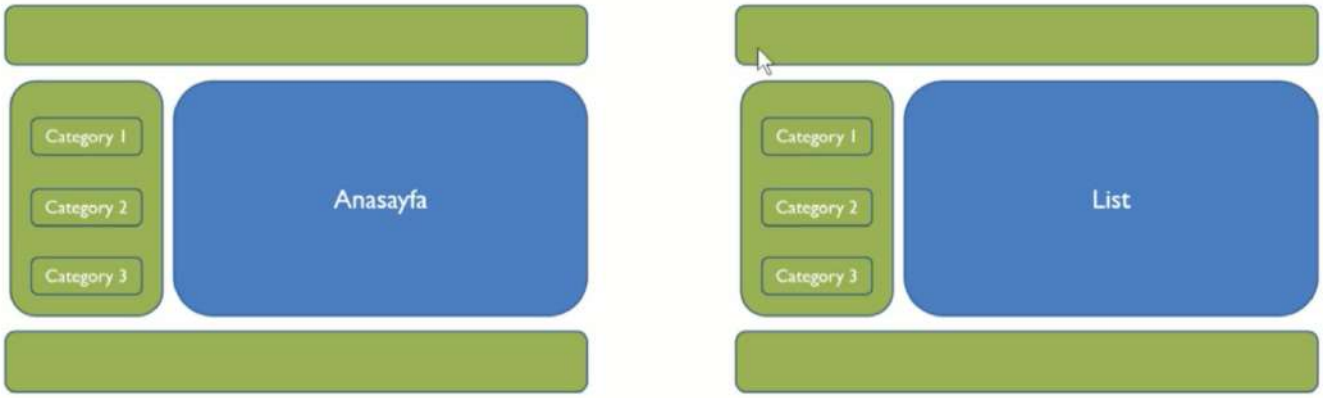
İşleyiş şu şekilde olmaktadır. Home altındaki Index sayfasını açmaya çalıştığımızda o sayfa içine yazdığımız kodlar Component lerin altındaki Categories klasörü içindeki Default sayfayı kendi içine dahil ediyor. Default.cshtml

sayfası bir Components sayfası olduğundan oda sınıf yapısına uygun bilgileri CategoriesViewComponents.cs içinden okuyor. Orada ise kategoriler liste şeklinde bulunmaktadır.

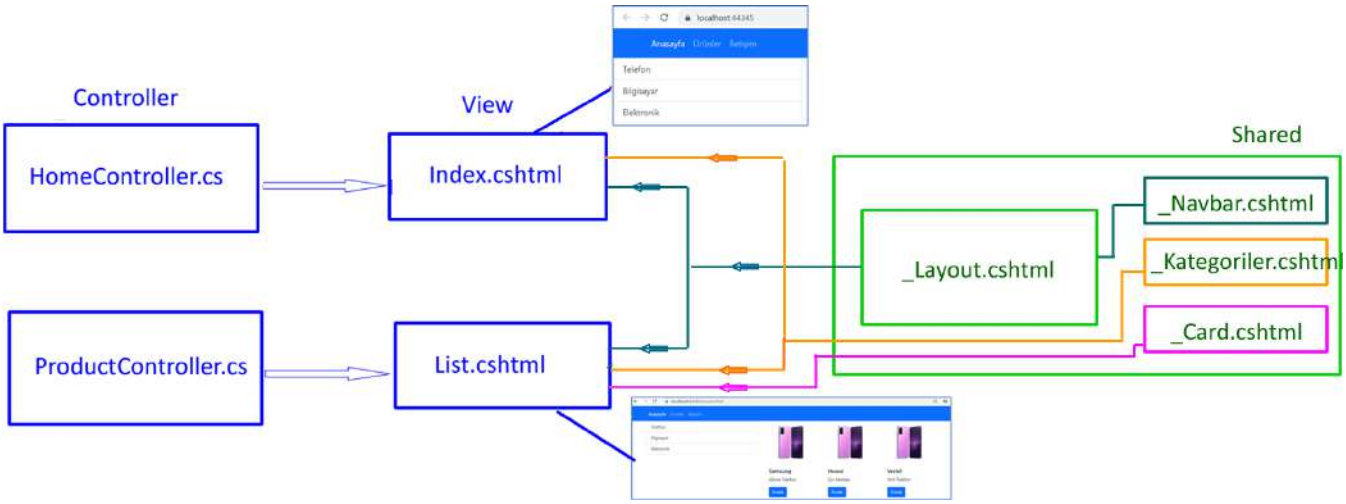


(_Layout) Sayfalardaki tekrarlı alanları Şablondan getirme

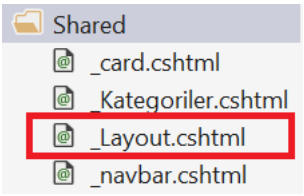
Sayfalarımızın bir çok kısımları ortaktır. Bu ortak kısımlar için tek bir **_layout.cshtml** sayfası oluşturursak bu yapıyı tüm sayfalarda kullanabiliriz. Dolayısı ile diğer sayfalarda sadece içerik kısımlarını değiştirebiliriz. Dış şablon kısmı ise ortak **Shared** klasörü içerisinde **_Layout.cshtml** dosyasında bulunacak. Tüm sayfalarda NavBar nesnemiz bulunsun. Dolayısı ile tüm sayfalara bu bölüm Layout sayfasından gelsin. Tabii Layout sayfası içerisinde de bu bölüm yine alt kısım olan _navbar.cshtml dosyasından gelsin.



Aşağıdaki grafiğe göre sayfalarımızı şu şekilde oluşturuyoruz. Index.cshtml ve List.cshtml sayfalarını oluştururken üç tane alt sayfa parçası kullanıyoruz. Bunlar _NavBar, _Kategoriler, _Card dır. _NavBar ı _Layout şablon sayfası içine atıyoruz. Bu şablonu da bağlı olduğu iki sayfada (Index ve List) çağırıyoruz. Dolayısı ile her iki sayfada da _Layout gözükürken içindeki _NavBar göstermiş oluyor. Her sayfa kendi içinde kullandığı Kategori ve Kart görünümlerini Şablon içinden değilde dışarıdan bağımsız olarak dahil ediyor.



Şimdi Shared klasörü içerisinde **_Layout.cshtml** dosyasını oluşturalım. Bu dosyanın yapısı temel html yapısını içersin. Üzerine Boostrap linkini de ekleyelim.



_Layout.cshtml

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta2/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-
BmbxuPwQa2lc/FVzBcNJ7UAyJxM6wuqIj61tLrc4wSX0szH/Ev+nYR
RuWlolflfl" crossorigin="anonymous">
</head>
<body>
    @*Sayfaya navbar kısmı partial olarak
_navbar.cshtml dosyasından geliyor.*@
    @await Html.PartialAsync("_navbar")

        <main>
            <div class="container">
                @RenderBody()
            </div>
        </main>

</body>
</html>
```

_navbar.cshtml

```
@*********NAVBAR*************@
<nav class="navbar navbar-expand-lg navbar-dark bg-
primary">

<div class="container-fluid">

<div class="collapse navbar-collapse"
id="navbarSupportedContent">
<div class="container">

<ul class="navbar-nav me-auto mb-2 mb-lg-0">
<li class="nav-item">
<a class="nav-link active" aria-current="page"
href="#">Anasayfa</a>
</li>

<li class="nav-item">
<a class="nav-link" href="#">Ürünler</a>
</li>

<li class="nav-item">
<a class="nav-link" href="#">İletişim</a>
</li>
</ul>
</div>
</div>
</div>
</nav>
```

_card.cshmtl

```
@model Urun_Model


<div class="cart">
    <img class="card-img-top"
src="https://images.samsung.com/is/image/samsung/tr-
galaxy-s20-plus-bts-sm-g985fzpdtur-frontbpurple-
263911073?$684_547_PNG$" alt="" />
    <div class="card-body">
        <h5 class="card-title">@Model.Ad</h5>
        <p class="card-text">@Model.Tanim </p>
        <a class="btn btn-primary">İncele</a>
    </div>
</div>
```

_Kategoriler.cshmlt

```
@model List<Kategori_Model>
```

Index.cshtml

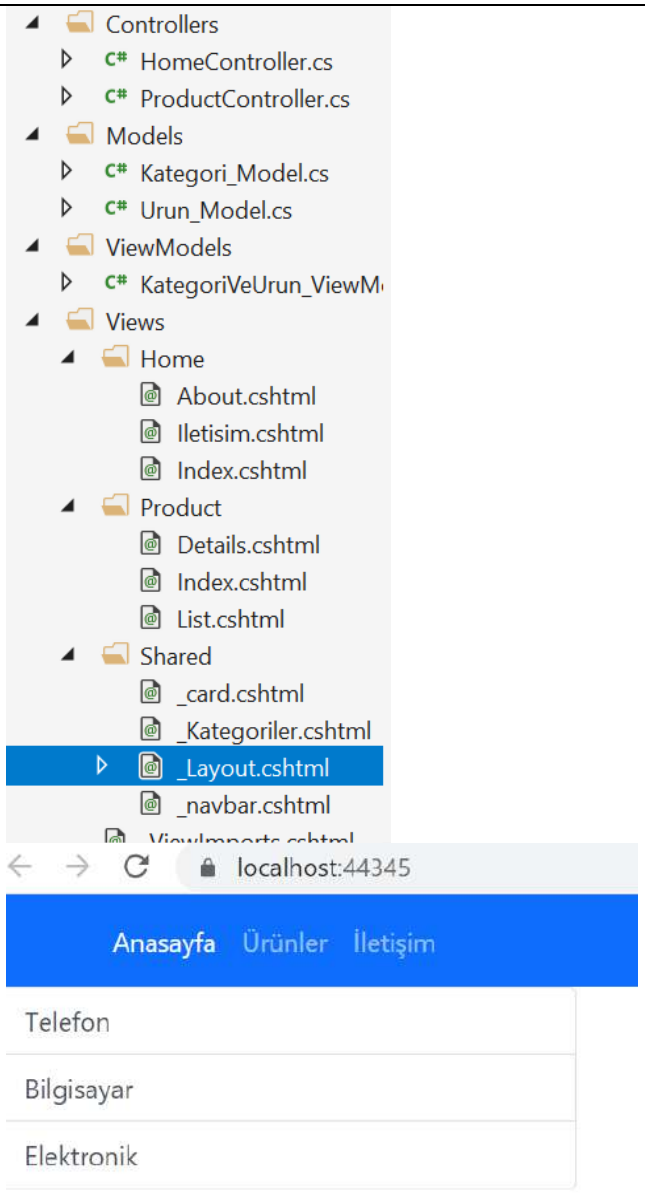
```
@model List<Kategori_Model>
```

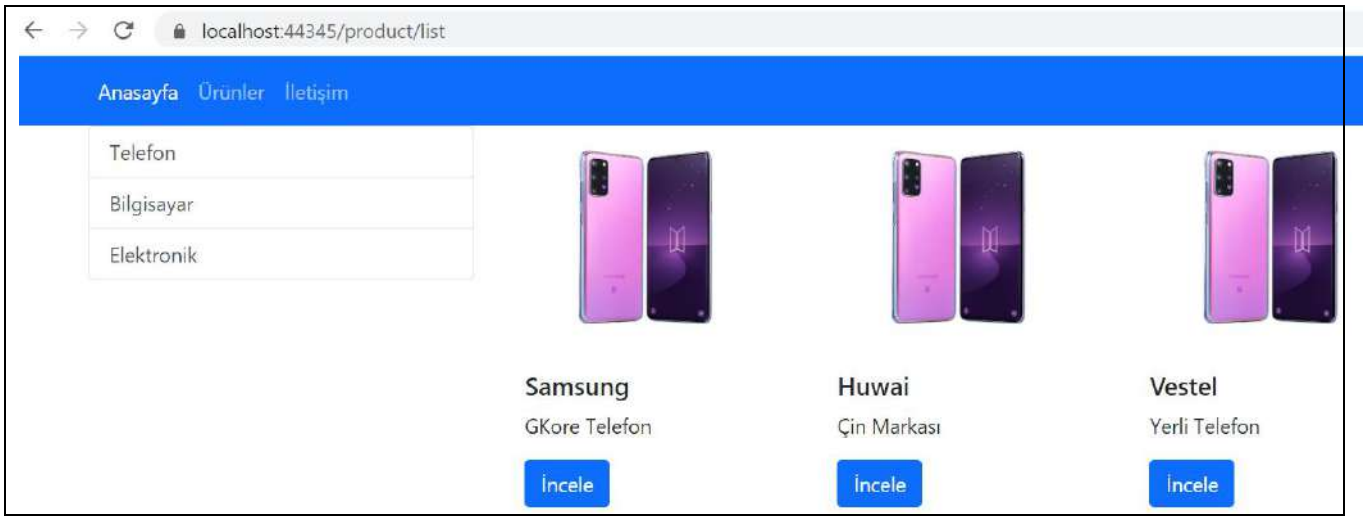
```
<div class="list-group">
    @foreach (var Kategori in Model)
    {
        <a href="#" class="list-group-item list-group-
item-action">@Kategori.Ad</a>
    }
</div>
```

```
@{
    Layout = "_Layout";
}

<head>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta2/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-
BmbxuPwQa2lc/FVzBcNJ7UAyJxM6wuqIj61tLrc4wSX0szH/Ev+nYR
RuWlolflfl" crossorigin="anonymous">
</head>

<body>

    <main>
        <div class="col-md-3">
            @await Html.PartialAsync("_Kategoriler",
Model)

@await Component.InvokeAsync("Categories")

        </div>

    </main>
</body>
```

List.cshtml

```
@model
NetCoreProjesi2.ViewModels.KategoriVeUrun_ViewModel

@{
    var Kategoriler = Model.Kategoriler;
    var Urunler = Model.Urunler;
}

@{
    Layout = "_Layout";
}

<div class="row">

    <div class="col-md-3">
        @await Html.PartialAsync("_Kategoriler",
Kategoriler)
    </div>

    <div class="col-md-9">

        @if (Urunler.Count > 0)
        {
            <div class="row">
                @foreach (var Urun in Urunler)
                {
                    <div class="col-md-3">
                        @*Dikkat kart görünümü share
klasörü altından alıyor*@
                        @await
Html.PartialAsync("_card", Urun)
                    </div>
                }
            </div>
        }
        else
        {
            <div class="row">
                <div class="col-md-12">
                    <div class="alert alert-danger">
                        Ürün Bulunamadı
                    </div>
                </div>
            </div>
        }
    </div>
</div>
```

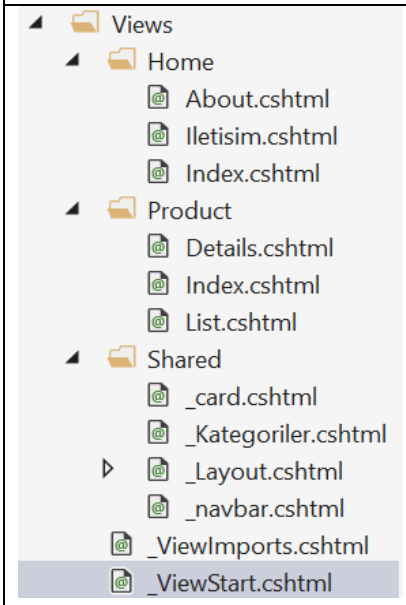




Dikkat edilirse buradaki iki sayfamızın üst kısmındaki NavBar , layout sayfamızdan geldi. Ve biz Layout şablon sayfamızı tüm sayfalarda olmasını istiyorsak, her sayfanın üst kısmına aşağıdaki kodları yazmamız gerekiyordu.

```
@{
    Layout = "_Layout";
}
```

Her sayfaya bu kodları yazmak yerine Views altındaki tüm sayfalarda bu _Layout şablonun olmasını istiyorsak tek bir yere yazarak bu görünümün tüm sayfalarda çıkmasını sağlayabiliriz. Bunun için yazacağımız yere Views altındaki ilk seviye içerisidir. Buraya ekleyeceğimiz boş bir **_ViewStart.cshtml** dosyası içerisine bu kodları ekleyelim. List ve Index sayfalarından da bu kodları kaldıralım. Dikkat edilirse orada bir tanede _ViewImports.cshml sayfası vardır. Bu sayfada tüm sayfalara eklememiz gereken bir kütüphane (bildirim) varsa onu tek bir merkezden eklemek içindir. Daha önce kullanmıştık. Onun içeriğini de buradan verelim.



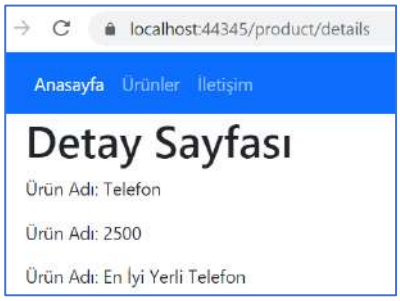
_ViewStart.cshtml

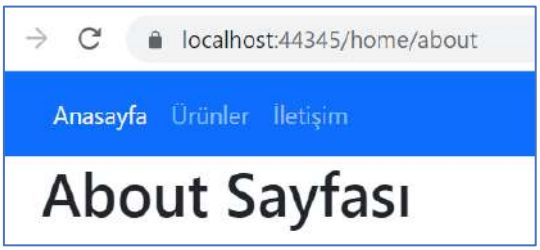
```
@*Buraya yazdığımız kodlar tüm sayfalara otomatik olarak tek bir yerden eklenmiş olmaktadır
Böylece aşağıdaki Layout şablonu tüm sayfalara eklenmiş olacaktır.*@

@{
    Layout = "_Layout";
}
```

_ViewImports.cshtml

```
@*Bu dosya tüm sitedeki Kütüphaneleri her sayfada çağırma işini tek bir yerden yapmamızı sağlıyor.*@

@using NetCoreProjesi2.Models
```

Projemizi çalıştırdığımızda bu sefer daha önce Layout şablonunu eklemediğimiz diğer bütün sayfalarda onun içindeki Navbar çubuğu gelecektir.





Bu şekilde _Layout tanımlamasını merkezi bir yerden yaptığımızda tüm sayfalara eklenmesi sorun olabilir. Tüm sitedeki sayfalarda olsun fakat çok özel bir sayfada belki bu NavBar ı gösteren Layout sayfasının bulunmasını istemeyiz. Bu durumda sadece özel olarak istemediğimiz sayfaların üzerine şu kodu eklersek artık bu sayfa oralara eklenmeyecektir.

Örneğin About sayfası içerisine bu kodu eklersek artık yukarıda gösterildiği gibi Navbar gelmeyecektir.

About.cshtml

```
@{
    Layout = null;
}

<h1>About Sayfası</h1>
```

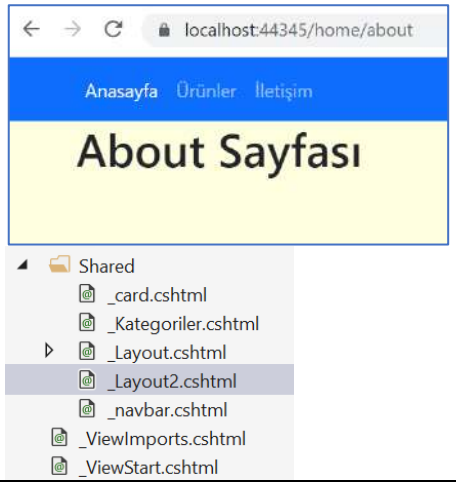
Buradaki kullanımda _Layout sayfamız sitede bir tane tipte kullanılmış oldu. Oysa bir sitede Yönetici sayfalarının belki _Layout şablonlarını farklı isteyebiliriz. Örneğin buradaki About sayfasına farlı bir _layout yerleşimi oluşturalım. Arkaplan renkleri bu sayfalarda farklı çıkmış olsun. Bunun için **_Layout2.cshtml** sayfası oluşturalım.

_Layout2.cshtml

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta2/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-
BmbxuPwQa2lc/FVzBcNJ7UAyJxM6wuqIj61tLrc4wSX0szH/Ev+nYRRuWlolflfl"
crossorigin="anonymous">
</head>
<body style="background-color:lightyellow ">
    @*Sayfaya navbar kısmı partial olarak _navbar.cshtml
dosyasından geliyor.*@
    @await Html.PartialAsync("_navbar")

        <main>
            <div class="container">
                @RenderBody()
            </div>
        </main>

</body>
</html>
```

About.cshtml

```
@{
    Layout = "_Layout2";
}

<h1>About Sayfası</h1>
```



(Section) _Layout sayfasında istediğimiz sayfa parçalarını görüntüleme

Yukarıdaki uygulamada _Layout şablon sayfası içinde tanımlanan bilgiler bağlantılı olduğu tüm sayfalarda çıkıyordu. Oysa _Layout sayfası içine konulan bilgilerin tüm sayfalarda değil istenilen sayfalarda çıkmasını istediğimiz durumlarda olabilir. Örneğin bir Alert (uyarı) mesajı Ana sayfada (Index sayfası) karşımıza gelsin fakat diğer sayfalarda çıkmasın. Bu uyarı sayfası da _Layout içerisine konulmuş olsun. Bu işlem için aşağıdaki adımları takip edelim.

Örnek 1:

Tüm sayfalarda olmasını istiyoruz fakat bunun zorunlu olmasını istemiyorsak _Layout sayfasının içerisine aşağıdaki kodları ekleyelim. Buradaki false ifadesi zorunluğu kaldırır.

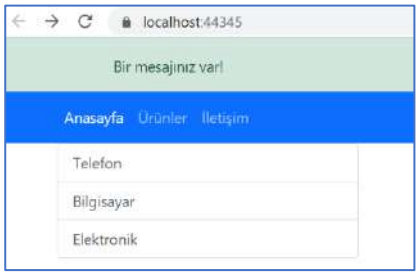
_Layout.cshtml içine ekle

```
@RenderSection("MessageBox", false)
```

Bu section kullanacak olan sayfaların içerisine de aşağıdaki ifadeleri ekleyelim. Örneğin Index sayfasında alert (uyarı) mesajının çıkmasını istiyoruz. Bunun içerisine aşağıdaki kodları ekleyelim.

Index.cshtml içine ekle

```
@section MessageBox
 {
     <div class="alert alert-success text-center m-0">
         Bir mesajınız var!
     </div>
 }
```



Peki burada _Layout sayfası içerisine Render ifadesini niye koyuyoruz. Aslında olay şöyle. Index sayfasının üst kısımlarının tasarımı _Layout sayfasından geliyor. Örneğin üst kısımda NavBar var. Eğer biz mesajı Index sayfasında üst kısmana koysak en üstte gösteremeyiz. Tasarımda üst kısımlar _Layout sayfasından geliyor.

O zaman Mesaj kutusunu direk _Layout içerisine üst kısma koyalım. Bu seferde mesaj kutusu tüm sayfalarda gözükmeye başlar. Bu durumda Index sayfa gibi alt sayfaların hangilerinde mesaj kutusu çıkmasını istiyorsak, mesaj kutusunun kodlarını o sayfanın içerisine herhangi bir yere "section" olarak koyuyoruz. Üstte olması şart değil sayfanın içinde herhangi bir yerde olabilir. Çünkü bu bir bölüm (section) dır ve tasarımda çağrıldığı yede gözükecektir. Çağırma işlemide _Layout sayfasında yapılmaktadır. Layout sayfasından çağırırken "false" ifadesini de kullanıyoruz ki, alt sayfaların hepsinin içine mesaj kutusu var mı diye bakacaktır. Bulamazsa hata vermesin.

Örnek 2: Bir önceki örnekte messagebox ı hangi sayfada kullandıysak orada görüntüleme yaptık. Mesaj kutusunun olmadığı sayfalarda da başka bir ifade çıkmasını isteyebilir miyiz? Örneğin kişi ile ilgili bir uyarı vereceksek bunu mesaj görüntü içinde verdik. Peki bir uyarı vermeyeceksek ona hoş geldin demek istesek ve bunu tüm sayfalarda yapmak istesek nasıl bir uygulama yapmalıyız. Şimdi onu görelim.

Burada _Layout sayfası içinde If yapısını kullanarak Section kısmı açılan sayfalarda Uyarı mesajı (alert) çıksın. Section bölümü açılmayan sayfalarda ise Hoş geldin! mesajı çıkartalım.

_Layout.cshtml için aşağıdaki renkli kısım eklendi.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta2/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-BmbxuPwQa2lc/FVzBcNJ7UAyJxM6wuqIj61tLrc4wSX0szH/Ev+nYRRuWlolflfl" crossorigin="anonymous">
</head>
<body>
    @if (IsSectionDefined("MessageBox"))
    {
        @RenderSection("MessageBox")
    }
    else
    {
        <div class="alert alert-success text-center m-0">
            Hosgeldiniz!
        </div>
    }

@*Sayfaya navbar kısmı partial olarak _navbar.cshtml dosyasından geliyor.*@
@await Html.PartialAsync("_navbar")

        <main>
            <div class="container">
                @RenderBody()
            </div>
        </main>

    </body>
</html>
```

Index.cshtml içine aşağıdaki renkli kısım eklendi. Diğer sayfalara eklenmedi.

```
@model List<Kategori_Model>

<head>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta2/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-BmbxuPwQa2lc/FVzBcNJ7UAyJxM6wuqIj61tLrc4wSX0szH/Ev+nYRRuWlolflfl" crossorigin="anonymous">
</head>

<body>
    @section MessageBox
    {
        <div class="alert alert-warning text-center m-0">
            Bir mesajınız var!
        </div>
    }

    <main>
        <div class="col-md-3">
            @await Html.PartialAsync("_Kategoriler", Model)
        </div>

    </main>
</body>
```

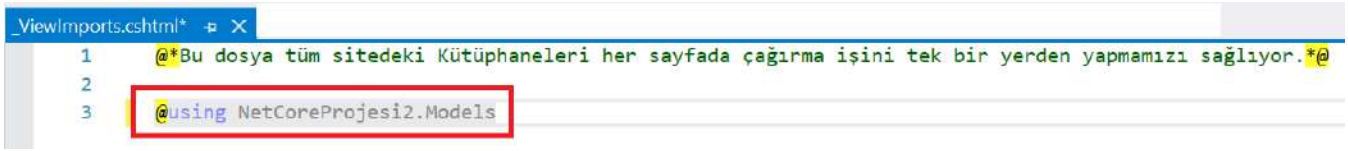
Not: Sayfalara @Section bölümü konulursa bunu tarayacak kod mutlaka _Layout sayfasında olmalıdır. Bu kod If yapısı içine gizlenmiş olsa bile bulunmalıdır.

Örnek 3: Index ana sayfamızda Kategorilerin olmasını istemeyebiliriz. Ama List sayfasında ise kategorilerin ve ürünlerin olmasını isteyelim. Benzer şekilde sitenin başka sayfalarında da bu şekilde uygulamalara ihtiyaç duyabiliriz. Böyle bir durumda Kategorilerin olduğu sayfalarda sol taraftaki alanı 3 kolon genişliğinde, sağ tarafındaki kısımları ise 9 kolon genişliğinde tasarlamamız gerekir. Kategori olmayan sayfalarda ise tüm içerik 12 kolon içinde gösterilmeli. Bu uygulamayı Section kullanarak yapalım.

Böyle bir uygulama _Layout sayfası içinde daha tasarımın nasıl olması gerektiğini de belirlememiz gerekir. Şimdi bunun kodlarına bakalım.

Dikkat: Projenin klasörlere kadar olan ana yolu _ViewImports.cshtml dosyası içine aşağıdaki şekilde yazıldığında artık tüm sayfalarda model tanımlarken buraya kadar olan yolu yazmak gerektirmez. Eğer aşağıdaki gibi tanımlama yapılmadı ise model bildirimleri şu şekilde yapılmalıdır.
`@model List<NetCoreProjesi2.Models.UrunModel>`
Eğer aşağıdaki gibi tanımlama yapıldı ise model tanımlamaları şu şekilde olabilir.

`@model List< UrunModel>`
Kodları denerken buna dikkat edilmelidir.



Index.cshtml

```
@model List<Urun_Model>

<head>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta2/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-
BmbxuPwQa2lc/FVzBcNJ7UAyJxM6wuqIj61tLrc4wSX0szH/Ev+nYRRuWlolf
lfl" crossorigin="anonymous">
</head>

<body>
    @section MessageBox
    {
        <div class="alert alert-warning text-center m-0">
            Bir mesajınız var!
        </div>
    }

    @*RenderBody içinde gözükecek bilgiler *@

    @if (Model.Count > 0)
    {
        <div class="row">
            @foreach (var Urun in Model)
            {
                //Elimizde 3 tane ürün olduğu için satır 4
lük dilimler halinde bölüyoruz. 4x3
                <div class="col-md-4">
                    @await Html.PartialAsync("_card", Urun)

                </div>
            }
        </div>
    }
    else
    {
            <div class="alert alert-danger">
                Ürün Bulunamadı
            </div>
    }

</body>
```

List.cshtml

```
@model
NetCoreProjesi2.ViewModels.KategoriVeUrun_ViewMo
del

@{
    var Kategoriler = Model.Kategoriler;
    var Urunler = Model.Urunler;
}

@*Kategoriler bölümü*@

@section Categories
{
    @await Html.PartialAsync("_Kategoriler",
Kategoriler)
}

@* RenderBody içinde gözükecek bilgiler *@

@if (Urunler.Count > 0)
{
    <div class="row">
        @foreach (var Urun in Urunler)
        {
            //Elimizde 3 tane ürün olduğu için
satırı 4 lük dilimler halinde bölüyoruz. 4x3=12
            <div class="col-md-4">
                @await
Html.PartialAsync("_card", Urun)
            </div>
        }
    </div>
}
else
{
    <div class="alert alert-danger">
        Ürün Bulunamadı
    </div>
}
```

_Layout.cshtml

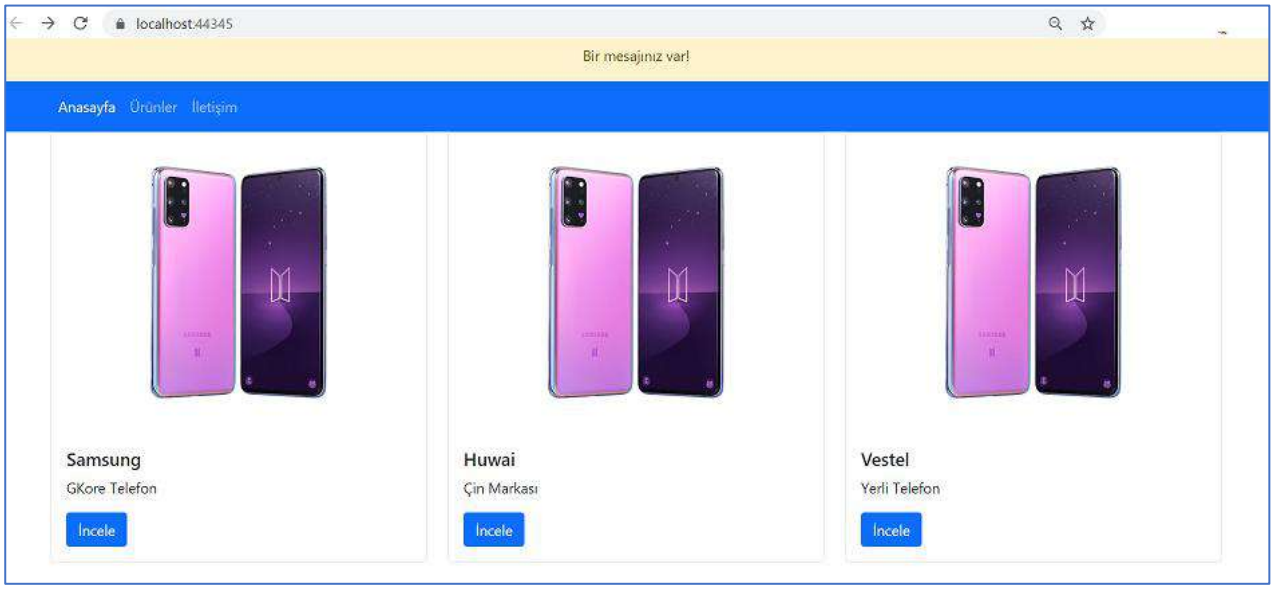
HomeController.cshtml sayfası içine eklenen

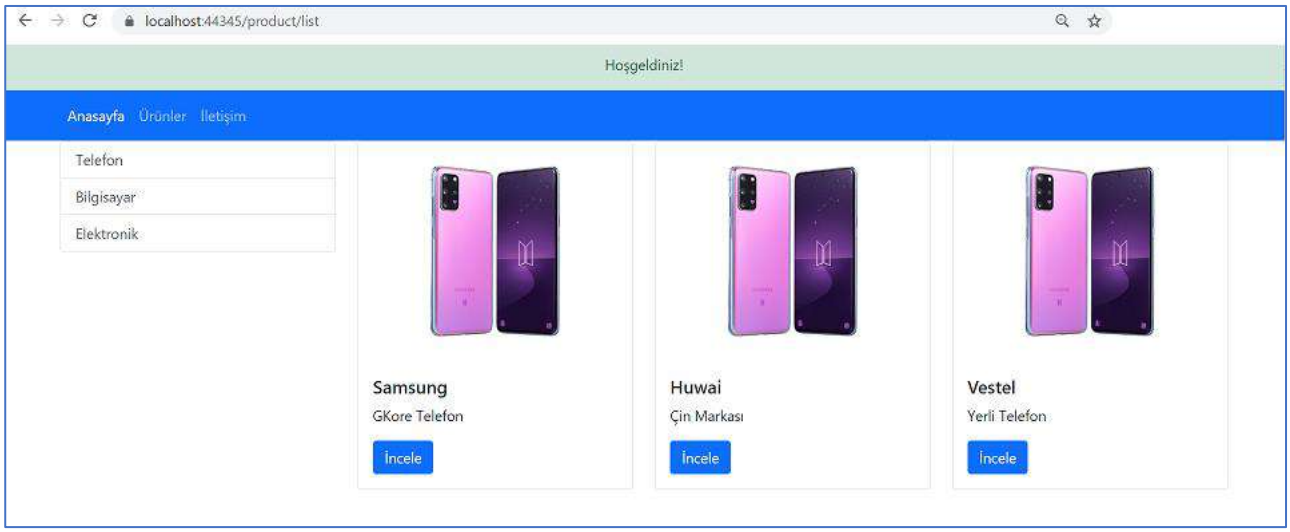
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta2/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-
BmbxuPwQa2lc/FVzBcNJ7UAyJxM6wuqIj61tLrc4wSX0szH/Ev+nYRRuWlolf
lfl" crossorigin="anonymous">
</head>
<body>
    @if (IsSectionDefined("MessageBox"))
    {
        @RenderSection("MessageBox")
    }
    else
    {
        <div class="alert alert-success text-center m-0">
            Hoşgeldiniz!
        </div>
    }

    @await Html.PartialAsync("_navbar")

    <main>
        <div class="container">

            @if (IsSectionDefined("Categories"))
            {
                <div class="row">
                    <div class="col-md-3">
                        @RenderSection("Categories")
                    </div>
                    <div class="col-md-9">
                        @RenderBody()
                    </div>
                </div>
            }
            else
            {
                <div class="row">
                    <div class="col-md-12">
                        @RenderBody()
                    </div>
                </div>
            }

        </div>
    </main>

</body>
</html>
```

```
public IActionResult Index()
{
    var UrunListesi = new List<Urun_Model>()
    {
            new Urun_Model{Ad= "Samsung", Fiyat
= 2000,Tanim = "GKore Telefon", StokDurumu=false
},
            new Urun_Model{Ad= "Huwai", Fiyat =
2500,Tanim = "Çin Markası", StokDurumu=true },
            new Urun_Model{Ad= "Vestel", Fiyat =
1500,Tanim = "Yerli Telefon", StokDurumu=true }
    };

    return View(UrunListesi);
}
```

ProductController.cshtml sayfası içine eklenenler

```
public IActionResult list()
{

    var KategoriListesi = new
List<Kategori_Model>()
    {
            new Kategori_Model{Ad= "Telefon",
Aciklama="Akıllı Telefonlar"},
            new Kategori_Model{Ad= "Bilgisayar",
Aciklama ="Diz Üstü Bilgisayarlar"},
            new Kategori_Model{Ad= "Elektronik",
Aciklama ="Tüketici Elektroniği Ürünleri"}
    };

    var UrunListesi = new List<Urun_Model>()
    {
            new Urun_Model{Ad= "Samsung", Fiyat
= 2000,Tanim = "GKore Telefon", StokDurumu=false
},
            new Urun_Model{Ad= "Huwai", Fiyat =
2500,Tanim = "Çin Markası", StokDurumu=true },
            new Urun_Model{Ad= "Vestel", Fiyat =
1500,Tanim = "Yerli Telefon", StokDurumu=true }
    };

    var KategorilerVeUrunler = new
KategoriVeUrun_ViewModel()
    {
        Kategoriler = KategoriListesi,
        Urunler = UrunListesi
    };

    return View(KategorilerVeUrunler);
}
```





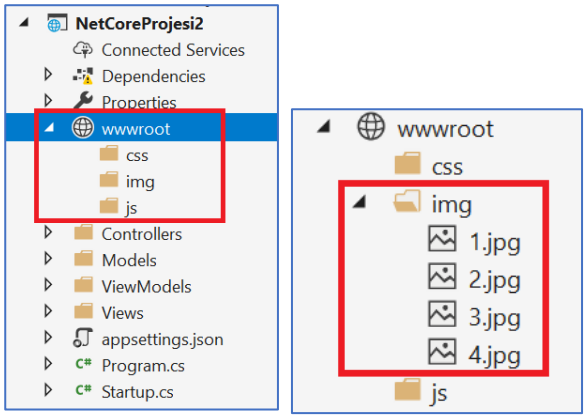
(Static Files) Site içerisinden (resimler, css, js) sabit dosyaları okutma

Projemiz içerisinde çeşitli sabit dosyaları kullanmamız gerekebilir. Bunlar Bootstrap dosyaları, Css dosyaları Js dosyaları, resimler vs olabilir. Bu dosyaları proje içerisinde kullanabilmemiz için **wwwroot** klasörünün altında **css, js, img** gibi sabit klasörler oluşturup bunların içerisine atabiliriz. Fakat serverın wwwroot görebilmesi için bunun için aşağıdaki ifadeyi **startup.cs** dosyasının içerisindeki Configure metodu içine gösterildiği şekilde **app.UseStaticFiles();** ifadesini yazmalıyız.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{

app.UseStaticFiles();

}
```

Şimdi ana dizin altına wwwroot isimli bir klasör ekleyelim ve bunun içerisine de css, img ve js klasörlerini oluşturalım. Klasör eklemek için proje üzerine sağ tuşa tıklayıp Add/New Folder seçebiliriz. Resimler klasörü içerisine birkaç tane resim ekleyelim. Ve yukarıdaki projelerde kullandığımız ürün resimlerini buradan okutalım. Dışarıdan **img** klasörü içerisine resimlerimizi yükleyelim. Projemize geldiğimizde resimler artık tanınır durumdadır.

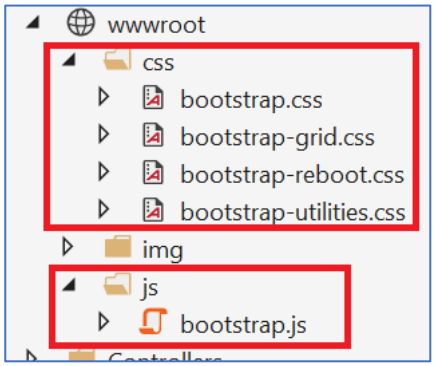
Ürünleri görüntülediğimiz **_Card.cshtml** parçalı dosyasının içerisinde internetten okuduğumuz resimlerin yolunu kendi projemiz olarak değiştirebiliriz.




```
@model Urun_Model


<div class="card">
    <img class="card-img-top" src="~/img/2.jpg" alt="" />
    <div class="card-body">
        <h5 class="card-title">@Model.Ad</h5>
        <p class="card-text">@Model.Tanim </p>
        <a class="btn btn-primary">İncele</a>
    </div>
</div>
```

Şimdi bootstrap dosyalarını proje içerisine ekleyelim.

İndirilen css ve js içindeki dosyaları aynı isimdeki projemizin kalasörleri içine kopyalayalım. Artık projemiz içinde dosyaları görebiliyoruz. Bootstrap kütüphanesine bağlanmak için kullandığımız linki artık kaldırabiliriz. Bu linki tüm sayfalar için şablon olarak kullandığımız _Layout.cshtml dosyası içine yazmıştık. Buradan linkimizi kaldırıp aşağıdaki satırı onun yerine ekleyelim.



Artık bootstrap dosyalarını kullanmak için aşağıdaki linki yazabiliriz.

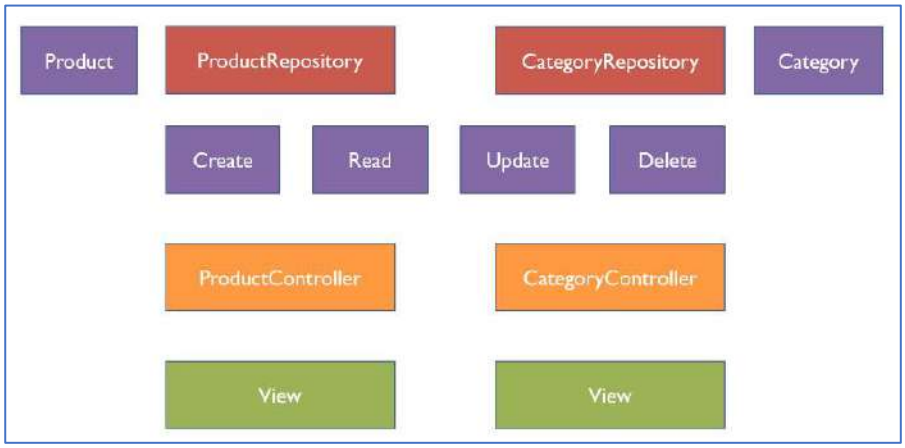
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <link href="~/css/bootstrap.min.css" rel="stylesheet"> @*Bootsrap: 5.0.0 Versiyonu
yüklendi*@
</head>
<body>
```

Sayfalarımızı deneyelim (Index ve List) tekrar çalıştığını görüyoruz. Ürün resimlerimizide atık yerel klasörden çağırıyoruz.

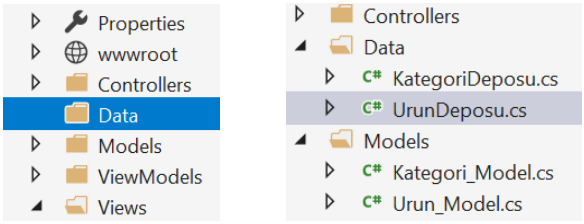
DİNAMİK VERİ İLE ÇALIŞMA

(Repository) Sanal Veritabanı işlemleri

Veritabanı işlemlerine hazırlık olması için alt yapıyı oluşturacağız. Veritabanında yer alacak bilgilerin yerine geçecek bilgileri tutmak için Repository (depo) sınıfları oluşturacağız. Bu sınıflardan alınan bilgileri ise 4 tane temel işlemden geçireceğiz (Yeni kayıt (create), Listeleme (Read), Güncelleme (Update), Kayıt silme (Delete). Yani sanal bir veritabanı oluşturmuş olacağız.



Bunun için Ana dizinde bir **Data** klasörü oluşturalım. Bu klasör içinde ürünleri (product) **UrunDeposu** (ProductRepository) classı içinde tutalım. Kategorileri ise **KategoriDeposu** (CategoryRepository) içinde tutalım. Repositorylerin dosya yapısı içinde C# Class yapısını örneğini formatında ekleyeceğiz.



Depo içindeki bilgiler taşınırken ise Model sınıfları ile taşınacaktır. Yani Kategorileri taşırken **KategoriModel** sınıfı, Ürünler içinde **UrunModel** sınıfı kullanılacaktır.

Ürün modeli içine **ID** ve **Resim** bilgilerinide ekleyelim. Oluşturacağımız depo bilgileri **static** olarak tanımlandı. Böylece kopyası üretilemeyecek, sabit bir sınıf olacak. (`public static class UrunDeposu`)

Bu bölüm için düzenlenen dosyaların içerikleri şu şekilde olmuştur. Açıklamalar kodlar arasına yazılmıştır.

UrunDeposu.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using NetCoreProjesi2.Models;

namespace NetCoreProjesi2.Data
{
    public static class UrunDeposu
    {
        //Ürünlerin tutulacağı listenin tanımlanması.
        private static List<UrunModel> _urunListesi =
null;

        //Veritabanı gibi kullanılacak ürünlerin
listesi. Yukarıda tanımlandı burada içi dolduruluyor.
        static UrunDeposu()
        {
            _urunListesi = new List<UrunModel>
            {
                new UrunModel{UrunID=1, Ad= "Samsung",
Fiyat = 2000,Ozellik = "GKore Telefon",
OnayDurumu=false, ResimAdi="1.jpg" },
                new UrunModel{UrunID=2, Ad= "Huwai",
Fiyat = 2500,Ozellik = "Çin Markası", OnayDurumu=true,
ResimAdi="2.jpg" },
                new UrunModel{UrunID=3, Ad= "Vestel",
Fiyat = 1500,Ozellik = "Yerli Telefon",
OnayDurumu=true, ResimAdi="3.jpg" },
                new UrunModel{UrunID=4, Ad=
"GeneralMobile", Fiyat = 2200,Ozellik = "Yerli
Telefon", OnayDurumu=true, ResimAdi="4.jpg" }
```

KategoriDeposu.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using NetCoreProjesi2.Models;

namespace NetCoreProjesi2.Data
{
    public class KategoriDeposu
    {
        //Kategorilerin tutulacağı listenin
tanımlanması.
        private static List<KategoriModel>
_kategoriListesi = null;

        //Veritabanı gibi kullanılacak kategorilerin
listesi. Yukarıda tanımlandı burada içi dolduruluyor.
        static KategoriDeposu()
        {
            _kategoriListesi = new List<KategoriModel>
            {
                new KategoriModel{KategoriID=1, Ad=
"Telefon", Aciklama="Akıllı Telefonlar"},
                new KategoriModel{KategoriID=2, Ad=
"Bilgisayar", Aciklama ="Diz Üstü Bilgisayarlar"},
                new KategoriModel{KategoriID=3, Ad=
"Elektronik", Aciklama ="Tüketici Elektroniği
Ürünleri"}
            };
        }
```

```
            };
        }

        //Bu metod listeleme işlemlerinde
kullanılacak. Dolayısı ile geri bir liste gönderecek.
        public static List<UrunModel> Urunler
        {
            get
            {
                return _urunListesi;
            }
        }

        //Bu metod ürün eklemek için kullanılacak.
Dolayısı ile geri değer göndermeyecek.
        public static void UrunEkle(UrunModel urun)
        {
            _urunListesi.Add(urun);
        }

        //Bu motod ID ye bağlı ürün getirmek için
oluşturuldu.
        public static UrunModel  UrunGetirByID(int ID)
        {
            //Dışarıdan istenen ID nin karşığı ürün
listesindeki her bir ürünün UrunID si
            //ile karşılaştırılır. Karşılığı varsa
ürünü gönderir. Yoksa null gönderir.
            return _urunListesi.FirstOrDefault(p =>
p.UrunID == ID);
        }

    }
}
```

```
        //Bu metod listeleme işlemlerinde
kullanılacak. Dolayısı ile geri bir liste gönderecek.
        public static List<KategoriModel> Kategoriler
        {
            get
            {
                return _kategoriListesi;
            }
        }

        //Bu metod yeni kategori eklemek için
kullanılacak. Dolayısı ile geri değer göndermeyecek.
        public static void KategoriEkle(KategoriModel
kategori)
        {
            _kategoriListesi.Add(kategori);
        }

        //Bu motod ID ye bağlı kategori getirmek için
oluşturuldu.
        public static KategoriModel
KategoriGetirByID(int ID)
        {
            //Dışarıdan istenen ID nin karşığı
kategori listesindeki her bir kategorinin KategoriID
si
            //ile karşılaştırılır. Karşılığı varsa
kategori gönderir. Yoksa null gönderir.
            return _kategoriListesi.FirstOrDefault(k
=> k.KategoriID == ID);
        }

    }
}
```

UrunModel.cs

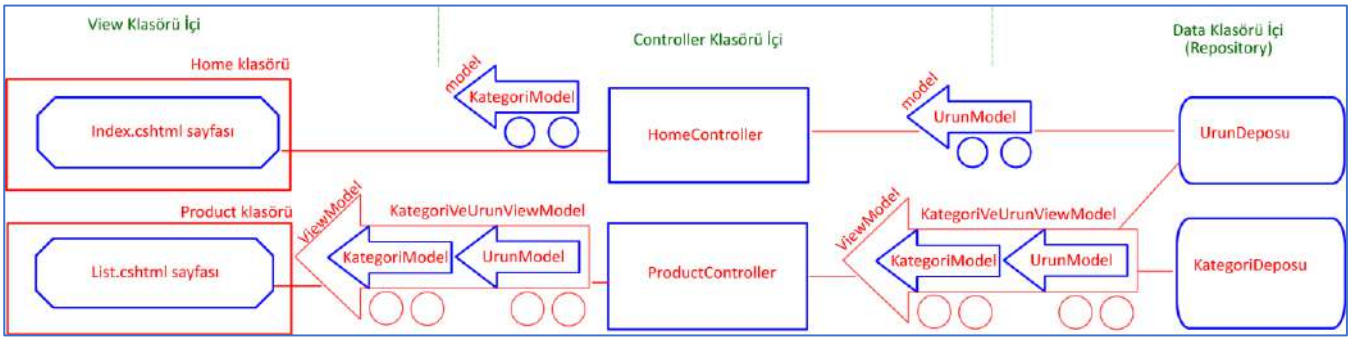
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace NetCoreProjesi2.Models //Dikkat namespace
adını bulunduğu klasörden aldı.
{
    public class UrunModel
    {   //Bu yazıyı kısayol şeklinde çıkarmak için
prog yazıp iki defa tab tuşuna tıklayın.
        public int UrunID { get; set; }
        public string Ad { get; set; }
        public double Fiyat { get; set; }
        public string Ozellik { get; set; }
        public string ResimAdi { get; set; }
        public bool OnayDurumu { get; set; }
    }
}
```

KategoriModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace NetCoreProjesi2.Models
{
    public class KategoriModel
    {
        public int KategoriID { get; set; }
        public string Ad { get; set; }
        public string Aciklama { get; set; }
    }
}
```

Listeleme Sayfası

Ürün ve Kategori bilgilerimiz artık Depolar (Repository) içindedir. Daha önceden bilgileri Controller içerisinden Views sayfalarına gönderiyorduk. Bundan sonra artık bilgilerimizi Repository den Controllera oradanda Views sayfalarına aktarabiliriz. Bu esnada taşıma işlemleri Model sınıfı içinde yapılacaktır.



Yukarıdaki bir önceki konudaki dosyalarda buranın devamıdır.

HomeController

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using NetCoreProjesi2.Models; //Sayfaya eklendi
using NetCoreProjesi2.ViewModels; //Sayfaya eklendi
using NetCoreProjesi2.Data; //Sayfaya eklendi


namespace NetCoreProjesi2.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            var Urunler = new List<UrunModel>();

            Urunler = UrunDeposu.Urunler;

            return View(Urunler);

        }
    }
}
```

ProductController

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using NetCoreProjesi2.Models; //Sayfaya eklendi
using NetCoreProjesi2.ViewModels; //Sayfaya eklendi
using NetCoreProjesi2.Data; //Sayfaya eklendi

namespace NetCoreProjesi2.Controllers
{
    public class ProductController : Controller
    {
        public IActionResult list()
        {

            var KategorilerVeUrunler = new
KategoriVeUrun_ViewModel()
            {
                Kategoriler =
KategoriDeposu.Kategoriler,
                Urunler =UrunDeposu.Urunler
            };

            return View(KategorilerVeUrunler);
        }
    }
}
```

Index.cshtml

```
@model List<NetCoreProjesi2.Models.UrunModel>


<body>
    @section MessageBox
    {
        <div class="alert alert-warning text-center m-
0">
            Bir mesajınız var!
        </div>
    }

    @*RenderBody bölümü*@

    @if (Model.Count > 0)
    {
        <div class="row">
            @foreach (var Urun in Model)
            {
                //Elimizde 4 tane ürün olduğu için
satır 3 lük dilimler halinde bölüyoruz. 4x3
                <div class="col-md-3">
                    @await Html.PartialAsync("_card",
Urun)
                </div>
            }
        </div>
    }
    else
    {
```

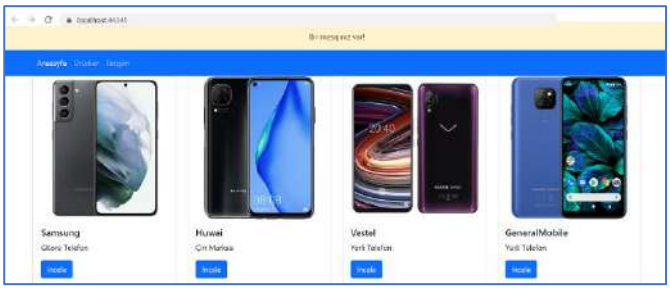
List.cshmtl

```
@model
NetCoreProjesi2.ViewModels.KategoriVeUrun_ViewModel

@{
    var Kategoriler = Model.Kategoriler;
    var Urunler = Model.Urunler;
}

@*Kategoriler bölümü*@

@section Categories
{
    @await Html.PartialAsync("_Kategoriler",
Kategoriler)
}

@*RenderBody bölümü*@

@if (Urunler.Count > 0)
{
    <div class="row">
        @foreach (var Urun in Urunler)
        {
            //Elimizde 3 tane ürün olduğu için satırı
4 lük dilimler halinde bölüyoruz. 4x3
            <div class="col-md-3">
                @await Html.PartialAsync("_card",
Urun)
            </div>
        }
```

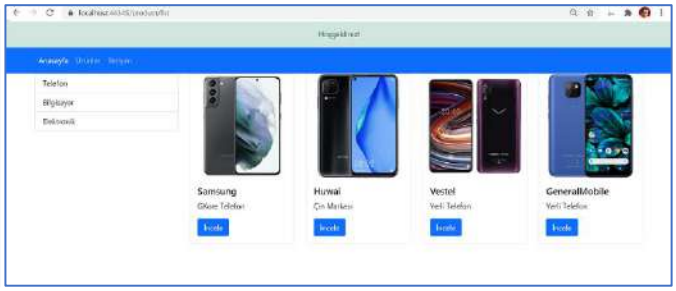
```
        <div class="alert alert-danger">
            Ürün Bulunamadı
        </div>
    }

</body>
```

```
    </div>
}
else
{
    <div class="alert alert-danger">
        Ürün Bulunamadı
    </div>
}
```

Localhost/ sayfası

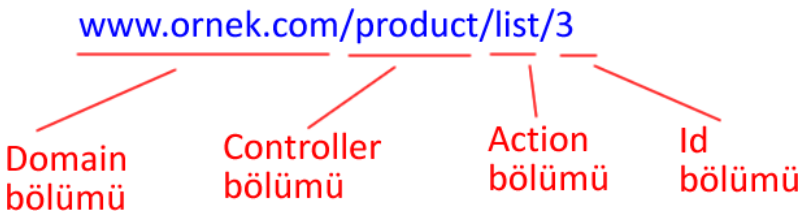


Product/List sayfası



Detay Sayfası

Adres satırına bir istek yazdığımızda bu istek üç bölümden oluşmaktadır. Bunlar: Controller bölümü, Action bölümü ve ID bölümüdür. Bu üç bölüm bir adres satırında şu şekildedir.



Bu isteği ilk karşılayan yer, **Startup.cs** içindeki Route kısmıdır.

Startup.cs

```
app.UseEndpoints(endpoints =>
          {
              endpoints.MapControllerRoute(
                  name: "default",
                  pattern: "{controller=home}/{action=index}/{id?}"
                  );
          });
```

Buradaki adres deseni (pattern) incelendiğinde bu üç bölüm açıkça gözükmektedir. Adres satırında domainden sonra herhangi bir şey yazılmazsa (Örneğin: http://www.ornek.com/) şeklinde yazdığımızda **Controller**, **action** ve **Id** bölümleri bulunmamaktadır. Bu durumda varsayılan değerlerin ne olacağı burada gösterilmiştir. Varsayılan değerler controller için "**home**", action için "**index**" dir. Id kısmının yanına konulan soru işaretinin bu üçüncü parametrenin zorunlu olmadığını, isteğe bağlı olduğunu gösterir. Yani Id yazılmasa da link çalışır.

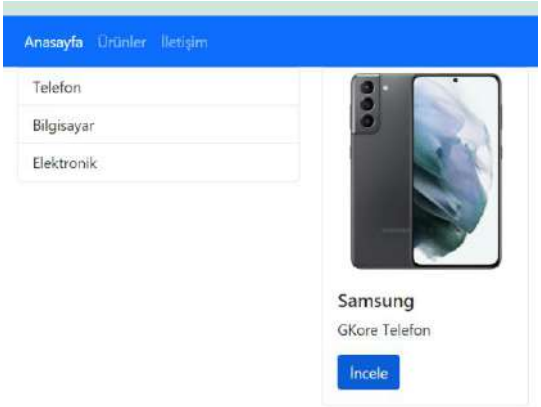
Startup.cs dosyasında bu route yazan yerden sonra yönlendirme ilgili Controller dosyasına olacaktır. Yani **controller=home** olursa **homecontroller** içine gidecektir. Yada **controller=product** olursa **productcontroller** içine gidecektir.

Şimdi ürünleri gösterdiğimiz Kart görünümün olduğu kodlara gidelim. Buradaki Butona tıklayınca ürünün detayını gösteren sayfaya göndersin. Bu sayfa "product" controlleri altında, "detail" action ile temsil edilecek. detail.cshtml içinde detay bilgiler gösterilecektir.

Ürünleri görüntülediğimiz neresi varsa onun Card görünümünün altında "İncele" butonu gelecektir. Bu butona tıklayınca detay sayfasına gidecek. Bu sayfaya giderken yanımızda ürüne ait ID yi de götüreceğiz. Ürün sayfası içinde ise Ürünün model bilgisinden gelen detay bilgileri gösterilecektir.

_card.cshtml

```
@model UrunModel

<div class="card">
    <img class="card-img-top"
src="~/img/@Model.ResimAdi" alt="" />
    <div class="card-body">
        <h5 class="card-title">@Model.Ad</h5>
        <p class="card-text">@Model.Ozellik </p>
        <a href="/product/Details/@Model.UrunID"
class="btn btn-primary">İncele</a>

        @*Linkleri verirken aşağıdaki şekildede
verebiliriz. Çalışmadı bak..
           <a asp-controller="Product" asp-
action="Details" asp-route-UrunID="@Model.UrunID"
class="btn btn-primary">İncele</a>*@

    </div>
</div>
```



UrunDeposu.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using NetCoreProjesi2.Models;

namespace NetCoreProjesi2.Data
{
    public static class UrunDeposu
    {
        //Ürünlerin tutulacağı listenin tanımlanması.
        private static List<UrunModel> _urunListesi = null;

        //Veritabanı gibi kullanılacak ürünlerin listesi.
Yukarıda tanımlandı burada içi dolduruluyor.
        static UrunDeposu()
        {
            _urunListesi = new List<UrunModel>
            {
                new UrunModel{UrunID=1, Ad= "Samsung", Fiyat =
2000,Ozellik = "GKore Telefon", OnayDurumu=false,
ResimAdi="1.jpg" },
                new UrunModel{UrunID=2, Ad= "Huwai", Fiyat =
2500,Ozellik = "Çin Markası", OnayDurumu=true,
ResimAdi="2.jpg" },
                new UrunModel{UrunID=3, Ad= "Vestel", Fiyat =
1500,Ozellik = "Yerli Telefon", OnayDurumu=true,
ResimAdi="3.jpg" },
                new UrunModel{UrunID=4, Ad= "GeneralMobile",
Fiyat = 2200,Ozellik = "Yerli Telefon", OnayDurumu=true,
ResimAdi="4.jpg" }
            };
        }

        //Bu metod listeleme işlemlerinde kullanılacak.
Dolayısı ile geri bir liste gönderecek.
        public static List<UrunModel> Urunler
        {
            get
            {
                return _urunListesi;
            }
        }

        //Bu metod ürün eklemek için kullanılacak. Dolayısı
ile geri değer göndermeyecek.
        public static void UrunEkle(UrunModel urun)
        {
            _urunListesi.Add(urun);
        }

        //Bu motod ID ye bağlı ürün getirmek için
oluşturuldu.Dikkat! giriş değişkeni int Id şeklinde
yazılmalıdır.
        public static UrunModel  UrunGetirByID(int ID)
        {
            //Dışarıdan istenen ID nin karşığı ürün
listesindeki her bir ürünün UrunID si
            //ile karşılaştırılır. Karşılığı varsa ürünü
gönderir. Yoksa null gönderir.
            return _urunListesi.FirstOrDefault(p => p.UrunID
== ID);
        }

    }
}
```

ProductController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using NetCoreProjesi2.Models; //Sayfaya eklendi
using NetCoreProjesi2.ViewModels; //Sayfaya eklendi
using NetCoreProjesi2.Data; //Sayfaya eklendi

namespace NetCoreProjesi2.Controllers
{
```

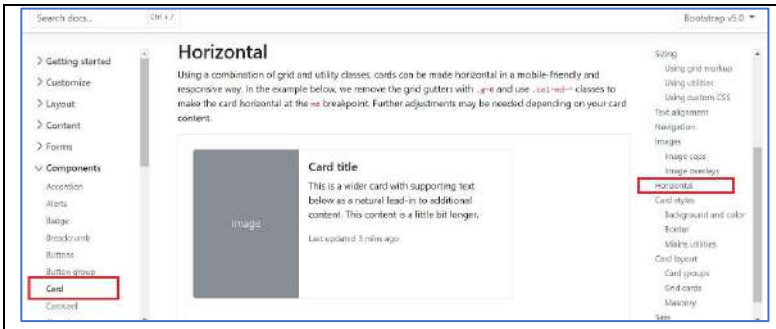
Details.cshmtl

```
@model NetCoreProjesi2.Models.UrunModel

<h1>Detay Sayfası</h1>

<p>Ürün ID: @Model.UrunID </p>
<p>Ürün Adı: @Model.Ad </p>
<p>Ürün Fiyatı: @Model.Fiyat </p>
<p>Ürün Özellikleri: @Model.Ozellik </p>
<p>Ürün ResimAdı: @Model.ResimAdi </p>
```

```
    public class ProductController : Controller
    {
        //localhost:5001/product/list linki için
metod.
        public IActionResult list()
        {

            var KategorilerVeUrunler = new
KategoriVeUrun_ViewModel()
            {
                Kategoriler =
KategoriDeposu.Kategoriler,
                Urunler =UrunDeposu.Urunler
            };

            return View(KategorilerVeUrunler);
        }

        //localhost:5001/product/details linki için
metod.
        public IActionResult details(int ID)
        {
            return View(UrunDeposu.UrunGetirByID(ID));
        }
    }
}
```

Detay Sayfasını Düzenleme: Şimdi detay sayfasını biraz daha kullanışlı hale getirelim. Ürün detayları listelenirken sol tarafta da Kategoriler gözüksün.

Detay sayfamızda ürün bilgilerini görüntülerken yine bir Bootsrap dan Kart gönümü ekleyelim. Bunun için Bootsrap sayfasındaki Horizontal kart görünümünü alalım.




```
<div class="card mb-3" style="max-width: 540px;">
 <div class="row g-0">
  <div class="col-md-4">
   <img src="..." class="img-fluid rounded-start" alt="...">
  </div>
  <div class="col-md-8">
   <div class="card-body">
    <h5 class="card-title">Card title</h5>
    <p class="card-text">This is a wider card with
supporting text below as a natural lead-in to additional
content. This content is a little bit longer.</p>
    <p class="card-text"><small class="text-muted">Last
updated 3 mins ago</small></p>
   </div>
  </div>
 </div>
</div>
```

Buradan alınan tasarım kodlarını aşağıdaki şekilde Details.cshtml sayfası içinde kullanabiliriz. Dikkat edilirse Details sayfasında Model içinde ürün bilgisi tutulmaktadır. Fakat bu sayfada aynı zamanda Kategori bilgileri de gösterilmektedir. ProductControllerdan Details sayfasına giderken yanımızda sadece Urun bilgilerini götürebilmekteyiz. Bu nedenle Details sayfasında Kategorileri başka bir yerden getirmekteyiz. Bu amaçla daha öncede kullandığımız ViewComponent sınıfını kullanmaktayız. Bu esnada _Layout sayfası ile Details sayfasını birleştirirken Section yapısını kullandık. (Bu konuyu daha önceki notlardan inceleyin)

_Layout.cshtml	**Details.cshtml**

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <link href="~/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
</head>
<body>
    @if (IsSectionDefined("MessageBoxSection"))
    {
        @RenderSection("MessageBoxSection")
    }
    else
    {
        <div class="alert alert-success text-center m-0">
            Hoşgeldiniz!
        </div>
    }
    @*Sayfaya navbar kısmı partial olarak _navbar.cshtml
dosyasından geliyor.*@
    @await Html.PartialAsync("_navbar")
    <main class="mt-3">
        <div class="container">
            @*İçerik sayfaları bu kısımda gösterilecek*@
             @if (IsSectionDefined("KategorilerSection"))
            {
                <div class="row">
                    <div class="col-md-3">
                        @RenderSection("KategorilerSection")
                    </div>
                    <div class="col-md-9">
                        @RenderBody()
                    </div>
                </div>
            }
            else
            {
                <div class="row">
                    <div class="col-md-12">
                        @RenderBody()
                    </div>
                </div>
            }
        </div>
    </main>
    <footer class="py-5 bg-dark text-white text-center">
        Hakkımızda | İletişim
    </footer>
</body>
</html>
```

```
@model UrunModel ;

@*********SECTION*************@

@section KategorilerSection
{
    @await
Component.InvokeAsync("Kategoriler")
}

@* ************ RENDERBODY ************ *@

@{
    string ResimAdi = "/img/" +
@Model.UrunID  + ".jpg";
}

<div class="card mb-3" >
    <div class="row g-0">
        <div class="col-md-4">
            <img src="@ResimAdi" class="img-
fluid rounded-start" height="300">
        </div>
        <div class="col-md-8">
            <div class="card-body">
                <h5 class="card-
title">@Model.Ad </h5>
                <p class="card-
text">@Model.Aciklama</p>
                <p class="card-text"><small
class="text-muted">3 gün önce </small></p>
            </div>
        </div>
    </div>
</div>
```

CategoriesViewComponent

```
public class CategoriesViewComponent :
ViewComponent
{
  public IViewComponentResult Invoke()
  {
    var KategoriListesi = new
List<CategoryModel>();
    KategoriListesi =
KategoriDeposu.Kategoriler;
    return View(KategoriListesi);
  }
}
```

ProductController.cs

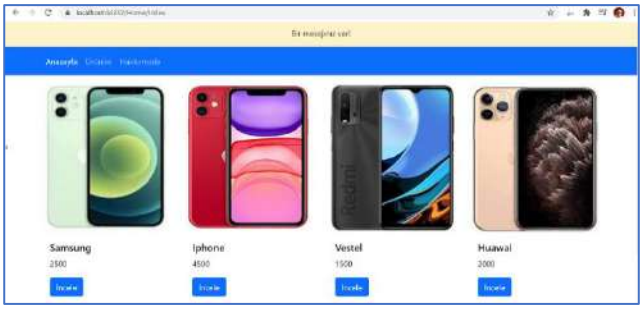
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using WebApplication1.Models;
using WebApplication1.Data;
using WebApplication1.ViewModels;

namespace WebApplication1.Controllers
{
    public class ProductController : Controller
    {
        //Views sayfaları hazır olduğunda buradan oraya
yönlendirme bu şekilde yapılacak.
        public IActionResult Index()
        {
            return View();

        }

        public IActionResult Details(int ID)
```

UrunRepository.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using WebApplication1.Models;

namespace WebApplication1.Data
{
    public static class UrunRepository
    {

        //Bu motod ID ye bağlı ürün getirmek
için oluşturuldu.
        public static UrunModel
UrunGetirByID(int UrunID)
        {
            //Dışarıdan istenen ID nin
karşığı ürün listesindeki her bir ürünün
UrunID si
            //ile karşılaştırılır. Karşılığı
varsa ürünü gönderir. Yoksa null gönderir.
```

```
        {//Dikkat bu fonksiyonda giriş değişkeni int ID şeklinde olmak zorundadır.
            return View(UrunRepository.UrunGetirByID(ID));
        }
    }
}
```

```
            return UrunListesi.FirstOrDefault(p => p.UrunID == UrunID);
        }


    }
}
```



Seçilen Kategoriye Göre Ürünleri Listeleme

Bu işlem için ürün bilgileri içinde bağlı olduğu Kategoriyi gösteren bir alanın olması gerekir. KategoriID ismiyle bir alanı ürünün model bilgisine ekleyelim. Ürünlerin bilgilerini tuttuğumuz UrunRepository.cs dosyamızın içinde de her ürünü bir Kategoriye bağlayalım.

Ürünleri Listelerken kullandığımız linkin sonuna bir sayı getirirsek "/Product/List/2" şeklinde yazdığımızda 2 numaralı kategoriye bağlı ürünleri listeleyelim. Ayrıca Kategoriler içine "Tüm Kategoriler" adıyla bir link ekleyelim. Buna tıklandığında Id kısmı boş bir link olsun ve tüm kategorileri getirsin.

UrunModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace WebApplication1.Models
{
    public class UrunModel
    {
        public int UrunID { get; set; }
        public int KategoriID { get; set; }
        public string Ad { get; set; }
        public double Fiyat { get; set; }
        public string Aciklama { get; set; }
        public bool StokDurumu { get; set; }
    }

  }
```

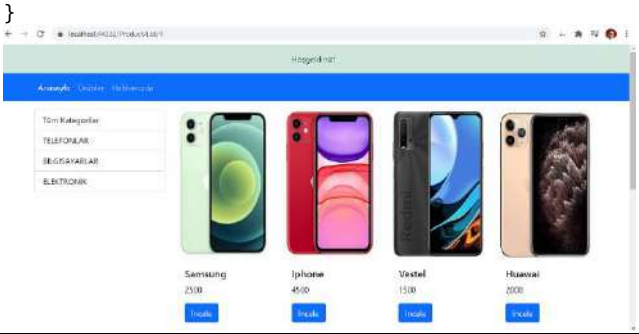
ProductController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using WebApplication1.Models; //Model sınıflarının çalışması için sayfaya eklendi.
using WebApplication1.Data;
using WebApplication1.ViewModels;
```

UrunRepository.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using WebApplication1.Models;

namespace WebApplication1.Data
{
    public static class UrunRepository
    {

        //Ürünlerin tutulacağı listenin tanımlanması.
        private static List<UrunModel> UrunListesi = null;

        static UrunRepository()
        {
            UrunListesi = new List<UrunModel>
            {
                new UrunModel {UrunId=1, KategoriId=1, Ad = "Samsung", Fiyat = 2500, Aciklama = "İyi Telefon", StokDurumu=false },
                new UrunModel {UrunId=2, KategoriId=1, Ad = "Iphone", Fiyat = 4500, Aciklama = "Pahalı Telefon", StokDurumu=true },
                new UrunModel {UrunId=3, KategoriId=1,  Ad = "Vestel", Fiyat = 1500, Aciklama = "Yerli Telefon", StokDurumu=true },
                new UrunModel {UrunId=4, KategoriId=1,  Ad = "Huawai", Fiyat = 2000, Aciklama = "Çin Telefonu", StokDurumu=true },
                new UrunModel {UrunId=5, KategoriId=1,  Ad = "Sony", Fiyat = 2000, Aciklama = "Japon Telefonu", StokDurumu=true },
                new UrunModel {UrunId=6, KategoriId=2, Ad = "Lenova", Fiyat = 6500, Aciklama = "İyi Bilgisayar", StokDurumu=false },
```

```
namespace WebApplication1.Controllers
{
    public class ProductController : Controller
    {
        public IActionResult List(int? Id) //? Id olsada olur, olmasada olur manasında.
        {
            var Urunler = UrunRepository.Urunler;

            if (Id != null)
            {
                Urunler = Urunler.Where(p => p.KategoriId == Id).ToList();
            }
            return View(Urunler);
        }
    }
}
```




```
                new UrunModel {UrunId=7, KategoriId=2, Ad = "Hp", Fiyat = 7500, Aciklama = "Güzel Bilgisayar", StokDurumu=true },
                new UrunModel {UrunId=8, KategoriId=2,  Ad = "Casper", Fiyat = 8000, Aciklama = "Yerli Bilgisayar", StokDurumu=true },
                new UrunModel {UrunId=9, KategoriId=2,  Ad = "Sony", Fiyat = 9000, Aciklama = "Japon Markası", StokDurumu=true }
                                    };
        }

    }
}
```

Shared>Components>Kategoriler>Default.cshtml

```
@model List<KategoriModel>

    <div class="list-group">

        <a href="/Product/List/" class="list-group-item list-group-item-action">Tüm Kategoriler</a>

        @foreach (var Kategori in Model)
        {
            <a href="/Product/List/@Kategori.KategoriId" class="list-group-item list-group-item-action">@Kategori.Ad</a>
        }
    </div>
```

Seçilen Kategorinin Aktif olarak gösterimi-Link İçindeki Bilgilerin Alınması

"/Product/List/3" gibi bir linkin içerisinden bilgileri aşağıdaki komutlar ile alabiliriz. List sayfasına gitmek istediğimizde eğer yanında Id bilgisi de varsa o zaman bir Kategori seçilmiş demektir. Seçilen kategorinin numarasına bağlı olarak Listeden o kategoriyi aktif renklere boyayabiliriz.

- RouteData.Values["Controller"]; ifadesi bize linkin (Route) içindeki "Product" bilgisini verir.
- RouteData.Values["Action"]; ifadesi bize linkin içinde "List" bilgisini verir.
- RouteData.Values["Id"]; ifadesi bize linkin içindeki "Id" bilgisini verir.

Kategorileri görüntülerken ViewComponent sınıfını kullanmıştık. Bu sınıfın sayfa görüntüleme kodları Share>Components>Kategoriler>Default.cshtml sayfası içinde idi. Bu sayfaya giderken Kategori bilgilerini Veritabanından (yada şu anda KategoriRepository) çekip götürmemizi sağlayan ise ViewComponents klasörü altında oluşturduğumuz KategorilerViewComponent.cs sayfası idi. İşte bu C# sayfasından html sayfasına Action ve Id bilgilerini ViewBag içinde götüreceğiz.

KategorilerViewComponent.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using WebApplication1.Data;

namespace WebApplication1.ViewComponents
{
    public class KategorilerViewComponent:ViewComponent
    {
        public IViewComponentResult Invoke()
        {
            var Kategoriler = KategoriRepository.Kategoriler;
            //Sadece List sayfasına gidildiğinde yanında Id bilgisi varsa bu bilgi kategori bilgisidir.
            //Oysa Details sayfasına da gidilmekte ve oradaki Id bilgisi Ürünün Id sidir.
            //Bu durumun karışmaması için if yapısı içinde list kontrolü yapıldı.
```

Shared>Components>Kategoriler>Default.cshtml

```
@model List<KategoriModel>

    <div class="list-group">

        <a href="/Product/List/"
           class="list-group-item list-group-item-action
           @(ViewBag.SecilenKategori == null?"active":"")  ">Tüm Kategoriler</a>

        @foreach (var Kategori in Model)
        {
            <a href="/Product/List/@Kategori.KategoriId"
               class="list-group-item list-group-item-action
               @(ViewBag.SecilenKategori == Kategori.KategoriId.ToString()?"active":"") ">
```

```
            //Dikkat burada ? işareti Id bilgisi gelmediği
zamanlar da olabiliceğinden bu durumda null alınacağını
gösterir.
            if (RouteData.Values["Action"].ToString() ==
"List" )
                ViewBag.SecilenKategori =
RouteData?.Values["Id"];

            return View(Kategoriler);
        }
    }
}
```

```
                @Kategori.Ad
            </a>
        }
    </div>
```

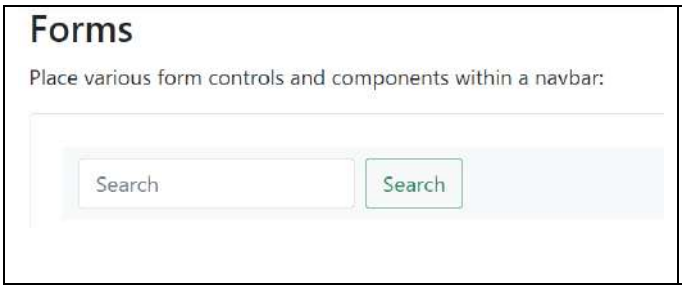
FORM İŞLEMLERİ

Arama Formu Hazırlama (Get Metodu)

Web sitemizde doldurmuş olduğumuz formları (içerisine bilgi girdiğimiz, textbox ların olduğu yapıları) servera gönderirken iki metod kullanırız. **Get** ve **Post** metodu. Bu metodlardan Get metodunu kullandığımızda bilgileri servera götürürken Linkin içinde, kullanıcılarında görebileceği şekilde götürülür. Örneğin google da bir arama yaptığımızda arama kelimelerinin servera giderken üstte link alanı içinde gözükmesi fazla bir sorun oluşturmaz ve bu şekilde gidebilir. Fakat şifre girişi yaptığımız bir formu doldurduğumuzda bu bilgilerin açıktan link içinden gitmesi sorun oluşturur. Bu durumda bilgileri götürürüken Post metodunu kullanmalıyız. Bu metod kullanıldığında bilgiler sayfa içinde dışarıya gösterilmeden götürülür.

Şimdi NavBar içinde basit bir arama formu oluşturalım. Bu forma yazdığımız anahtar kelimeyi içeren ürünleri sayfamızda görüntüleme yaptıralım. Bu işlem için Get metodu kullanacağız. Yani link içinde arama yapılan kelimeleri görebileceğiz.

Bunun için sitemizde kullandığımız NavBar içine, Boostrap dan indirdiğimiz bir form yapısını yerleştirelim. Dosyayı indirdiğimiz sayfanın linki (https://getbootstrap.com/docs/5.0/components/navbar/#forms) şeklindedir.




```
<nav class="navbar navbar-light bg-light">
  <div class="container-fluid">
    <form class="d-flex">
      <input class="form-control me-2" type="search"
placeholder="Search" aria-label="Search">
      <button class="btn btn-outline-success"
type="submit">Search</button>
    </form>
  </div>
</nav>
```

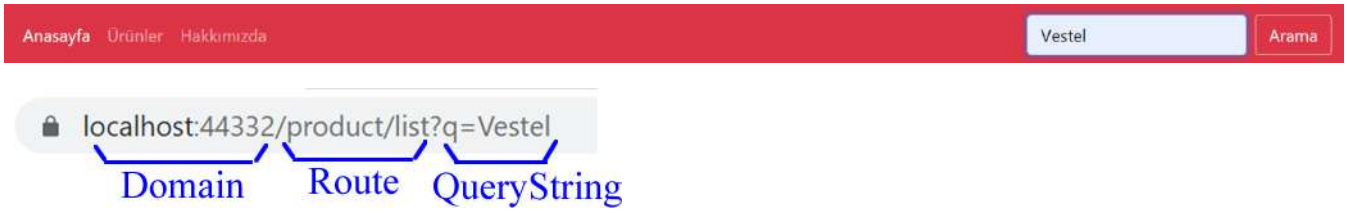
İndirdiğimiz kodların içindeki form etiketlerinin olduğu kısmı alıp NavBar oluşturduğumuz dosyanın içerisine yerleştirelim (_NavBar.cshtml). Orada dışına container içine kodları yerleştirelim. Böylece NavBar üzerinde sol tarafta linkler sağ tarafta ise arama formu olsun. Kodlar üzerinde ufak bazı değişikliklerde yapalım. _NavBar dosyamızın son hali aşağıda verilmiştir.

_NavBar.cshtml

```
@********NAVBAR*************@
    <nav class="navbar bg-danger navbar-dark navbar-expand-sm">
        <div class="container-fluid">

            <ul class="navbar-nav me-auto mb-2 mb-lg-0">
                <li class="nav-item">
                    <a class="nav-link active" aria-current="page" href="~/Home/Index">Anasayfa</a>
                </li>

                <li class="nav-item">
```

```
                    <a class="nav-link" href="~/Product/List">Ürünler</a>
                </li>

                <li class="nav-item">
                    <a class="nav-link" href="~/Home/About">Hakkımızda</a>
                </li>
            </ul>

            <form action="/product/list" class="d-flex">
                <input name="q" type="text" class="form-control me-2" placeholder="Arama yapın" aria-
label="Kelime Gir" />
                <button type="submit" class="btn btn-outline-light mr-0">Arama</button>
            </form>

        </div>
    </nav>
```

Bu hazırlıktan sonra textbox içerisine bir kelime yazdığımızda butona tıkladığımızda formun gönderileceği adres `action="/product/list"` metoduyla belirtilen adrestir. Bu adrese giderken yanımızda sorgulama parametrelerini de götürecektir ve parametreler linki içinde son kısma (soru işaretinden sonra) aşağıdaki şekilde eklenecektir. Dikkat edilirse burada "q" parametresi textbox da kullandığımız name="q" alanından gelmektedir. İçerisindeki bilgi de "Vestel" olarak geçmektedir. Burada linkin ana kısmı Route ismiyle anılacak, sorgulama kısımları ise Querystring adı ile anılacak.



Şimdi QueryString içinden sorgu parametrelerini ve değerleri nasıl alacağımızı görelim. Route kısmındaki adrese göre linkimiz bizi Product controller içindeki List metodunun olduğu yere götürecektir. Oradan parametreleri almamız gerekir.

```
public IActionResult List(int? Id, string q) //? Id olsada olur, olmasada olur manasında.
{
       Console.Write(q); //q parametresini alıp Consol da yazdırıyoruz.
       Console.Write(HttpContext.Request.Query["q"].ToString()); //Gelen istek içindeki query
ifadesindeki q paremetresinin içeriğini alıyoruz.
}
```

Şimdi arama işini yaptırmak için kullanacağımız kodları oluşturalım productcontroller son halini verelim.

ProductController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using WebApplication1.Models; //Model sınıflarının çalışması için sayfaya eklendi.
using WebApplication1.Data;
using WebApplication1.ViewModels;

namespace WebApplication1.Controllers
{
    public class ProductController : Controller
    {
        //Views sayfaları hazır olduğunda buradan oraya yönlendirme bu şekilde yapılacak.
        public IActionResult Index()
        {
            return View();
        }
```

```
        public IActionResult List(int? Id, string q) //? Id olsada olur, olmasada olur manasında.
        {
            var Urunler = UrunRepository.Urunler;

            if (Id != null)
            {
                Urunler = Urunler.Where(p => p.KategoriId == Id).ToList();
            }

            //Eğer q parametresi null yada boş değilse işlem yap.
            if (!string.IsNullOrEmpty(q))
            {
                //Urunlerin içinde arama yaparken i nesnesi kullanılacak. Bu nesne UrunModel sınıfı tipindedir.
                //O zaman i içindeki Ad alanı içinde ara. Eğer q parametresi içindeki bilgi varsa true gönder.
                //Yanlız bu işlemi yaparken hem Ad alanı hemde q parametresi küçük harfe çevriliyorki, büyük
küçük karakter sorunu çıkmasın.
                //Sonra veya (||) işlemi ile Aciklama alanı içinde de aynı şekilde arama yapıyoruz.
                Urunler = Urunler.Where(i => i.Ad.ToLower().Contains(q.ToLower()) ||
i.Aciklama.ToLower().Contains(q.ToLower())).ToList();
            }

            return View(Urunler);
        }

        public IActionResult Details(int ID)
        {//Dikkat bu fonksiyonda giriş değişkeni int ID şeklinde olmak zorundadır.

            return View(UrunRepository.UrunGetirByID(ID));
        }

    }
}
```

Not: Eğer arama parametremizi çoğaltmak istiyorsak, örneğin min ve maks fiyat aralığını getirmek istiyorsak bunu ilgili form elemanlarından aldıktan sonra linkimizi şu şekilde oluşturabiliriz.

https://localhost:44332/product/list?q=Vestel&min_fiyat=1000&maks_fiyat=5000

linkimizi karşılayan list metodunun girişi ise aşağıdaki şekilde olabilir.

```
public IActionResult List(int? Id, string q, double? min_Fiyat, double? maks_Fiyat)
```

İçerisinde fiyat değişkenlerinin null olmadığını tıpkı Id de olduğu gibi kontrol edip, içeriği dolu geldiyse sorgu içinde kullanabiliriz.

Eğer list sınıfının girişinden bu parametreleri almak istemiyorsak aşağıdaki Request içinden Query metoduyla da istediğimiz parametreleri alabiliriz.

```
HttpContext.Request.Query["min_Fiyat"].ToString();
```

Burada arama işlemi yaparken link içinde arama kelimelerinin görülmesi sorun teşkil etmediği için bu şekilde GET metodunu kullandık. Şimdi Kişisel bilgilerin girildiği bir kayıt formu oluşturalım ve bununda link içinde gözükmeyecek şekilde POST metodu ile gönderelim.

Ürün Kayıt Formu Hazırlama (Post Metodu)

Bu uygulamada bir ürün kayıt formu oluşturalım ve bilgileri servera Post Metodu ile (sayfa içinde gizleyerek) gönderelim. Önce ürün kayıt sayfamızı oluşturalım.

Bunun için NavBar içinde sayfamıza gidecek linki oluşturalım.

```
<li class="nav-item">
       <a class="nav-link" href="~/Product/Create">Yeni Ürün Ekle</a>
</li>
```

Burada oluşturduğumuz linkin Create sayfasına gidebilmesi için ProductController içinde buna ait metodu oluşturalım.

```
public IActionResult Create()
{
       return View();
}
```

Ardından Create sayfamızı oluşturalım. Bu sayfa içerisinde Kategoriler bulunmasın. Dolayısı ile sayfa genişliği 12 kolon olacaktır. Formumuz şimdilik basitçe aşağıdaki şekilde olsun. Formda kullanılan alan bilgilerini Ürün için oluşturulan Model yapısı ile uyumlu olduğuna dikkat edin. Kullanıcıya gösterilen kısımlar Türkçe yazıldı fakat arka plandaki alan bilgileri İngilizce kullanıldı

(**Not** bundan önceki sayfalarda bu kısımlarda Türkçe idi fakat sitenin yapısı uluslararası alışkanlıklara daha uygun olması için İngilizce kullanıldı. Bundan sonra arka planda kalan değişkenler ve alanlar İngilizce kullanılacaktır.)

Create.cshtml

```
<form method="GET">
    <div class="form-group">
        <label for="Name">Ürün Adı</label>
        <input type="text" name="Name" class="form-control">
    </div>
    <div class="form-group">
        <label for="Price">Fiyatı</label>
        <input type="text" name="Price" class="form-control">
    </div>
    <div class="form-group">
        <label for="Description">Açıklama</label>
        <textarea name="Description" class="form-control"></textarea>
    </div>
    <div class="form-group">
        <label for="ImageUrl">Resim Adresi</label>
        <input type="text" name="ImageUrl" class="form-control">
    </div>
    <div class="form-group">
        <label for="CategoryId">Kategori ID</label>
        <input type="text" name="CategoryId" class="form-control">
    </div>
    <button type="submit" class="btn btn-primary">KAYDET</button>
</form>
```

Siteye Header Ekleme:

Not: Sitemizin görünümünü biraz daha güzelleştirmek için NavBar altında bir tane Header ekleyelim. O kısım için içerisi boş bir reklam alanı olsun. Öncelikle Shared klasörü içerisine _Header.cshtml ismiyle bir tasarım parçası ekleyelim.

_Header.cshtml

```
    <img src="~/img/Header.jpg" class="img-fluid" alt="...">
```

Header uygulamasını Index ana sayfamızda kullanalım. Diğer sayfalarda zorunlu olmasın. Header parçasını tasarıma eklemek için Section yöntemini kullanalım. _Layout.cshtml sayfasına aşağıdaki kodu ekleyelim. Buradaki false ifadesi her sayfada kullanmanın zorunlu olmadığını gösterir. Malum _Layout sayfasına eklenen her tasarım tüm sayfalarda oluşacaktır.

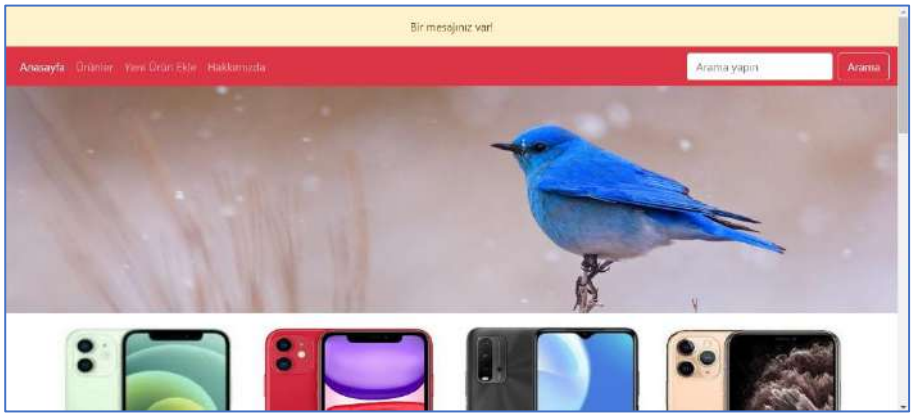
```
@RenderSection("HeaderSection",false)
```

Index.cshtml sayfası içerisine de aşağıdaki kodu ekleyelim. Header sayfa bilgisi burada Partial olarak Shared klasörü içinden alınmaktadır.

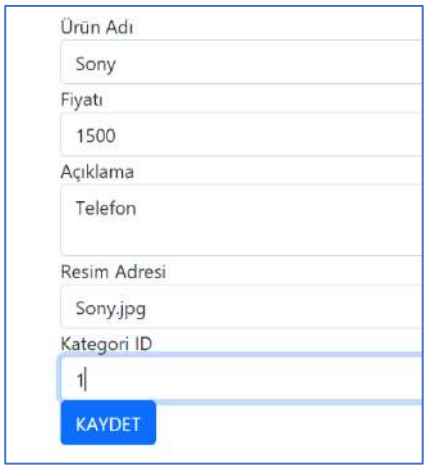
```
@section HeaderSection
```

```
{
    @await Html.PartialAsync("_Header")
}
```

Açılış sayfasının görünümü şu şekilde oldu.



Ürün Ekle Sayfamız içinde (Create.cshtml) form yapımız şu şekilde olacaktır.



Bu formda GET metodu kullandığımızdan form içinde kullanılan nesnelerin "name" alanlarındaki kontrol bilgileri otomatik olarak Link içinde değişken olarak kullanıldığını ve değerlerini de yine formdaki kontrollerden aldığını görürüz. Yani form yapımızdaki tüm alanlar Link bilgisi içinde gönderilmektedir. Linki incelediğimizde görebiliyoruz.

Bu linkin içindeki sayfa yine Create.cshtml sayfasına gittiği için ilk olarak ProductControler içindeki Create metodunun olduğu yere gidecektir. Linkin içindeki değişkenleri istersek oradan aşağıdaki şekilde yazarak alabiliriz. Şu aşamada bunu kullanmayacağız.

```
public IActionResult Create(string Name, double Price)
{
       return View();
}
```

Burada aslında formumuzu GET metodu ile Link içinde göndermeyi gördük. Fakat böyle bir yöntem tercih edilmez. Çünkü formdaki bilgilerimizi Link içinde gözükmesi istenmez. Bu nedenle bilgilerin sayfa içinde gizlenerek gönderilmesi gerekir. Buda POST metoduyla olur. Şimdi bu yöntemi görelim.

Post Metoduyla Formdan Bilgilerin Gönderilmesi

ProductController içinde oluşturulan Create ve diğer metodların üzerinde aslında **[HttpGet]** tipi vardır. Yani bu metodlar link içinden bilgileri aldığından GET requesti ile çalışmaktadır. Varsayılan Request tipi Get metodudur. Metodların görünümü şu şekildedir.

```
[HttpGet]
public IActionResult Create()
{
       return View();
}
```

Peki biz formumuzu POST yöntemiyle gönderdiğimizde nasıl olacaktır. Bu sefer Create metodumuzu şu şekilde oluşturmamız gerekiyor. Hem **HttpGet** hemde **HttpPost** şeklinde iki tane Create metodu oluşturmalıyız. Get tipinde olan Linke tıklandığında formun açılmasını sağlar. Post tipinde olan ise butona tıklandığında tekrar aynı formun içindeki bilgileri almamızı sağlar. Kodlarımız şu şekildedir.

ProductController.cs

```
[HttpGet]
public IActionResult Create()
{
    return View();
}
[HttpPost]
public IActionResult Create(string Name,
string Price, string Description, string
ImgeUrl, string CategoryID )
{
    return View();
}
```

Create.cshtml

```
<form method="POST">
    <div class="form-group">
        <label for="Name">Ürün Adı</label>
        <input type="text" name="Name" class="form-
control">
    </div>
    <div class="form-group">
        <label for="Price">Fiyatı</label>
        <input type="text" name="Price" class="form-
control">
    </div>
    <div class="form-group">
        <label for="Description">Açıklama</label>
        <textarea name="Description" class="form-
control"></textarea>
    </div>
    <div class="form-group">
        <label for="ImageUrl">Resim Adresi</label>
        <input type="text" name="ImageUrl"
class="form-control">
    </div>
    <div class="form-group">
        <label for="CategoryId">Kategori ID</label>
        <input type="text" name="CategoryId"
class="form-control">
    </div>
    <button type="submit" class="btn btn-
primary">KAYDET</button>
</form>
```

Not: [HttpPost] requestinin olduğu yerde metodun aldığı parametreleri tek tek yazmak yerine hepsinin içinde bulunduğu ProductModel sınıfın tipinde alırsak işimiz daha kolay olmuş olur.

```
public IActionResult Create(string Name, string Price, string Description, string ImgeUrl, string
CategoryID)
```

Bunun kullanımını ise şu şekilde yapacağız.

ProductController.cs

```
[HttpPost]
public IActionResult Create(ProductModel p)
{
    Console.Write(p.Name);
    Console.Write(p.Price);
    Console.Write(p.ProductId);
    Console.Write(p.ImageUrl);
    //***Bu şekilde p nesnesi içindeki bilgileri alıp  console da gösterebiliriz.

    return View();
```

```
}
```

Peki benzer şekilde form içinde de tek tek alanları name içerisinde sınıf yapımıza uygun olarak tanımlamıştık. Bu kullanımda alanları yazarken hatada yapabiliriz. Onun yerine form sayfamız içinde de bir model tanımlayıp onun alt alanlarını kullanabiliriz. Bu durumda kodlarımız şu şekilde olacaktır. Dikkat edersek `type="text" name="Name"` çıkarıldı, onun yerine `asp-for="Name"` geldi. Böylece tip tanımları sınıf içinden geldi ve name alanları ise yine benzer şekilde yazıldı. Bunun başka yararları da var, onlardan da bahsedeceğiz.

Create.cshtml

```
@model ProductModel

<form method="POST">
    <div class="form-group">
        <label for="Name">Ürün Adı</label>
        <input asp-for="Name" class="form-control">
    </div>
    <div class="form-group">
        <label for="Price">Fiyatı</label>
        <input asp-for="Price" class="form-control">
    </div>
    <div class="form-group">
        <label for="Description">Açıklama</label>
        <textarea asp-for="Description" class="form-control"></textarea>
    </div>
    <div class="form-group">
        <label for="ImageUrl">Resim Adresi</label>
        <input asp-for="ImageUrl" class="form-control">
    </div>
    <div class="form-group">
        <label for="CategoryId">Kategori ID</label>
        <input asp-for="CategoryId" class="form-control">
    </div>
    <button type="submit" class="btn btn-primary">KAYDET</button>
</form>
```

Bu model sınıfını Controller içindeki Create metodunda kullanalım ve yeni bir ürün kaydetmek için AddProduct() metodunu kullanalım. Kayıt işlemi bittikten sonra return View() diyerek tekrar aynı sayfaya gitmeyelim. Başka bir sayfaya yönlendirelim. Örneğin eklenen ürünü görmek için List sayfasına yönlensin. Bunun için `return RedirectToAction("List");` ifadesini kullanacağız.

Not: Burada form sayfamızda POST işlemi yapılırken Action komutunu kullanmadık. Bunu kullanmadığımızda yine Controller altındaki [HttpPost] tipinde oluşturulan Create metoduna otomatik olarak gitmektedir. Eğer biz Form içinde bir action="/product/AddProduct" şeklinde bir action tanımlarsak Controller içindeki metodumuzun adını da o şekilde değiştirmemiz gerekir.

---------- 000 ----------

Not: Burada Create sayfası içinde Model bilgisi içerisindeki alanları ve tipleri otomatik olarak alan bir üstteki kod yapısı çalıştırılabilmesi için _ViewImports.cshmtl dosyası içinde

`@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers`

İfadesinin bulunması gerekir. Bu ifade html etiketleri içindeki asp etiketlerini çalışır hale getirir. (Not: Denemelerde textbox larda etiketler çalışıyor fakat Dropdown da çalışmıyordur. Yani bu kod Dropdown için gerekli olduğu anlaşılıyor. Deneme yaparak teyid et..!). _ViewImports içine bu şekilde her sayfada çalışmasını istediğimiz bildirimleri yazmak istemezsek her sayfanın başına bu bildirimleri koymamız gerekir.

---------- 000 ----------

Çalışan tüm kodları toptan verelim.

_ViewImports.cshtml

```
@*View altındaki tüm sayfalarda Model
bilgilerini çağıran kodlar konulması
istenmezse buras açılmalıdır.*@

@using WebApplication1.Models;
@using WebApplication1.ViewModels;

@*DİKKAT: Create.cshtml ürün ekleme
sayfasındaki Select Kutusunun modelden
bilgileri otomatik çekebilmesi için bu bu
kodların olması gerekiyor.*@
@addTagHelper *,
Microsoft.AspNetCore.Mvc.TagHelpers
```

ProductController.cs

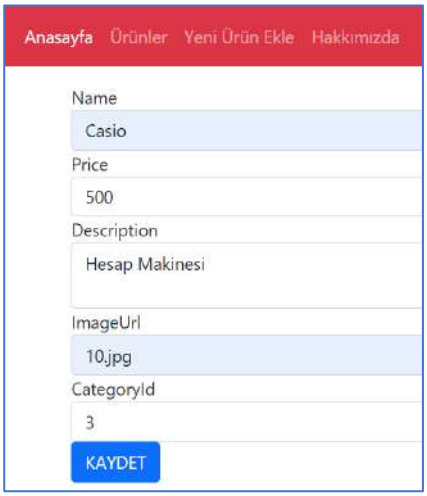
```
public IActionResult Create()
{
    return View();
}
[HttpPost]
public IActionResult Create(ProductModel p)
{
    ProductRepository.AddProduct(p);
    return RedirectToAction("List");
}
```

ProductRepository.cs

```
public static class ProductRepository
{ //Bu metod ürün eklemek için kullanılacak.
Dolayısı ile geri değer göndermeyecek.
public static void AddProduct(ProductModel
Product)
{
    ProductList.Add(Product);
}

}
```

ProductModel.cs

```
public class ProductModel
{
    public int ProductId { get; set; }
    public string Name { get; set; }
    public double Price { get; set; }
    public string Description { get; set; }
    public string ImageUrl { get; set; }
    public bool IsApproved { get; set; }
    public int CategoryId { get; set; }
}
```



Dikkat ettiysek burada 3. Kategoriye yeni bir ürün ekledik ve bu ürün liste sayfasında gösterildi. Tabi bu ürüne ait resim daha önceden wwwroot/img altına 10.jpg adıyla eklenmişti. Oradan resmini getirdi. Peki bizim diğer 9 ürünümüzde nerde kayıtlı idi. Bunlar ProductRepository altında tutuluyordu. Yeni eklenen bu üründe onların devamına hafızada tutulacak şekilde eklendi. Ve böylece 10 tane ürün gösterilmiş olmaktadır.

Kategori Select Kutusunu Forma Ekleme

Form içerisinde ürün kaydederken CategoriID sini elle yazmıştık. Oysa böyle bir formda bir Select kutusu içerisinde tüm kategorilerin gelmesi ve oradan seçim yaparak ID bilgisini otomatik almamız gerekir. Bu işlem için form içerisinde bir Categori select kutusu oluşturalım. İçerisine kategorileri otomatik getirilim. Ürünü kaydederken kategorisini de bu select kutusundan seçelim.

A-Kategorileri Elle Select Kutusuna Doldurma

```
@model ProductModel

<form method="POST">
    <div class="form-group">
        <label for="Name">Ad</label>
        <input type="text" name="Name" class="form-
control">
    </div>
    <div class="form-group">
        <label for="Price">Fiyat</label>
        <input type="text" name="Price" class="form-
control">
    </div>
    <div class="form-group">
        <label for="Description">Açıklama</label>
        <textarea name="Description" class="form-
control"></textarea>
    </div>
    <div class="form-group">
        <label for="ImageUrl">Resim Adı</label>
        <input type="text" name="ImageUrl" class="form-
control">
    </div>
    <div class="form-group">
        <label for="CategoryId">Kategori Seçimi</label>
        <select name="CategoryID" class="form-control">
            <option selected disabled >Seçiniz</option>
            <option value="1">Telefon</option>
            <option value="2">Bilgisayar</option>
            <option value="3">Elektronik</option>
        </select>
    </div>
    <button type="submit" class="btn btn-
primary">KAYDET</button>
</form>
```

Burada select kutusu içinde value bilgisi CategoriID alanı içinde taşınmaktadır. “selected disabled” ifadesi ise ilk açılışta seçili olarak gelmesini sağlar. Aynı zamanda kişinin bu alanı seçmesine müsaade etmez. Böylece oluşacak hata önlenmiş olmaktadır.

B-Select Kutusuna Kategorileri Otomatik Olarak Doldurma

Select kutusu içerisindeki Kategorileri elle yazmak olmaz. Bunları CategoryRepository de (Sanal veritabanı olarak düşünebiliriz) tanımladığımız kategorileri otomatik olarak yükleyelim. Bunun için öncelikle Formumuz ilk açılışta GET metodu ile açılıyordu. Bu requestin olduğu Create metodu içerisinde bilgileri forma gönderelim. Kodlar aşağıdadır. CategoryRepository den bilgileri alırken SelectList tipinde alıyoruz. Bu liste tipi bizden bir Value değeri bir de Text değeri için bilgi ister. Sonraki iki parametrede Value değeri için CategoryId alanını kullanıyoruz. Üçüncü parametrede ise Name alanını kullanıyoruz.

ProductController.cs

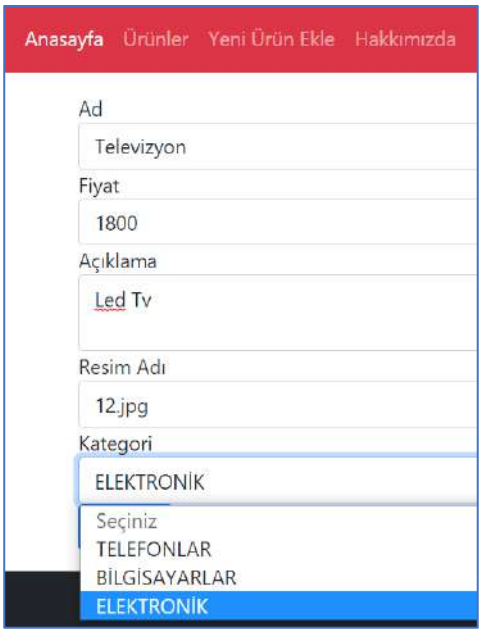
```
using Microsoft.AspNetCore.Mvc.Rendering;
[HttpGet]
public IActionResult Create()
{
```

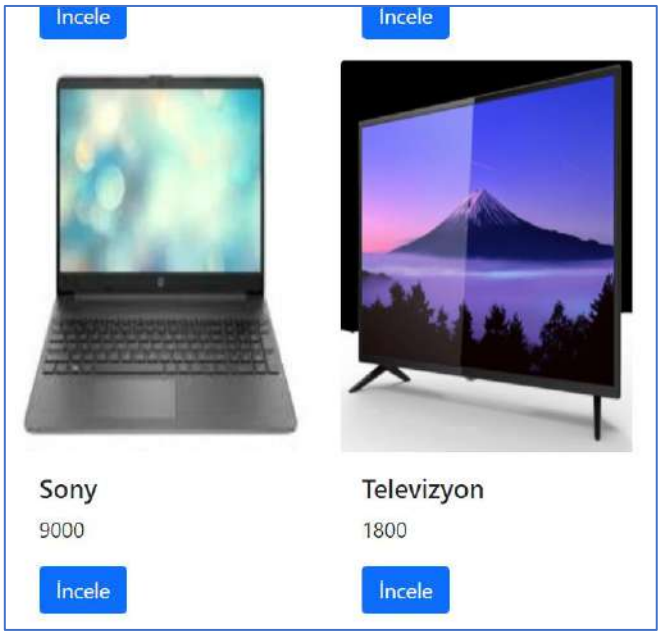
Create.cshtml

```
@model ProductModel

<form method="POST">
    <div class="form-group">
        <label for="Name">Ad</label>
        <input asp-for="Name" class="form-control">
    </div>
```

```
    ViewBag.Categories = new SelectList(CategoryRepository.Categories, "CategoryId", "Name");
    return View();
}
[HttpPost]
```

```
    <div class="form-group">
        <label for="Price">Fiyat</label>
        <input asp-for="Price" class="form-control">
    </div>
    <div class="form-group">
        <label for="Description">Açıklama</label>
        <textarea asp-for="Description" class="form-control"></textarea>
    </div>
    <div class="form-group">
        <label for="ImageUrl">Resim Adı</label>
        <input asp-for="ImageUrl" class="form-control">
    </div>
    <div class="form-group">
        <label for="CategoryId">Kategori</label>
        <select asp-for="CategoryId" asp-items="@ViewBag.Categories" class="form-control">
            <option selected disabled>Seçiniz</option>
        </select>
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
</form>
```



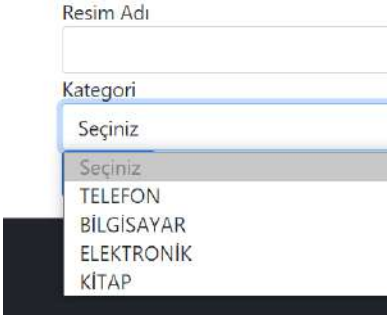


Peki mevcut kategoriler içerisine bir tane daha Kategori ekleyelim. Bunun için CategoryRepository içerisinde eklemeyi yapalım.

CategoryRepository.cs

```
static CategoryRepository()
{
    CategoryList = new List<CategoryModel>
    {
        new CategoryModel { CategoryId=1, Name = "TELEFON", Description = "Telefon Kategorisi"},
        new CategoryModel { CategoryId=2, Name = "BİLGİSAYAR", Description = "Bilgisayar Kategorisi"},
        new CategoryModel { CategoryId=3, Name = "ELEKTRONİK", Description = "Elektronik Ürünler Kategorisi"}
```

Create.cshmtl



List.cshtml

```
        new CategoryModel { CategoryId=4,
Name = "KİTAP", Description = "Kitap
Kategorisi"}
    };
}
```

Form Üzerinden Ürün Güncelleme

Liste üzerinden herhangi bir ürüne ait "Değiştir" butonuna tıkladığımızda bir form içerisinde ürün bilgileri gelsin. Ardından form üzerinde ürüne ait bilgilerde değişiklik yaptıktan sonra "Güncelle" butonuna tıkladığımızda ürün bilgileri güncellenmiş olsun.

Güncelleme formu, ürün eklerken kullandığımız formun bir benzeri olacak. Bunun için oluşturacağımız sayfalar Edit sayfası olacak.

Öncelikle _Card.cshtml sayfasında ürünlerin altında çıkan bir "İncele" butonu olsun. Bu butona tıkladığımızda Product kontrolleri altındaki Edit sayfasına giderken ürünün Id bilgilsinide götürsün. Linkte şöyle gözükecektir "product/Edit/1".

Link ProductController içindeki [HttpGet] tipindeki Edit metoduna gidecektir. Bu metedoun girişinde Linki içindeki Id bilgiside alınmaktadır. Buradan View içindeki Edit sayfasına giderken, Id li ürüne ait Ürün bilgilerini ProductRepository den GetProductById metodu ile almakta ve yanında götürmektedir.

Edit.cshtml sayfasına vardığında yanında götürdüğü ürün bilgilerini (model yapısı içinde gidiyor) form elemanlarında otomatik olarak görüntülemektedir. Bu form içinde herhangi bir bilgide değişiklik yaptığımızda "Güncelle " butonuna tıklarsak form bilgileri model yapısı içinde tekrar ProductControler.cs gelecekt fakat bu sefer form içinden POST yöntemi ile gönderildiğinden HttpPost tipindeki Edit metoduna giriş yapacaktır. Bu metodun girişinde model bilgisi içinde ürünün yeni bilgileri gelecektir. Metod içinde alınan ürün bilgileri ProductRepository nin alt metodu olan EditProduct içinde güncellenecektir.

EditProduct metodu içinde bilgiler güncellenirken tüm listedeki ürünler içinde tek tek aranan ürün taranmaktadır. Formdan gelen ürün Id si ile taranan listedeki ürün Id si eşleştiğinde yeni bilgiler orada güncellenmektedir.

_Card.cshtml

```
@model ProductModel;


<div class="cart">
    <img class="card-img-top"
src="/img/@Model.ImageUrl" height="300" />
    <div class="card-body">
        <h5 class="card-
title">@Model.Name</h5>
        <p class="card-text">@Model.Price</p>

        <a
href="/Product/Details/@Model.ProductId"
class="btn btn-primary btn-sm">İncele</a>
        <a
href="/Product/Edit/@Model.ProductId"
class="btn btn-primary btn-sm">Değiştir</a>
```

ProductController.cs

```
[HttpGet]
public IActionResult Edit(int Id)
{
    ViewBag.Categories = new
SelectList(CategoryRepository.Categories,
"CategoryId", "Name");
    return
View(ProductRepository.GetProductById(Id));
}

[HttpPost]
public IActionResult Edit(ProductModel p)
{
    ProductRepository.EditProduct(p);
    return RedirectToAction("List");
}
```

```
    </div>

</div>
```

Edit.cshtml

```
@model ProductModel

    <form method="POST">
        <input type="hidden" name="ProductId"
value="@Model.ProductId">
        <div class="form-group">
            <label for="Name">Ad</label>
            <input asp-for="Name" class="form-
control">
        </div>
        <div class="form-group">
            <label for="Price">Fiyat</label>
            <input asp-for="Price" class="form-
control">
        </div>
        <div class="form-group">
            <label for="Description">Açıklama</label>
            <textarea asp-for="Description"
class="form-control"></textarea>
        </div>
        <div class="form-group">
            <label for="ImageUrl">Resim Adı</label>
            <input asp-for="ImageUrl" class="form-
control">
        </div>
        <div class="form-group">
            <label for="CategoryId">Kategori</label>
            <select asp-for="CategoryId" asp-
items="@ViewBag.Categories" class="form-control">
                <option selected
disabled>Seçiniz</option>
            </select>
        </div>
        <button type="submit" class="btn btn-
primary">GÜNCELLE</button>
    </form>
```

ProductRepository.cs

```
public static void EditProduct(ProductModel
Product)
{
    foreach(ProductModel p in ProductList)
    {
        if(p.ProductId ==Product.ProductId)
        {
            p.Name = Product.Name;
            p.Price = Product.Price;
            p.ImageUrl = Product.ImageUrl;
            p.Description =
Product.Description;
            p.IsApproved =
Product.IsApproved;
            p.CategoryId =
Product.CategoryId;
        }
    }

}
```

Form Validation-Girilen Bilgilerin Kontrolü

Form üzerinden girilen bilgilerin tiplerine uygun şekilde doğru girilmesi, boş bırakılmaması yada uygun formatta girilmesi gibi kontrollerin yapılması gerekir. Bu kontroller kişinin tarayıcısında yada Server tarafında da yapılabilir. Kişinin tarayıcısında yapılırken JavaScript kodlamaları kullanıldığından, eğer kişi bu özelliği kapatırsa, hatalı girişler devam edeceğinden, server tarafında da kontrollerin yapılması gerekir.

Form için gerekli olan Validation (doğrulama) özelliklerini ProductModel yapısı içinde tanımlayacağız. Bunun için aşağıdaki uygulamaları yapalım.

Öncelikle validationları model yapısı içine ekleyebilmek için DataAnnotaions kütüphanesini sayfanın en üstüne eklemeliyiz.

```
using System.ComponentModel.DataAnnotations;
```

Model içindeki Name alanı doldurmayı zorunlu yapmak için [Required] ifadesi kullanılır. Karakter uzunluğunu ve hata mesajını belirtmek için de [StringLength..] ifadesi kullanılır.

```
   [Required]
   [StringLength(60,MinimumLength =3,ErrorMessage ="3-60 Karakter arasında olmalıdır")]
public string Name { get; set; }
```

Not: Fiyatı girerken Price alanı double olduğu için herhangi bir değer girilmezse 0 değeri gidecektir. Ama eğer sıfır değerinin gitmesi problemse [Required] ifadesini kullanabilmek için alanın Nullable yapılması gerekir. Bunun için tipin sonunda "?" işareti kullanılır. Bu durumda kullanıcı herhangi bir değer girmezse Null ifadesi gidecektir bu durumda da [Required] ifadesi çalışacaktır. Aşağıdaki şekilde kullandığımızda hem hata mesajını vermiş olduk hemde aralık ifade etmiş olduk.

```
   [Required(ErrorMessage ="Fiyat girmelisiniz")]
   [Range(1,10000)]
public double? Price { get; set; }
```

Not: Google dan "Data Annotations" ifadesi ile arama yaptığımızda diğer Validation kullanımlarınıda görmüş oluruz.

Model yapımız aşağıdaki şekilde olmuş oldu.

ProductModel.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;

namespace WebApplication1.Models
{
    public class ProductModel
    {
        public int ProductId { get; set; }
        [Required]
        [StringLength(60, MinimumLength = 3, ErrorMessage = "3-60 Karakter arasında olmalıdır")]
        public string Name { get; set; }

        [Required(ErrorMessage = "Fiyat girmelisiniz")]
        [Range(1, 10000)]
        public double? Price { get; set; }
        public string Description { get; set; }
        public string ImageUrl { get; set; }
        public bool IsApproved { get; set; }
        [Required]
        public int? CategoryId { get; set; }    }

}
```

Form üzerinde yeni bir ürün bilgisini girip Butona tıkladığımızda bilgiler ProductController içindeki [HttpPost] tipindeki Create metoduna gelecektir. Sayfadan gelen alanların Validation işleminden geçip geçmediğini biz "ModelState" ifadesi ile anlayabiliyoruz. Hata yoksa kaydediyoruz, hata varsa tekrar sayfaya geri gidiyoruz fakat yanımızda bize gelen ürün bilgilerini de geri götürüyoruz.

ProductController.cs

```
[HttpPost]
public IActionResult Create(ProductModel p)
{
    //Form üzerindeki alanlar Validation kurallarına göre oluşturulmuş ise, yeni ürünün bilgilerini kaydedecek.
    if(ModelState.IsValid)
    {
        ProductRepository.AddProduct(p);
        return RedirectToAction("List");
    }
    //Kategori bilgilerinide götürüyoruz.
    ViewBag.Categories = new SelectList(CategoryRepository.Categories, "CategoryId", "Name");

    //Validationlara uyulmadıysa burası çalışacaktır.
    //Tekrar sayfaya gönderiyor fakat bu esnada kendisine gelen hatalı ürün bilgilerini de geri götürüyor.
    return View(p);
}
```

Fakat Create.cshtml sayfasına geri gittiğimizde ve yanımızda ürünün model bilgileri var ise bu bilgileri form içinde görüntülememiz gerekir. Yani hatalı bilgiler tekrar kullanıcının karşısında olmalıdır. Model alanlarını controller içinde gösterebilmek içinde Value değerlerini doldurmamız gerekir. Tıpkı Edit.cshtml sayfasında yaptığımız gibi. Orada içi model bilgisi form içinde gösteriliyordu. Sayfamızın son hali şu şekilde olacaktır.

Create.cshtml

```
@model ProductModel

<form method="POST">
    <div class="form-group">
        <label for="Name">Ad</label>
        <input asp-for="Name" value="@Model.Name" class="form-control">
    </div>
    <div class="form-group">
        <label for="Price">Fiyat</label>
        <input asp-for="Price" value="@Model.Price" class="form-control">
    </div>
    <div class="form-group">
        <label for="Description">Açıklama</label>
        <textarea asp-for="Description" value="@Model.Description" class="form-control"></textarea>
    </div>
    <div class="form-group">
        <label for="ImageUrl">Resim Adı</label>
        <input asp-for="ImageUrl" value="@Model.ImageUrl" class="form-control">
    </div>
    <div class="form-group">
        <label for="CategoryId">Kategori</label>
        <select asp-for="CategoryId" asp-items="@ViewBag.Categories" class="form-control">
            <option selected disabled>Seçiniz</option>
        </select>
    </div>
    <button type="submit" class="btn btn-primary">KAYDET</button>
</form>
```

Burada Create sayfası içine Value değerlerini eklediğimizde bizden Model bilgilerini istemektedir. Eğer bu sayfaya ilk geldiğimizde yani [HttpGet] Create metodu içinden geldiğimizde Model bilgisini getirmezsek hata alırız. Bunu önlemek için içinde ürün bilgileri olmasa da Model bilgisi dolu olarak gelmelidir. Bunu sağlamak için GET yöntemini kullanan metodun içi aşağıdaki şekilde yazılmalıdır.

ProductController.cs

```
[HttpGet]
public IActionResult Create()
{
    ViewBag.Categories = new SelectList(CategoryRepository.Categories, "CategoryId", "Name");
    return View(new ProductModel());
}
```

Hatalı ürün bilgileri sayfaya geldiğinde hata olan alanlarda Validation Error mesajlarını göstermemiz gerekir. Bunun her alanın yanına aşağıdaki gibi bir etiketi ekleyelim ve içerisinde hangi alandaki hatayı göstereceğini de belirtelim.

```
<span asp-validation-for="Name" class="text-danger"></span>
```

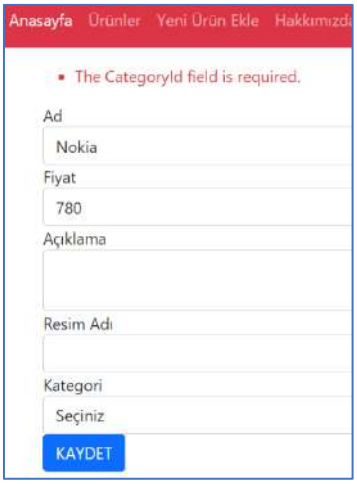
Hata mesajları her alanın yanında gözükmesini istediğimiz gibi tümünün özetini üstte bir özet şeklinde verilmesi de önemlidir. Özellikle uzun formlarda böyle bir uygulama iyi olacaktır. Bunun için aşağıdaki satırları formun üst kısmına ekleyelim.

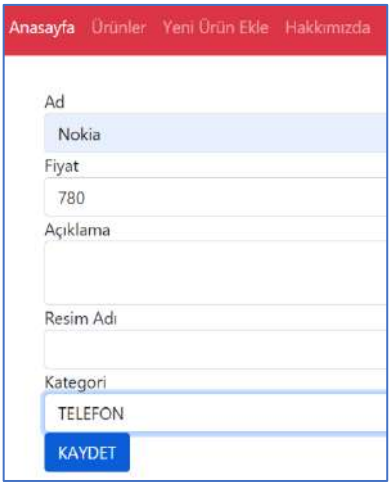
```
<div asp-validation-summary="All" class="text-danger"></div>
```

Kodlarımızın son hali aşağıdaki şekilde olur.

Create.cshtml

```
@model ProductModel

<form method="POST">
    <div asp-validation-summary="All" class="text-danger"></div>

    <div class="form-group">
        <label for="Name">Ad</label>
        <input asp-for="Name" value="@Model.Name" class="form-control">
        <span asp-validation-for="Name" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label for="Price">Fiyat</label>
        <input asp-for="Price" value="@Model.Price" class="form-control">
        <span asp-validation-for="Price" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label for="Description">Açıklama</label>
        <textarea asp-for="Description" value="@Model.Description" class="form-control"></textarea>
        <span asp-validation-for="Description" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label for="ImageUrl">Resim Adı</label>
        <input asp-for="ImageUrl" value="@Model.ImageUrl" class="form-control">
        <span asp-validation-for="ImageUrl" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label for="CategoryId">Kategori</label>
        <select asp-for="CategoryId" asp-items="@ViewBag.Categories" class="form-control">
            <option selected disabled>Seçiniz</option>
        </select>
       <span asp-validation-for="CategoryId" class="text-danger"></span>
    </div>
    <button type="submit" class="btn btn-primary">KAYDET</button>
</form>
```





Ürün Silme

Ürün silme işini iki yöntemle yapalım. Birinci yöntemde <a> linkine tıkladığımızda direk Controller altına gitsin ve oradan da Repository altında silme işini gerçekleştirelim. İkinci yöntemde ise Silme işlemini gerçekleştireceğimiz butonu bir form içine yerleştirelim. Bu form içinden bilgileri POST yöntemiyle Controller içine gönderelim. Oradan da silme işlemini Repository içinde gerçekleştirelim.

1.Yöntem: Direk Link (buton) Üzerinden Silme İşlemi

_Card.cshtml	ProductController.cs

```
@model ProductModel;
<div class="cart">
    <img class="card-img-top"
src="/img/@Model.ImageUrl" height="300" />
    <div class="card-body">
        <h5 class="card-title">@Model.Name</h5>
        <p class="card-text">@Model.Price</p>

        <a
href="/Product/Details/@Model.ProductId"
class="btn btn-primary btn-sm">İncele</a>
        <a href="/Product/Edit/@Model.ProductId"
class="btn btn-primary btn-sm">Değiştir</a>
        <a href="/Product/Delete/@Model.ProductId"
class="btn btn-danger btn-sm">Sil</a>
    </div>

</div>
```

```
public IActionResult Delete(int Id)
{
    ProductRepository.DeleteProduct(Id);
    return RedirectToAction("List");
}
```

ProductRepository.cs

```
public static void DeleteProduct(int Id)
{
    var Product = GetProductById(Id);

    if (Product != null)
    {
        ProductList.Remove(Product);
    }
}
```

2. Yöntem: Form Üzerinden Silme İşlemi

_Card.cshtml

```
@model ProductModel;

<div class="cart">
    <img class="card-img-top"
src="/img/@Model.ImageUrl" height="300" />
    <div class="card-body">
        <h5 class="card-title">@Model.Name</h5>
        <p class="card-text">@Model.Price</p>

        <a href="/Product/Details/@Model.ProductId"
class="btn btn-primary btn-sm">İncele</a>
        <a href="/Product/Edit/@Model.ProductId"
class="btn btn-primary btn-sm">Değiştir</a>

        @*<a href="/Product/Delete/@Model.ProductId"
class="btn btn-danger btn-sm">Sil</a>*@

    </div>
</div>
```

ProductController.cs

```
[HttpPost]
public IActionResult Delete(int ProductId)
{

ProductRepository.DeleteProduct(ProductId);
    return RedirectToAction("List");
}
```

ProductRepository.cs

```
public static void DeleteProduct(int Id)
{
    var Product = GetProductById(Id);

    if (Product != null)
    {
        ProductList.Remove(Product);
    }
}
```

Kaynak: Udemy-Komple Uygulamalı Web Geliştirme Kursu-Sadık Turan