ASP.NET CORE-MVC

İçindekiler

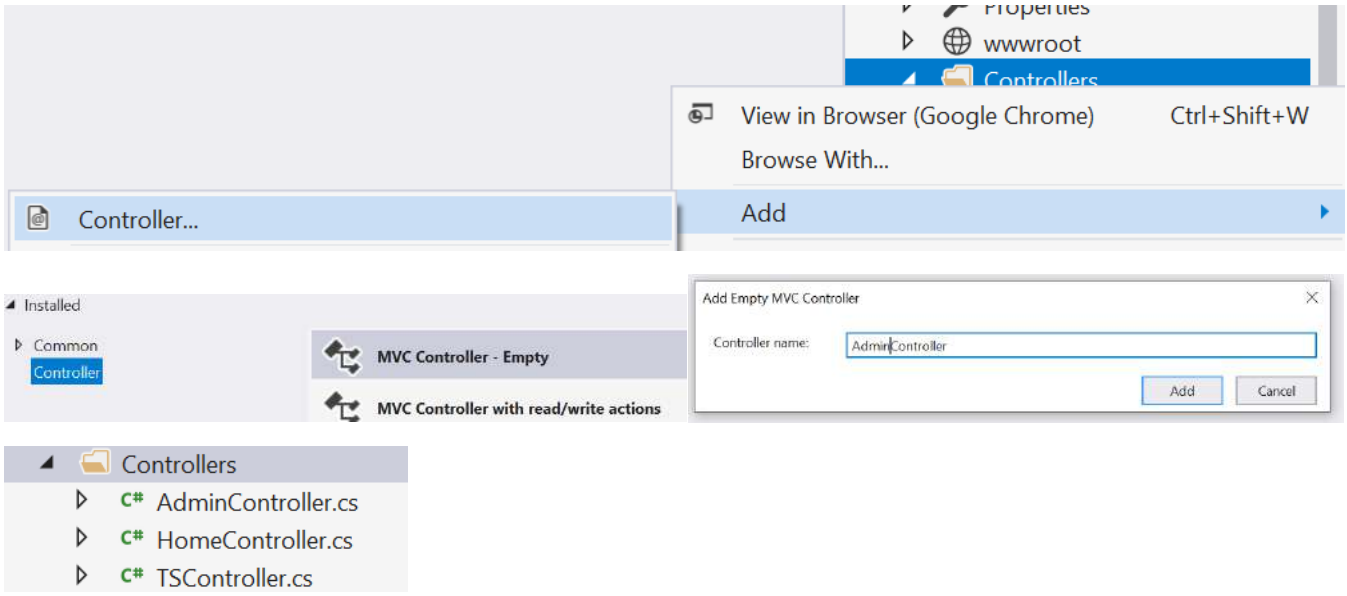
YÖNETİCİ SAYFALARININ HAZIRLANMASI

Ürün Listeleme Tablosu

Admin sayfalarını hazırlarken Farklı bir Layout sayfası (sayfa parçalarının içinde toplandığı dış şablon) kullanabiliriz. Böylece farklı bir tasarımla admin karşına çıkmış olabiliriz. Ancak burada biz aynı layout sayfasını (Views>Shared>_Layout.cshtml) kullanacağız.

Öncelikle Admin işlemlerini yönetmek için ayrı bir Controller oluşturabiliriz. Bunun için Controllers>AdminController şeklinde yeni bir controller oluşturalım. Admin giriş yaptığında karşısına gelecek linkleri ona göre getirmemiz gerekir. Bunun yetkilendirme yaparak gösterebiliriz. Fakat şu aşamada bununla uğraşmayıp, adminin kullanacağı linkleri de NavBar içinde kullanalım.

Dikkat edersek ifadenin controllers içinde verilen dosya adlarının sonları _Controller olarak bitmesi gerekiyor. "AdminController" gibi.. Bu kontroller içinde ProductList() isminde bir metod oluşturalım.



Not: Model yapılarımızı iki klasörün altında oluşturmuştuk. Birisi Models diğer ViewModels idi. Artık bu iki klasör yerine tüm model yapılarımızı Models klasörü altında toplayalım ve ona göre NameSpace isimlerinide (klasörlerin adreslerini) düzeltelim. Bu işlemleri burada göstermedik.

Daha önce NavBar içine admin için aşağıdaki linkleri eklemiştik. Bunları kullanabiliriz.

```
<li class="nav-item">
    <a asp-controller="Admin" asp-action="createProduct" class="nav-link">(Admin)Ürün Ekle</a>
</li>

<li class="nav-item">
    <a asp-controller="Admin" asp-action="productList" class="nav-link">(Admin)Listele</a>
</li>
```

AdminController içinde ProductList() adında bir metod oluşturalım. Bu metod sayfaya ProductListViewModel() yapısında yani içerisinde sayfalama bilgilerinin ve ürün bilgilerinin olduğu bir nesne göndersin. Ürün bilgileri ise Servis katmanından (business) geliyordu. Dolayısı ile metod içerisinde servis katmanını kullanabilmek için de bir Injection yapmamız gerekir (`private IProductService _productService;`). Injection satırından sonrada bir Constructor satırları yazacağız (`public AdminController(IProductService productService) {_productService = productService;}`). Bunu yaptıktan sonra dışarıdan alınan nesneyi sayfa içinde kullanacağımız parametre içine atacağız. Böylelikle artık sayfamızda herhangi bir yerde service üzerinden kullanacağımız getAll() metodunu (tüm ürünleri getiren) çalıştırabileceğiz.

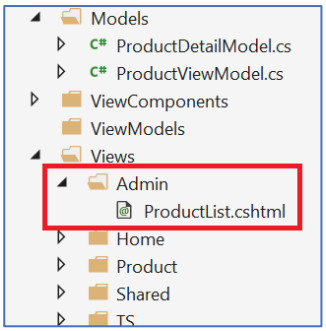
AdminController'ın son hali aşağıdaki şekilde olmuş oldu.

Controllers>**AdminController.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using TS.Business.Abstract;
using TS.WebUI.Models;

namespace TS.WebUI.Controllers
{
    public class AdminController : Controller
    {
```

```
        private IProductService _productService;

        public AdminController(IProductService productService)
        {
            _productService = productService;
        }
        public IActionResult ProductList()
        {
            return View(new ProductListViewModel()
            {
                Products = _productService.GetAll()
            }) ;
        }
    }
}
```

Artık controller ımız hazır ise burada kullandığımız ProductList() metodunun gideceği bir View (görüntüleme) sayfasına ihtiyaç vardır. Sayfamızı Views>Admin>… içinde oluşturmalıyız. Burada klasöre Admin isminin verilmesi onun AdminController a bağlı olduğunu gösterecektir. İsimleri verirken Controller ve Action seviyelerine dikkat etmeliyiz. Admin view klasörü içinde de ProductList.cshmtl sayfamızı oluşturalım.



Şimdi ProductList.cshtml sayfamızı oluşturmaya geçelim. Bu sayfa dış kısmına giydirilen _Layout.cshmtl sayfası içinde görüntüleneceğinden önce Layout sayfasında ufak bir düzenleme yapalım. Sayfamız RenderBody içinde görüntülenecektir. Burada kullanılan row etiketlerini layout dan alıp alt sayfa olan ProductList sayfasında oluşturalım

```
<main class="mt-3">
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                @RenderBody()
            </div>
        </div>
    </div>
</main>
```

Yeni hali

```
<main class="mt-3">
    <div class="container">
        @RenderBody()
    </div>
</main>
```

Şimdi linkimizi desen yapısını startup.cs içinde oluşturalım. Gelen istek (`pattern: "admin/products/")`, şeklinde ise (`controller = "Admin", action = "ProductList"`) gidecektir. Buradan yapılan yönlendirme ile de productlist.cshmtl sayfasına gidecektir. Link desenimiz şu şekilde olacaktır.

```
//admin/products/
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "adminproductlist",
        pattern: "admin/products/",
        defaults: new { controller = "Admin", action = "ProductList" }
    );
});
```

Burada oluşturulan Route ın her zaman varsayılan route dan önce gelmesi gerekir. Çünkü 2 bölümlü bir link varsayılan route tarafından da karşılanır (`pattern: "{controller=home}/{action=index}/{id?}"`).

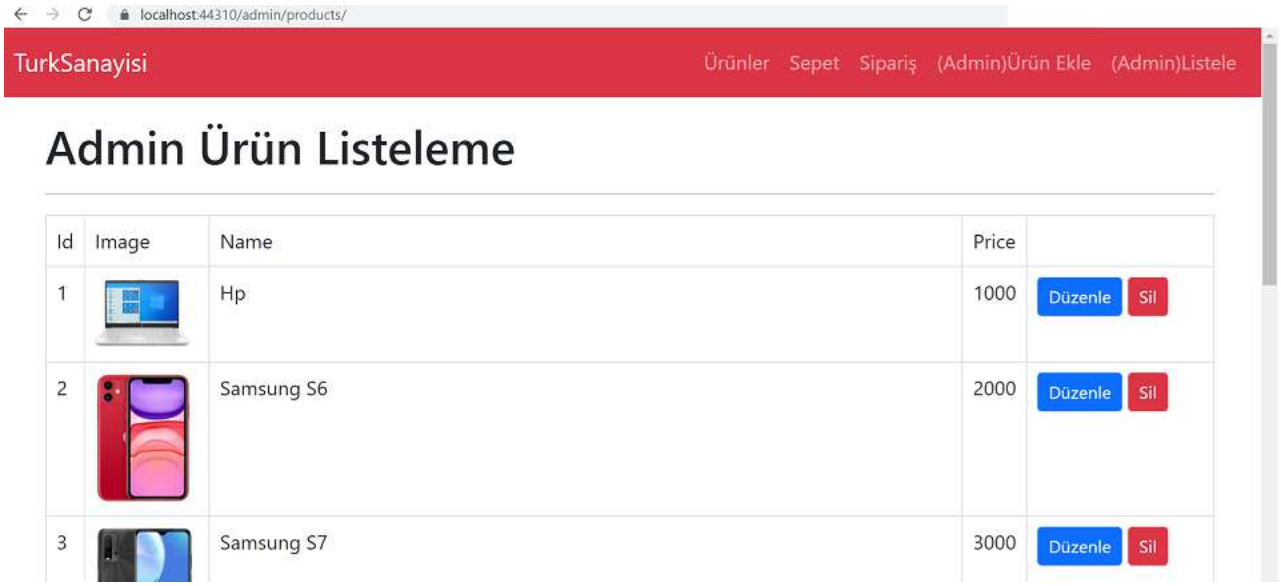
Link adresimizi alıp NavBar içinde aşağıdaki şekilde kullanalım. Dikkat edersek link içinde Controller ve Action ları ayrı ayrı (asp-controller="Admin" asp-action="productList") şeklinde yazabiliriz yada beraberce adres içinde (href="/admin/products/") şeklinde kullanabiliriz. Bu durumda Controller ve Action ayrışmasını startup.cs içindeki link desenimiz yapacaktır.

```
<li class="nav-item">
    <a href="/admin/products/" class="nav-link">(Admin)Listele</a>
</li>
```

ProductList.cshtml sayfamızın son hali aşağıdaki şekilde oldu.

Views>Admin>**ProductList.cshtml**

```
@model ProductListViewModel

<div class="row">
    <div class="col-md-12">
        <h1>Admin Ürün Listeleme</h1>
        <hr>
        <table class="table table-bordered">
            <thead>
                <tr>
                    <td style="width:30px">Id</td>
                    <td style="width:100px">Image</td>
                    <td>Name</td>
                    <td style="width:20px">Price</td>
                    <td style="width:150px"></td>
                </tr>
            </thead>
            <tbody>
                @if (Model.Products.Count > 0)
                {
                    @foreach (var item in Model.Products)
                    {
                        <tr>
                            <td>@item.ProductId</td>
                            <td><img src="~/img/@item.ImageUrl" width="80" /> </td>
                            <td>@item.Name</td>
                            <td>@item.Price</td>
                            <td>
                                <a href="/admin/products/@item.ProductId" class="btn btn-primary btn-sm mr-
2">Düzenle</a>
                                <a href="#" class="btn btn-danger btn-sm">Sil</a>
                            </td>
                        </tr>
                    }
                }
                else
                {
                    <div class="alert alert-varning">
                        <h3>No Products</h3>
                    </div>
                }

            </tbody>
        </table>
    </div>
</div>
```



Bir sonraki dersimizde ürün kaydetme sayfasını oluşturalım.

Admin Yeni Ürün Kaydetme

Adminin yeni ürün eklemesi için kullanacağı link NavBar içinde bulunacak. Bu linki aşağıdaki şekilde düzenleyelim.

```
<li class="nav-item">
    <a asp-controller="Admin" asp-action="createProduct" class="nav-link">(Admin)Ürün Ekle</a>
</li>
```

Bu linki karşılayacak olan Admin controller için CreateProduct isimli bir action metodu yazalım. Form işlemlerinde controller içinde iki tane metod yazmalıyız. Birinci metod View içindeki sayfayı getirecektir. Bu şekilde form sayfasını getirme işlemi [HttpGet] yöntemini kullanır. Form doldurulduktan sonra servera giderken ise [HttpPost] yöntemi kullanılır. Dolayısı ile aynı isimde iki ayrı action metodu yazmamız gerekecek.

[HttpPost] metodunda formdan gelen ürün bilgilerinin kaydedilme işlemi yapılacaktır. Metod sonunda ise Listeleme sayfasına gidilecektir.

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

[HttpPost]
public IActionResult CreateProduct(Product product)
{
    return RedirectToAction("ProductList");
}
```

AdminController içinde oluşturulan bu metodlar tam olarak bitmiş değildir. Şimdi metodların gideceği CereateProduct.cshtml dosyasını oluşturalım.

Sayfanın en üstünde <h1>seviyesinde başlığımız olsun. Yalnız bu başlık büyük gözüktüğü için class=”h3” diyerek daha küçük bir başlık elde edelim. Neden h1 ifadesinde ısrarcı olmamızın sebebi ise sayfanın en önemli başlığı olduğu için arama motorlarının bu önemde sayfayı taramasını sağlamış oluyoruz. Form etiketlerimizi oluştururken iki şekilde yazabiliriz. Ya (`<form action="/admin/createproduct" method="post"></form>`) yada (`<form asp-controller="Admin" asp-action="CreateProduct" method="post"></form>`) şeklinde olabilir. Form içinde kullanılacak elemanlar için Boostrap sayfasından örnek satır alarak düzenleyelim.

Form üzerinde doldurulacak alanlar ürünün Product sınıfındaki alanlar olacaktır. Bu sınıf içindeki alanlardan boolean olan IsAproved ve IsHome alanlarını forma eklemeyeceğiz. Çünkü veritabanında bu alanlar oluşturulurken zaten false olarak oluşturulmakta. Bizde daha baştan bunların false olmasını tercih ederiz. Bu konudaki onayları daha sonra verilecektir.

Bir ürünün Url bilgisi ise isminden otomatik olarak oluşturacak bir fonksiyon yazılarak yapılabilir. Yani ürünün adı üzerindeki boşluklara tire konulup, tekrarsız bir isimle kayıt yapılabilir. Burada Url bilgisini biz form dan alacağız.

Dikkat edersek elemanlar üzerinde hem "id" bilgisi var hemde "name" bilgisi vardır. İd bilgisi tarayıcı kısmında Javascripler için kullanılacak. Name ise server tarafından post işlemi yaparken elemanlar içinde bilgiler alınırken bu isimle kullanılacaktır.

Ürünün Description (açıklama) alanı textarea nesnesi ile alınacak. Böylece birden fazla satır girilebilmiş olacak. Textbox lardan type="" parametreleri çıkarıldı.

Butonun bulunduğu satırda label etiketi bulunmadığı için ve butonun Textboxlarla aynı hizada bulunması için iki kolon offset atarak ötelenmesi şu class adı ile sağlandı (`<div class="col-sm-10 offset-sm-2">`)

Not: Burada formun etiketleri oluşturulurken iki farklı yöntem kullanılabilir. Form sayfasında bir Model yapısı çağrılırsa (örn: TS.Entity>Product için `@model Product`) bu model yapısı içindeki alanlar form nesnelerinde Asp-For="" şeklinde kullanılır. Böylece butona tıklandığında tüm elemanların alanları Action Metodu içerisine alınırken sadece Model sınıfı yazılır (Örn: `public IActionResult CreateProduct(Product product) gibi`). Eğer form sayfasında Model bilgisi kullanılmaz ise o zaman her bir alan için yukarıda bahsedilen id="" ve name="" bilgileri verilmeli. Bu bilgilerden name bilgisi action metodu içine alınırken her bir alan değişkenler vasıtasıyla içeri alınır (örn: `public IActionResult CreateProduct(string Name, string Price, string Description, string ImgeUrl) gibi`)

Biz burada @model yapısı kullanarak formumuzu oluşturacağız ve action metoduna alınırken de tek seferde tüm alanlar model çatısı altında içeriye alınacaktır.

Not: Burada önemli bir konuya dikkat çekmek gerekiyor. Formlarda kullanacağımız model yapıları ile Entityler için kullanacağımız model yapılarını birbirinden ayırmakta fayda vardır. Bunun birkaç gerekçesi sayılabilir. Entity için oluşturulan model içindeki alanların tümünü Form üzerinde kullanmak durumunda olmayabiliriz. Çok daha önemlisi form üzerinde kullanılacak bir elemanın üzerindeki metni data Annotation özelliği kullanılarak farklı şekilde çıkartabiliriz. "Name" adı şeklinde göstermek yerine "Ürün adı" şeklinde gösterebiliriz.

Form içinde artık Model yapısı içinde gelen alanları "Id" ve "Name" bilgisi şeklinde değilde "asp-for" adıyla kullanacağız (bunlara takehelper deniyor). Böylece alanlar otomatik olarak dolmuş olacak. Bu alanlar için Type="text" gibi alanların tanımlanmasına gerek yoktur. Model de tanımlanan tip ne ise burada o alan o tipte olacaktır. "asp-for" şeklindeki takehelper alanlarını hem textboxlarda hemde labellarda kullanabiliriz.

Şimdi öncelikle form içinde kullanacağımız model yapısını oluşturalım. Entity için oluşturduğumuz model sınıfı TS.Entity projesi içinde Product ismiyle oluşturulmuştu. Formlar için oluşturacağımız model yapısını ise TS.WebUI projesi içindeki Model klasörü içinde "ProductModel" ismiyle oluşturalım. Dikkat edersek alanların bir kısmını kullandık.

TS.WebUI>Models>**ProductModel.cs**

```
namespace TS.WebUI.Models
{
    public class ProductModel
    {
        public int ProductId { get; set; }
        public string Name { get; set; }
        public string Url { get; set; }
        public double? Price { get; set; }
        public string Description { get; set; }
        public string ImageUrl { get; set; }
    }
```

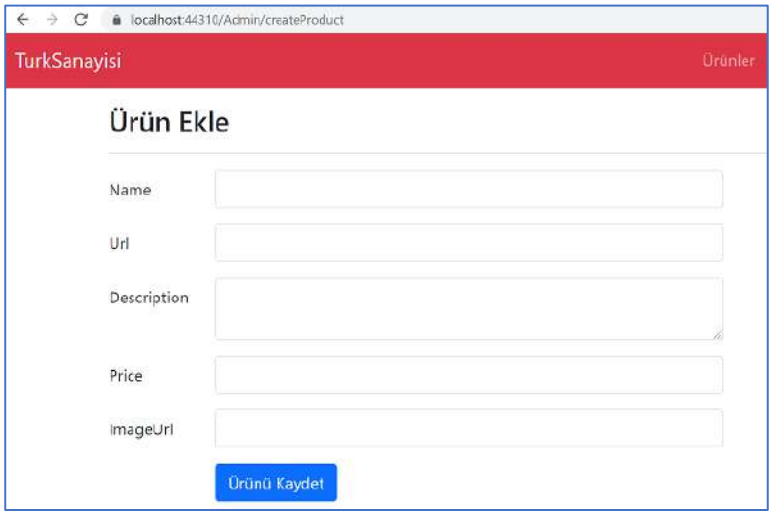
```
}
```

Form sayfamızın son hali aşağıdaki şekilde olmuş olur.

Views>Admin>**CreateProdut.cshmtl**

```
@model ProductModel

<h1 class="h3">Ürün Ekle</h1>
<hr>

<div class="row">
    <div class="col-md-8">
        <form asp-controller="Admin" asp-action="CreateProduct" method="POST">
            <div class="form-group row mb-3">
                <label asp-for="Name" class="col-sm-2 col-form-label"></label>
                <div class="col-sm-10">
                    <input class="form-control" asp-for="Name">
                </div>
            </div>
            <div class="form-group row mb-3">
                <label asp-for="Url" class="col-sm-2 col-form-label"></label>
                <div class="col-sm-10">
                    <input class="form-control" asp-for="Url">
                </div>
            </div>
            <div class="form-group row mb-3">
                <label asp-for="Description" class="col-sm-2 col-form-label"></label>
                <div class="col-sm-10">
                    <textarea class="form-control" asp-for="Description"></textarea>
                </div>
            </div>
            <div class="form-group row mb-3">
                <label asp-for="Price" class="col-sm-2 col-form-label"></label>
                <div class="col-sm-10">
                    <input class="form-control" asp-for="Price">
                </div>
            </div>
            <div class="form-group row mb-3">
                <label asp-for="ImageUrl" class="col-sm-2 col-form-label"></label>
                <div class="col-sm-10">
                    <input class="form-control" asp-for="ImageUrl">
                </div>
            </div>
            <div class="form-group row mb-3">
                <div class="col-sm-10 offset-sm-2">
                    <button type="submit" class="btn btn-primary">Ürünü Kaydet</button>
                </div>
            </div>
        </form>
    </div>
</div>
```

Ekran görünümü şu şekilde olacaktır.



Sayfanın üzerinde sağ tuşa tıklayıp sayfa kaynağını görüntülersek her bir alan içinde gerekli alt parametrelerin sınıf yapısından alınarak otomatik olarak oluşturulduğunu görmüş oluruz. Sayfamızdan henüz bilgi girmediğimizden value değeri boş olarak gelmiştir.

```
<div class="row">
    <div class="col-md-8">
        <form method="POST" action="/Admin/CreateProduct">
            <div class="form-group row mb-3">
                <label class="col-sm-2 col-form-label" for="Name">Name</label>
                <div class="col-sm-10">
                    <input class="form-control" type="text" id="Name" name="Name" value="">
                </div>
            </div>
            <div class="form-group row mb-3">
                <label class="col-sm-2 col-form-label" for="Url">Url</label>
                <div class="col-sm-10">
                    <input class="form-control" type="text" id="Url" name="Url" value="">
                </div>
            </div>
```

Biz şimdi form üzerindeki label kısımlarındaki metinlerin farklı dilllerde gözükmesini istersek aşağıdaki şekilde model bilgisine ekleme yapabiliriz. Bu durumda form üzerinde ilgili alan Türkçe olarak yazılmış olacaktır.

```
using System.ComponentModel.DataAnnotations;

namespace TS.WebUI.Models
{
    public class ProductModel
    {
        public int ProductId { get; set; }

        [Display(Name="Ürün adı")]
        public string Name { get; set; }
```

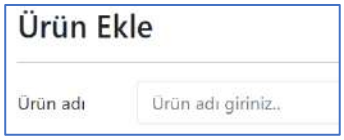
Sayfa kaynağına baktığımızda ilgili alan Label ın görüntülenme alanına yazılmış olmaktadır.

```
<form method="POST" action="/Admin/CreateProduct">
    <div class="form-group row mb-3">
        <label class="col-sm-2 col-form-label" for="Name">&#xDC;r&#xFC;n ad&#x131;</label>
        <div class="col-sm-10">
            <input class="form-control" type="text" id="Name" name="Name" value="">
        </div>
    </div>
```

Bazen textboxlar içinde kişiyi yönlendirmek için açıklama bilgileride yazılabilir. Bunun için aynı yerde Prompt parametresi kullanılabilir.

```
[Display(Name="Ürün adı", Prompt ="Ürün adı giriniz..")]
public string Name { get; set; }
```

Bu durumda çıktı ve sayfa kaynağı aşağıdaki şekilde olacaktır.




```
<div class="form-group row mb-3">
    <label class="col-sm-2 col-form-label" for="Name">&#xDC;r&#xFC;n ad&#x131;</label>
    <div class="col-sm-10">
        <input class="form-control" type="text" id="Name" name="Name" placeholder="&#xDC;r&#xFC;n ad&#x131; giriniz.." value="">
    </div>
</div>
```

Gördüğümüz gibi form üzerinde kullanılan bir çok özellik model yapısı üzerinden kontrol edilebilmekte. Peki sonradan eklediğimiz bu eksra özellikleri neden Entity model içerisinde tanımlayalım ki? Gereksiz yere orayı karmaşık hale getirmiş oluruz. Bu nedenle formlarda kullanılacak model yapılarını Entitylerde kullanılan model yapılarından ayırmakta fayda vardır.

Peki biz artık formdan model içinden gelen bilgileri controller üzerinden veritabanına kaydedeceğiz. Oysa AdminController içindeki CreateProduct() metodumuz bizden Product tipinde bir paket beklemektedir. Form içerisinde ise ProductModel tipinde bir paketleme kullandık. Dolayısı ile metod içerisine girişte karşılayacak olan paket bilgisi ProductModel olacak fakat veritabanına kaydederken bize Product tipinde bir paket lazım

olacağından bu ikisi arasında dönüşümü yapmamız gerekecek. Önce Product tipinde bir entity oluşturalım. Daha sonra bu entity nin içerisini ProductModel paketi ile dolduralım.

Burada henüz validationları (bilgilerin doğru girildiği kontrolünü) kullanmadık. Bu işlemleri Business katmanında yada form modeli içinde DataAnnotionlar kullanarak yapabiliriz. Bu konuyu ileride ele alacağız. Ayrıca bura bir ürünü kaydederken ürüne haz Url bilgisinin daha önceden veritabanında kullanılmadığı doğrulamasını yaptıktan sonra kaydedilmesi gerekir. Bu işlem ise bir business katmanında yapılmalıdır. İleride bunları ele alacağız.

Kodlarımız şu şekidle olacaktır. Denersek veritabanına kaydettikten sonra admin listeleme sayfasına gidip ürünleri gösterecektir. Henüz klasör içerisine 7 nolu resmi eklemediğimizden resim alanı boş olarak gelecektir.

Controllers>**AdminController.cs**

```
using TS.Entity;
using TS.WebUI.Models;

namespace TS.WebUI.Controllers
{
    public class AdminController : Controller
    {
        private IProductService _productService;

        public AdminController(IProductService productService)
        {
            _productService = productService;
        }
        public IActionResult ProductList()
        {
            return View(new ProductListViewModel()
            {
                Products = _productService.GetAll()
            }) ;
        }

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

        [HttpPost]
        public IActionResult CreateProduct(ProductModel model)
        {
            var entity = new Product()
            {
                Name = model.Name,
                Url = model.Url,
                Price = model.Price,
                Description =model.Description,
                ImageUrl =model.ImageUrl
            };
            _productService.Create(entity);

            return RedirectToAction("ProductList");
        }
    }
}
```

Ürün Listesinde Ürün Onaylarının gösterilmesi

Ürünler kaydedilirken veritabanına onaylı olup olmayacağı verilmedi ve varsayılan olarak ürünler veritabanında IsApproved=false olarak kaydedildi. Admin ürünleri listelediğinde hangi üründe ne onayı var, görebilmelidir.

Admin Listeleme sayfasına iki tane kolon ekleyelim ve buradan IsAproved ve IsHomePage özelliklerini atayalım. Öncelikle sayfamızda iconlar kullanacağız. Öncesinde listemizde onaylı ve onaysız ürünleri farklı iconlarla göstermek için aşağıdaki linki aşağıdaki linki _Layout.cshtml sayfasının en üstündeki <head> içerisine yerleştirelim.

```
<head>
    <meta charset="UTF-8">
    <title>TurkSanayisi.com</title>
    <link href="~/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC" crossorigin="anonymous">
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.4.1/css/all.css">
</head>
```

Listenin olduğu ProductList.cshtml sayfasına gidip tabloya yeni iki sütun ekleyelim. Eklenen sütünlar içerisine bir if bloğu açarak eğer veritabanında o alan onaylı ise check-circle ismindeki iconla gösterilsin. Değilse “” icon ile gösterilsin. İconları fontawesome.com sitesine girip oradan html etiketlerini alarak yapalım. Örnek: Bu etiketler içine eklenen text-success ifadesi yeşil renkli, text-danger ifadesi ise iconların kırmızı gözükmesini sağlamıştır.

Listeleme sayfasının kodları aşağıdaki şekilde olmuştur.

Views>Admin>**ProductList.cshtml**

```
@model ProductListViewModel

<div class="row">
    <div class="col-md-12">
        <h1>Admin Ürün Listeleme</h1>
        <hr>
        <table class="table table-bordered">
            <thead>
                <tr>
                    <td style="width:30px">Id</td>
                    <td style="width:100px">Image</td>
                    <td>Name</td>
                    <td style="width:20px">Price</td>
                    <td style="width:20px">IsApproved</td>
                    <td style="width:20px">IsHome</td>
                    <td style="width:150px"></td>
```

```
                </tr>
            </thead>
            <tbody>
                @if (Model.Products.Count > 0)
                {
                    @foreach (var item in Model.Products)
                    {
                <tr>
                    <td>@item.ProductId</td>
                    <td><img src="~/img/@item.ImageUrl" width="80" /> </td>
                    <td>@item.Name</td>
                    <td>@item.Price</td>
                    <td>
                        @if (item.IsApproved)
                        {
                            <i class="fas fa-check-circle text-success"></i>
                        }
                        else
                        {
                            <i class="fas fa-times-circle text-danger"></i>
                        }
                    </td>
                    <td>
                        @if (item.IsHome)
                        {
                            <i class="fas fa-check-circle text-success"></i>
                        }
                        else
                        {
                            <i class="fas fa-times-circle text-danger"></i>
                        }
                    </td>
                    <td>
                        <a href="/admin/products/@item.ProductId" class="btn btn-primary btn-sm mr-
2">Düzenle</a>
                        <a href="#" class="btn btn-danger btn-sm">Sil</a>
                    </td>
                </tr>
                    }
                }
                else
                {
                    <div class="alert alert-varning">
                        <h3>No Products</h3>
                    </div>
                }
            </tbody>
        </table>
    </div>
</div>
```

Çıktısı ise aşağıdaki şekilde olmuştur.

Admin Ürün Güncelleme Sayfasının Oluşturulması

Burada admin ürün listeleme kısmında herhangi bir ürünün Edit butonua tıkladığımızda "/admin/product/5" şeklinde bir Url oluştuğunda buradaki 5 numaralı Id li ürünü Edit sayfasına gütüreceğiz ve orada güncellemeleri yapacağız.

İlk olarak bu şekilde bir URL yi karşılayacak olan bir route oluşturalım. Startup.cs içinde bu url ye uygun route yazalım. Url sonuna Id parametresi gelirse, yani şemaya uyan bir adres gelirse bu durumda Admin controlleri içindeki Edit action metoduna gidecek.

```
//admin/products/
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "adminproductlist",
        pattern: "admin/products/{Id?}",
        defaults: new { controller = "Admin", action = "Edit" }
    );
});
```

{Id?} ifadesi içindeki ? işareti buranın nullable olduğunu gösterir. Yani boş bırakılabilir olduğunu gösteriyor. Eğer burayı nullable yapmazsak boş geldiğinde metod girişinde int ifadesinin karşılığı olarak 0 gelecektir. Bu durumda 0 numaralı ürünü getirmeye çalışacaktır ve hata alırız. Bu soru işaretini eklediğimizde boş gelirse o zaman burasını yok gibi görecektir.

İlk olarak listeleme sayfamızın içindeki Edit (Düzenle) butonu içindeki Url adresimizi kontrol edelim. Burada link adresimizin verildiğini görmekteyiz.

```
<td>
    <a href="/admin/products/@item.ProductId" class="btn btn-primary btn-sm mr-2">Düzenle</a>
    <a href="#" class="btn btn-danger btn-sm">Sil</a>
</td>
```

Şimdi Admin controller içinde Edit() işlemi için iki adet action metodu oluşturacağız. Bir tanesi bilgilerin form sayfasına dolması için HttpGet() metodu ile çalışacak, diğeri ise dolan bilgilerin güncellenmesi için HttpPost() metodu ile çalışacak.

Get metodu ile çalışan metod gelen Id bilgisine göre bir sorgulama yapacak ve bu bilgileri form'a dolduracak. (`int? Id`) ifadesindeki soru işareti Id bilgisinin boş gelebilir olduğunu (nullable) gösterir. Yani Id bilgisi gelmez ise int ifadesi nedeniyle içerisine sıfır atamayacak. Çünkü Id=0 olması durumunda ürün bulunamayacağından bir hata oluşur.

Dikkat edersek id null olduğunda yada entity (product tipinde bir nesne yani bir ürün) bulunamadığında NotFound() sayfasına gidecek (henüz daha sayfa oluşturulmadı). Eğer bir ürün var ise (yani entity bulundu ise) bu entity Product sınıfı tipindedir. Fakat bizim form sayfamıza gitmesi gereken ürün tipi ise ProductModel tipinde olmalıdır. Çünkü hatırlarsak form sayfalarında bazı alanları çıkarıp ona uygun sınıf tipi olarak ProductModel oluşturmuştuk. Bu tipe dönüştürerek göndececeğiz.

```
        [HttpGet]
        public IActionResult Edit(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var entity=_productService.GetById((int)id);

            if (entity == null)
            {
                return NotFound();
            }

            var model = new ProductModel()
            {
                ProductId = entity.ProductId,
                Name=entity.Name,
                Url = entity.Url,
                Price = entity.Price,
                ImageUrl =entity.ImageUrl,
                Description = entity.Description
            };
```

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

Şimdi bu Get metodu ile çalışacak olan metodun yönleneceği Views>Admin>Edit.cshtml form sayfasını aşağıdaki şekilde oluşturmuş oluruz. CreateProduct sayfasındaki form yapısını aynen kopyalayarak üzerinde düzenleme yapabiliriz.

Burada form sayfasına ürüne ait bilgiler ProductModel içinde taşınmaktadır ve ilgili alanlar form nesneleri içinde otomatik olarak görüntülenmektedir. Görüntülenen bu bilgiler üzerinde bir değişiklik yapıp Butona tıkladığımızda yine bu bilgiler model içerisinde [HttpPost] tipindeki Edit() metoduna gidecektir. Yalnız bura model içinde taşınan bilgiler sayfamızda kullandığımız bilgiler olacaktır. Yani sayfada nesneler içinde gösterilen alanlar taşınacaktır. Oysa biz veritabanında güncelleme yapabilmek için butona tıkladığımızda ProductId bilgisinin de taşınması gerekir. Fakat bu alan form içinde gösterilmemektedir. En azından kullanmış olmak ve kullanıcıya göstermemek için hidden şeklinde forma eklemeliyiz.

Views>Admin>**Edit.cshtml**

```
@model ProductModel
<h1 class="h3">Ürün Güncelle</h1>
<hr>
<div class="row">
    <div class="col-md-8">
        <form asp-controller="Admin" asp-action="Edit" method="POST">
            <input type="hidden" name="ProductId" value="@Model.ProductId" />
            <div class="form-group row mb-3">
                <label asp-for="Name" class="col-sm-2 col-form-label"></label>
                <div class="col-sm-10">
                    <input class="form-control" asp-for="Name">
                </div>
            </div>
            <div class="form-group row mb-3">
                <label asp-for="Url" class="col-sm-2 col-form-label"></label>
                <div class="col-sm-10">
                    <input class="form-control" asp-for="Url">
                </div>
            </div>
            <div class="form-group row mb-3">
                <label asp-for="Description" class="col-sm-2 col-form-label"></label>
                <div class="col-sm-10">
                    <textarea class="form-control" asp-for="Description"></textarea>
                </div>
            </div>
            <div class="form-group row mb-3">
                <label asp-for="Price" class="col-sm-2 col-form-label"></label>
                <div class="col-sm-10">
                    <input class="form-control" asp-for="Price">
                </div>
            </div>
            <div class="form-group row mb-3">
                <label asp-for="ImageUrl" class="col-sm-2 col-form-label"></label>
                <div class="col-sm-10">
                    <input class="form-control" asp-for="ImageUrl">
                </div>
            </div>
            <div class="form-group row mb-3">
                <div class="col-sm-10 offset-sm-2">
                    <button type="submit" class="btn btn-primary">Ürünü Güncelle</button>
                </div>
            </div>
        </form>
    </div>
</div>
```

Submit butonuna tıkladığımızda ProductModel içinde değiştirilmiş ürün bilgileri Controller içindeki post yöntemiyle çalışan Edit() metodu içine gelecektir. Buradan ürün bilgilerini yine (ProductModel model) şeklinde model nesnesi içine alırız. Bu nesne içindeki bilgilerde değişiklik olmuştur. Bu değişiklikleri entity şeklindeki Product dan üretilmiş nesnenin içine atıp onu veritabanına güncellemeye göndermemiz gerekir. Yani burada

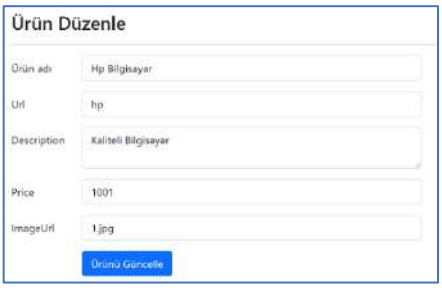
yeniden bir tane daha Product model tipinde entity nesnesi üretmeliyiz ve sayfadan gelen bilgileri (model nesnesi içinde) bunun içine atmalıyız.

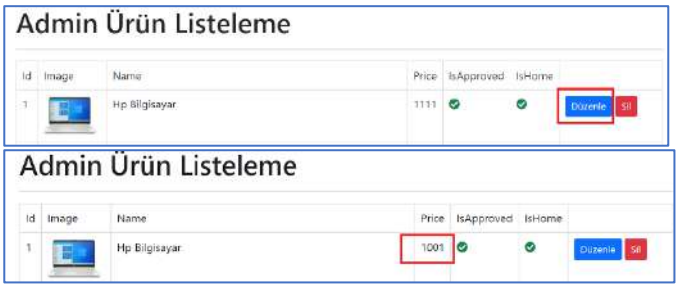
```
[HttpPost]
public IActionResult Edit(ProductModel model)
{
    //Ürünün varlığını VT den kontrol için eklendi ürün bilgileri VT den okunuyor.formdan gelen Id ile aynı olan ürünün bilgileri okunuyor.
    var entity = _productService.GetById(model.ProductId);
    if(entity==null)
    {
        return NotFound();
    }

    //sayfaya model olarak gelen bilgiler sayfada oluşturulan entity nesnesinin içine güncel haliyle taşınıyor
    entity.Name = model.Name;
    entity.Price = model.Price;
    entity.Url = model.Url;
    entity.Description = model.Description;
    entity.ImageUrl = model.ImageUrl;

    //bilgileri değiştirilmiş entity nesnesi veritabanında güncelleniyor.
    _productService.Update(entity);

    //güncelleme yapıldıktan sonra yukarıdaki ProductList() metoduna yönlendiriliyor ve tüm ürünler burada son haliyle listeleniyor.
    return RedirectToAction("ProductList");
}
```

Daha önceden iş katmanında ProductManager.cs içinde update ile ilgili metodun içi yazılmamıştı. Bu kısmı aşağıdaki şekilde düzeltelim.

```
public void Update(Product entity)
{
    // iş kuralları buraya yazılır
    _productRepository.Update(entity);
}
```

Kodları çalıştırdığımızda çıktımız aşağıdaki şekilde olmuş olur.





Admin Ürün Silme

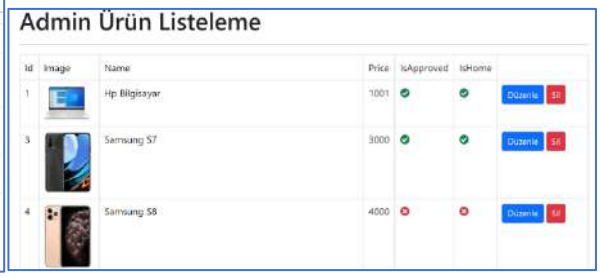
Silme işlemini Listeleme tablosu içinden yapacağız. İlgili satırdaki "sil" butonuna tıkladığımızda ürünü silmiş olacağız. Bunun için Delete butonunu bir form içine alacağız ve formun metodunu Post olarak ayarlayacağız. İlgili satırdaki hidden olarak saklanan Id bilgisi ile controllera gidip oradan silme işlemini yapacak metod içinde işlemi gerçekleştireceğiz.

Şimdi ilk olarak Views>Admin>ProductList.cshtml sayfasından Delete butonun bir form içine alalım. Formun gideceği controller ve action metodunu `action="/admin/deleteproduct"` şeklinde atayalım. Giderken yanımızda ProductId yi götüreceğimizden `<input type="hidden" name="productId" value="@item.ProductId" />` satırı ile bu alanı hidden olarak form içine ekledik. Butonumuz link olarak değil (yani <a>) submit butonu olarak belirledik. Yani butonun içinde bulunduğu formu servera gönderecek. Düzenle butonu ve Sil butonu normalda yan yana görünüyordu. Fakat <form> etiketi eklenince burası blok olarak görüntüleneceğinden alt alta gözükecektir. Bunları yan yana göstermek için formun içine `style="display:inline;"` özelliği eklendi.

```
<td>
    <a href="/admin/products/@item.ProductId" class="btn btn-primary btn-sm mr-2">Düzenle</a>

    <form action="/admin/deleteproduct" method="post" style="display:inline;">
        <input type="hidden" name="productId" value="@item.ProductId" />
        <button type="submit" class="btn btn-danger btn-sm">Sil</button>
    </form>

</td>
```

Kodları çalıştırınca görünmüünde bir sorun gözükmemektedir.

Şimdi AdminController içine action metodumuzu oluşturalım.



Bilgilendirme Mesajları

Herhangi bir veritabanı işlemi yaptıktan sonra (ürün silme, güncelleme, kaydetme vs) kullanıcıya bir bilginin verilmesi gerekir. Burada bu tür bir bilgiyi ürünle ilgili işlemi yaptıktan sonra Listeleme sayfasına gidiyorduk. Bu listeleme sayfası içinde mesajlarımızı gösterelim. Listeleme sayfasına giderken yanımızda mesaj içinde hangi ürünle ilgili işlem yaptığımız bilgisini ve gösterim rengi ile ilgili bilgiyide götürmemiz gerekir. Zira ürün kaydetme, güncelleme gibi işlemler yeşil tonlarında gösterilirken, ürün silme işlemini kırmızı tonlarında göstermek gerekir. Burada mesaj ve renk bilgisini Listeleme sayfasına taşırken 3 farklı yöntem kullanarak yapacağız.

a) TempData içinde sadece metin bilgisini taşıma:

Ürünün kaydettikten sonra aşağıdaki şekilde Listeleme sayfasına yönlenmeden önce ViewData içinde “Message” isminde bir değişken oluşturup metni taşıyalım. Dikkat edersek metnin içine ürün adını ekledik. Yazım şekline dikkat ediniz.

```
[HttpPost]
public IActionResult CreateProduct(ProductModel model)
{
    var entity = new Product()
    {
//***
    };
    _productService.Create(entity);

    TempData["Message"] = $"{entity.Name} isimli ürün eklendi";

    return RedirectToAction("ProductList");
}
```

Listeleme sayfasının üst kısmına da aşağıdaki kodları ekleyelim.

Not: Yukarıdaki kodları ViewData[“Message”] şeklindede göndermeyi deneyebilirdik fakat ViewData sadece Controllar içindeki bir metoddan direk View sayfasına gidiyorsa kullanılabilir. Oysa burda ürünü kaydeden metoddan listeleme metoduna gidiyoruz, ordanda View sayfasına gidiyoruz. Bu nedenle ViewData içinde bilgileri taşıyamıyoruz. Onun yerine metodlar arasında da bilgiyi taşıyabilmek için TempData[“”] nesnesi kullanıldı.

ProductList.cshtml sayfasının baş tarafına mesajı göstermek için eklenen kodlar aşağıdaki gibi oldu.

```
@model ProductListViewModel
@if (TempData["Message"] != null)
{
    <div class="row">
        <div class="col-md-12">
            <div class="alert alert-success">
                @TempData["Message"]
            </div>
        </div>
    </div>
}
```

Ürün ekleme için verdirilen bu mesajın benzeri, güncelleme ve silme işlemleri içinde yapılabilir. Bunun için ilgili metodların içine benzer kodları ekleyelim. Her işlemden sonra Listeleme yaptığımızdan oluşturulan mesaj listelenin başında gösterilecektir.

```
[HttpPost]
public IActionResult Edit(ProductModel model)
{
TempData["Message"] = $"{entity.Name} isimli ürün güncellendi";
}
public IActionResult DeleteProduct(int productId)
{
TempData["Message"] = $"{entity.Name} isimli ürün silindi";
}
```

Bir çok sayfada bu tipte bir mesaj kullanabiliriz. Dolayısı ile bu mesajları sadece Listeleme sayfası içinde kullanmak yerine sayfaların en üstünde istendiğinde göstermek üzere kullanmak için _Layout.cshtml şablonunda RenderBody() üzerinde bu gösterimi aktif edebiliriz.

```
<main class="mt-3">
    <div class="container">

        @if (TempData["Message"] != null)
        {
```

```
            <div class="row">
                <div class="col-md-12">
                    <div class="alert alert-success">
                        @TempData["Message"]
                    </div>
                </div>
            </div>
        }

        @RenderBody()
    </div>
</main>
```

Bu yapıyı ile ürün güncelleme ve silme işlemleri için deneyelim.

Fakat burada bir sorunumuz var. Hangi mesajı gösterirsek gösterelim tüm mesajlar yeşil gelmektedir. Oysa mesajın içeriğine uygun olarak renklendirme yapmak gerekir. Örneğin silme işlemlerinde kırmızı tonlarında bir mesaj kullanılması daha iyi olur. Bu durumda mesaj metni ile birlikte renk bilgisini de götürmeliyiz.

Böyle bir işlem için bir string taşımak yerine bir class taşıyabiliriz. Class içerisinde de hem mesaj bilgisi hemde renk bilgisi bulunabilir.

Bunun için Model klasörü içinde bir class oluşturup içerisini aşağıdaki şekilde dolduralım.

```
namespace TS.WebUI.Models
{
    public class AlertMessage
    {
        public string Message { get; set; }
        public string AlertType { get; set; }
    }
}
```

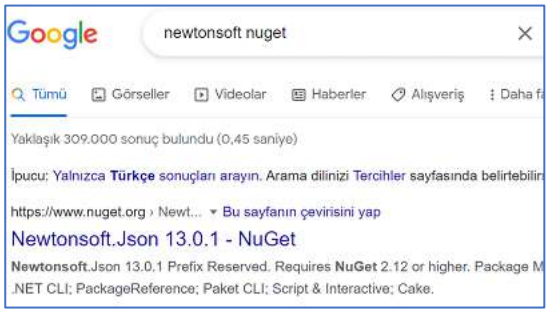
Bu yapıyı AdminController içindeki metodlarda şu şekilde kullanmaya çalıştığımızda hata alırız. Çünkü bir nesnenin elemanlarını TempData içinde taşıyabilmek için Serilization işlemi yapmak gerekir. TempData ise serverın Session objesini kullandığı için bu mümkün olmayacaktır.

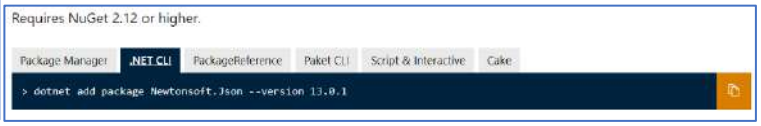
```
TempData["Message"] = new AlertMessage()
{
    Message = $"{entity.Name} isimli ürün eklendi",
    AlertType = "success"
};
```

Bu nedenle Class yapımızı Json objesi oluşturarak onun içinde taşıyalım.

JSON (JavaScript Object Notation); bütün programlama dilleri arasında, yapılandırılmış veri değişimini kolaylaştıran bir metin biçimidir. Yay ayraç, köşeli ayraç, iki nokta ve virgüllü yazımı ile birçok bağlam ve uygulamada kullanışlıdır.

Json dilini serilization işlemi olarak kullanabilmek için newtonsoft kütüphanesini eklememiz gerekiyor. Google aşağıdaki ifadeyi yazıp karşımıza gelen nuget.org linkine tıklayalım. Dotnet kullanarak yüklemek için olan komut grubunu kopyalayalım.



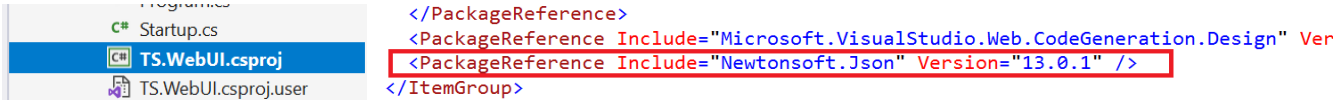


Bu linki komut satırından TS.WebUI projesi içine kadar gidip oraya yapıştıralım ve entera tıklayıp kütüphaneyi kurmuş olalım.

```
C:\Users\pc1>cd desktop

C:\Users\pc1\Desktop>cd TS.com

C:\Users\pc1\Desktop\TS.com>cd TS

C:\Users\pc1\Desktop\TS.com\TS>cd TS.WebUI

C:\Users\pc1\Desktop\TS.com\TS\TS.WebUI>dotnet add package Newtonsoft.Json --version 13.0.1
```

```
info : Varlıklar dosyası diske yazılıyor. Yol: C:\Users\pc1\Desktop\TS.com\TS\TS.WebUI\obj\project.assets.json
log  : C:\Users\pc1\Desktop\TS.com\TS\TS.WebUI\TS.WebUI.csproj geri yüklendi (3 sec içinde).

C:\Users\pc1\Desktop\TS.com\TS\TS.WebUI>
```

Kütüphane kurulduktan sonra WebUI projesinin csproj dosyası içine bakarsak ilgili kütüphanenin eklendiğini görebiliriz.



Bu kütüphane eklendikten sonra sınıf yapımızı Json nesnesi ile serilize yapıp TempData içinde ilgili sayfalarımıza taşıyabiliriz.

```
using Newtonsoft.Json;

//*****

var msg  = new AlertMessage()
{
    Message = $"{entity.Name} isimli ürün eklendi",
    AlertType = "success"
};

TempData["Message"] = JsonConvert.SerializeObject(msg);
```

Burada Json nesnesinin yaptığı işlem aslında bilgileri bir formatlı string şekline dönüştürmektir. Şunun gibi bir ifade olmuş oluyor.

??????

Serilize edilmiş ifadeler (tek bir string olarak formatlanmış metin) kullanırken tekrar Deserilization yapılması gerekir. Yani string ifade çözülüp tekrar değişkenlere yüklenmelidir.

_layout.cshtml sayfası içinde aşağıdaki kodları kullandığımızda TempData içinde serilize bir şekilde gelen ifade AlertMessage sınıfı yapısına uygun olacak şekilde Deserilize edilerek (çözme) message nesnesi içine bilgiler yüklenmektedir. Daha sonra message alt özellikleri kullanılarak alert kutusu gösterilmektedir.

```
@if (TempData["Message"] != null)
{
    var message = JsonConvert.DeserializeObject<AlertMessage>(TempData["message"] as String)
    <div class="row">
        <div class="col-md-12">
            <div class="alert alert-@message.AlertType">
                @message.Message
            </div>
        </div>
    </div>
}
```

.cshtml sayfaları içinde Json komutlarının kullanabilmek için üst kısımda kütüphanesini eklemeliyiz. Fakat biz projede bu tip sayfalar için kütüphaneyi sayfa içine değil, diğer tüm sayfalarda kullanabilmek için _ViewImports.cshml sayfası içine ekliyorduk. Oraya aşağıdaki şekilde kütüphanesini ekleyelim.

```
@using TS.Entity;

@using TS.WebUI.Models;

@using Newtonsoft.Json;
```

Benzer şekilde üç işlem içinde (yeni kayıt, güncelleme ve silme) metodlar içinde alert mesajlarını oluşturalım..

Kodları denersek aşağıdaki şekilde her bir farklı işlem için uygun renkte alert mesajları çıkacaktır.

Admin Kategori Sayfalarının Hazırlanması

Kategori için yapacağımız adımlar Ürün için yaptığımız adımların aynısıdır. Yani admin kategori ekleyip bu kategorileri listeleyebilmesi gerekir. Ürünleri listelediğimizde ise Edit ve Delete işlemlerinide yapmamız gerekir.

NavBar içinde Admin Yeni Ürün ekleme linki oluşturmuş. Bu linki buradan alıp listeleme sayfasının içine koyalım. Böylece NavBar içinde yer kazanmış olalım. NavBar içinde sadece Ürünler ve Kategoriler şeklinde iki link olsun.

Admin için _navBar.cshmtl içindeki linklerimiz şu ikisinden oluşsun.

```
<li class="nav-item">
    <a href="/admin/products/" class="nav-link">(Admin)Ürünler</a>
</li>

<li class="nav-item">
    <a href="/admin/categories/" class="nav-link">(Admin)Kategoriler</a>
</li>
```

Ürün Listeleme ve Kategori listeleme sayfaları içine Ürün Ekle ve Kategori ekle linklerini koyalım. Bu linki koyarken URL linklerimizide biraz değiştirelim. “admin/product/create” dediğimizde bir ürün ekleyeceğimiz anlaşılsın. Benzer şekilde “admin/category/create” dediğimizde de bir kategori ekleyeceğimiz anlaşılsın.

ProductList.cshtml nin üst kısmına aşağıdaki kodları ekleyelim.

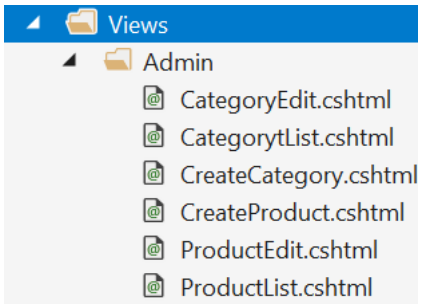
```
        <h1>Admin Ürün Listeleme</h1>
        <hr>
        <a class="btn btn-primary btn-sm" href="admin/products/create" >Ürün Ekle</a>
```

Bu linkin route adres desenlerini starup.cs içinde düzeltelim. Yeni Route larımız aşağıdaki şekilde olacaktır.

```
//1---admin/products/  PRODUCT -- LIST
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "adminproductlist",
        pattern: "admin/products/",
        defaults: new { controller = "Admin", action = "ProductList" }
    );
});

//2---admin/products/create   PRODUCT -- CREATE
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "adminproductcreate",
        pattern: "admin/products/create",
        defaults: new { controller = "Admin", action = "ProductCreate" }
    );
});

//3---admin/products/Id?    PRODUCT --- EDİT
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "adminproductedit",
        pattern: "admin/products/{Id?}",
        defaults: new { controller = "Admin", action = "ProductEdit" }
    );
});

//4---admin/categories/       CATEGORY --- LİST
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "admincategorylist",
        pattern: "admin/categories/",
        defaults: new { controller = "Admin", action = "CategoryList" }
    );
});

//5---admin/categories/create    --- CATEGORY --- CREATE
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "admincategorycreate",
        pattern: "admin/categories/create",
        defaults: new { controller = "Admin", action = "CategoryCreate" }
    );
});

//6---admin/categories/Id?    CATEGORY --- EDİT
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "admincategoryedit",
        pattern: "admin/categories/{Id?}",
        defaults: new { controller = "Admin", action = "CategoryEdit" }
    );
});
```

Burada temel 4 işlemden 3 için (Listeleme, oluşturma, düzenleme) bir sayfaya gideceğimizden link desenlerimizi oluşturduk. Silme işlemi için bir sayfaya gitmeyeceğimizden bunun için link deseni oluşturmadık.

View>Admin klasörü içinde eksik sayfalarımızı oluşturalım. İlgili sayfalarımız aşağıdaki şekilde olsun. Product için oluşturduklarımızın bir benzerini Kategori içinde oluşturalım.



CategoryList.cshtml sayfamızın içeriği aşağıdaki şekilde olacaktır.

Views>Admin> **CategoryList.cshtml**

```
@model CategoryListViewModel

<div class="row">
    <div class="col-md-12">
        <h1 class=”h3”>Admin Kategori Listeleme</h1>
        <hr>
        <a class="btn btn-primary btn-sm" href="/admin/categories/create">Kategori Ekle</a>

        <table class="table table-bordered mt-3 ">
            <thead>
                <tr>
                    <td style="width:30px">Id</td>
                    <td style="width:100px">Name</td>
                </tr>
            </thead>
            <tbody>
                @if (Model.Categories.Count > 0)
                {
                    @foreach (var item in Model.Categories)
                    {
                        <tr>
                            <td>@item.CategoryId</td>
                            <td>@item.Name</td>

                            <td>
                                <a href="/admin/categories/@item.CategoryId" class="btn btn-primary btn-sm mr-
2">Düzenle</a>

                                <form action="/admin/deletecategory" method="post" style="display:inline;">
                                    <input type="hidden" name="productId" value="@item.CategoryId" />
                                    <button type="submit" class="btn btn-danger btn-sm">Sil</button>
                                </form>

                            </td>
                        </tr>
                    }
                }
                else
                {
                    <div class="alert alert-varning">
                        <h3>No Categories</h3>
                    </div>
                }
            </tbody>
        </table>
    </div>
</div>
```

Sayfalarda kullanacağımız Model class larınıda oluşturalım. ProductViewModel sınıfımızın bir benzerini Kategori içinde oluşturursak aşağıdaki şekilde olabilir.

```
using System.Collections.Generic;
using TS.Entity;

namespace TS.WebUI.Models
{
    public class CategoryListViewModel
    {
        public List<Category> Categories { get; set; }
```

```
    }
}
```

Şimdi AdminController.cs sayfamızın içinde adreslerin yönlendiği Action metodlarımızı oluşturalım. İlk Başta IcategorService injection yapalım. Böylece categoryService içindeki bütün metodları kullanmış olalım. Tüm kodlar şu şekilde olacaktır.

AdminController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using TS.Business.Abstract;
using TS.Entity;
using TS.WebUI.Models;

namespace TS.WebUI.Controllers
{
    public class AdminController : Controller
    {
        private IProductService _productService;
        private ICategoryService _categoryService;
        public AdminController(IProductService productService, ICategoryService categoryService)
        {
            _productService = productService;
            _categoryService = categoryService;
        }
        public IActionResult ProductList()
        {
            return View(new ProductListViewModel()
            {
                Products = _productService.GetAll()
            }) ;
        }
        public IActionResult CategoryList()
        {
            return View(new CategoryListViewModel()
            {
                Categories = _categoryService.GetAll()
            });
        }
        [HttpGet]
        public IActionResult ProductCreate()
        {
            return View();
        }

        [HttpPost]
        public IActionResult ProductCreate(ProductModel model)
        {
            var entity = new Product()
            {
                Name = model.Name,
                Url = model.Url,
                Price = model.Price,
                Description =model.Description,
                ImageUrl =model.ImageUrl
            };
            _productService.Create(entity);
            var msg  = new AlertMessage()
            {
                Message = $"{entity.Name} isimli ürün eklendi",
                AlertType = "success"
            };
            TempData["Message"] = JsonConvert.SerializeObject(msg);
            return RedirectToAction("ProductList");
        }
        [HttpGet]
        public IActionResult CategoryCreate()
        {
            return View();
        }
        [HttpPost]
        public IActionResult CategoryCreate(CategoryModel model)
```

```
        {
            var entity = new Category()
            {
                Name = model.Name,
                Url = model.Url,
            };
            _categoryService.Create(entity);
            var msg = new AlertMessage()
            {
                Message = $"{entity.Name} isimli kategori eklendi",
                AlertType = "success"
            };
            TempData["Message"] = JsonConvert.SerializeObject(msg);
            return RedirectToAction("CategoryList");
        }
        [HttpGet]
        public IActionResult ProductEdit(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var entity = _productService.GetById((int)id);
            if (entity == null)
            {
                return NotFound();
            }
            var model = new ProductModel()
            {
                ProductId = entity.ProductId,
                Name = entity.Name,
                Url = entity.Url,
                Price = entity.Price,
                ImageUrl = entity.ImageUrl,
                Description = entity.Description
            };
            return View(model);
        }
        [HttpPost]
        public IActionResult ProductEdit(ProductModel model)
        {
            var entity = _productService.GetById(model.ProductId);
            if (entity == null)
            {
                return NotFound();
            }
            entity.Name = model.Name;
            entity.Price = model.Price;
            entity.Url = model.Url;
            entity.ImageUrl = model.ImageUrl;
            entity.Description = model.Description;
            _productService.Update(entity);
            var msg = new AlertMessage()
            {
                Message = $"{entity.Name} isimli ürün güncellendi",
                AlertType = "success"
            };
            TempData["Message"] = JsonConvert.SerializeObject(msg);
            return RedirectToAction("ProductList");
        }
        [HttpGet]
        public IActionResult CategoryEdit(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            var entity = _categoryService.GetById((int)id);
            if (entity == null)
            {
                return NotFound();
            }
            var model = new CategoryModel()
            {
                CategoryId = entity.CategoryId,
                Name = entity.Name,
                Url = entity.Url
            };
```

```
            return View(model);
        }
        [HttpPost]
        public IActionResult CategoryEdit(CategoryModel model)
        {
            var entity = _categoryService.GetById(model.CategoryId);
            if (entity == null)
            {
                return NotFound();
            }
            entity.Name = model.Name;
            entity.Url = model.Url;
            _categoryService.Update(entity);
            var msg = new AlertMessage()
            {
                Message = $"{entity.Name} isimli kategori güncellendi",
                AlertType = "success"
            };
            TempData["Message"] = JsonConvert.SerializeObject(msg);
            return RedirectToAction("CategoryList");
        }
        public IActionResult DeleteProduct(int productId)
        {
            var entity = _productService.GetById(productId);
            if(entity !=null)
            {
                _productService.Delete(entity);
            }
            var msg = new AlertMessage()
            {
                Message = $"{entity.Name} isimli ürün silindi",
                AlertType = "danger"
            };
            TempData["Message"] = JsonConvert.SerializeObject(msg);
            return RedirectToAction("ProductList");
        }
        public IActionResult DeleteCategory(int categoryId)
        {
            var entity = _categoryService.GetById(categoryId);
            if (entity != null)
            {
                _categoryService.Delete(entity);
            }
            var msg = new AlertMessage()
            {
                Message = $"{entity.Name} isimli kategori silindi",
                AlertType = "danger"
            };
            TempData["Message"] = JsonConvert.SerializeObject(msg);
            return RedirectToAction("CategoryList");
        }
    }
}
```

Burada bizim için gerekli olan CategoryModel.cs yapımısız oluşturalım. Dikkat edersek normalde kategorileri taşımak için TS.Entity içindeki Categories.cs class yapılarını kullanıyorduk. Fakat formlarda bu class yapısını kullanmıyoruz. Bunun gerekçesi yukarılarda geçmişti. Formlarda kullandığımız yapı biraz daha farklı olabilir ayrıca form işlemleri için sınıf yapısı içine bazı ek kodlamalar yazmamız gerekebilir. Örneğin Validation kodlamaları gibi.

Model>**CategoryModel.cs**

```
using System.ComponentModel.DataAnnotations;

namespace TS.WebUI.Models
{
    public class CategoryModel
    {
        public int CategoryId { get; set; }
        public string Name { get; set; }
        public string Url { get; set; }

    }
```

```
}
```

Kategorilere ait diğer sayfalarda Product sayfalarının kopyası çıkarılıp değiştirilerek oluşturulur. Bunlar CategoryList, cshtml, CategoryCreate.cshtml, CategoryEdit.cshtml sayfalarıdır.

CategoryCreate.cshtml

```
@model CategoryModel
<h1 class="h3">Kategori Ekle</h1>
<hr>
<div class="row">
    <div class="col-md-8">
        <form asp-controller="Admin" asp-action="CategoryCreate" method="POST">
            <div class="form-group row mb-3">
                <label asp-for="Name" class="col-sm-2 col-form-label"></label>
                <div class="col-sm-10">
                    <input class="form-control" asp-for="Name">
                </div>
            </div>
            <div class="form-group row mb-3">
                <label asp-for="Url" class="col-sm-2 col-form-label"></label>
                <div class="col-sm-10">
                    <input class="form-control" asp-for="Url">
                </div>
            </div>
            <div class="form-group row mb-3">
                <div class="col-sm-10 offset-sm-2">
                    <button type="submit" class="btn btn-primary">Kategori Kaydet</button>
                </div>
            </div>
        </form>
    </div>
</div>
```

CategoryEdit.cshtml

```
@model CategoryModel
<h1 class="h3">Kategori Düzenle</h1>
<hr>
<div class="row">
    <div class="col-md-8">
        <form asp-controller="Admin" asp-action="CategoriEdit" method="POST">
            <input type="hidden" name="CategoryId" value="@Model.CategoryId" />
            <div class="form-group row mb-3">
                <label asp-for="Name" class="col-sm-2 col-form-label"></label>
                <div class="col-sm-10">
                    <input class="form-control" asp-for="Name">
                </div>
            </div>
            <div class="form-group row mb-3">
                <label asp-for="Url" class="col-sm-2 col-form-label"></label>
                <div class="col-sm-10">
                    <input class="form-control" asp-for="Url">
                </div>
            </div>
            <div class="form-group row mb-3">
                <div class="col-sm-10 offset-sm-2">
                    <button type="submit" class="btn btn-primary">Kategori Güncelle</button>
                </div>
            </div>
        </form>
    </div>
</div>
```

Category için TS.Entity, TS.Data, TS.Business katmanlarındaki sınıf yapıları yine benzer olarak Product için oluşturulanlar gibi yapılır. Çoğu metod zaten generic idi (hangi tip entity olursa olsun silme, güncelleme, Id ye göre getirme vs gibi işlemler) hepsi için otomatik yapılıyordu.Sayfaları çalıştıralım.

Kategori-Ürün Arasındaki İlişkilerin Oluşturulması

Herhangi bir Kategoriyi düzenlediğimizde, o kategoriye ait ürünlerimizide listeleyelim. Bunun için kategorileri GetById() metodu ile getiriyorduk. Bu sefer bu kategoriye bağlı ürünlerle birlikte gelmesi içinde GetByIdWithProducts(Id) şeklinde bir metod oluşturabiliriz.

Bu işlem için TS.Data>Abstract>ICategoryRepository.cs içinde dışırıdan CategoryId alıp geriye bir Category gönderen satırı yazalım.

TS.Data.Abstract> **ICategoryRepository.cs**

```
using System;
using System.Collections.Generic;
using System.Text;
using TS.Entity;

namespace TS.Data.Abstract
{
    public interface ICategoryRepository : IRepository<Category>
    {
        Category GetByIdWithProducts(int CategoryId);
    }
}
```

Metodumuzun özet kısmını oluşturduk. Dolu kısmını concrete içinde oluşturalım. Burada özet metodumuzun dolu versiyonunu "Impletment Interface" yöntemiyle Concrete içindeki EfCoreCategoryRepository.cs içinde daha önce de çok kez yaptığımız gibi oluşturalım. Metod içinde TSContext den bir contex isminde nesne türetelim. TSContext içinde standart Database işlemleri için (DbSet kullanarak) metod oluşturmuştuk. Burada yapılan sorgulamada Where ifadesi ile dışarıdan gönderelen kategori bilgisi ile aynı olan kategoriyi alacaktır. Bu sorgulamanın üzerine Include ifadesi ile ekstra bir sorgulama yapıyoruz. Yani burada Include(i => i.ProductCategories) ifadesi ile Kategori tablosundan ProductCategories tablosuna geçiyoruz. Bu iki tablo arasında zaten bir bağlantı vardı. Ardından ThenInclude ifadesi ile de ortak tablo olan ProductCategories tablosundan da Products tablosuna geçiyoruz. Böylece en başta belirlediğimiz Kategorilere ait olan Ürünleri getirmiş oluyoruz. Bu ikisi arasında ise ilişki tablosu olan ProductCategories tablosunu kullanmış oluyoruz.

Kodların devamı şu şekilde olacaktır. Where ve Include komutları için İlgili kütüphaneleride üstte eklemeliyiz. Include ifade Entity framework un bir komutudur. Where ise SQL komutudur Linq kütüphasini ister.

TS.Data.Concrete.EfCore> **EfCoreCategoryRepository**

```
using Microsoft.EntityFrameworkCore;
using System.Linq;
```

```
using TS.Data.Abstract;
using TS.Entity;

namespace TS.Data.Concrete.EfCore
{
    public class EfCoreCategoryRepository : EfCoreGenericRepository<Category, TSContext>, ICategoryRepository
    {
        public Category GetByIdWithProducts(int categoryId)
        {
            using (var context = new TSContext())
            {
                return context.Categories
                              .Where(i => i.CategoryId == categoryId)
                              .Include(i => i.ProductCategories)
                              .ThenInclude(i => i.Product)
                              .FirstOrDefault();
            }
        }

    }
}
```

Bu metodları Servis (Business) katmanına da eklemeliyiz. Business katmanında Abstract ve Concrete içinde ilgili classların içine aşağıdaki kodları eklemiş olalım.

```
public interface ICategoryService
 {
     Category GetByIdWithProducts(int categoryId);
```

Concrete içine ise

```
public class CategoryManager : ICategoryService
 {

     public Category GetByIdWithProducts(int categoryId)
     {
         return _categoryRepository.GetByIdWithProducts(categoryId);
     }
```

Artık AdminController içindeki Category yi edit yaptığımız metodların içinde GetById() metodlarını değil GetByIdWithProducts() metodlarını kullanacağız. Önceki metodların isimlerini bu şekilde değiştireceğiz.

`var entity = _categoryService.GetByIdWithProducts((int)id);` satırı ile kategorimizi ve ona bağlı ürünlerimizi alıp bunları entity içinde tutuyoruz. Bu entity içindeki ürünlerimizle birlikte sayfaya taşımak için sayfadaki form yapısı için oluşturduğumuz Model>CategoryModel.cs sınıfı içinde ürünleri taşıyabilecek bir yapının da oluşturulmuş olması gerekir.

```
public class CategoryModel
{

   public List<Product> Products { get; set; }
```

Controller içinde ilgili metod içinde aşağıdaki değişiklikleri yapıyoruz.

```
public class AdminController : Controller
{

    public IActionResult CategoryEdit(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }

        var entity = _categoryService.GetByIdWithProducts((int)id);
```

```
            if (entity == null)
            {
                return NotFound();
            }

            var model = new CategoryModel()
            {
                CategoryId = entity.CategoryId,
                Name = entity.Name,
                Url = entity.Url,
                Products = entity.ProductCategories.Select(p => p.Product).ToList()
            };
            return View(model);
        }
```

`Products = entity.ProductCategories.Select(p => p.Product).ToList()` işleminde entity içinde bulunan ProductCategories bilgisine geçiş yapacağız. Bu bilgiler içinden bir Select işlemi gerçekleştireceğiz ve işaret ettiği ürünleri alıp bir liste içine atıyoruz. Bu listede yine List tipinde olan Products içinde atılmış oluyor. Biz bu ifade ile bir sorgu oluşturmuyoruz. Zaten entity içine bilgileri veritabanından alınırken data katmanında sorgu yazılmıştı. Buraya bu bilgiler zaten geldi. Şimdi bu entity içinde bilgilerin içinden ürünleri bir liste içine atıyoruz.

Controllerdan sayfaya giderken yanımızda hem kategori bilgisi hemde ona bağlı ürün bilgilerini götürmüş oluyoruz. Bu bilgileri ilgili sayfada (CategoryEdit.cshtml) görüntüleyeceğiz.

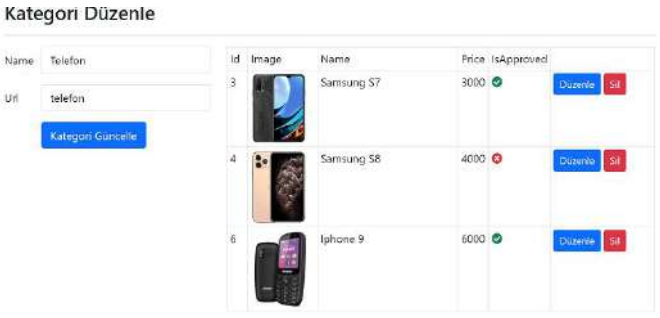
Bu sayfa içinde genişliği 4 ve 8 sütünlük iki bölüme ayıralım. 4 lük bölümde kategori bilgisini, 8 lik kısımda ise ona bağlı ürünleri gösterelim. Zaten düzenleme sayfasında kategori ile ilgili Name ve Url alanlarımız var idi. Bu kısımlar 4 lük bölüm içinde kaldı. 8 lik bölüme ise ProductList.cshtml sayfasındaki ürünleri gösterdiğimiz tabloyu alıp kopyalayalım. Bu durumda sayfamızın model yapısı içinde hem kategori bilgisi hemde ürünler bilgisi olduğundan kolaylıkla bu bilgiler form ve tablo içinde gözükecektir.

CategoryEdit.cshtml

```
@model CategoryModel
<h1 class="h3">Kategori Düzenle</h1>
<hr>
<div class="row">
    <div class="col-md-4">
        <form asp-controller="Admin" asp-action="CategoriEdit" method="POST">
            <input type="hidden" name="CategoryId" value="@Model.CategoryId" />
            <div class="form-group row mb-3">
                <label asp-for="Name" class="col-sm-2 col-form-label"></label>
                <div class="col-sm-10">
                    <input class="form-control" asp-for="Name">
                </div>
            </div>
            <div class="form-group row mb-3">
                <label asp-for="Url" class="col-sm-2 col-form-label"></label>
                <div class="col-sm-10">
                    <input class="form-control" asp-for="Url">
                </div>
            </div>
            <div class="form-group row mb-3">
                <div class="col-sm-10 offset-sm-2">
                    <button type="submit" class="btn btn-primary">Kategori Güncelle</button>
                </div>
            </div>
        </form>
    </div>

    <div class="col-md-8">
        <table class="table table-bordered table-sm">
            <thead>
                <tr>
                    <td style="width:30px">Id</td>
                    <td style="width:100px">Image</td>
                    <td>Name</td>
                    <td style="width:20px">Price</td>
                    <td style="width:20px">IsApproved</td>
```

```
                    <td style="width:150px"></td>
                </tr>
            </thead>
            <tbody>
                @if (Model.Products.Count > 0)
                {
                    @foreach (var item in Model.Products)
                    {
                        <tr>
                            <td>@item.ProductId</td>
                            <td><img src="~/img/@item.ImageUrl" width="80" /> </td>
                            <td>@item.Name</td>
                            <td>@item.Price</td>
                            <td>
                                @if (item.IsApproved)
                                {
                                    <i class="fas fa-check-circle text-success"></i>
                                }
                                else
                                {
                                    <i class="fas fa-times-circle text-danger"></i>
                                }
                            </td>

                            <td>
                                <a href="/admin/products/@item.ProductId" class="btn btn-primary btn-sm mr-
2">Düzenle</a>

                                <form action="/admin/deleteproduct" method="post" style="display:inline;">
                                    <input type="hidden" name="productId" value="@item.ProductId" />
                                    <button type="submit" class="btn btn-danger btn-sm">Sil</button>
                                </form>
                            </td>
                        </tr>
                    }
                }
                else
                {
                    <div class="alert alert-varning">
                        <h3>No Products</h3>
                    </div>
                }
            </tbody>
        </table>
    </div>
```

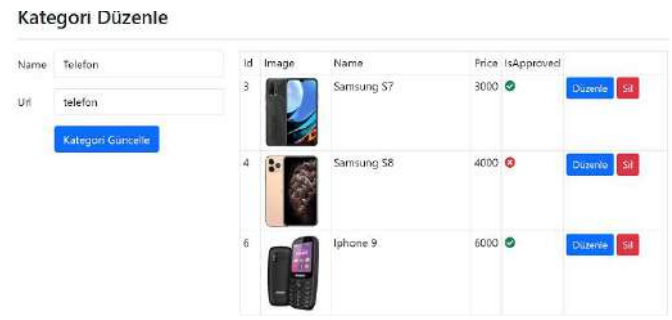
Çalışan sayfalarımızın görünümü şu şekilde olacaktır.



Bu sayfada kategoriyi düzenlerken orada bulunan ürünleri silme işlemi yaptığımızda veritabanından değil, ilgili kategoriden siliyor olmamız gerekir. Yani ürün ve kategori arasındaki ilişkiyi tutan çoka-çok tablodan silmek istiyoruz. Bir sonraki derste bu işlemi yapalım.

Kategoriden Ürün Silme

CategoryEdit.cshtml sayfamızdki Sil (delete) butonuna tıkladığımızda ürünle kategori arasında bağlantımızı sağlayan ara tablodan (ProductCategory) o ürün ile kategorinin olduğu kaydı sileceğiz. Bir ürün birden fazla kategoriye bağlanabildiği için böyle bir ara tabloya ihtiyaç duymuştuk.



Tablo: ProductCategory

	CategoryId	ProductId
	Filtre	Filtre
1	2	1
2	3	1
3	1	3
4	3	3
5	1	4
6	3	4
7	1	6

Sil butonuna tıkladığımızda admin controller içinde deletefromcategory action metoduna gitsin. Burada formdan bilginin gideceği action tipi Post olarak işaretlenmeli. Buradaki metod hem productId, hemde categoryId parametrelerini almalı. Çünkü ilişki tablosunda her iki bilginin eşleştiği satırı sileceğiz. Biz zaten formdan productId gönderiyoruz. Benzer şekilde aynı satırdan bir tane daha ekleyip categoryId gönderelim. CategoryId bilgisi ise sayfamızda kullandığımız model içinde alabiliriz.

```
<form action="/admin/deletefromcategory" method="post" style="display:inline;">
    <input type="hidden" name="productId" value="@item.ProductId" />
    <input type="hidden" name="categoryId" value="@Model.CategoryId" />
    <button type="submit" class="btn btn-danger btn-sm">Sil</button>
</form>
```

Admin controller içindeki ilgili action metodu her iki Id bilgisini alıp silme işlemini gerçekleştirecek fakat bu işi yapacak sınıf ve metod alt yapımız olmalı.

```
public IActionResult DeleteFromCategory(int productId, int categoryId)
{
    _categoryService.DeleteFromCategory(productId, categoryId);

    return Redirect ("/admin/categories/" + categoryId);
}
```

Sınıf ve metod alt yapımızı oluşturalım. Bunun için Data ve Business katmanlarındaki alt yapıyı oluşturalım. Silme işleminde geri dönen değer olmadığı için fonksiyonların başına void ekleyeceğiz. Data>Concrete içindeki kodlarımız aşağıdaki şekilde oluşturalım. Burada Tscontext alt yapısı kullanılarak bir Sql sorgusu çalıştıracağız. Bu sorgudan geri dönüş değeri olmayacak.

Data>Abstract içeriği

```
public interface ICategoryRepository : IRepository<Category>
{
    void DeleteFromCategory(int productId, int categoryId);
}
```

Data>Concrete>EfCore içeriği

```
public class EfCoreCategoryRepository : EfCoreGenericRepository<Category, TSContext>, ICategoryRepository
{
    public void DeleteFromCategory(int productId, int categoryId)
    {
        using (var context = new TSContext())
        {
            string cmd = "DELETE FROM ProductCategory WHERE ProductId=@p0 AND CategoryId=@p1";
            context.Database.ExecuteSqlRaw(cmd, productId, categoryId);
        }
    }
}
```

Business katmanındaki metodlar.

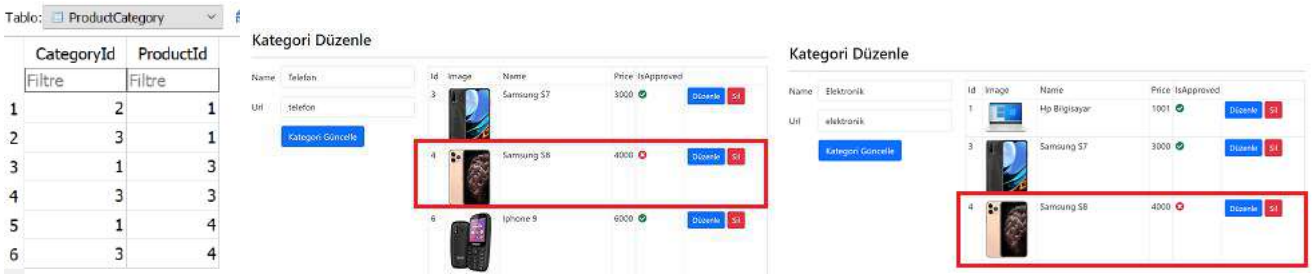
Abstract içeriği

```
public interface ICategoryService
{
    void DeleteFromCategory(int productId, int categoryId);
```

Concrete içeriği

```
public class CategoryManager : ICategoryService
{
    public void DeleteFromCategory(int productId, int categoryId)
    {
        _categoryRepository.DeleteFromCategory(productId, categoryId);
    }
```

Kodları deneyelim. VT de 4 numaralı ürün hem 1 numaralı Telefon kategorisinde hemde 3 numaralı elektronik kategorisinde gözüküyor.



Bu ürünü elektronik kategorisinden silersek bir daha bu kategori altında karşımıza gelmez ama diğer kategoride gözükür. Bundan sonrası için ürünü düzenlerken ilgili kategorilere bağlama işlemi yapalım.

Ürüne Kategori Ekleme

Ürün detay sayfasından ürünü bağlayacağımız kategorileri seçebilelim. Yapılacak olan bir önceki uygulamamızın benzeri. Ürünü Id bilgisi ile getirmek yerine yani GetById() metodu yerine GetByWithCategories() metoduna dönüştürelim.

```
var entity = _productService.GetByWithCategories((int)id);
```

Bu metodun alt yapısını Data ve Business katmanlarında oluşturalım.

Data Katmanında Abstract içeriği

```
public interface IProductRepository: IRepository<Product>
{
    Product GetByIdWithCategories(int id);
```

Data katmanında Concrete içeriği

```
public class EfCoreProductRepository :
    EfCoreGenericRepository<Product, TSContext>, IProductRepository
{

    public Product GetByIdWithCategories(int id)
    {
        using (var context= new TSContext())
        {
            return context.Products
                .Where(i => i.ProductId == id)
                .Include(i => i.ProductCategories)
                .ThenInclude(i => i.Category)
                .FirstOrDefault();
        }
    }
```

Business katmanında Abstract içeriği

```
public interface IProductService
{

   Product GetByIdWithCategories(int id);
```

Business katmanında Concrete içeriği

```
public class ProductManager : IProductService
{

    public Product GetByIdWithCategories(int id)
    {
        return _productRepository.GetByIdWithCategories(id);
    }
```

Sınıf alt yapımız hazır. Artık AdminController içindeki metodlarımızı oluşturalım. Öncelikle ürünle birlikte onun bağlı olduğu Categoriler geleceğinden form sayfalarında kullandığımız ProductModel içinde ilgili tanımlamayı ekleyelim.

```
public class ProductModel
{

    public List<Category> SelectedCategories { get; set; }
```

AdminController içindeki metod içine yazılan şu ifade ile entity içinde zaten ürünün bağlı olduğu kategoriler geliyordu. Bunlar seçilip category sınıfı tipinde listeye dönüştürülüp ProductModel sınıfının SelectedCategories listesinin içine atılıyor.

```
SelectedCategories = entity.ProductCategories.Select(i => i.Category).ToList()
```

Sayfada seçilen kategorilerin yanında VT deki tüm kategorilerinde getiliyor olması gerekir. Seçili olanlar işaretli gösterilir ama diğerleride olmalı ki istenildiğinde ürün başka kategorilerede aktarılabilmeli.

Bütün kagetori bilgilerini ProductModel içinde yine bir liste oluşturarak taşıyabiliriz yada ViewBag içinde sayfaya götürebiliriz. Biz ViewBag tercih edelim. Aşağıdaki komutla tüm kategori bilgilerini ViewBag içine yükleyebiliriz.

```
ViewBag.Categories = _categoryService.GetAll();
```

Views>Admin içindeki ProductEdit.cshtml sayfasını düzenleyelim. Burada sol tarafta 8 sütünluk ürün bilgilerini gösteren form bilgileri vardı 4 sütünlu sağ taraftada kategori bilgilerini gösterelim.

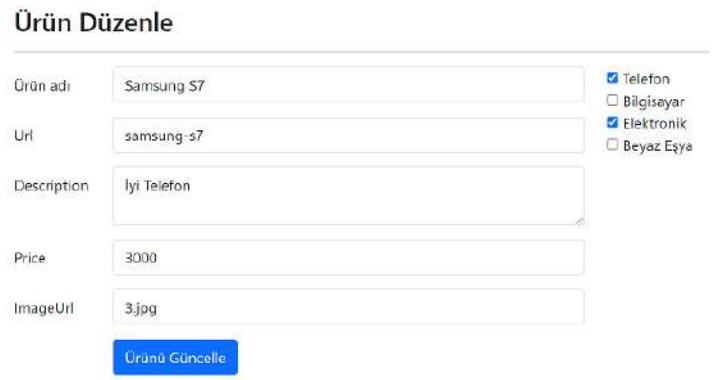
Kategori bilgilerini ViewBag içinden önce listeye çevirerek foreach döngüsü ile listeleyelim.

`@(Model.SelectedCategories.Any(i=>i.CategoryId==item.CategoryId)?"checked":"")` şeklinde verilen satır ile model içerisinde gelen ürüne ait kategorilerden birinin Id si Veritabanında kayıtlı categorilerden birinin Id sine eşitse bu durumda checked olarak gösterilsin. Any metodu burada şart sağlanırsa bir bool değeri gönderir. Ona göre checked yada boş olarak gösterilir. ProductEdit.cshtml sayfamızın içindeki kategorilerin gösteildiği bölümün son hali aşağıdaki şekilde olur.

```
<div class="col-md-4">
    @foreach(var item in (List<Category>)ViewBag.Categories)
    {
        <div class="custom-control costum-checkbox">
            <input class="custom-control-input" type="checkbox" value=""
                   id="category_@item.CategoryId"

@(Model.SelectedCategories.Any(i=>i.CategoryId==item.CategoryId)?"checked":"")   />
            <label class="custom-control-label" for="category_@item.CategoryId">
                @item.Name
            </label>
        </div>
    }
```

```
    </div>
```

Kodları denersek aşağıdaki şekilde bir görüntü alırız.



İlgili ürüne atanacak kategoriler listeden seçildikten sonra Güncelle dediğimizde bilgileri kaydetmemiz lazım. Şimdi bu işlemi yapalım.

Burada sayfada olan bütün bilgileri toptan action metoduna gönderiyor olmamız gerekiyor. Sayfada bir tane Submit butonu var. Dolayısı ile safyamızda bir tane form yapısı olması gerekir. İçerisinde hem ürün hemde kategori bilgileri olmalıdır. Form etiketelerimizi sayfada en dış kısma yerleştirelim. Form içindeki checkbox ait bilgiler Name alanı içinde gidecek fakat bu bilginin değeride Value alanı ile gidecek. Fakat kategorilerin tek bir tane olmadığını görüyoruz. Döngü içinde çok sayıda kategori olacağından bu bilgilerin metod içine alınırken dizi şeklinde alınması gerekir. Metod girişinde dizi adı yine categoriIds şeklinde olabilir ama int [] categoriIds şeklinde integer bir dizi olmalıdır.

ProductEdit.cshtml sayfamızın kategori kısmı aşağıdaki şekilde olur.

```
<div class="col-md-4">
    @foreach (var item in (List<Category>)ViewBag.Categories)
    {
        <div class="custom-control costum-checkbox">
            <input class="custom-control-input" type="checkbox"
                   name="categoriIds"
                   value="@item.CategoryId"
                   id="category_@item.CategoryId"
                   @(Model.SelectedCategories.Any(i => i.CategoryId == item.CategoryId) ?
"checked" : "") />
            <label class="custom-control-label" for="category_@item.CategoryId">
                @item.Name
            </label>
        </div>
    }
</div>
```

Formdan action metoduna gelecek olan bilgiler entity bilgileri ile dizi şeklinde categoryIds ler olacaktır. Bu durumda her iki bilgiyi bir tane Update() metodunda çalıştırabilmek için aşırı yüklenmiş başka bir Update metodu oluşturmamız gerekir.

```
_productService.Update(entity, categoryIds);
```

Buradaki Update metodu aşırı yüklenmiş farklı bir versiyon olmuş olacak. Onunda alt yapısını Data ve Business katmanında oluşturalım.

Business>Abstract içerisinde ve Data>Abstract içerisinde ilgili dosyaların içinde aşağıdaki tanımlamaları yapalım.

```
        void Update(Product entity, int[] categoryIds);
```

Concrete versiyonu ProductManger içerisi;

```
public void Update(Product entity, int[] categoryIds)
{
    _productRepository.Update(entity, categoryIds);
}
```

Olur.

EfCoreProductRepository içinde aşağıdaki mantıkla kodlarımızı oluşturalım.

İlk olarak dışarıdan gönderilen entitinin productId si ile eşleşen ürünün bilgilerini VT den context aracılığı ile çekelim.

```
var product = context.Products.FirstOrDefault(i => i.ProductId == entity.ProductId);
```

Bu bilgileri gönderirken yanında da ProductCategories leride göndersin. Bunu sorgu içine include edelim. Yeni hali şu şekilde olur.

```
var product = context.Products
                    .Include(i=>i.ProductCategories)
                    .FirstOrDefault(i => i.ProductId == entity.ProductId);
```

Böylece bize bir product bilgileri ve içerisinde kategori bilgileri de gelmiş olacak. Böylece VT den okuduğumuz ürün bilgileri product nesnesi içinde olacaktır. Dışarıdan formdan gelen entity içindeki ürün bilgilerini VT den okuduğumuz nesnenin alanları içine dolduralım. Aynı zamanda VT den yeni haliyle kategori bilgileride gelecektir. Onlarıda atayacağız. Bütün bu işlemleri eğer product içerisi dolu ise yapalım. `if(product!=null)`

`product.ProductCategories =` için bir liste hazırlayıp onu eşitleyeceğiz. Bu listeyide formdan gelen categorIds dizisine göre oluşturacağız. Zaten ürünün Id si belli ve o ürünün bağlanacağı kategori Id değerleride formdan gelen dizinin içinde var. Bu yüzden burda yapmamız gereken bu dizi üzerinden bir Select işlemi oluşturacağız.

Burada productId ve categoriId den oluşan çiftler oluşturacağız. Yani kaç tane kategoriye bağlanmışsa o kadar çift oluşturmamız gerekiyor. Oluşturulan çiftlerde bir liste şekline dönüştürülüp ProductCategories nesnesi içine atılacak. Bu arada `categoryIds.Select(catId => new ProductCategory()` ifadesi ile dizi (categoryIds) içindeki herbir sayı select ifadesi ile alınıp catId değişkeni içine atılmaktadır. Her bir catId içindeki bilgilerde her seferinde `new ProductCategory()` ifadesi ile oluşturulan nesnenin içine atılmaktadır. Bu kısmın son hali şu şekilde olur.

```
product.ProductCategories = categoryIds.Select(catId => new ProductCategory()
{
    ProductId = entity.ProductId,
    CategoryId =catId
}
).ToList();
```

Bunların hepsi bittikten sonra `context.SaveChanges();` diyere değişiklikleri kaydedelim.

Kodlarımızın son hali EfCoreProductRepository içinde aşağıdaki şekilde olur.

```
public void Update(Product entity, int[] categoryIds)
{
    using(var context = new TSContext())
    {
        var product = context.Products
                             .Include(i=>i.ProductCategories)
                             .FirstOrDefault(i => i.ProductId == entity.ProductId);
        if(product!=null)
        {
            product.Name = entity.Name;
            product.Price = entity.Price;
            product.Description = entity.Description;
            product.Url = entity.Url;
            product.ImageUrl = entity.ImageUrl;

            product.ProductCategories = categoryIds.Select(catId => new ProductCategory()
            {
                ProductId = entity.ProductId,
                CategoryId =catId
            }
            ).ToList();
```

```
                context.SaveChanges();
            }

        }
    }
```

Kodları denediğimizde çalıştığını görürüz.

Sayfa üzerinde sağtuşa tıklayıp kaynağı görüntülersek sadece checked özelliği olanlar dizi içinde metoda gönderildiğini görürüz.

```
<div class="custom-control custom-checkbox">
        <input type="checkbox"
                class="custom-control-input"
                name="categoryIds"
                value="3"
                id="category_3"
                checked>
        <label class="custom-control-label" for="category_3">Elektronik</label>
</div>
<div class="custom-control custom-checkbox">
        <input type="checkbox"
                class="custom-control-input"
                name="categoryIds"
                value="4"
                id="category_4"
                checked>
        <label class="custom-control-label" for="category_4">Beyaz E&#x15F;ya</label>
</div>
```

Burada foeeach döngüsü dönerken bulduğu checkboxların sadece checked özelliği işeretlenmiş olanların value değerlerini Name bilgisi içinde (categoryIds) dizi şeklinde action metoduna taşımaktadır. Checked özelliği false olanlar name bilgisi içinde taşınmıyor. Bu işlem varsayılan otomatik olarak yapılmaktadır.

FORM BİLGİLERİNİN DOĞRULANMA İŞLEMİ (VALIDATION)

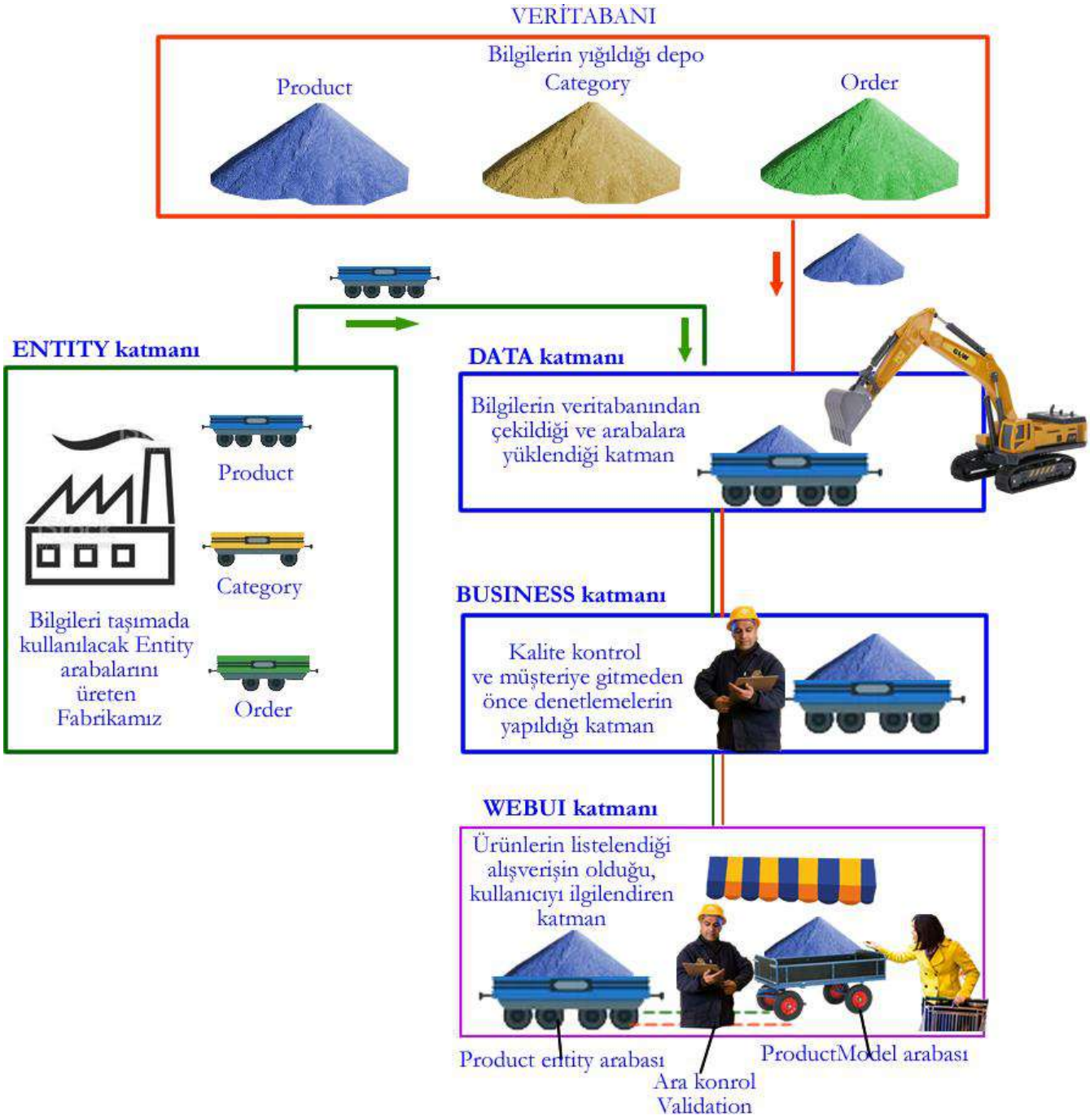
Server Tarafında Product İçin Validation İşlemi (Product Validation)

Web sitemiz üç katmandan oluşuyordu. En altta bilgilerin kullanıcıya gösterildiği WebUI, onun bilgileri çektiği İş Katmanı, Daha sonra sonra sorgulama ve VT işlemlerinin yapıldığı Data katmanıdır. Bütün bu yapılarla çalışan sınıf yapılarımızı gösteren Entity katmanıdır. Bilgilerin birbirine aktarıldığı bu üç katmanın herhangi birinde kontrolü gerçekleştirilebilir. Bunların her birinin avantaj, dezevantajı vardır. Bunları daha sonra göreceğiz. Şu aşamada kontrol işlemini WebUI katmanı içinde yapalım.

Veritabanından bilgiler taşınırken kullanılan sınıf yapısı (entity diyeceğiz) ile formlarda bilgilerin gösterildiği sınıf yapısını ayrı kullanacağız. Böylece formlarda bizim için gerekli olacak bir validation (doğrulama) işlemleri için yazacağımız kod bloklarını entity içine yazmak yerine form için oluşturduğumuz sınıf yapısı içine yazmış olacak. Çünkü bizim burada yapacağımız doğrulama işlemleri sadece WebUI katmanını ilgilendiriyor. Form giriş çıkışları burada olmaktadır. Oysa kullandığımız Entity yapıları başka projelerde kullanılabilir. Dolayısı ile o sınıf yapılarının içine yazmış olacağımız kodlar o projeleri ilgilendirmeye bilir.

Validation için oluşturacağımız sınıf yapılarının sonuna Model ifadesini ekleyelim. Örneğin Product dediğimizde entity sınıf yapısı olsun. ProductModel dediğimizde ise validation için kullanacağımız sınıf yapısı olacak. Validation için bu sınıf yapısı içine **DataAnnotation** (aşağıda örneklerini göreceğimiz) ifadeleri ekleyeceğiz.

Aşağıda işleyişin mantığı katmanlar halinde gösterilmiştir. Esas bilgilerin kontrollerinin yapıldığı işler için Business katmanı oluşturulmuştu fakat şu ana kadar bu katmanı hiç kullanmadık. Direk sadece aynı tip vagondan diğer vagona bilgileri aktardık. Oysa kontrolü (Validation) şu anda WebUI içirisinde sadece formyapıları için (müşteriye çıkılan tezgahı temsil eder) kullandığımız ProductModel sınıfları içinde yapacağız (Product entitysindeki alanları içeren ama ondan daha küçük arabayı temsil eder. Formlarda kullanılmayan gereksiz alanlar çıkarılmıştır).



Product entity şema yapımız veritabanı şema yapısını gösterir. ProductModel şema yapısı ise formdan gelen bilgilerin şema yapısını gösterir. Veritabanında kullanılan bazı alanların formda kullanılması gerekmez. Bu nedenle ProductModel şema yapımız Product entity şema yapımızın daha kısmi bir versiyonu şeklindedir.

Yeni bir ürün kayıtı için ProductCreate.cshtml sayfamızdan bilgileri aldığımızda bu bilgiler ProductModel nesnesi içerisinde taşırız. Taşınan bilgiler AdminController içindeki [httpPost] tipindeki ProductCreate metodu olacaktır. Formdan buraya gelen bilgiler Veritabanına kaydedilirken yine Product entity arabasına bindirilmesi gerekir. Çünkü veritabanından bilgileri alırken ve götürürken Product entity arabasını kullanıyoruz. Dolayısı ile formdan gelen bilgiler bu arabaya burada tekrar bindirilmesi gerekir.

Buna göre form üzerindeki bilgilerin doğruluğunu kontrol için ProductModel sınıf yapımızı kullanalım. DataAnnotation (veri yazım şekli) ismi verilen Validation (doğrulama) kodlarını ProductModel sınıf yapısı içine ekleyelim.

Model yapısı içinde form için örnek bir alan doldurma aşağıdaki gibi olabilir. Burada Display kısmı form üzerindeki labellerda kullanıcıya gösterilecek olan yazıları ifade eder. Required komutu alanı doldurmayı zorunlu kılar. StringLenght ile de girilecek karekter sayısı belirlenebilir.

```
 [Display(Name="Ürün adı", Prompt ="Ürün adı giriniz..")]
 [Required(ErrorMessage ="Ad alanını doldurmak zorunludur")]
 [StringLength(60, MinimumLength =5, ErrorMessage = "Ürün adı 5-60 karakter aralığında
olmalıdır")]
public string Name { get; set; }
```

Fiyat alanı nullable (boş bırakabilir) olmalıdır. Bunun için soru işareti şu şekilde eklenir (`public double? Price { get; set; }`) Eğer bu tip alanları nullable yapmazsak hiçbir şey yazılmasa sayısal ifadeye 0 değerini atar oda bir fiyat olarak anlam ifade eder. Oysa ? konursa boş bırakıldığı zaman null değeri almış olur ve null olan değerlerde kontrol edilebilir ve boş bırakıldığı anlaşılmış olur. Bu şekilde yapılırsa Required komutu boş bırakılmayı alagılayabilir. Range ifadesiyle de girilecek fiyatın aralığı gösterilebilir. Böylece negatif sayıda girilememiş olur.

```
[Required(ErrorMessage = "Fiyat alanını doldurmak zorunludur")]
[Range(1,100000, ErrorMessage ="Fiyat için 1-100000 TL arasında bir değer girmelisiniz")]
public double? Price { get; set; }
```

ProductModel.cs sınıf yapımızın son hali aşağıdaki şekilde oldu.

TS.WebUI>Models> **ProductModel.cs**

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using TS.Entity;

namespace TS.WebUI.Models
{
    public class ProductModel
    {
        public int ProductId { get; set; }

        [Display(Name="Ürün adı", Prompt ="Ürün adı giriniz..")]
        [Required(ErrorMessage ="Ad alanını doldurmak zorunludur")]
        [StringLength(60, MinimumLength =5, ErrorMessage = "Ürün adı 5-60 karakter aralığında olmalıdır")]
        public string Name { get; set; }

        [Required(ErrorMessage = "Url alanını doldurmak zorunludur")]
        public string Url { get; set; }

        [Required(ErrorMessage = "Fiyat alanını doldurmak zorunludur")]
        [Range(1,100000, ErrorMessage ="Fiyat için 1-100000 TL arasında bir değer girmelisiniz")]
        public double? Price { get; set; }

        [Required(ErrorMessage = "Açıklama alanını doldurmak zorunludur")]
        [StringLength(250, MinimumLength = 10, ErrorMessage = "Ürün açıklaması 10-250 karakter aralığında
olmalıdır")]
        public string Description { get; set; }

        [Required(ErrorMessage = "Resim bilgisini doldurmak zorunludur")]
        public string ImageUrl { get; set; }

        public List<Category> SelectedCategories { get; set; }
    }
}
```

Şimdi bu sınıf yapısında gösterildiği şekilde formdan gelen bilgilerin doğru olup olmadığını ModelState.IsValid komutu ile Controller içinde kontrolünü yapalım. Aşağıdaki if bloğu içinde normalde yapmamız gereken bütün ifadeleri koyabiliriz. Eğer IsValid false geldiyse yani formda giriş bilgilerinde hata varsa o zaman burada yapılacak işler yapılmaz bir sonraki satır ile (`return View(model);`) tekrar form sayfasına göndeririz. Tabii tekrar oraya gittiğinde yine ordan gelen bilgileri model nesnemiz içinde götürmeliyiz. Model yapımızdan bilgiler form alanlarına (name ve value alanlarına) otomatik dolduğu için eksra bir işlem yapmamız gerekmiyor.

```
if(ModelState.IsValid)
{//***
```

```
        return RedirectToAction("ProductList");
}
return View(model);
```

Tekrar form sayfasına gittiğimizde bilgiler form içine otomatik dolacak fakat hatalı olan alanlarla ilgili açıklamaları kullanıcıya göstermek gerekir. Bu tür işlemler sayfanın en üstünde özet şeklinde hatalı alanları gösterdikten sonra her alanın yanında da hata mesajlarını göstermek şeklinde uygulanır. Sayfanın en üstünde özet şeklinde bilgileri göstermek aşağıdaki kodla yapılır.

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

Her alanın altında bilgileri aşağıdaki etiketi ile yapabiliriz. Hata mesajlarını taşıyan ModelState komutu burada verilen **asp-validation-for** özelliğini kullanarak mesajı otomatik olarak bu kısımda gösterir.

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

Buraya kadar olan kodlarımızı denersek eksik olan form alanlarının çalıştığını görürüz.

Burada Validation ları sadece Create sayfasına ekledik. Edit sayfasına da eklememiz gerekir. Aynı işlemleri orası içinde yapalım. Fakat şu ana kadar Create (ürün ekleme) sayfamızda Kategorileri kaydetme eklemedik. Ama Edit sayfasında Kategorilerde vardır. O yüzden burada işler biraz daha karışacaktır.

Eğer Edit sayfasında herhangi bir değişiklik yapmadan Güncelle butonuna tıklarsak form IsValid ifadesi true geleceğinden normal kaydı yapar. Ama formdan bazı alanları eksik bırakırsak IsValid ifadesi false geleceğinden parantez dışında ise sayfaya tekrar gönderilirken model içinde Kategoriler bulunmayacağından hata oluşacaktır. Bu nedenle tekrar View sayfasına giderken yanımızda tüm Kategori bilgilerini de götürmeliyiz. ViewBag içerisine alınan bilgiler bir metoddan bir View sayfasına gidene kadar aktif olur. Yani içerisinde sürekli olarak kalıcı değildir. O yüzen action metodundan tekrar View sayfasına giderken içerisini yeniden doldurmalıyız. Aşağıdaki kodları post metodunun sonuna yazdığımızda View sayfasına giderken hem model bilgilerini hemde tüm kategori bilgilerini götürür (tabi kategori bilgilerini ViewBag içinde taşıyacaktır). Fakat View sayfasına giderken orada hangi kategorilerin seçili olduğu bilgisine de ihtiyaç olacak. O nedenle bu bilgilerinde gitmesi gerekirdi. Bu tür bir zorluğu aşmak için daha baştan View sayfasında seçili olan kategorileri gizli (hiden alanında) for döngüsü ile tutarsak yani formun içinde gizli bir nesne üzerine yazarsak bu sayfa tekrar post actiona geldiğinde yine o bilgilerde form içinde model bilgisine aktarılarak gelmiş olur. Tekrar geri gönderdiğimizde yine aynı bilgiler gidecektir. Kısaca form üzerinde seçili kategori bilgilerini gizli alanlarda tutarak kullanacağız.

```
        ViewBag.Categories = _categoryService.GetAll();
        return View(model);
```

Şimdi View sayfasında hidden alanlarda seçili kategori bilgilerini tutmak için döngü ile hidden alanı oluşturalım. Hidden alanlara verilen Name isimlerini ise model içinde kullandığımız SelectedCategories listesindeki isimlere uygun olsun (`List<Category> SelectedCategories`). Yani bu liste bir dizi dir. Herbir Name in sonunda indis numaraları da olsun. Aşağıdaki kodlar ile form üzerinde oluşturulan gizli alanlar üzerindeki bilgiler model içindeki bilgilerden çekilerek oluşturulmaktadır. Form içinde hidden şeklinde bu alanlarımız oluşunca sayfayı post ettiğimizde bu bilgiler model içine tıpkı onun sınıf yapısına uygun şekilde konulmuş olacaktır. Burada sınıf yapısı içindeki alan ve alt bilgileri ile hidden nesnelerinin name ve value değerleri aynı isimde ve tipte olmalıdır. CategoryId ve Name ismindeki alt alanlar Category entitisinde geçen alt alanlardır. `name="SelectedCategories[@i].CategoryId"` cümlesi seçilen kategorilerden 0 ıncı alandaki kategorinin CategoriId si şudur gibi bir anlam ifade eder.

```
@for (int i = 0; i < Model.SelectedCategories.Count; i++)
{
    <input type="hidden" name="SelectedCategories[@i].CategoryId"
           value="@Model.SelectedCategories[@i].CategoryId" />

    <input type="hidden" name="SelectedCategories[@i].Name"
           value="@Model.SelectedCategories[@i].Name" />
}
```

İlgili Sayfalarımız şu şekilde olacaktır.

ProductCreate.cshtml

```
@model ProductModel

<h1 class="h3">Ürün Ekle</h1>
<hr>
<div class="row">
    <div class="col-md-12">
        <div asp-validation-summary="All"
class="row text-danger"></div>
    </div>
</div>

<div class="row">
    <div class="col-md-8">
        <form asp-controller="Admin" asp-
action="ProductCreate" method="POST">
            <div class="form-group row mb-
3">
                <label asp-for="Name"
class="col-sm-2 col-form-label"></label>
                <div class="col-sm-10">
                    <input class="form-
control" asp-for="Name">
                    <span asp-validation-
for="Name" class="text-danger"></span>
                </div>
            </div>
            <div class="form-group row mb-
3">
                <label asp-for="Url"
class="col-sm-2 col-form-label"></label>
                <div class="col-sm-10">
                    <input class="form-
control" asp-for="Url">
                    <span asp-validation-
for="Url" class="text-danger"></span>
                </div>
            </div>
            <div class="form-group row mb-
3">
                <label asp-
for="Description" class="col-sm-2 col-
form-label"></label>
                <div class="col-sm-10">
                    <textarea class="form-
control" asp-for="Description"></textarea>
```

ProductEdit.cshmtl

```
@model ProductModel

<h1 class="h3">Ürün Güncelleme</h1>
<hr>
<div class="row">
    <div class="col-md-12">
        <div asp-validation-summary="All" class="row text-
danger"></div>
    </div>
</div>

<form asp-controller="Admin" asp-action="ProductEdit"
method="POST">

    <div class="row">
        <div class="col-md-8">

            <input type="hidden" name="ProductId"
value="@Model.ProductId">
            <div class="form-group row mb-3">
                <label asp-for="Name" class="col-sm-2 col-form-
label"></label>
                <div class="col-sm-10">
                    <input class="form-control" asp-for="Name">
                    <span asp-validation-for="Name" class="text-
danger"></span>
                </div>
            </div>
            <div class="form-group row mb-3">
                <label asp-for="Url" class="col-sm-2 col-form-
label"></label>
                <div class="col-sm-10">
                    <input class="form-control" asp-for="Url">
                    <span asp-validation-for="Url" class="text-
danger"></span>
                </div>
            </div>
            <div class="form-group row mb-3">
                <label asp-for="Description" class="col-sm-2 col-
form-label"></label>
                <div class="col-sm-10">
                    <textarea class="form-control" asp-
for="Description"></textarea>
                    <span asp-validation-for="Description"
class="text-danger"></span>
                </div>
            </div>
```

```
                    <span asp-validation-
for="Description" class="text-
danger"></span>
                </div>
            </div>
            <div class="form-group row mb-
3">
                <label asp-for="Price"
class="col-sm-2 col-form-label"></label>
                <div class="col-sm-10">
                    <input class="form-
control" asp-for="Price">
                    <span asp-validation-
for="Price" class="text-danger"></span>
                </div>
            </div>
            <div class="form-group row mb-
3">
                <label asp-for="ImageUrl"
class="col-sm-2 col-form-label"></label>
                <div class="col-sm-10">
                    <input class="form-
control" asp-for="ImageUrl">
                    <span asp-validation-
for="ImageUrl" class="text-danger"></span>
                </div>
            </div>
            <div class="form-group row mb-
3">
                <div class="col-sm-10
offset-sm-2">
                    <button type="submit"
class="btn btn-primary">Ürünü
Kaydet</button>
                </div>
            </div>
        </form>
    </div>
</div>
```

```
            <div class="form-group row mb-3">
                <label asp-for="Price" class="col-sm-2 col-form-
label"></label>
                <div class="col-sm-10">
                    <input class="form-control" asp-for="Price">
                    <span asp-validation-for="Price" class="text-
danger"></span>
                </div>
            </div>
            <div class="form-group row mb-3">
                <label asp-for="ImageUrl" class="col-sm-2 col-
form-label"></label>
                <div class="col-sm-10">
                    <input class="form-control" asp-
for="ImageUrl">
                    <span asp-validation-for="ImageUrl"
class="text-danger"></span>
                </div>
            </div>
            <div class="form-group row mb-3">
                <div class="col-sm-10 offset-sm-2 mt-3">
                    <button type="submit" class="btn btn-
primary">Ürünü Güncelle</button>
                </div>
            </div>

        </div>
        <div class="col-md-4">
            @for (int i = 0; i < Model.SelectedCategories.Count;
i++)
            {
                <input type="hidden"
name="SelectedCategories[@i].CategoryId"

value="@Model.SelectedCategories[@i].CategoryId" />

                <input type="hidden"
name="SelectedCategories[@i].Name"
                       value="@Model.SelectedCategories[@i].Name"
/>
            }

            @foreach (var item in
(List<Category>)ViewBag.Categories)
            {
                <div class="custom-control custom-checkbox">
                    <input type="checkbox"
                           class="custom-control-input"
                           name="categoryIds"
                           value="@item.CategoryId"
                           id="category_@(item.CategoryId)"
                           @(Model.SelectedCategories.Any(i =>
i.CategoryId == item.CategoryId) ? "checked" : "")>
                    <label class="custom-control-label"
for="category_@(item.CategoryId)">@item.Name</label>
                </div>
            }
        </div>
    </div>

</form>
```

AdminController.cs>**ProductCreate()**

```
[HttpPost]
public IActionResult ProductCreate(ProductModel model)
{
    if(ModelState.IsValid)
    {
        var entity = new Product()
        {
            Name = model.Name,
            Url = model.Url,
            Price = model.Price,
            Description = model.Description,
            ImageUrl = model.ImageUrl
        };
```

AdminController.cs>**ProductEdit()**

```
[HttpPost]
public IActionResult ProductEdit(ProductModel model,
int[] categoryIds)
{
    if (ModelState.IsValid)
    {
        var entity =
_productService.GetById(model.ProductId);
        if (entity == null)
        {
            return NotFound();
        }
```

```
        _productService.Create(entity);

        var msg = new AlertMessage()
        {
            Message = $"{entity.Name} isimli ürün
eklendi",
            AlertType = "success"
        };
        TempData["Message"] =
JsonConvert.SerializeObject(msg);

        return RedirectToAction("ProductList");
    }
    return View(model);

}
```

```
        entity.Name = model.Name;
        entity.Price = model.Price;
        entity.Url = model.Url;
        entity.ImageUrl = model.ImageUrl;
        entity.Description = model.Description;

        _productService.Update(entity, categoryIds);

        var msg = new AlertMessage()
        {
            Message = $"{entity.Name} isimli ürün
güncellendi",
            AlertType = "success"
        };
        TempData["Message"] =
JsonConvert.SerializeObject(msg);

        return RedirectToAction("ProductList");
    }
    ViewBag.Categories = _categoryService.GetAll();
    return View(model);

}
```

Kodları çalıştırdığımızda hatalı alanları engellediğini görürüz ancak esas burada öğrenmemiz gereken konu şudur. Get() tipindeki bir action metoddan bilgiler View sayfasına giderken model içerisinde gerekli bilgiler gidiyordu. Yani ürünle birlikte, hangi kategorilerin seçili olduğu bilgileri de gidiyordu. Bu bilgiler form üzerindeki alanlarda görüntüleniyordu. Fakat butona tıklanıldığında Post() metoduna giderken bu sefer model içinde form üzerinde işaretlenmiş alanlar gönderiliyor. Bu durumda form üzerindeki ürüne ait bilgiler gidiyor fakat Category bilgileri gitmiyordu. Bizde geri giderken kategori bilgilerini de götürmek için atanmış categori bilgilerini gizli alanlar içinde tutuyoruz. Böylece bu alanlarda model içinde Post() metoduna geri gitmiş oluyordu. Bu gizli alanları görmek için sayfa üzerinde sağ tuşa tıklarsak görebiliriz.


```
<div class="col-md-4">
    <input type="hidden" name="SelectedCategories[0].CategoryId"
           value="1" />
    <input type="hidden" name="SelectedCategories[0].Name"
           value="Telefon" />
    <input type="hidden" name="SelectedCategories[1].CategoryId"
           value="3" />
    <input type="hidden" name="SelectedCategories[1].Name"
           value="Elektronik" />
    <input type="hidden" name="SelectedCategories[2].CategoryId"
           value="4" />
    <input type="hidden" name="SelectedCategories[2].Name"
           value="Beyaz E&#x15F;ya" />

    <div class="custom-control custom-checkbox">
        <input type="checkbox"
               class="custom-control-input"
               name="categoryIds"
               value="1"
               id="category_1"
               checked>
```

Server Tarafında Kategori İçin Validation İşlemi (Category Validation)

Product validation lar için yaptığımız işlemlerin aynısını Kategoriler içinde yapalım. Tüm adımları tekrar anlatmak yerine özet olarak adımları yazalım.

a) Öncelikle Models>ProductModel.cs nin bir benzeri olarak formlarda kullanacağımız CategoryModel.cs sınıfımızı oluşturalım.

Models> **CategoryModel.cs**

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using TS.Entity;
```

```
namespace TS.WebUI.Models
{
    public class CategoryModel
    {
        public int CategoryId { get; set; }

        [Required(ErrorMessage = "Kategori ismi girilmelidir")]
        [StringLength(100,MinimumLength =5,ErrorMessage ="Kategori ismi 5-100 karakter arası olmalıdır")]
        public string Name { get; set; }

        [Required(ErrorMessage = "Url adresi girilmelidir")]
        public string Url { get; set; }

        public List<Product> Products { get; set; }

    }
}
```

b) AdminController.cs içindeki [HttpPost] tipindeki hem CategoryCreate() hemde CategoryEdit() içine aşağıdaki if bloklarımızı ekleyelim.

```
if (ModelState.IsValid)
{

}

return View(model);
```

c) Views içindeki CategoryCreate.cshtml ve CategoryEdit.cshtml sayfaları içindeki form elemanlarının olduğu yerlere şu komutları ekleyelim.

```
<div class="row">
    <div class="col-md-12">
        <div asp-validation-summary="All" class="row text-danger"></div>
    </div>
</div>

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

d) Burada Create sayfasında fazla bir sorun olmayacaktır. Hem Get() metodundan View sayfasına giderken hemde Sayfadan Post() metoduna gelirken model içinde bilgiler taşınabiliyor.

Fakat Edit sayfaları için aynı durum söz konusu değildir. Get() metodundan View giderken model içinde hem Kategori bilgileri hemde bu kategoriye ait ürün bilgilerini taşıyabiliyorduk. Burda sorun yoktu. Fakat tekrar Vew sayfasından Post() metoduna geri giderken ürün bilgileri model içinde taşınamadığı için bir Validation hatası olduğunda tekrar view sayfasına yönlenmek istediğimiz bu bilgileri kaybediyorduk ve bir hata alıyorduk. Bu sorunu yukarıda Product için çözmüştük. Tıpkı ordaki gibi View sayfası içinde Kategorinin yanında ona ait ürünleri gizli (hidden) form elemanları atayarak bilgileri onlar üzerinde saklayacağız. Böylece geri Post() metoduna giderken model içinde gizli olarak tuttuğumuz bu bilgileri yanımızda götüreceğiz. Gizli alanların içinde sadece sayfa içinde kullandığımız alt alanlar olsa yeterlidir. Entity nin tüm alanlarını tutmamız gerekmez. Model içinde götürülecek alanların <form> etiketleri arasında olması gerektiğini unutmayalım.

Bu gizli alanlard IsAproved alanı bool tipinde olduğundan varsayılan değeri true dur. Sayfadaki hidden alanların prametrelerinde Value=”value” olarak gözükecektir. False olduğunda ise value=”false” olarak

gözükecektir. Bu durumu sayfa üzerinde sağ tuşa tıkladığımızda görebiliriz. Eğer biz bu tip bool bilgileri "true" yada "false" şeklinde görmek istersek son kısmınad .toString() ifadesini ekleyebiliriz. Bu komut aşağıda boyanarak gösterilmiştir. Bu şekilde bool türlerini String olarak algılamasını sağladık.

CategoryEdit.cshtml

```
@model CategoryModel

<h1 class="h3">Kategori Düzenleme</h1>
<hr>

<div class="row">
    <div class="col-md-4">
        <form asp-controller="Admin" asp-action="CategoryEdit" method="POST">

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

            <input type="hidden" name="CategoryId" value="@Model.CategoryId">
            <div class="form-group row mb-3">
                <label asp-for="Name" class="col-sm-2 col-form-label"></label>
                <div class="col-sm-10">
                    <input class="form-control" asp-for="Name">
                    <span asp-validation-for="Name" class="text-danger"></span>
                </div>
            </div>
            <div class="form-group row mb-3">
                <label asp-for="Url" class="col-sm-2 col-form-label"></label>
                <div class="col-sm-10">
                    <input class="form-control" asp-for="Url">
                    <span asp-validation-for="Url" class="text-danger"></span>
                </div>
            </div>
            <div class="form-group row">
                <div class="col-sm-10 offset-sm-2">
                    <button type="submit" class="btn btn-primary">Kategoriyi Güncelle</button>
                </div>
            </div>

            <div id="products">
                @for (int i = 0; i < Model.Products.Count; i++)
                {
                    <input type="hidden" name="Products[@i].ProductId" value="@Model.Products[@i].ProductId">
                    <input type="hidden" name="Products[@i].ImageUrl" value="@Model.Products[@i].ImageUrl">
                    <input type="hidden" name="Products[@i].Name" value="@Model.Products[@i].Name">
                    <input type="hidden" name="Products[@i].Price" value="@Model.Products[@i].Price">
                    <input type="hidden" name="Products[@i].IsApproved"
value="@Model.Products[@i].IsApproved.ToString()">
                }
            </div>

        </form>
    </div>
    <div class="col-md-8">
        <div class="row">
            <div class="col-md-12">

                <table class="table table-bordered table-sm">
                    <thead>
                        <tr>
                            <td style="width: 30px;">Id</td>
                            <td style="width: 100px;">Resim</td>
                            <td>Name</td>
                            <td style="width: 20px;">Fiyat</td>
                            <td style="width: 20px;">Onaylı</td>
                            <td style="width: 150px;"></td>
                        </tr>
                    </thead>
                    <tbody>
                        @if (Model.Products.Count > 0)
                        {
                            @foreach (var item in Model.Products)
                            {
                                <tr>
                                    <td>@item.ProductId</td>
                                    <td><img src="~/img/@item.ImageUrl" width="80"></td>
                                    <td>@item.Name</td>
                                    <td>@item.Price</td>
```

```
                                    <td>
                                        @if (item.IsApproved)
                                        {
                                            <i class="fas fa-check-circle text-success"></i>
                                        }
                                        else
                                        {
                                            <i class="fas fa-times-circle text-danger"></i>
                                        }
                                    </td>
                                    <td>
                                        <a href="/admin/products/@item.ProductId" class="btn btn-primary btn-sm
mr-2">Düzenle</a>

                                        <form action="/admin/deletefromcategory" method="POST" style="display:
inline;">
                                            <input type="hidden" name="productId" value="@item.ProductId">
                                            <input type="hidden" name="categoryId" value="@Model.CategoryId">
                                            <button type="submit" class="btn btn-danger btn-sm">K.Sil</button>
                                        </form>
                                    </td>
                                </tr>
                            }
                        }
                        else
                        {
                            <div class="alert alert-warning">
                                <h3>No Products</h3>
                            </div>
                        }
                    </tbody>
                </table>
            </div>
        </div>
    </div>
</div>

@section Scripts
{
    <script src="/modules/jquery-validation/dist/jquery.validate.min.js"></script>
    <script src="/modules/jquery-validation-unobtrusive/dist/jquery.validate.unobtrusive.min.js"></script>
}
```

CategoryEdit (POST)

```
        [HttpPost]
        public IActionResult CategoryEdit(CategoryModel model)
        {
            if(ModelState.IsValid)
            {
                var entity = _categoryService.GetById(model.CategoryId);
                if (entity == null)
                {
                    return NotFound();
                }

                entity.Name = model.Name;
                entity.Url = model.Url;

                _categoryService.Update(entity);

                var msg = new AlertMessage()
                {
                    Message = $"{entity.Name} isimli kategori güncellendi",
                    AlertType = "success"
                };
                TempData["Message"] = JsonConvert.SerializeObject(msg);
                return RedirectToAction("CategoryList");
            }
            return View(model);
        }
```

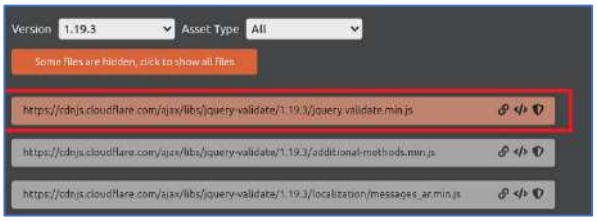
Kullanıcı Tarayıcısında Validation İşlemleri (Client Validation)

Şu ana kadar form üzerindeki bilgilerin doğrulama işlemlerini server taraflı kodlar ile yaptık. Bu doğrulama işlemlerini kullanıcının tarayıcısı üzerinde de yaptırabiliriz.

Herhangi bir form saymızın üzerinde sağ tuşa tıkladığımızda asp.net core tarafından data-annotation işlemine ait form alanları ile bir çok bilginin kullanıcının tarayıcısına geldiğini görüyoruz. Bunlar aslında kullanıcının tarayıcısında kullanılmayı bekleyen hazır parametrelerdir. Kullanıcının tarayıcısında çalışacak bir script ise JavaScript dir. Sayfaya ekleyeceğimiz JavaScript kütüphanesi buradaki paramatre yada özellikleri alabilecek özelliğe sahiptir. Bu şekilde yaptığımız bir işlemde boşuna server tarafını yormamış oluruz. Daha kişi butona tıklamadan (sayfa servera gitmeden) JavaScript kullanıcıyı uyarıp engelleyebilir.


```
<div class="form-group row mb-3">
<label class="col-sm-2 col-form-label" for="Name">&#xDC;r&#xFC;n ad&#x131;</label>
<div class="col-sm-10">
        <input class="form-control" type="text" data-val="true" data-val-length="&#xDC;r&#xFC;n ad&#x131; 5-60 karakter aral&#x131;&#x11F;&#x131;nda olmal&#x131;d&#x131;r" data-val-length-max="60" data-val-length-min="5" data-val-required="Ad alan&#x131;n&#x131; doldurmak zorunludur" id="Name" maxlength="60" name="Name" placeholder="&#xDC;r&#xFC;n ad&#x131; giriniz.." value="">
        <span class="text-danger field-validation-valid" data-valmsg-for="Name" data-valmsg-replace="true"></span>
</div>
</div>
```

Bu işlem için bizim yapmamız gereken data-attribution parametrelerini okuyabilecek JavaScript kütüphanesini sayfaya dahil etmektir. Bunun için npm üzerinden kurma işlemide yapabiliriz ama biz direk kütühanenin yolunu sayfamıza dahil edelim. Google “jquery validate cdn link” ifadesini yazıp karşımıza çıkan linkden devam edelim ve kütüphanenin yolu alıp sayfamızda kullanalım.



https://cdnjs.cloudflare.com/ajax/libs/jquery-validate/1.19.3/jquery.validate.min.js

https://cdnjs.cloudflare.com/ajax/libs/jquery-validation-unobtrusive/3.2.12/jquery.validate.unobtrusive.min.js

Bu scriptlerin tüm sayfalarda çalışması için _layout.cshtml sonundaki scriplerin olduğu yere ekleyebiliriz yada her form sayfasının sonuna ekleyebiliriz. Biz kullanıldığı sayfalara eklemeyi tercih edelim. _layout.cshtml sayfasının en sonunda aşağıdaki kodlar bulunsun. Validation scriplerinin çalışması için en üstte Jquery kütüphanesinin bulunmasını ister.

```
    <script src="https://code.jquery.com/jquery-3.4.1.min.js" integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n"
crossorigin="anonymous"></script>
```

```
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.8.1/css/all.css">

    @*scriplerin nerede olacağının yerini aşağıdaki ifadeyle işaretledik*@
    @RenderSection("Scripts", false)
```

Form sayfalarının sonuna ise aşağıdaki satırları ekleyelim.

```
@section Scripts
{
    @*Bu kütüphaneler client taraflı validation lar için eklendi.*@
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-
validate/1.19.3/jquery.validate.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-validation-
unobtrusive/3.2.12/jquery.validate.unobtrusive.min.js"></script>
}
```

Sayfalar çalıştğında tüm kütüphaneleri aşağıdaki şekilde görebiliriz. Tıkladığımızda kodlar açılıyorsa sayfaya kütüphane yüklenmiş demektir.

```
    <script src="https://code.jquery.com/jquery-3.4.1.min.js" integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n"
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.8.1/css/all.css">


    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-validate/1.19.3/jquery.validate.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-validation-unobtrusive/3.2.12/jquery.validate.unobtrusive.min.js"></script>
</body>
</html>
```

Form üzerindeki butona tıkladığımızda sayfa servera gitmeden direk olduğu yerde uyarı mesajlarını verecektir. Hatta yazı yazarken eksik karakter tamamlanmadan uyarı yine gözükecektir. Bu işlemler server taraflı yapıldığında butona tıklayıp eksikler giderilmeye çalışılır. Bu daha az yorucu bir işlem olmaktadır.

Benze şekilde bu script linkleri diğer formlara da eklenmelidir. Kullanıcı açısından bu uygulamaları tercih etmemiz önemlidir.

Not: Bu video daki Jquery validation çalışmadı. Sonra tekrar bak.

https://www.udemy.com/course/komple-web-developer-kursu/learn/lecture/19346398#overview

Link vererek çalışmayınca paketi kurmayı deneyelim. Google “jquery.validate download” yazıp çıkan sayfadan (https://www.nuget.org/packages/jquery.validation) alınan aşağıdaki dotnet linki aracılığ ile kuralım.

dotnet add package jQuery.Validation --version 1.19.3

```
C:\Users\pc1\Desktop\TS.com\TS\TS.WebUI>dotnet add package jQuery.Validation --version 1.19.3
  Geri yüklenecek projeler belirleniyor...
  Writing C:\Users\pc1\AppData\Local\Temp\tmp2C3A.tmp
info : 'C:\Users\pc1\Desktop\TS.com\TS\TS.WebUI\TS.WebUI.csproj' projesine 'jQuery.Validation' paketi için PackageRefere
nce ekleniyor.
info : C:\Users\pc1\Desktop\TS.com\TS\TS.WebUI\TS.WebUI.csproj için paketler geri yükleniyor...
info :   GET https://api.nuget.org/v3-flatcontainer/jquery.validation/index.json
```

Tekrar google a “jquery.validate.unobtrusive download” yazıp çıkan sayfadan alınan aşağıdaki dotnet komutu aracılığı ile kuralım.

dotnet add package jquery.validate.unobtrusive.bootstrap --version 1.2.3

```
C:\Users\pc1\Desktop\TS.com\TS\TS.WebUI>dotnet add package jquery.validate.unobtrusive.bootstrap --version 1.2.3
  Geri yüklenecek projeler belirleniyor...
  Writing C:\Users\pc1\AppData\Local\Temp\tmpF01E.tmp
info : 'C:\Users\pc1\Desktop\TS.com\TS\TS.WebUI\TS.WebUI.csproj' projesine 'jquery.validate.unobtrusive.bootstrap' paket
i için PackageReference ekleniyor.
info : C:\Users\pc1\Desktop\TS.com\TS\TS.WebUI\TS.WebUI.csproj için paketler geri yükleniyor...
info :   GET https://api.nuget.org/v3-flatcontainer/jquery.validate.unobtrusive.bootstrap/index.json
```

Yüklendiyse aşağıdaki ekranlardan referencelarını görebiliriz.

Script yollarını aşağıdaki şekilde _layout.cshmtl içine yazdık fakat yine çalışmadı. Doğru script yolunu öğrenmek gerekiyor

```
<script src="~/jquery.validate.min.js"></script>
<script src="~/jquery.validate.unobtrusive.min.js"></script>
```

İş Katmanında Validation İşlemleri (Business Validation)

Şu ana kadar Validasyonları WebUI projesi içerisinde gerçekleştirdik. İlk uygulamamız server taraflı idi, ikincisi ise istemci (Client) taraflı idi. Fakat verilerimiz Web projemizden başka projelerde de kullanmak isteyebiliriz. Böyle bir durumda verilerin tekrar o proje için yeniden süzülmesi gerekecektir. Oysa biz VT den gelen bu verileri eğer iş katmanında (business) süzersek o yeni kullanılan projede süzülerek gelmiş olacaktır. Bu açıdan bu tür işlemleri Business katmanında yapmak daha mantıklı olacaktır.

İş katmanında Concrete içindeki ProductManager dosyası içinden verileri aktarıyorduk. Buradaki Create metodu içinde bize gelen bir entity bilgisini veritabanına gönderiyorduk. Eğer burada gelen veri istenilen kuralları sağlıyorsa veritabanına gönderme işlemini yapmalıyız. Bu metod içerisinde örneğin gelen entity nin name alanı 5-100 karakter arasında değilse geri gönderilebilir. Onun yerine biz burada daha genel bir yöntem yapalım. Tüm entity ler için çalışacak bir generic bir Interface yapısı oluşturalım.

TS.Business>Abstract>Ivalitador.cs adı ile bir interface tipinde dosya oluşturalım. Bu dosya için aşağıdaki metodu yazalım. Burada geçen `bool Validation(T entity);` ifadeyle gelen etity doğrulama işlemine tabii tutsun sonuç başarılımı yada başarısızmı bool olarak geri göndersin.

TS.Business>Abstract>**Ivalidator.cs**

```
using System;
using System.Collections.Generic;
using System.Text;

namespace TS.Business.Abstract
{
    public interface IValidator<T>
    {
        string ErrorMessage { get; set; }
        bool Validation(T entity);
    }
}
```

Artık aynı Abstract klasörü içindeki diğer Interface ler (ICategoryService ve IProductService) bu interface den türeticek.

```
public interface ICategoryService:IValidator<Category>
public interface IProductService:IValidator<Product>
```

Bu şekilde yazdığımızda herbir servis mutlaka Validator u imlement etmek zorundadır.

Concrete versiyonları olan ProductManager a gelelim. Ara birimi uygula dediğimizde ilgili Validator ve ilgili Error message buraya eklenecektir. Bu işlemleri yaptığımızda ilgili metodlar bu dosyalar içine eklenecektir. Bir tanesi aşağıda verildi.

```
public class ProductManager : IProductService
{

    public string ErrorMessage { get => throw new System.NotImplementedException(); set =>
                                 throw new System.NotImplementedException(); }
    public bool Validation(Product entity)
    {
        throw new System.NotImplementedException();
    }
```

Bu ifadelerin düzenlenmiş hali aşağıdaki şekilde olabilir. Burada ErrorMessage tüm metodlar içinde kullanılmış genel bir değişken gibi düşünebiliriz. İçerisinde her eklenen hata mesajı alt satıra inilerek toplanmış olacaktır. Eğer her alan için ayrı bir mesaj yapısı tutacaksak bir Dictionary tipinde tanımlama yapabiliriz.

```
public class ProductManager : IProductService
{

    public string ErrorMessage { get; set; }

    public bool Validation(Product entity)
    {
        var IsValid = true;

        if(string.IsNullOrEmpty(entity.Name))
        {
            ErrorMessage += "İsim alanını girmelisiniz.\n";
            IsValid = false;
        }
        if (entity.Price<0)
        {
            ErrorMessage += "Ürün fiyatı negatif olamaz. \n";
            IsValid = false;
        }
        return IsValid;
    }
```

Şu anda Validation metodumuz ve onun içinde kullanılan birikimli hata mesajı ifadesini oluşturduk. Artık bu metodu Create vs gibi değişik metodlar içinde kullanabiliriz. Böylece her kullanıldığı yerde ayrı ayrı mesaj yazmak yerine gelen entity yi Validation metoduna göndermektir. Örneğin Create() metodu içinde aşağıdaki şekilde kullanabilir. Entity yi Validation içine gönderdiğimizde gelen sonuç true ise bir sorun yok demektir ve bilgileri Veritabanına gönderebiliriz. Web katmanında bu durumdan haberdar olması için Create metodu da geriye bool değerini göndermesi gerekir.

```
public class ProductManager : IProductService
{

    public bool Create(Product entity)
```

```
        {
            if(Validation(entity))
            {
                _productRepository.Create(entity);
                return true;
            }
            return false;
        }
```

Bool değerini Servideki metotda da yapmalıyız. Başırılı bir durum varsa geriye True değerini göndeririz. Hata varsa da geriye false göndeririz.

```
    public interface IProductService:IValidator<Product>
    {

        bool Create(Product entity);
```

Artık AdminController içinde bunun kullanımına geçebiliriz. Buradaki ProductCrate metodu içinde kayıt işleminde servisten gelen nesne true ise kayıtı yaparız. False bir problem var demektir ve sayfayı tekrar View göndeririz.

```
    public class AdminController : Controller
    {

        [HttpPost]
        public IActionResult ProductCreate(ProductModel model)
        {
            if(ModelState.IsValid)
            {
                var entity = new Product()
                {
                    Name = model.Name,
                    Url = model.Url,
                    Price = model.Price,
                    Description = model.Description,
                    ImageUrl = model.ImageUrl
                };

                if(_productService.Create(entity))
                {
                    var msg = new AlertMessage()
                    {
                        Message = $"{entity.Name} isimli ürün eklendi",
                        AlertType = "success"
                    };
                    TempData["Message"] = JsonConvert.SerializeObject(msg);

                    return RedirectToAction("ProductList");
                }
                return View(model);
            }
            return View(model);

        }
```

Artık bizim Validation kurallarımız İş katmanından gelmektedir. Dolayısı ile ProductModel içine eklediğimiz Validation kurallarını kaldıralım.

Şu ana kadar bu kodlar çalışır fakat hata mesjı View gitmeyecektir. Daha önceden sayfalarda bir hata oluştuğunda sayfanın en üstünde gösterilen bir mesaj kutumuz vardı ve oraya mesajları ViewData içinde götürüyorduk. Bu yapıyı aynen kullanabiliriz. TempData oluşturmak üzere bir metod içinde kodlarımızı yazabiliriz. Bu metodumuz dışarıdan Message ve AlertType bilgilerini alsın. Bu bilgileri TempData içine yerleştirsin. Böylece messajı oluşturan bu metodu AdminController içinde kullanabiliriz.

```
public class AdminController : Controller
{

    public void CreateMessage(string message, string alertType)
    {
        var msg = new AlertMessage()
        {
            Message = message,
            AlertType = alertType
        };
        TempData["Message"] = JsonConvert.SerializeObject(msg);
    }
```

Artık bu mesaj metodunu kullanalım. Sadece hata olduğunda değil başarılı bir kayıt yapıldığında da kullanabiliriz. Kayıt başarılı ise olumlu mesajı direk oluşturabiliriz. Ama bir hata olduğunda bir çok sebepten olabilir bu durumda servis içinden gelen ErrorMessajı kullanıcıya gösterelim.

```
public class AdminController : Controller
{

    public IActionResult ProductCreate(ProductModel model)
    {

            if(_productService.Create(entity)) //dikkat burada hem kayıt yapıyor hemde sonuç
bool olarak geri döndüğünden kontrol ediliyor.
            {
                CreateMessage("Kayıt yapıldı", "success");
                return RedirectToAction("ProductList");
            }
            CreateMessage(_productService.ErrorMessage, "danger");
            return View(model);
```

Kodları çalıştırdığımızda isim alanı girilmemiş ve fiyatı negatif olarak girdiğimizde aşağıdaki iki hatalı mesajı bize kırmızı olarak göstermiş olur.

Burada tabii mesajlar yanyana görüntülendi. Hata mesajlarını bir liste içine ekleyerek yaparsak ve en sonda da mesajları foreach döngüsü ile alt alta gösterebiliriz. Bu şekilde de yapılabilir.

Yada bir Dictionary kullanabiliriz. Burada key ve value ifadelerini kullanarak key alanında hangi alana takıldığını value içinde de mesajın türünü verebiliriz. Bu şekilde yaparsak kullanıcıyı çok daha güzel bilgilendirebiliriz.

İş Kurallarının Uygulanması

Biz şu ana kadar iş katmanını formlardan gelen bilgilerde oluşabilecek hataları uyarmak üzerine kullandık. Ancak projelerimizde bundan farklı iş kurallarımızda olabilir. Bu derste iş katmanında diğer bazı kurallarıda nasıl kullanırız bunu öğrenelim.

Validation dışındaki diğer kurallar ne olabilir. Örneğin bir ürünü silmek isteyelim. Fakat bu ürün bir sipariş sürecinde ise hataya yol açacaktır. Böyle bir durumda silmeden önce bir kontrolün yapılması gerekir. Bu durumda sipariş sürecinde olan bir ürünü Order tablosunda aramamız gerekir. Eğer böyle bir ürün bu tabloda varsa sipariş sürecinde demektir bu durumda bu ürünü silmemiz gerekir.

Yada bunun gibi bir ürünü güncellerken IsApproved (onaylı mı?, satışa çıkarıldı mı?) gibi bir alanı güncellemeden önce ürünün fiyatı doğru girilmiş mi, kategorisi atanmış gibi bir çok kontrolü yapmamız gerekebilir. Böyle bir ihtiyaç bir iş kuralı anlamındadır.

Veritabanı süreçlerinde bilgilerin hiçbir zaman bir tabloda silinmemesi gerekir. Tabloda bool türünde IsDeleted alanı oluşturup bu alanın silinmediğini kontrol etmemiz gerekir. Eğer bu ifade True ise veritabanından ürün silinmiş demektir.

Şimdi biz burada uygulama olarak bir Update metodunu çalıştırmadan önce bazı kontroller yapalım. Bir update işleminde daha önceki yaptığımız bir entity nin bilgilerinin doğru girilmiş olması yine burada da gereklidir. Daha önceki yaptıklarımızı buraya da ekleyelim. Bu işlemde de Update metodları da geri bool değeri göndermelidir.

```
public interface IProductService:IValidator<Product>
 {

    bool Update(Product entity, int[] categoryIds);
```

Bir ürün güncellenmeden önce kategori seçilmiş mi kontrol edelim. Eğer kategorilerin boyutu 0 dan büyük değilse bir kategori seçmemiş demektir. Aşağıdaki kodlarda Validation(entity) ifadesi ile önce entity kontrol ediliyor. Zaten entity bilgilerinde bir hata varsa aşağılardaki o metod içinde ErrorMessage içine bazı uyarılar eklenmiş olur. Eğer bir hata yoksa bu sefer burada categori seçilmiş mi ona bakılmış olacak. Return false ifadesinden sonra gelen satırlar artık çalıştırılmaz.

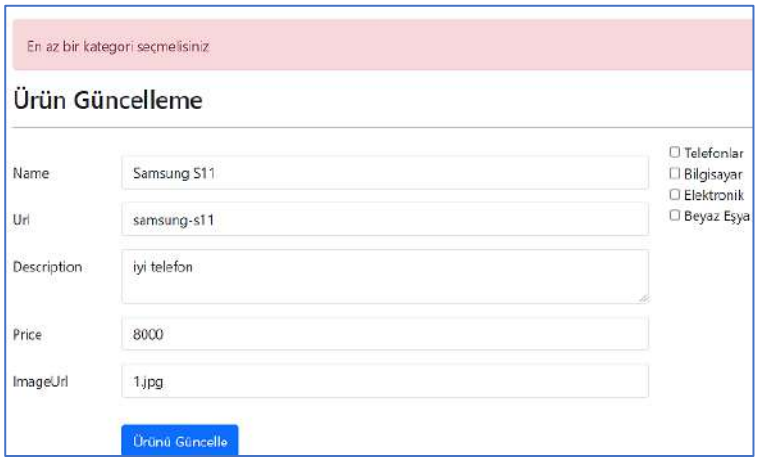
```
public class ProductManager : IProductService
{

    public bool Update(Product entity, int[] categoryIds)
    {
        if (Validation(entity))
        {
            if(categoryIds.Length==0)
            {
                ErrorMessage += "En az bir kategori seçmelisiniz";
                return false;
            }
            _productRepository.Update(entity, categoryIds);
            return true;
        }
        return false;
    }
```

AdminController içindeki düzenlemeleri yapalım.

```
public class AdminController : Controller
{


    [HttpPost]
    public IActionResult ProductEdit(ProductModel model, int[] categoryIds)
    {
        if (ModelState.IsValid)
        {
            var entity = _productService.GetById(model.ProductId);
```

```
                if (entity == null)
                {
                    return NotFound();
                }

                entity.Name = model.Name;
                entity.Price = model.Price;
                entity.Url = model.Url;
                entity.ImageUrl = model.ImageUrl;
                entity.Description = model.Description;

                if (_productService.Update(entity, categoryIds)) //dikkat burada hem kayıt
yapıyor hemde sonuç bool olarak geri döndüğünden kontrol ediliyor.
                {
                    CreateMessage("Kayıt güncellendi", "success");
                    return RedirectToAction("ProductList");
                }
                CreateMessage(_productService.ErrorMessage, "danger");

            }
            ViewBag.Categories = _categoryService.GetAll();
            return View(model);
        }
```

Herhangi bir kategori seçmeden ürünü güncellemeye çalışırsak karşımıza hata mesajı gelecektir.



Ürün Güncellemede Onaylama ve AnaSayfaya alma işlemlerinin eklenmesi

Ürün güncelleme sayfasına IsApproved ve IsHome seçeneklerini ekleyelim. IsApproved seçeneği ürünün onaylı olmasını gösterir. Onaylanan ürünler listeleme sayfalarında gözükmeye başlar. IsHome ise ürünün ana sayfada gözükmesini sağlar.

Öncelikle formlarda kullandığımız yapı olan ProductModel.cs içine bu iki özelliği ekleyelim. TS.Entitiy içinde Product entitisine ait özellikler vardı. Ordan alıp ProductModel içine ekleyebiliriz.

```
    public class ProductModel
    {
        public int ProductId { get; set; }
        public string Name { get; set; }
        public string Url { get; set; }
        public double? Price { get; set; }
        public string Description { get; set; }
        public string ImageUrl { get; set; }
        public bool IsApproved { get; set; }
        public bool IsHome { get; set; }
        public List<Category> SelectedCategories { get; set; }
```

```
    }
```

Bu bilgiler AdminController içinde Get metodunun olduğu yerden sayfaya giderken ve sayfadan gelirken taşnması gerekir.

```
        [HttpGet]
        public IActionResult ProductEdit(int? id)
        {

            var model = new ProductModel()
            {
                ProductId = entity.ProductId,
                Name = entity.Name,
                Url = entity.Url,
                Price = entity.Price,
                ImageUrl = entity.ImageUrl,
                Description = entity.Description,
                IsApproved = entity.IsApproved,
                IsHome = entity.IsHome,
                SelectedCategories = entity.ProductCategories.Select(i => i.Category).ToList()
            };
```

Sayfadan dönüşte Post içine ise

```
        [HttpPost]
        public IActionResult ProductEdit(ProductModel model, int[] categoryIds)
        {
                entity.Name = model.Name;
                entity.Price = model.Price;
                entity.Url = model.Url;
                entity.ImageUrl = model.ImageUrl;
                entity.Description = model.Description;
                entity.IsApproved = model.IsApproved;
                entity.IsHome = model.IsHome;
```

Veritabanına kaydederken kullanılan EfCoreProductRepository içinde aşağıdaki satırlarda değişiklikleri yapmalıyız.

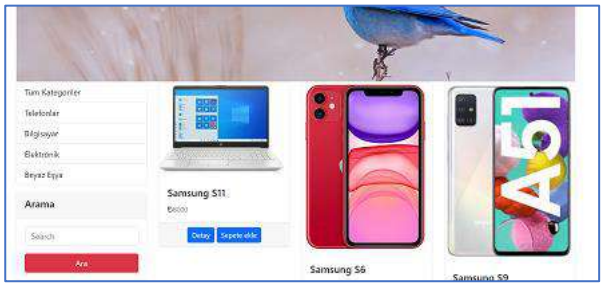
```
       public void Update(Product entity, int[] categoryIds)
       {

              if(product!=null)
              {
                  product.Name = entity.Name;
                  product.Price = entity.Price;
                  product.Description = entity.Description;
                  product.Url = entity.Url;
                  product.ImageUrl = entity.ImageUrl;
                  product.IsApproved = entity.IsApproved;
                  product.IsHome = entity.IsHome;

                  product.ProductCategories = categoryIds.Select(catId => new ProductCategory()
                  {
                      ProductId = entity.ProductId,
                      CategoryId =catId
                  }
                  ).ToList();

                  context.SaveChanges();
              }
```

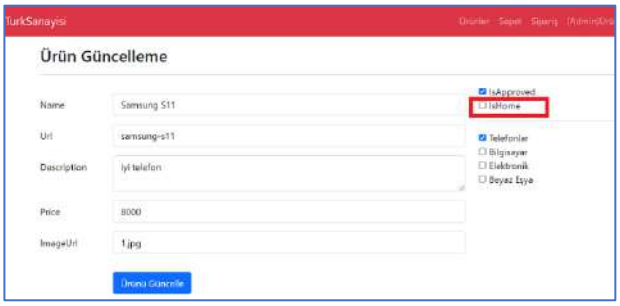
View sayfasında ise Kategorilerin hem üstüne aşağıdaki satırları ekleyebiliriz.

```
       <div class="col-md-4">

            <div class="custom-control custom-checkbox">
                <input asp-for="IsApproved" type="checkbox" class="custom-control-input" />
                <label asp-for="IsApproved" class="custom-control-label"></label>
```

```
        </div>
        <div class="custom-control custom-checkbox">
            <input asp-for="IsHome" type="checkbox" class="custom-control-input" />
            <label asp-for="IsHome" class="custom-control-label"></label>
        </div>

        <hr />
```

Tüm projeler Build işlemleri yapıldıktan sonra çalıştırılırsa ürünlerin ana sayfada gözükmesi yada listelemelerde gözüküp gözükmeyeceği ayarlanmış olur.

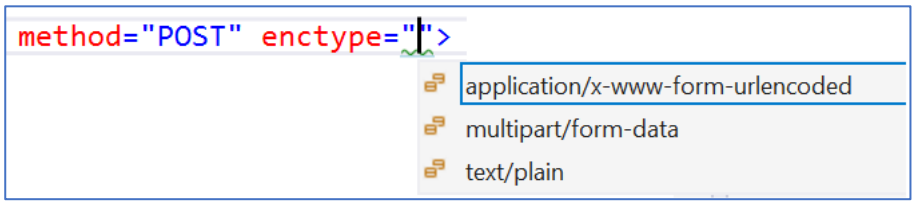




Ürün Resminin Yüklenmesi

Şu ana kadar resim eklerken texbox a elle resim adını yazdık. Oysa ürün resmini bilgisayarımızdan seçip, resim adını otomatik oluşturup aynı anda resmin hem servera yüklenmesini hemde veritabanına adının kaydedilmesini sağlamamız gerekir. Bu işlem için form sayfamızda FileUpload kullanmalıyız. Resimlerimiz wwwroot içinde ki img klasörüne yüklemeli ve adıda veritabanına kaydedilmelidir.

Resim ekleme işlemini Edit sayfasında yapalım. Şu ana kadar edit sayfamızdan form içinde sadece string bilgisini taşıdık. Oysa artık form içinde bir dosya da taşıyacağız. Bu nedenle form etiketi içindeki enctype özelliğini aşağıdaki şekilde değiştirmemiz gerekiyor. Bu parametrenin varsayılanı appliacation diye başlayan en üstteki seçenektir. Biz burada alttaki ikinci seçeneği kullanacağız. Yani “multipar/form-data” seçeneğini kullanacağız.




```
<form asp-controller="Admin" asp-action="ProductEdit" method="POST" enctype="multipart/form-data">
```

Artık formda resmimizin adını göstermek yerine img nesnesi içinde resmin kendisini gösterelim. Resimlerin içinde bulunduğu wwwroot klasör adını yazmamız gerekmiyor. Direk ~/img klasörünü yol olarak verebiliriz. Resmin genişliğini 80 olarak sabit gösterelim. Formuz post olarak giderken bir Validation işlemine takılırsa tekrar döndüğünde resim adının kaybolmaması gerekir. Bu nedenle input nesnesinin içinde hidden olarak resim adını saklayalım. Sayfaya tekrar döndüğümüzde resim adının ne olduğunu biliyor olmamız gerekir. Eleman içinde asp-

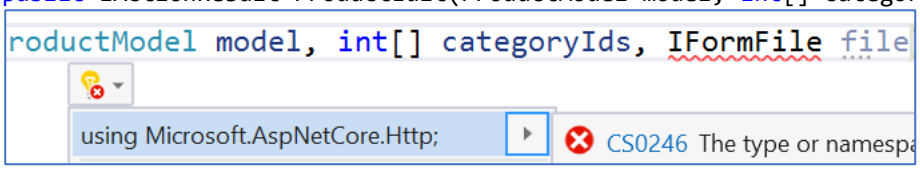
for=”ImgUrl” yazdığımız için Name ve Value bilgileri otomatik olarak zaten gelmiş olacak. Dosyamız <input type=file nesnesi içinde taşınacak. Burada bu nesneye verilen isim name=file şeklindedir. Kontroller içine alınırken bu isimli nesneden çekeceğiz.

ProductEdit.cshtml sayfasının içindeki ilgili alanın son hali aşağıdaki şekilde olacaktır.

```
<div class="form-group row mb-3">
    <label asp-for="ImageUrl" class="col-sm-2 col-form-label"></label>
    <div class="col-sm-10">
        <input type="hidden" asp-for="ImageUrl" />
        <img src="~/img/@Model.ImageUrl" width="80" />
        <input type="file" name="file" />
    </div>
</div>
```

Kontroller içine dosya alınırken aşağıdaki IformFile komutu kullanılır. Bunun …. http kütüphaneside yukarı eklenir. Iformfile ın yanındaki file değişkeni aslında formda kullanılan name=”file” adını verdiğimiz isimdir.

```
public IActionResult ProductEdit(ProductModel model, int[] categoryIds, IFormFile file)
```



Artık form içindeki text bilgilerinin yanında dosya bilgiside gelmektedir. Gelen bu dosya “file” parametresi ile metoda giriş yapmaktadır.

Kontroller içindeki Metod içinden aşağıdaki satırı kaldıralım. Çünkü resim adı hidden olarak metoda gelecektir fakat veritabanına kayıt yaparken bu eski adı kullanmayacağız. Yeni bir dosya yükleyebiliyorsak entity içine onun adını yükleyeceğiz.

```
entity.ImageUrl = model.ImageUrl;
```

Eğer metod içine sayfadan gelen bir resim dosyası var ise bu durumda entity nin resim adını buradan gelen dosyanın adı olarak atanır. `entity.ImageUrl = file.FileName;` satırı ile resmin adını veritabanına kaydettik fakat resmin fiziksel olarak klasör içine de kaydedilmesi gerekir. Resmin fiziksel olarak kaydederken dosya adı ratgele yada ürünün Ip adresi şeklindede tanımlanabilir. Çalışan kodlar aşağıdaki şekildedir. Bu kodlar resim adı rastgele bir fonksiyonla Guid() fonksiyonu ile üretilmektedir (var randomName = string.Format($"{Guid.NewGuid()}{extention}");). Öncesinde metoda gelen resim dosyasının uzantısı alınmaktadır var extention = Path.GetExtension(file.FileName);). Daha sonra resmin kaydedileceği yol oluşturulmaktadır (var path = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot\\img", randomName);) . Sonra ilgili yola dosya kaydedilirken stream akışı oluşturulmakta. Güvenlik içinde using ifadesi kullanılmaktadır. Bu blok içerisinde dosya ilgili klasöre kaydedilirken çalışma durdurulmaktadır. Aktarma işlemi Asenkron yöntemiyle yapılmaktadır. Bu yöntem ileride anlatılacak.

```
public class AdminController : Controller
{

    [HttpPost]
    public async Task<IActionResult> ProductEdit(ProductModel model, int[] categoryIds, IFormFile file)
    {
        if (ModelState.IsValid)
        {
            var entity = _productService.GetById(model.ProductId);
            if (entity == null)
            {
                return NotFound();
            }

            entity.Name = model.Name;
            entity.Price = model.Price;
            entity.Url = model.Url;
            entity.Description = model.Description;
```

```
                entity.IsApproved = model.IsApproved;
                entity.IsHome = model.IsHome;

                if (file != null)
                {
                    var extention = Path.GetExtension(file.FileName);
                    var randomName = string.Format($"{Guid.NewGuid()}{extention}");
                    entity.ImageUrl = randomName;
                    var path = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot\\img", randomName);

                    using (var stream = new FileStream(path, FileMode.Create))
                    {
                        await file.CopyToAsync(stream);
                    }
                }

                if (_productService.Update(entity, categoryIds)) //dikkat burada hem kayıt yapıyor hemde sonuç
bool olarak geri döndüğünden kontrol ediliyor.
                {
                    CreateMessage("Kayıt güncellendi", "success");
                    return RedirectToAction("ProductList");
                }
                CreateMessage(_productService.ErrorMessage, "danger");

            }
            ViewBag.Categories = _categoryService.GetAll();
            return View(model);

        }
```

Kodları deneyelim

Resim adı olarak ürün id sini kullanalım. Bunun için kodları aşağıdaki şekilde düzenleyebiliriz.

```
    if (file != null)
    {
        var extention = Path.GetExtension(file.FileName);
        var imageName = string.Format($"{model.ProductId}{extention}");
        entity.ImageUrl = imageName;
        var path = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot\\img", imageName);

        using (var stream = new FileStream(path, FileMode.Create))
        {
            await file.CopyToAsync(stream);
        }
    }
```

Yada sistem saati aşağıdaki şekilde resim adı olarak kullanılabilir.

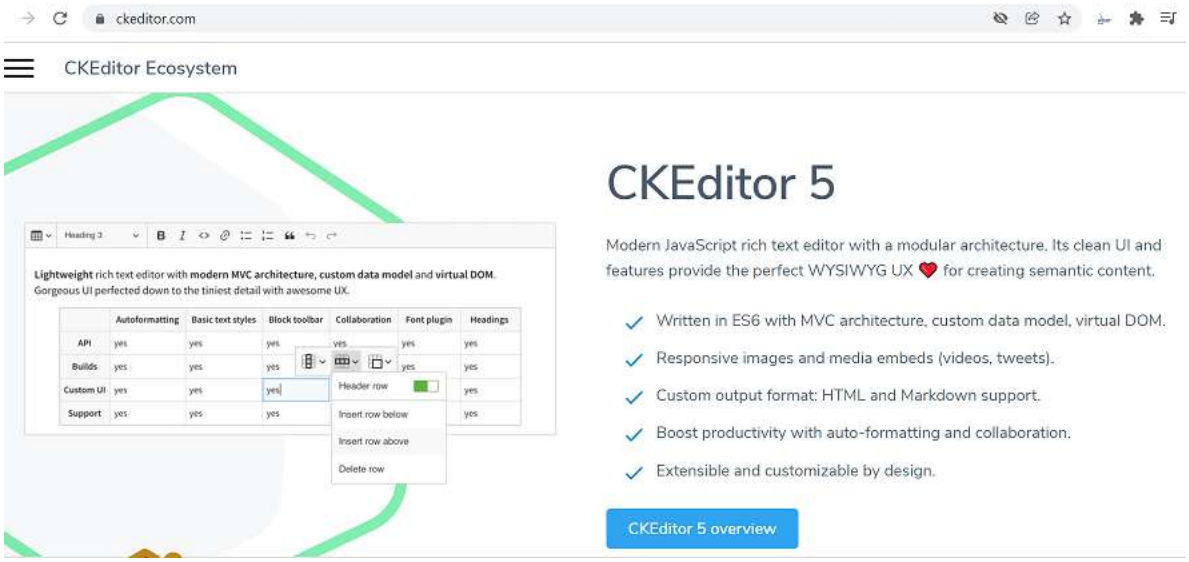
```
    var imageName = string.Format($"{DateTime.Now.Tics}{extention}");
```

SOE için (arama motorlarının bilgileri anlamlandırması için) resim adları için Url bilgisi de kullanılabilir. Bu durumda hem ürünün adı hemde sonunda Id bilgisi olmuş olur. Tanınırlığı daha da artmış olur.

```
var imageName = string.Format($"{model.Url}-{model.ProductId}{extention}");
```

Ürün Açıklamalarını Html Editörü ile daha Estetik Yazdırma

Genellik ürün detay kısımlarında (Description) yazı fontu tipi rengi büyüklüğü hizalaması vs bir çok bilgiyi belirlemek isteyebiliriz. Bu işlem için bir Html editörü kullanmamız gerekiyor. Bu editrölerden en popülerlerden bir tanesi için google **ckeditör** yazıp ulaşabiliriz.



Bunun bir javascript dosyası vardır. Bunu indirip kullanabiliriz ama biz projemiz içinde bunu indirip kuralım.

Bu editörü kurmak için komut satırından WebUI içine konumlanıp (/npm editor ckeditor) şeklinde komutu çalıştırmamız gerekiyor fakat öncelikle npm 'i kurmak gerekiyor.

Bilgi : Npm; Node Package Manager ya da Node Packaged Modules olarak da denmektedir. Isaac Z. Schlueter tarafından tamamen javascript dili kullanılarak geliştirilmiştir. Npm temel olarak 3. parti yazılımları yüklemeyi sağlayan bir araçtır. Aslında npm projemizdeki paketlerin yönetimini otomatikleştiriyor diyebiliriz. Npm ile temel olarak yapabileceğimiz şeyler ise şöyledir :⇨ Otomatik ya da manuel olarak paketleri yükleme ⇨ Sistemdeki paketleri silmek ⇨ Sistemdeki paketleri listeleme ⇨ Sistemdeki paketleri update etmek ❗ Npm komut satırı üzerinden çalışan bir uygulamadır. Npm için nodejs kurmanız gerekiyor ve nodejs kurulumu gerçekleşince npm'de otomatik olarak içinde kurulu gelmiş olacak. Bunun için nodejs sitesinden indirme işlemini yapabilirsiniz. Nodejs yüklendiğini kontrol etmek için :npm -v

https://nodejs.org/en/download/ adresinde nodejs indirip bilgisayarımıza varsayılanlar ile kuralım.

Npm -v

```
C:\Program Files\nodejs\node_modules>npm -v
8.1.2
```

Projemizin içerisinde aşağıdaki komutu çalıştırarak editörü kuralım.

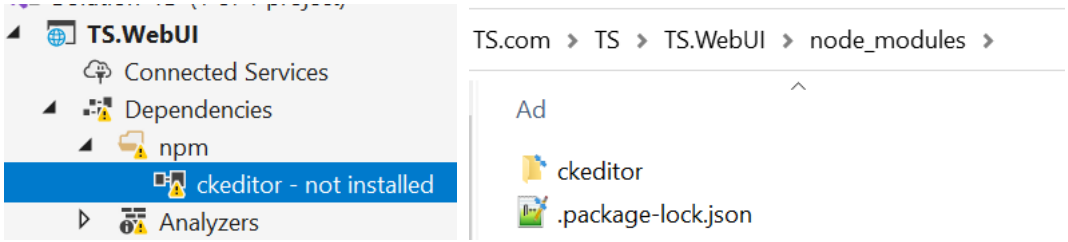
Npm install ckeditor

```
C:\Users\pc1\Desktop\TS.com\TS\TS.WebUI>npm install ckeditor
npm WARN deprecated ckeditor@4.12.1: We have renamed the @ckeditor package. New versions are available under the @ckedit
or4 name.

added 1 package, and audited 2 packages in 14s

found 0 vulnerabilities
```

Yüklenen paketleri görmek için “npm list” komutunu çalıştırabiliriz.

```
C:\Users\pc1\Desktop\TS.com\TS\TS.WebUI>npm list
TS.WebUI@ C:\Users\pc1\Desktop\TS.com\TS\TS.WebUI
`-- ckeditor@4.12.1
```

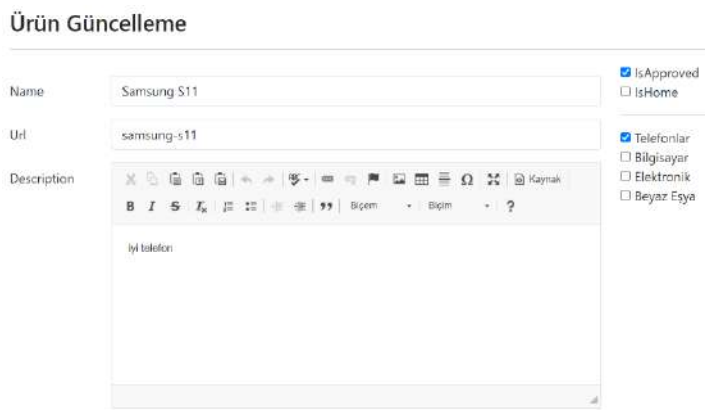
VS içinde yüklenen paket şu şekilde gözükmekte. Ayrıca klasörlerden şu şekilde gözükmektedir. Editör üzerinde ünlem işareti vardır. Üzerinde sağ tuşa tıklayıp “Restore Packages” işaretlersek düzelebilir.



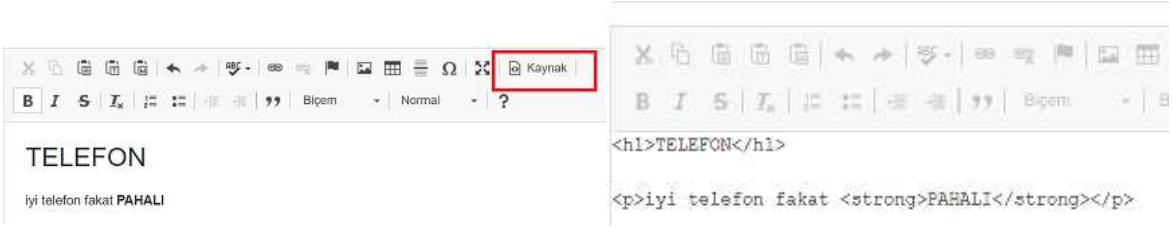
Buradan editörümüzün projeye kurulduğunu görüyoruz. Bu editörü sayfamızda (productedit.cshmlt) kullanabilmek için en altta script içine yolunu ekleyelim. Burada `CKEDITOR.replace('');` İle ilgili Text area nesnesinin adını yani Id bilgisini vereceğiz. vereceğiz. Yukarılardaki Description alanı için kullandığımız Textarenın Id bilgisini Id=”editor” olarak verelim.

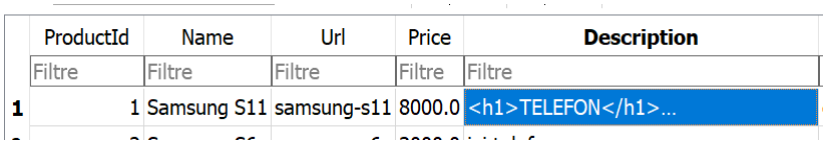
```
</form>

@section Scripts
{
    <script src="//cdn.ckeditor.com/4.17.1/standard/ckeditor.js"></script>

    @*<script src="/modules/ckeditor/ckeditor.js"></script>*@
    @*<script src="./node_modules/ckeditor/ckeditor.js"></script>*@

    <script>
        CKEDITOR.replace('editor');
    </script>
}----------------

<textarea id="editor" class="form-control" asp-for="Description"></textarea>
```

Bunların aynısını buradan alıp Ürün ekleme sayfasına da koyalım. Editörün görünümü aşağıdaki şekilde olacaktır. Yalnız burada dahili adresler çalışmadı. O yüzden internetten yükleyen adres kullanıldı (Bu tür adresler için CDN ismi kullanılır).



Editörün kaynak butonuna tıkladığımızda arka planda html etiketleri oluşturduğunu görürüz. Veritabanına kaydederken bu etiketlerle birlikte kaydeder.





Ürün detay sayfasından baktığımızda bu html etiketleri karşımıza gelmektedir. Bunların yorumlanarak gösterilmesi gerekiyor.

Bunu düzeltmek için detay sayfasına gidelim, description alanın gösterildiği yerdeki ifadeyi aşağıdaki şekilde değiştirelim. Bunu yazdığımızda gelen metin html olarak yorumlanarak gösterilecektir.

```
<div class="row">
    <div class="col-md-12">
        <p class="p-3">@Html.Raw(Model.Product.Description)</p>
    </div>
</div>
```

Kurs Kaynağı: Udemy-Komple Uygulamalı Web Geliştirme Kursu-Sadık Turan