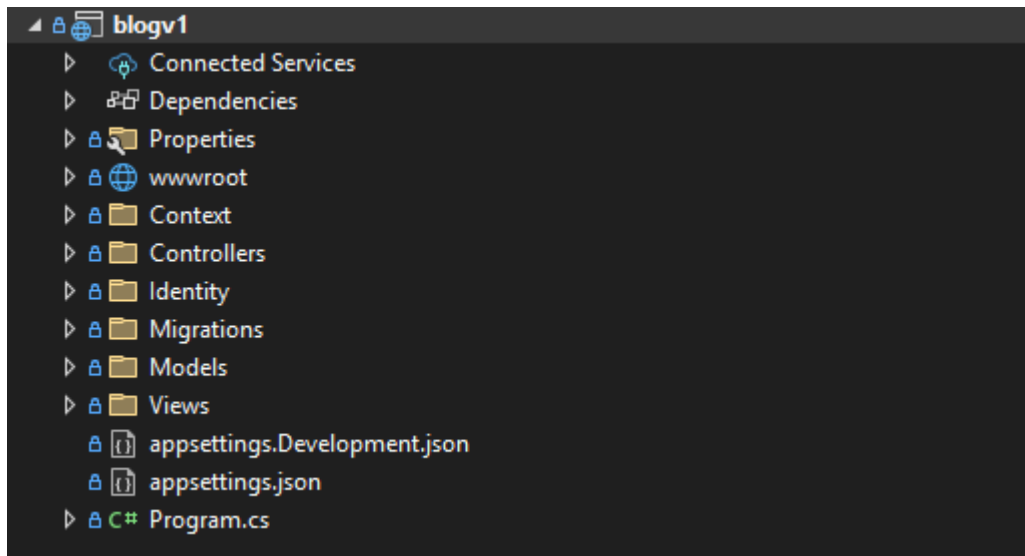


MVC Blog Project

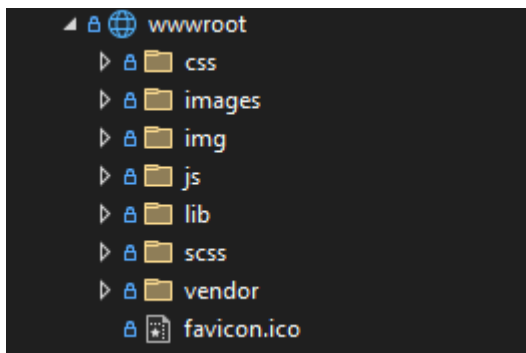
Documentation

Muhammet Emir Yüce

05/09/2025



Wwwroot daha frontend tarafı



Context



- **using blogv1.Models;**

Blogv1 projemizin altındaki listeler model

model = veritabanı tablosunu temsil eden bir sınıf diyebiliriz. Bu sınıfı kullanarak, veritabanındaki her bir blog yazısı için bir **nesne (object)** oluşturup, bu nesneler üzerinden kolayca işlem yaparsın.

- **public class BlogDbContext : DbContext**

`public class BlogDbContext : DbContext`: Bu satır, `BlogDbContext` adında bir sınıf tanımlar ve onu EF Core'un ana sınıfı olan **DbContext**'ten türetir. Bir sınıfın başka bir sınıftan türemesi, `DbContext`'in tüm özelliklerini ve yöntemlerini miras almasını sağlar.

- **protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)**

`protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)`: Bu yöntem, veritabanı bağlantı ayarlarının yapıldığı yerdir.

`override` anahtar kelimesi, bu yöntemin `DbContext` sınıfından miras alınan bir yöntemi geçersiz kıldığını (yeniden tanımladığını) belirtir.

`optionsBuilder.UseSqlServer(...)`: Bu satır, uygulamanızın **SQL Server** veritabanına bağlanacağını belirtir. Parantez içindeki metin, veritabanı bağlantı dizesidir (connection string).

"Data Source =NB3\SQLEXPRESS": Veritabanı sunucusunun adını belirtir.

"database=blogV1": Bağlanılacak veritabanının adını belirtir.

"Integrated Security=True": Uygulamanın Windows kimlik doğrulamasını kullanarak veritabanına bağlanacağını belirtir, yani kullanıcı adı ve şifreye gerek yoktur.

"TrustServerCertificate=True": Sunucu sertifikasına güvenileceğini belirtir, bu genellikle yerel geliştirme ortamlarında kullanılır.

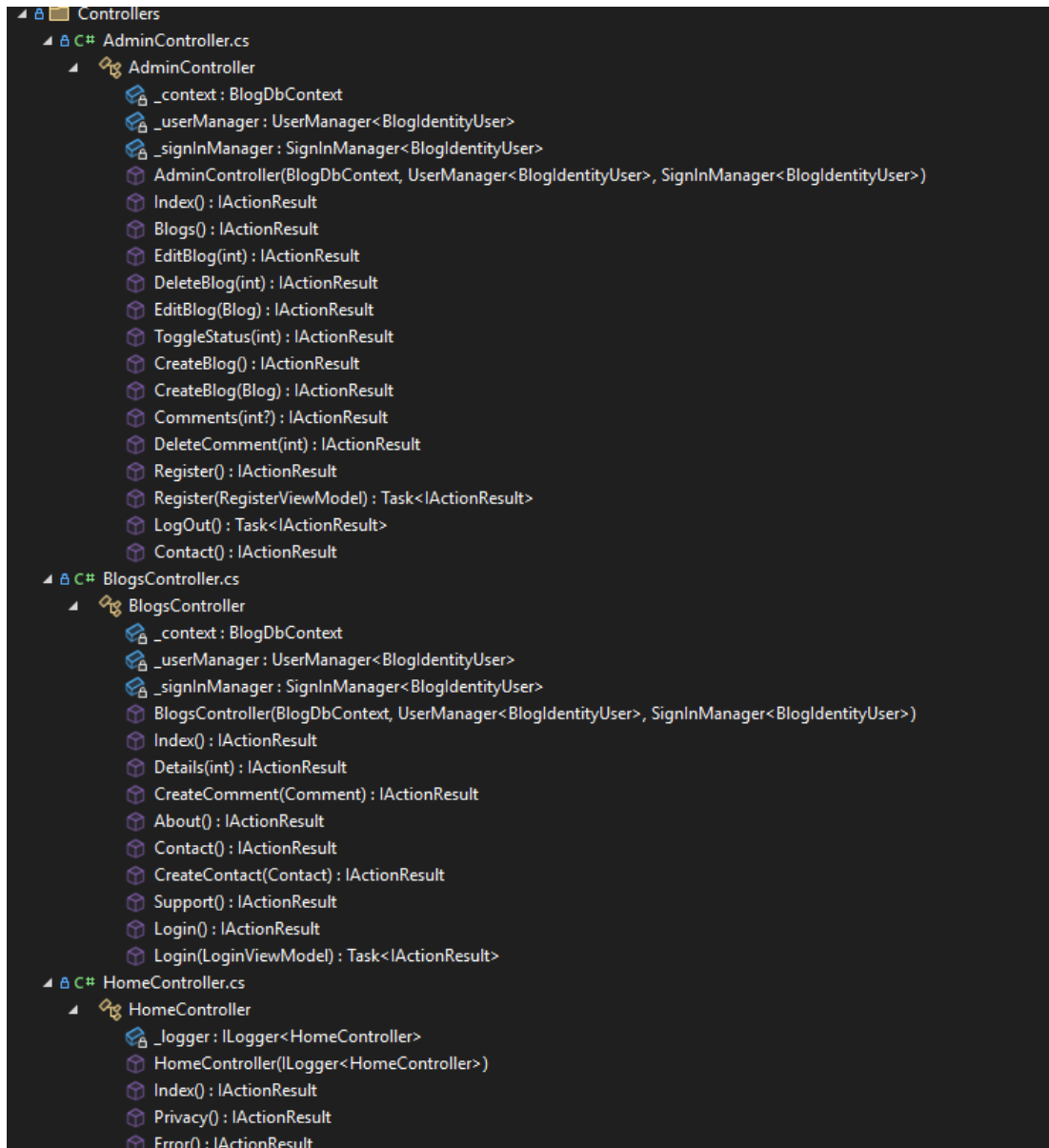
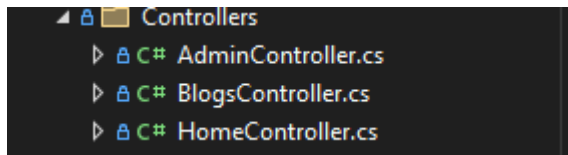
- **public DbSet<Blog> Blogs { get; set; }**

`DbSet<T>` genel bir türdür. T, buraya hangi model sınıfını koyarsanız, o tabloyu temsil eder.

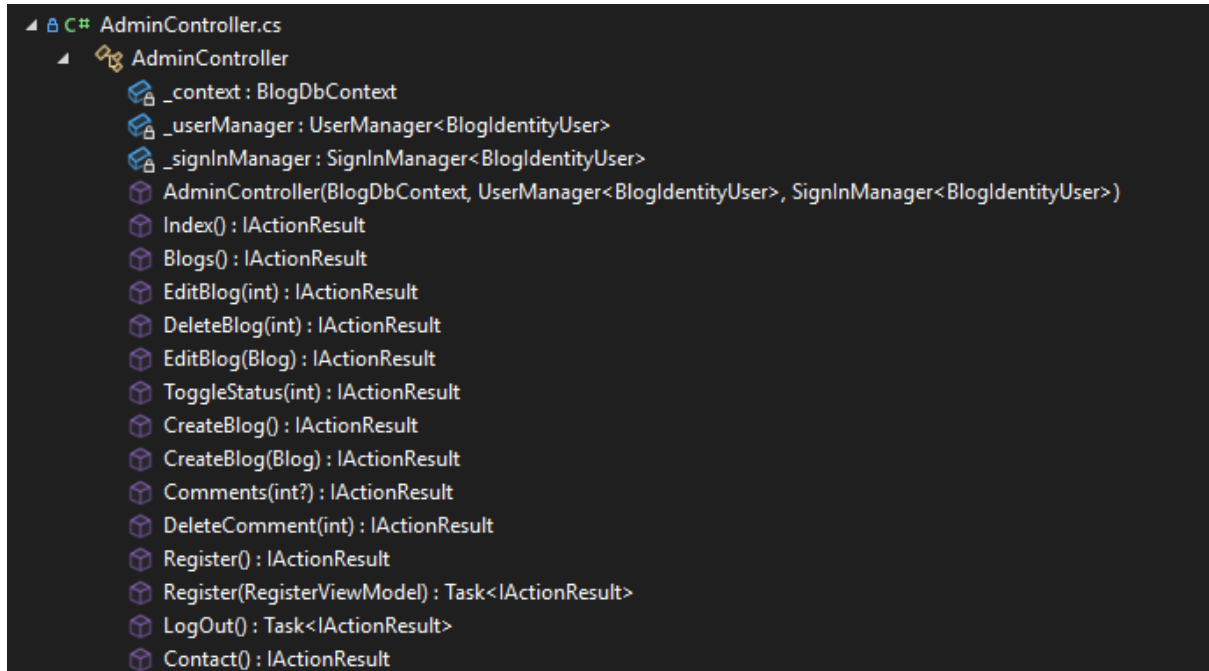
`DbSet<Blog>` ise, veritabanındaki **Blogs tablosunu** temsil eden özel bir türdür.

`{ get; set; }` ifadesi, bir özelliğin hem okunabilir hem de yazılabilir olduğunu belirtir ve bu işlemleri bizim için otomatik olarak yönetir

Controllers



•AdminController



using System.Threading.Tasks;

- **Ne yapıyor:** Bu satır, asenkron programlama için gerekli olan **Task** sınıfını kullanabilmenizi sağlar. Task, bir görevin arka planda çalışmasını ve sonucunu beklemesini yönetmek için kullanılır. **async** ve **await** anahtar kelimeleriyle birlikte kullanılır.

using blogv1.Context;

using blogv1.Identity;

using blogv1.Models;

using blogv1.Models.ViewModels;

- **Ne yapıyor:** Bunlar, projenin farklı katmanlarından (katman: bir uygulamanın birbiriyle ilgili kısımlarının oluşturduğu mantıksal bölümler, ör: veri katmanı, sunum katmanı vb.) sınıfları ve modelleri (Blog, Comment, Contact, DashboardViewModel vb.) kullanabilmek için gerekli olan referanslardır.

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

- **Ne yapıyor:** Bunlar, ASP.NET Core framework'ünün sağladığı temel kütüphaneleri içeri aktarır. **Authorization** (yetkilendirme), **Identity** (kimlik doğrulama) ve **Mvc** (Model-View-Controller) yapılarını kullanmak için gereklidirler.

namespace blogv1.Controllers

- **Ne yapıyor:** Bu, kodun bir ad alanına (namespace) ait olduğunu belirtir. Bu, benzer işlemlere sahip sınıfları gruplamak ve isim çakışmalarını önlemek için kullanılır.

[Authorize]

- **Ne yapıyor:** Bu bir niteliktir (attribute). AdminController sınıfının tamamına uygulanır. Bu, bu denetleyici içindeki her bir metodun çalıştırılmadan önce, kullanıcının sisteme giriş yapmış (yani kimliği doğrulanmış) olup olmadığını kontrol etmesini sağlar. Giriş yapmamış bir kullanıcı bu sayfalara erişemez.

public class AdminController : Controller

- **Ne yapıyor:** AdminController adında bir sınıf tanımlar ve bu sınıfın ASP.NET Core MVC'deki temel Controller sınıfından miras aldığını belirtir. Bu sayede, denetleyiciye özel tüm işlemlere (örneğin View() ve RedirectToAction() gibi metotlara) erişim sağlar.

private readonly BlogDbContext _context;

- **Ne yapıyor?** Bu değişken, uygulamanın veritabanı ile iletişim kurmasını sağlayan bir köprüdür. Bir veritabanı bağlamı (context) olarak, **Entity Framework Core**'un temelidir.
- **Kullanım Amacı:** Bu değişken sayesinde veritabanına sorgu gönderebilir, yeni veriler ekleyebilir, mevcut verileri güncelleyebilir veya silebilirsiniz. Örneğin, bir blog yazısı kaydetmek veya mevcut yazıları listelemek için `_context` kullanılır.
- **readonly ve private Nedir?**
 - **private:** Bu değişkenin sadece tanımlandığı sınıf içinde erişilebilir olduğunu belirtir. Dışarıdan doğrudan değiştirilemez veya okunamaz. Bu, kapsülleme (encapsulation) ilkesinin bir parçasıdır ve kodun daha güvenli ve düzenli olmasını sağlar.
 - **readonly:** Bu değişkenin değerinin sadece tanımlandığı anda (veya yapıcı metot içinde) bir kez atanabileceğini gösterir. Değer bir kere atandıktan sonra bir daha değiştirilemez. Bu, verinin bütünlüğünü korumaya yardımcı olur.

private readonly UserManager<BlogIdentityUser> _userManager;

- **Ne yapıyor?** Bu, ASP.NET Core Identity'nin bir parçasıdır ve kullanıcı hesaplarıyla ilgili işlemleri yönetir.
- **Kullanım Amacı:**
 - Yeni bir kullanıcı kaydetme (`CreateAsync`).
 - Kullanıcının şifresini değiştirme veya sıfırlama.
 - Bir kullanıcıyı role atama veya rollerini kontrol etme.
 - Kullanıcı bilgilerini bulma veya doğrulama.

private readonly SignInManager<BlogIdentityUser> _signInManager;

- **Ne yapıyor?** Bu da ASP.NET Core Identity'nin bir parçasıdır ve kullanıcıların uygulamaya giriş-çıkış (oturum açma/kapatma) işlemlerini yönetir.
- **Kullanım Amacı:**
 - Kullanıcının girdiği e-posta ve şifrenin doğruluğunu kontrol edip oturum açma (`PasswordSignInAsync`).
 - Kullanıcının oturumunu kapatma (`SignOutAsync`).
 - İki faktörlü kimlik doğrulama gibi gelişmiş oturum yönetimi senaryolarını ele alma.

Bu üç değişken, genellikle bir yapıcı metot (constructor) aracılığıyla "Dependency Injection" (Bağımlılık Enjeksiyonu) adı verilen bir mimari desen kullanılarak sınıfınıza

sağlanır. Böylece, ilgili servisleri manuel olarak oluşturmak zorunda kalmazsınız ve kodunuz daha test edilebilir ve esnek hale gelir.

Bu örnekte, bir `HesapMakinesi` sınıfımız var ve bu sınıfın içinde sayıları toplayan bir `Topla` metodu bulunuyor.

C#

```
public class HesapMakinesi
{
    // Bu bir metotdur. Sınıfın bir parçasıdır.
    public int Topla(int sayi1, int sayi2)
    {
        return sayi1 + sayi2;
    }
}

public class Program
{
    public static void Main(string[] args)
    {
        // HesapMakinesi sınıfından bir nesne (object) oluşturuyoruz.
        HesapMakinesi makine = new HesapMakinesi();

        // Nesnenin Topla metodunu çağırıyoruz.
        int toplam = makine.Topla(5, 10);

        Console.WriteLine("Toplam: " + toplam); // Çıktı: Toplam: 15
    }
}
```

AdminController Yapıcı Metodu (Constructor)

Verdiğiniz kod bloğu, **Dependency Injection (Bağımlılık Enjeksiyonu)** adı verilen temel bir yazılım tasarım desenini gösteriyor. Bu yöntem, bir sınıfın (burada `AdminController`) ihtiyaç duyduğu diğer nesneleri (bağımlılıkları) kendi içinde oluşturmak yerine, dışarıdan almasını sağlar. Bu, kodun daha esnek, bakımı kolay ve test edilebilir olmasını sağlar.

Bu kod satırını üç ana başlık altında inceleyebiliriz:

1. Metot Tanımı: `public AdminController(...)`

- **public:** Bu anahtar kelime, yapıcı metodun herkese açık olduğunu, yani uygulamanın herhangi bir yerinden çağrılabilceğini belirtir.
- **AdminController:** Bu, metodun ait olduğu sınıfın adıdır. Yapıcı metotların her zaman ait oldukları sınıf ile aynı adı taşıması gerekir.

2. Parametreler: `(BlogDbContext context, UserManager<BlogIdentityUser> userManager, SignInManager<BlogIdentityUser> signInManager)`

- Bu parantezler içinde yer alan üç parametre, AdminController sınıfının çalışmak için ihtiyaç duyduğu servislerdir.
- **BlogDbContext context:** Veritabanı işlemlerini yöneten servis.
- **UserManager<BlogIdentityUser> userManager:** Kullanıcı hesaplarının (kayıt, şifre değişikliği vb.) yönetilmesini sağlayan servis.
- **SignInManager<BlogIdentityUser> signInManager:** Kullanıcıların oturum açma ve kapatma işlemlerini yöneten servis.

`public IActionResult Index()`

- **var dashboard = new DashboardViewModel(); Tam Olarak Ne Oluyor?**
- Bu satır, bellekte yeni bir "kutucuk" oluşturup ona "dashboard" ismini veriyor ve bu kutucuğun içine DashboardViewModel sınıfına ait bir nesne yerleştiriyor.
- **new DashboardViewModel():** Bu kısım, DashboardViewModel adında bir sınıfın yeni bir örneğini (instance) oluşturur. Sınıfları bir nevi plan veya kalıp gibi düşünebilirsiniz. new anahtar kelimesi, bu kalıba uygun, somut bir nesne yaratır. Bu nesne, DashboardViewModel sınıfının içinde tanımlanmış tüm özelliklere (properties) ve metotlara sahip olur.
- **var dashboard =:** Bu kısım, oluşturulan bu yeni nesneyi saklamak için bir değişken tanımlar.
 - **var:** Bu anahtar kelime, C# dilinde tip çıkarımı (type inference) için kullanılır. Derleyici, new DashboardViewModel() ifadesinin döndürdüğü nesnenin tipini otomatik olarak anlar ve dashboard değişkenini DashboardViewModel tipinde tanımlar. Yani, bu satır aslında DashboardViewModel dashboard = new DashboardViewModel(); ile aynı anlama gelir. Bu, kodu daha kısa ve okunabilir hale getirir.

- **dashboard:** Bu, oluşturulan nesneye verdiğimiz isimdir. Artık kodun geri kalanında bu isimle bu nesneye erişebilir, onun özelliklerini (örneğin, `dashboard.KullaniciAdi`) veya metotlarını (`dashboard.Verigetir()`) kullanabiliriz.

```

3 references
public class DashboardViewModel
{
    //toplam blog sayisi
    2 references
    public int TotalBlogCount { get; set; }
    //toplam görüntülenme sayisi
    2 references
    public int TotalViewCount { get; set; }
    //en çok görüntülenen blog
    //public Blog MostViewedBlog { get; set; }
    ////en son yayınlanan blog
    //public Blog LatestBlog { get; set; }
    ////toplam yorum sayisi
    //public int TotalCommentCount { get; set; }
    ////en çok yorum alan blog
    //public Blog MostCommentedBlog { get; set; }
    ////bugün yapılan yorum sayisi
    2 references
    public int TodayCommentCount { get; set; }
}

```

- Burdan da görüldüğü üzere model aslında bir sınıf oluyor
- `var toplamblogsayisi = _context.Blogs.Count();` işlemi yapıyoruz
- `dashboard.TotalBlogCount = toplamblogsayisi;` işlem sonucunu dashboarddaki iteme eşitliyoruz
- **return View(dashboard);** Bu satır, **dashboard** nesnesini View (görünüm) adında bir web sayfasına gönderir. Bu View genellikle `Index.cshtml` gibi bir Razor sayfasıdır ve bu sayfadaki HTML kodları, gönderilen dashboard modelindeki verilere erişerek onları ekranda gösterebilir. Örneğin, toplam blog sayısını `@Model.TotalBlogCount` şeklinde kullanarak ekrana yazdırabilir. Çok mantıklı

```

@using blogv1.Models.ViewModels

@{
    Layout = "/Views/Shared/_AdminLayout.cshtml";
}

@model DashboardViewModel

<!-- Page Heading -->
<div class="d-sm-flex align-items-center justify-content-between mb-4">
    <h1 class="h3 mb-0 text-gray-800">Dashboard</h1>
    <a href="#" class="d-none d-sm-inline-block btn btn-sm btn-primary shadow-sm">
        <i class="fas fa-download fa-sm text-white-50"></i> Rapor Olustur
    </a>
</div>

<!-- Content Row -->
<div class="row">

    <!-- Earnings (Monthly) Card Example -->
    <div class="col-xl-2 col-md-6 mb-4">
        <div class="card border-left-primary shadow h-100 py-2">
            <div class="card-body">
                <div class="row no-gutters align-items-center">
                    <div class="col mr-2">
                        <div class="text-xs font-weight-bold text-primary text-uppercase mb-1">
                            Toplam Blog Sayisi
                        </div>
                        <div class="h5 mb-0 font-weight-bold text-gray-800">@Model.TotalBlogCount</div>
                    </div>
                    <div class="col-auto">
                        <i class="fas fa-calendar fa-2x text-gray-300"></i>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>

```

public IActionResult Blogs()

- **Ne yapıyor:** Blogs eylemini tanımlar. `_context.Blogs.ToList()` komutunu kullanarak veritabanındaki tüm blog gönderilerini bir liste olarak çeker ve bu listeyi **Blogs.cshtml** adlı bir görüntüleme (view) sayfasına gönderir.
- **.Blogs:** Bu, `BlogDbContext` içinde tanımlanmış olan ve veritabanındaki blog yazılarını temsil eden bir koleksiyondur (genellikle bir `DbSet`).
- **.ToList():** Bu metot, **LINQ (Language-Integrated Query)** sorgusunu çalıştırır. `_context.Blogs` ifadesi, henüz veritabanına bir sorgu göndermez, sadece bir sorgu tanımı oluşturur. `.ToList()` metodu çağrıldığında, sorgu veritabanına gönderilir ve tüm blog yazıları alınarak bir liste (`List<Blog>`) haline dönüştürülür. Bu liste, `blogs` adlı değişkene atanır.
- Koddaki yorum satırı, bu işlemin amacını özetliyor: `//sorguya gerek yok tum blogları gostericem`. Bu, herhangi bir filtreleme (Where) veya sıralama (OrderBy) işlemi yapılmadan doğrudan tüm blogların listeleneceğini belirtir.

```

@{
    Layout = "/Views/Shared/_AdminLayout.cshtml";
}
@model List<Blog>

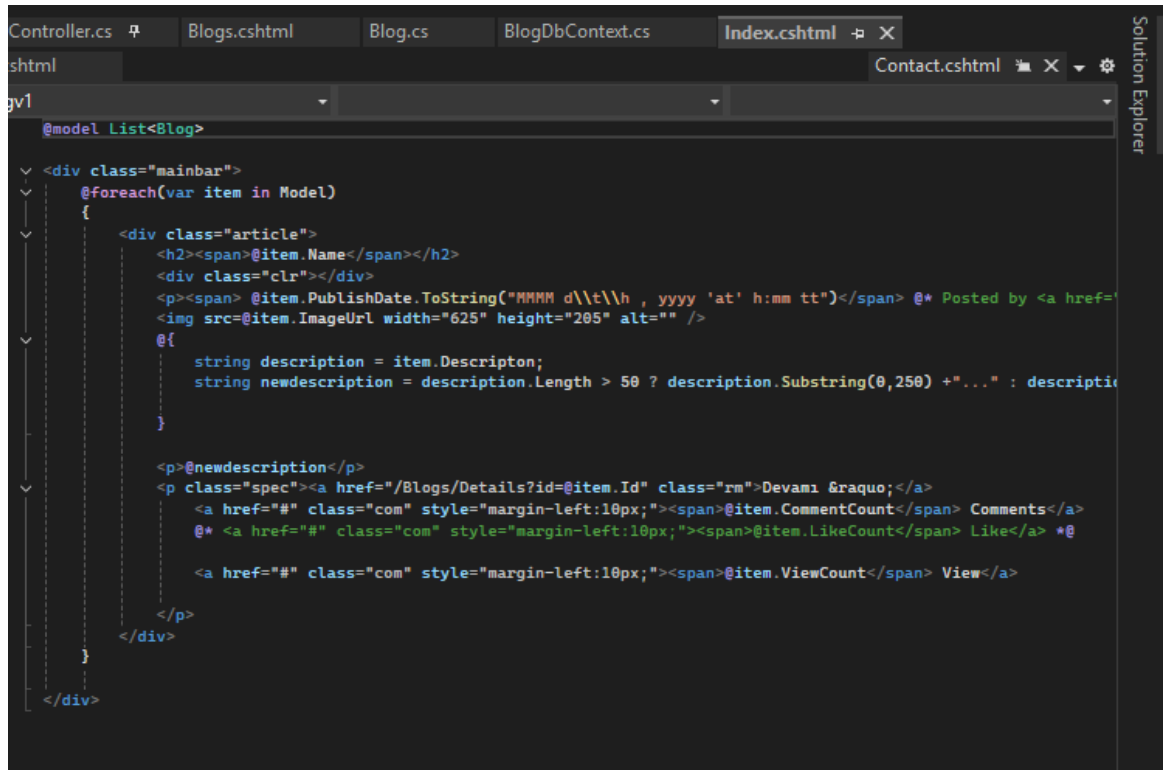
<!-- Page Heading -->
<div class="d-sm-flex align-items-center justify-content-between mb-4">
    <h1 class="h3 mb-0 text-gray-800">Bloglarım</h1>
    <a href="/Admin/CreateBlog" class="d-none d-sm-inline-block btn btn-sm btn-primary shadow-sm">
        <i class="fas fa-download fa-sm text-white-50"></i> Blok Ekle
    </a>
</div>

@if(Model.Count != 0) {
    @foreach(var item in Model)
    {
        <!-- Dropdown Card Example -->
        <div class="card shadow mb-4">
            <!-- Card Header - Dropdown -->
            <div class="card-header py-3 d-flex flex-row align-items-center justify-content-between">
                <h6 class="m-0 font-weight-bold text-primary">@item.Name</h6>
                <div class="dropdown no-arrow">
                    <a class="dropdown-toggle" href="#" role="button" id="dropdownMenuLink"
                        data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
                        <i class="fas fa-ellipsis-v fa-sm fa-fw text-gray-400"></i>
                    </a>
                    <div class="dropdown-menu dropdown-menu-right shadow animated--fade-in"
                        aria-labelledby="dropdownMenuLink">
                        <div class="dropdown-header">Aksiyonlar:</div>
                        <a class="dropdown-item" href="/Admin/EditBlog?id=@item.Id">Duzenle</a>
                        <a class="dropdown-item" href="/Admin/ToggleStatus?id=@item.Id">Acik/Kapali</a>
                        <div class="dropdown-divider"></div>
                        <a class="dropdown-item" href="/Admin/DeleteBlog?id=@item.Id">Sil</a>
                    </div>
                </div>
            </div>
            <!-- Card Body -->
            <div class="card-body">
                @* @item.Description *@
                @{
                    string description = item.Description;
                    string newdescription = description.Length > 50 ? description.Substring(0, 180) + "..." : description;
                }
                @newdescription
            </div>
        </div>
    }
} else
{
    <div class="card shadow mb-4">
        <div class="card-header py-3 d-flex flex-row align-items-center justify-content-between">
            <h6 class="m-0 font-weight-bold text-primary">Blog Olusturun</h6>
        </div>
    </div>
}

```

-
- Tam dediği gibi biz burad listeleme yapıyoruz sonrasında view ile modele gönderip orda liste hazır hale getirip foreach ile hepsini basıyoruz. Bu üstteki admin tarafı

bu alttaki de user tarafı bak



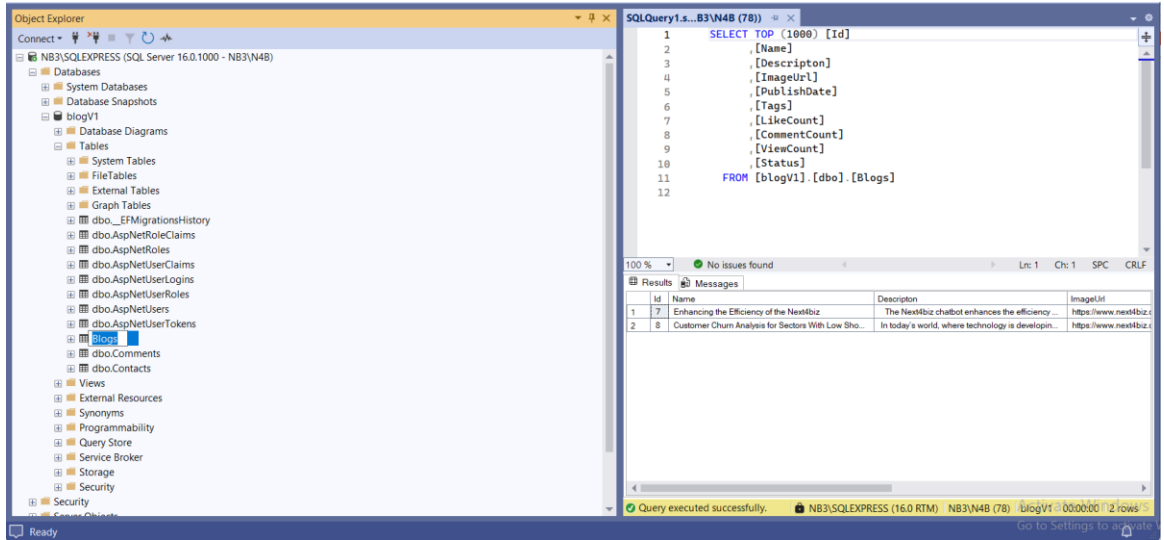
```
@model List<Blog>

<div class="mainbar">
    @foreach(var item in Model)
    {
        <div class="article">
            <h2><span>@item.Name</span></h2>
            <div class="clr"></div>
            <p><span> @item.PublishDate.ToString("MMM d\\t\\h , yyyy 'at' h:mm tt")</span> @* Posted by <a href=
            <img src=@item.ImageUrl width="625" height="205" alt="" />
            @{
                string description = item.Description;
                string newdescription = description.Length > 50 ? description.Substring(0,250) + "... " : description
            }

            <p>@newdescription</p>
            <p class="spec"><a href="/Blogs/Details?id=@item.Id" class="rm">Devamı &raquo;</a>
            <a href="#" class="com" style="margin-left:10px;"><span>@item.CommentCount</span> Comments</a>
            @* <a href="#" class="com" style="margin-left:10px;"><span>@item.LikeCount</span> Like</a> *@
            <a href="#" class="com" style="margin-left:10px;"><span>@item.ViewCount</span> View</a>

            </p>
        </div>
    }
</div>
```

- Gene bir model ve foreach
- **Model Tanımı: @model List<Blog>**
- Bu satır, bu Razor sayfasının, **List<Blog>** tipinde bir veri modeli beklediğini belirtir. Yani, sayfa yüklendiğinde, Controller'dan blog listesi gelmesini bekler. Bu, sayfadaki verilere @Model anahtar kelimesi ile erişilmesini sağlar.
- **Döngü ve Blog Verilerini İşleme: @foreach(var item in Model)**
- **@foreach:** Bu, C# dilinde bir döngüdür ve Razor sayfalarında HTML içinde C# kodu çalıştırmak için @ sembolü ile kullanılır. Bu döngü, Model olarak gelen her bir Blog nesnesi için aşağıdaki HTML bloğunu tekrarlar.
- **var item in Model:** Döngü her döndüğünde, mevcut blog nesnesi item değişkenine atanır.



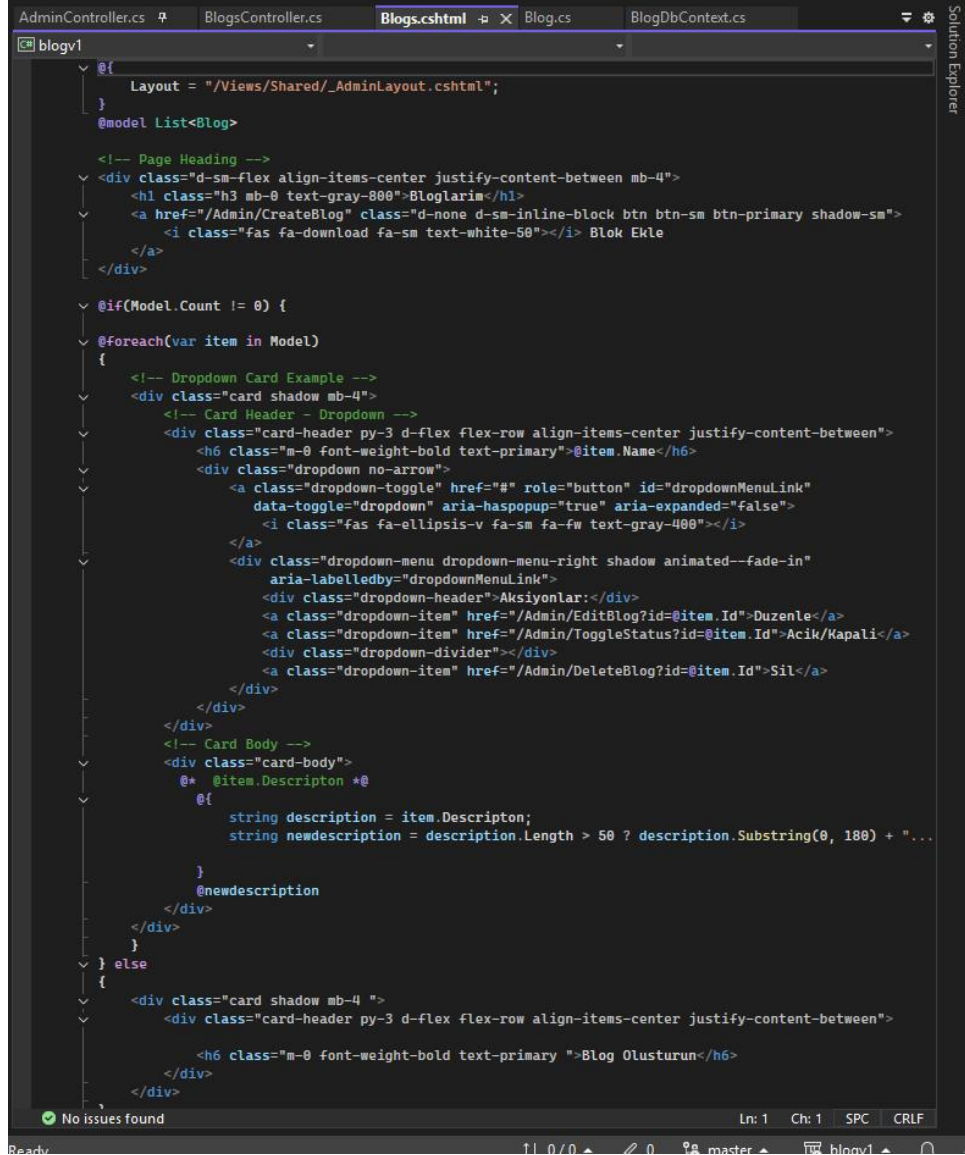
Controllerda buna ulasiyor

public IActionResult DeleteBlog(int id)

- **Blog ID'sini Alma**
- **public IActionResult DeleteBlog(int id):** Metot, id adında bir **integer (tamsayı)** parametre alır. Bu ID, silinecek blog yazısını benzersiz bir şekilde tanımlar.
- **Silinecek Blogu Bulma**
- **var blog = _context.Blogs.Where(x => x.Id == id).FirstOrDefault();**
 - **_context.Blogs:** Veritabanındaki Blogs tablosuna erişim sağlar.
 - **.Where(x => x.Id == id):** Bu **LINQ** sorgusu, Blogs tablosundaki tüm kayıtları ID'lerine göre filtreler. Yalnızca gelen id parametresiyle eşleşen blogları seçer.
 - **.FirstOrDefault():** Filtreleme sonucunda bulunan ilk kaydı alır. Eğer eşleşen bir blog bulunamazsa, null döner. Bu adım, veritabanından yalnızca bir tane blog kaydı getirmek için kullanılır.
- **Blogu Silme Komutunu Verme**
- **_context.Blogs.Remove(blog);** Bu satır, Remove metodunu kullanarak, bellekte bulunan blog nesnesini veritabanından silinmek üzere işaretler. Bu noktada blog henüz veritabanından silinmemiştir, sadece **Entity Framework** tarafından "silinecek" olarak etiketlenmiştir.
- **Değişiklikleri Veritabanına Kaydetme**
- **_context.SaveChanges();** Bu metot çağrıldığında, Entity Framework, işaretlenmiş olan tüm değişiklikleri (silme, ekleme, güncelleme) toplar ve tek bir

işlemede veritabanına göndererek kalıcı hale getirir. Bu satırdan sonra, ilgili blog kaydı veritabanından silinmiş olur.

- return RedirectToAction("Blogs"); blogs burda sitesel olarak gideceği yer ondan metin olarak alıyor



```
@{
    Layout = "/Views/Shared/_AdminLayout.cshtml";
}
@model List<Blog>

<!-- Page Heading -->
<div class="d-sm-flex align-items-center justify-content-between mb-4">
    <h1 class="h3 mb-0 text-gray-800">Bloglarım</h1>
    <a href="/Admin/CreateBlog" class="d-none d-sm-inline-block btn btn-sm btn-primary shadow-sm">
        <i class="fas fa-download fa-sm text-white-50"> Blok Ekle
    </a>
</div>

@if(Model.Count != 0) {
    @foreach(var item in Model)
    {
        <!-- Dropdown Card Example -->
        <div class="card shadow mb-4">
            <!-- Card Header - Dropdown -->
            <div class="card-header py-3 d-flex flex-row align-items-center justify-content-between">
                <h6 class="m-0 font-weight-bold text-primary">@item.Name</h6>
                <div class="dropdown no-arrow">
                    <a class="dropdown-toggle" href="#" role="button" id="dropdownMenuLink"
                        data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
                        <i class="fas fa-ellipsis-v fa-sm fa-fw text-gray-400">
                    </a>
                    <div class="dropdown-menu dropdown-menu-right shadow animated--fade-in"
                        aria-labelledby="dropdownMenuLink">
                        <div class="dropdown-header">Aksiyonlar:</div>
                        <a class="dropdown-item" href="/Admin/EditBlog?id=@item.Id">Düzenle</a>
                        <a class="dropdown-item" href="/Admin/ToggleStatus?id=@item.Id">Açık/Kapalı</a>
                        <div class="dropdown-divider"></div>
                        <a class="dropdown-item" href="/Admin/DeleteBlog?id=@item.Id">Sil</a>
                    </div>
                </div>
            </div>
            <!-- Card Body -->
            <div class="card-body">
                @* @item.Description *@
                @{
                    string description = item.Description;
                    string newdescription = description.Length > 50 ? description.Substring(0, 180) + "...
                }
                @newdescription
            </div>
        </div>
    }
} else {
    <div class="card shadow mb-4 ">
        <div class="card-header py-3 d-flex flex-row align-items-center justify-content-between">
            <h6 class="m-0 font-weight-bold text-primary ">Blog Oluşturun</h6>
        </div>
    </div>
}
```

- tekrar bu sayfayı yüklüyor yani blogların olduğu edit kısmını

[HttpPost]

- **Ne yapıyor:** Bu nitelik, bir metodun sadece HTTP POST isteği ile çalışabileceğini belirtir. Genellikle form gönderimlerinde kullanılır. EditBlog ve CreateBlog metotları için, kullanıcının bir form gönderdiğini anlar ve bu metotları çalıştırır.

`public IActionResult EditBlog(Blog model)`

- `public IActionResult EditBlog(Blog model)`: Bu metod, Blog türünde bir **model** parametresi alır. Bu model, kullanıcının formda doldurduğu ve sunucuya gönderdiği güncel blog verilerini (başlık, açıklama, etiketler, görsel URL'si vb.) içerir

```
11 references
public class Blog
{
    17 references
    public int Id { get; set; }
    6 references
    public string Name { get; set; }
    6 references
    public string Description { get; set; }

    4 references
    public string ImageUrl { get; set; }
    3 references
    public DateTime PublishDate { get; set; }

    5 references
    public string Tags { get; set; }

    0 references
    public int LikeCount { get; set; }
    3 references
    public int CommentCount { get; set; }

    3 references
    public int ViewCount { get; set; }
    6 references
    public int Status { get; set; }
}
```

-
- **Mevcut Blogu Veritabanından Bulma**
- `var blog = _context.Blogs.Where(x => x.Id == model.Id).FirstOrDefault();`
 - Bu satır, model nesnesinin Id özelliğini kullanarak, veritabanındaki mevcut blog kaydını bulur. Bu adım çok önemlidir çünkü doğrudan gelen model nesnesini güncellemek yerine, veritabanındaki **orijinal** nesne üzerinde değişiklik yapılır. Bu, Entity Framework'ün değişiklikleri doğru bir şekilde takip etmesini sağlar.
 - Bu noktada, bellekte iki farklı blog nesnesi vardır: veritabanından gelen **blog** nesnesi ve kullanıcının formdan gönderdiği **model** nesnesi.

- **Özellikleri Güncelleme**
- **blog.Name = model.Name;** Orijinal blog nesnesinin Name özelliği, formdan gelen model nesnesinin Name özelliği ile güncellenir.
- **blog.Descripton = model.Descripton;** Açıklama (Descripton) özelliği güncellenir.
- **blog.Tags = model.Tags;** Etiketler (Tags) özelliği güncellenir.
- **blog.ImageUrl = model.ImageUrl;** Görsel URL'si (ImageUrl) özelliği güncellenir.
- **Değişiklikleri Kaydetme ve Yönlendirme**
- **_context.SaveChanges();** Bu metot, yapılan tüm değişiklikleri veritabanına kalıcı olarak kaydeder.
- **return RedirectToAction("Blogs");** Güncelleme işlemi tamamlandıktan sonra, kullanıcıyı tekrar blog listeleme sayfasına yönlendirir. Bu, kullanıcının düzenlediği blogun güncel halini listede görmesini sağlar. Bu yönlendirme, GET isteğiyle çalışan Blogs eylemini tetikler ve tarayıcının sayfaı yeniden yüklemesini sağlar.

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
26
```

- Blog duzenleme sayfasına boyle veriyor modelle

```

</a>
*@
<button class="btn btn-success btn-icon-split" type="submit">
  <span class="icon text-white-50">
    <i class="fas fa-check"></i>
  </span>
  <span class="text">Kaydet</span>
</button>

@* <div class="my-2"></div> *@
<a href="/Admin/DeleteBlog?id=@Model.Id" class="btn btn-danger btn-icon-split">
  <span class="icon text-white-50">
    <i class="fas fa-trash"></i>
  </span>
  <span class="text">Sil</span>
</a>

<a href="/Admin/Comments?blogId=@Model.Id" class="btn btn-info btn-icon-split">
  <span class="icon text-white-50">
    <i class="fas fa-info-circle"></i>
  </span>
  <span class="text">Yorumları Gor</span>
</a>

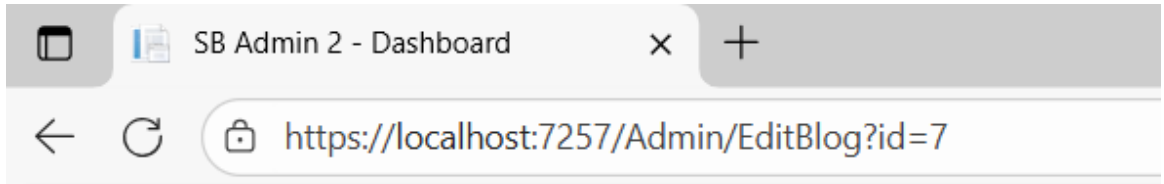
<a href="/Admin/Blogs" class="btn btn-secondary btn-icon-split">*@ //bir geri sayfa *@
  <span class="icon text-white-50">
    <i class="fas fa-arrow-right"></i>
  </span>
  <span class="text">Geri Don</span>
</a>

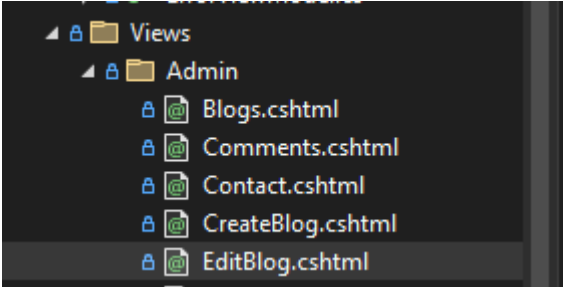
@if(Model.Status==1){
  <a href="/Admin/ToggleStatus?id=@Model.Id" class="btn btn-warning btn-icon-split">
    <span class="icon text-white-50">
      <i class="fas fa-exclamation-triangle"></i>
    </span>
    <span class="text">Kapa</span>
  </a>
}else{
  <a href="/Admin/ToggleStatus?id=@Model.Id" class="btn btn-primary btn-icon-split">
    <span class="icon text-white-50">
      <i class="fas fa-flag"></i>
    </span>
    <span class="text">Ac</span>
  </a>
}

```

public IActionResult ToggleStatus(int id)

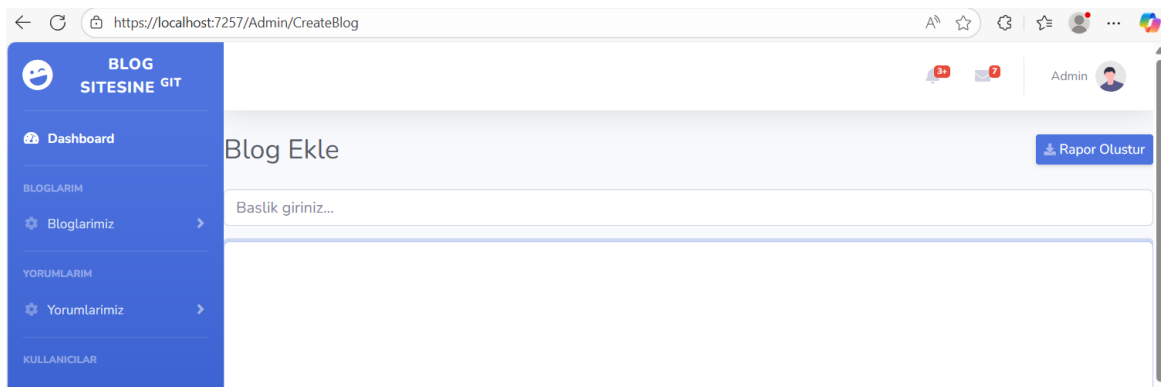
- Bu metot, bir blog yazısının yayımlanma durumunu (aktif/pasif) değiştirir.
- Şu yukardaki if else dongusu aslında yani orda kullanılıyor
- ``:
- Bu bir HTML bağlantı etiketidir.
- **href**: Butonun tıklanıldığında gideceği adresi belirtir.
- **`"/Admin/ToggleStatus?id=@Model.Id"`**: Bu adres, Admin kontrolcüsünün içindeki ToggleStatus eylemini çağırır ve bu eyleme, blogun benzersiz kimliğini (Model.Id) bir parametre olarak gönderir.



- 
- Bak bunlar burdan direkt üstteki linki alıyor

public IActionResult CreateBlog()

- **GET metodu** boş bir blog oluşturma formunu gösterir. Alttakını yani



```
@{
    Layout = "/Views/Shared/_AdminLayout.cshtml";
}
@* @model Blog *@
<!-- Page Heading -->
<div class="d-sm-flex align-items-center justify-content-between mb-4">
    <h1 class="h3 mb-0 text-gray-800">Blog Ekle</h1>
    <a href="#" class="d-none d-sm-inline-block btn btn-sm btn-primary shadow-sm">
        <i class="fas fa-download fa-sm text-white-50"></i> Rapor Olustur
    </a>
</div>

<form action="/Admin/CreateBlog" method="post" id="createblogform">

    <div class="form-group">
        <input type="text" class="form-control form-control-user"
            id="blogname" aria-describedby="blogname"
            placeholder="Baslik giriniz..." required name="name">
    </div>
    <div class="form-group">
        <textarea type="text" class="form-control form-control-user"
            id="blogdescription" placeholder="Aciklama Ekleyiniz..." required name="description" rows=
    </div>
    <div class="form-group">
        <input type="text" class="form-control form-control-user"
            id="blogimgurl" aria-describedby="blogimgurl"
            placeholder="Fotograf Url'sini Giriniz..." required name="imageUrl">
    </div>
    <div class="form-group">
        <input type="text" class="form-control form-control-user"
            id="blogtags" aria-describedby="blogtags"
            placeholder="Etiketleri giriniz..." required name="tags">
    </div>
</form>
```

public IActionResult CreateBlog(Blog model)

- Bu eylem, kullanıcı formu doldurup "Kaydet" veya "Oluştur" butonuna tıkladığında çalışır.
- **[HttpPost]**: Bu etiket, bu metodun yalnızca **HTTP POST** istekleri tarafından çağrılabilceğini belirtir. Bu, veritabanına veri gönderen işlemler için güvenlik ve doğruluk sağlar.
- **Blog model**: Bu, formdan gelen verilerin otomatik olarak Blog nesnesine bağlanmasını (Model Binding) sağlar. Kullanıcının formda girdiği başlık, içerik, etiketler gibi bilgiler bu model nesnesine aktarılır.
- **model.PublishDate = DateTime.Now;** Blogun PublishDate (yayınlanma tarihi) özelliği, işlemin yapıldığı anın tarihi ve saati ile doldurulur. Bu, kullanıcının girmesi gerekmeyen ancak sistemin otomatik olarak eklemesi gereken bir veridir.
- **model.Status = 1;** Blogun Status (durum) özelliği **1** olarak ayarlanır. **1** genellikle blogun "aktif" veya "yayınlanmış" olduğunu belirtir.
- **_context.Blogs.Add(model);** Bu komut, hazırlanan model nesnesini veritabanındaki Blogs koleksiyonuna eklemek için Entity Framework'e bildirir. Bu aşamada, veritabanında henüz bir değişiklik yapılmaz, yalnızca yeni bir kayıt için hazırlık yapılır.
- **_context.SaveChanges();** Bu komut, Entity Framework'e, bekleyen tüm değişiklikleri (bu durumda yeni blog ekleme) veritabanına kalıcı olarak

kaydetmesi talimatını verir. Bu adımdan sonra, yeni blog yazısı veritabanına kaydedilmiş olur.

- **return RedirectToAction("Blogs");**: İşlem tamamlandıktan sonra, kullanıcı, blog listeleme sayfasını gösteren Blogs eylemine yönlendirilir. Bu, kullanıcının yeni eklediği blogu listede görebilmesini sağlar.
- Baya iyi

public IActionResult Comments(int? blogId)

- **blogId** parametresi varsa o bloga ait yorumları, yoksa tüm yorumları gösterir.
- **?**: Soru işareti, bu tamsayının **null** olabileceğini, yani parametrenin opsiyonel olduğunu gösterir.
- **var comments = new List<Comment>();**
- **var**: C# dilinde değişken tipini derleyicinin otomatik olarak belirlemesini sağlar.
- **comments**: Değişkenin adıdır. Bu değişken, veritabanından çekilecek yorumları tutacaktır.
- **new List<Comment>()**: **Comment** nesnelerinden oluşacak boş bir liste oluşturur. Bu, değişkeni başlangıçta boş bir liste olarak başlatır. Kodun içindeki yorum satırı (hpsterıyor nereden geldiğini bu sefer 0 değilse) bu satırın amacını net olmasa da belirtiyor: Yorum listesinin başlangıç noktasını oluşturmak.
- **return View(comments);**
- **return View(...)**: Bu metot, hazırlanan **comments** listesini bir web sayfasına (**View**) gönderir.
- **View(comments)**: Otomatik olarak, metot adıyla eşleşen **Comments.cshtml** adındaki **View** dosyasını bulur ve **comments** listesini bu **View**'a bir model olarak gönderir. **View** dosyasında bu listeye **@Model** anahtar kelimesi ile erişilebilir.

```
namespace blogv1.Models
{
    5 references
    public class Comment
    {
        2 references
        public int Id { get; set; }

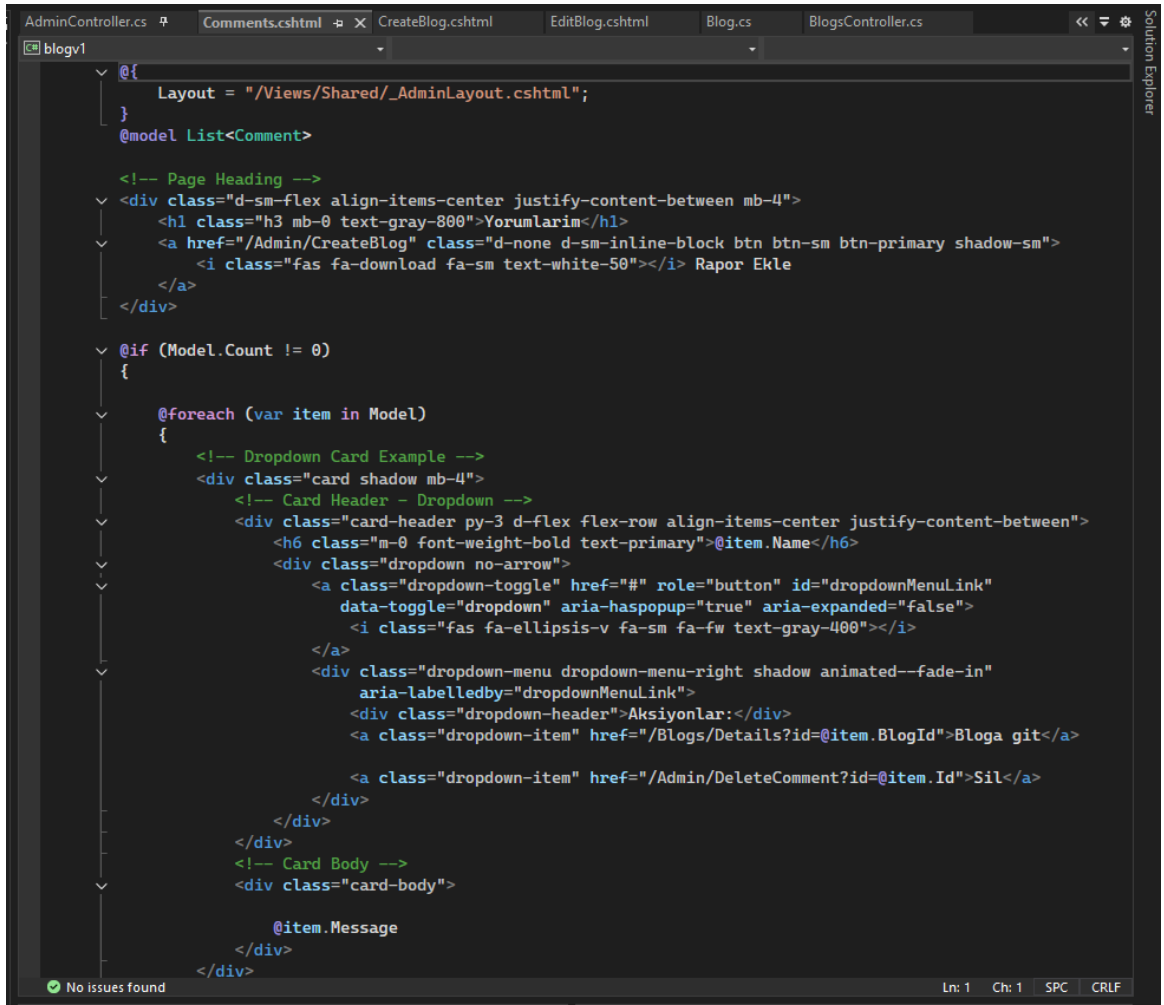
        5 references
        public int BlogId { get; set; } //hangisini yaptıysak ona

        2 references
        public DateTime PublishDate { get; set; }

        1 reference
        public string Name { get; set; }
        0 references
        public string Email { get; set; }

        1 reference
        public string Message { get; set; }
    }
}
```

- O gönderileni böyle de view de kullanıyoruz.



```
@{
    Layout = "/Views/Shared/_AdminLayout.cshtml";
}
@model List<Comment>

<!-- Page Heading -->
<div class="d-sm-flex align-items-center justify-content-between mb-4">
    <h1 class="h3 mb-0 text-gray-800">Yorumlarım</h1>
    <a href="/Admin/CreateBlog" class="d-none d-sm-inline-block btn btn-sm btn-primary shadow-sm">
        <i class="fas fa-download fa-sm text-white-50"></i> Rapor Ekle
    </a>
</div>

@if (Model.Count != 0)
{
    @foreach (var item in Model)
    {
        <!-- Dropdown Card Example -->
        <div class="card shadow mb-4">
            <!-- Card Header - Dropdown -->
            <div class="card-header py-3 d-flex flex-row align-items-center justify-content-between">
                <h6 class="m-0 font-weight-bold text-primary">@item.Name</h6>
                <div class="dropdown no-arrow">
                    <a class="dropdown-toggle" href="#" role="button" id="dropdownMenuLink"
                        data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
                        <i class="fas fa-ellipsis-v fa-sm fa-fw text-gray-400"></i>
                    </a>
                    <div class="dropdown-menu dropdown-menu-right shadow animated--fade-in"
                        aria-labelledby="dropdownMenuLink">
                        <div class="dropdown-header">Aksiyonlar:</div>
                        <a class="dropdown-item" href="/Blogs/Details?id=@item.BlogId">Bloga git</a>

                        <a class="dropdown-item" href="/Admin/DeleteComment?id=@item.Id">Sil</a>
                    </div>
                </div>
            </div>
            <!-- Card Body -->
            <div class="card-body">
                @item.Message
            </div>
        </div>
    }
}
```

- Varsayılan olarak, ASP.NET Core, View dosyalarını bulmak için belirli bir yolu takip eder: [Kontrolcü Adı]/[Eylem Adı].cshtml ve paylaşılan (Shared) View'lar için Shared klasörü. Bu yol, Views klasörünün içinde aranır.
- Eğer klasör adınızı Views yerine Goruntu veya başka bir şey yaparsanız, uygulamanıza View'ları nerede araması gerektiğini söylemeniz gerekir.

public IActionResult DeleteComment(int id)

- Belirli bir yorumu veritabanından siler.

public IActionResult Register()

- Register sayfasını gösterir

The screenshot shows a web application interface for user registration. The sidebar on the left contains navigation links: Dashboard, Bloglarim, Yorumlarim, Kullanicilar, and Iletisim. The main content area is titled 'Kullanici Kayit' and features a 'Rapor Olustur' button. Below the title are input fields for 'Isim', 'Soyisim', 'Email Adresi', 'Sifre', and 'Sifre Tekrar'. A large blue button labeled 'Kullanici Olustur' is positioned at the bottom of the form. The footer indicates 'Copyright © BlogV1 2025'.

Neden IActionResult Kullanıyoruz?

1. Esneklik ve Farklı Sonuç Tipleri

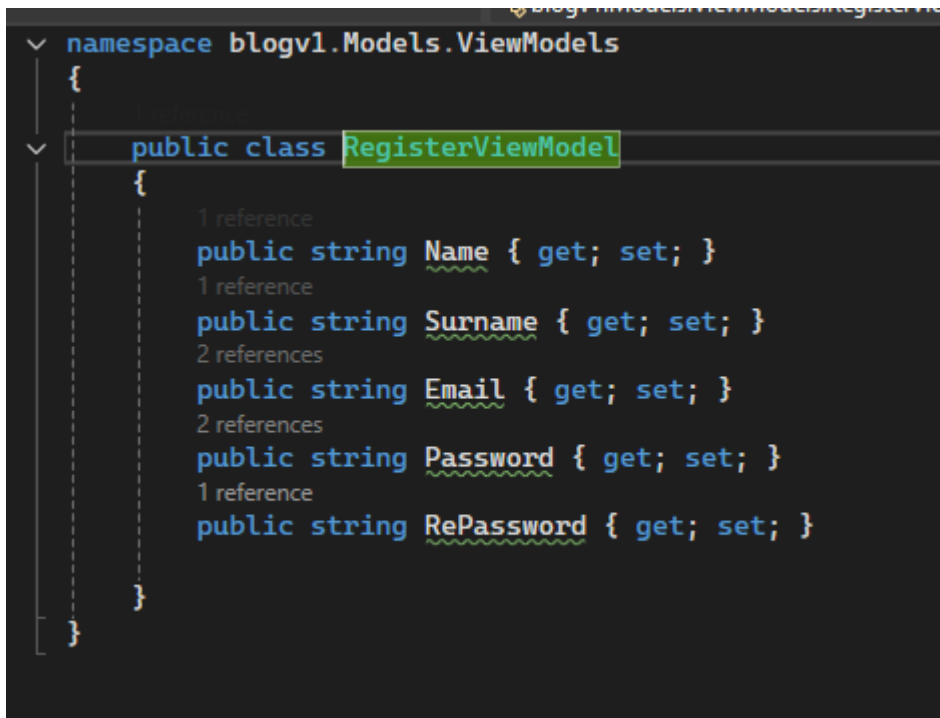
Bir kontrolcü metodu sadece bir View (sayfa) döndürmez. Farklı senaryolarda farklı sonuçlara ihtiyaç duyabilir:

- **View:** Bir web sayfası göstermek için kullanılır (örneğin, `return View(model)`).
- **Redirect (Yönlendirme):** Kullanıcıyı başka bir URL'ye yönlendirmek için kullanılır (örneğin, `return RedirectToAction("Blogs")`).
- **JSON:** Bir API çağrısında veri döndürmek için kullanılır (örneğin, `return Json(data)`).
- **File (Dosya):** Kullanıcıya bir dosya indirtmek için kullanılır.
- **NotFound:** Kaynak bulunamadığında 404 hatası döndürmek için kullanılır (örneğin, `return NotFound()`).
- **Ok/BadRequest:** Başarılı (200 OK) veya hatalı (400 Bad Request) bir işlem sonucunu belirtmek için kullanılır.

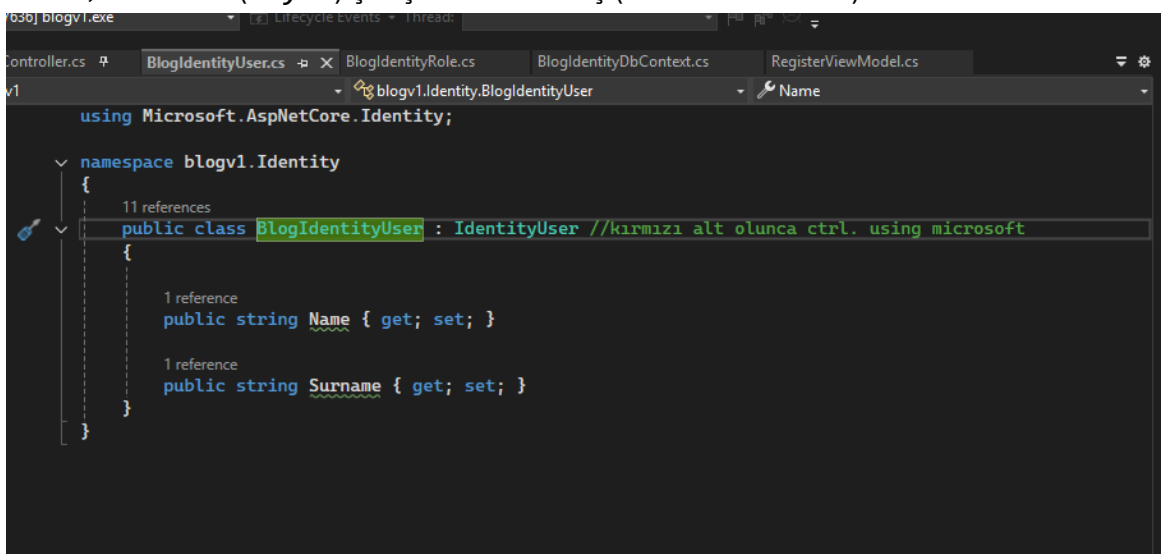
`IActionResult` bu farklı sonuç tiplerinin hepsini kapsayan bir şemsiye gibidir. Bu sayede, aynı metod içinde `if` veya `else` bloklarıyla farklı dönüş tipleri kullanabilirsiniz.

public async Task<IActionResult> Register(RegisterViewModel model)

- Az önceki login gibi bu da kullanıcı oluşturma basınca tetikleniyor



-
- Değişkenleri bunlar
- Username atamak gerekiyor
- **async Task<IActionResult> Register(RegisterViewModel model):**
Metot, RegisterViewModel tipinde bir **model** parametresi alır. Bu model, kullanıcının kayıt formuna girdiği ad, soyad, e-posta ve şifre gibi bilgileri içerir. Metot, asenkron (async) çalışır ve bir sonuç (IActionResult) döndürür.



-
- Tanımlarız yeni
- **Kullanıcı Nesnesi Oluşturma**

- **var user = new BlogIdentityUser { ... };** Şifreler eşleşirse, formdan gelen verilerle **BlogIdentityUser** adında yeni bir kullanıcı nesnesi oluşturulur.
 - Name, Surname, Email ve Username gibi özellikler, model nesnesinden alınan verilerle doldurulur. ASP.NET Core Identity'de Username genellikle Email ile aynı değere ayarlanır.
- **Kullanıcıyı Veritabanına Kaydetme**
- **var result = await _userManager.CreateAsync(user, model.Password);** Bu, **ASP.NET Core Identity**'nin ana bileşenlerinden biri olan **_userManager**'ın **CreateAsync** metodunu çağırır.
 - Bu metod, user nesnesini ve düz metin şifreyi (**model.Password**) parametre olarak alır.
 - **CreateAsync** metodu, şifreyi otomatik olarak **hash'ler** (güvenli bir şekilde şifreler), kullanıcıyı veritabanına kaydeder ve bu işlemin sonucunu (**IdentityResult**) döner.
 - **await** anahtar kelimesi, işlemin asenkron olduğunu ve sonucun beklenmesi gerektiğini belirtir. Bu, uygulamanın bu işlem sırasında engellenmemesini sağlar.
- Registerda şifre Büyük harf küçük harf sayı ve sembol içermesi gerekiyor kutuphaneden gelen özellik

Kütüphaneden Gelen Şifre Kuralları

_userManager.CreateAsync() metodu, kullanıcının girdiği şifreyi kaydetmeden önce varsayılan olarak şu kontrolleri yapar:

- **En az 1 büyük harf**
- **En az 1 küçük harf**
- **En az 1 sayı**
- **En az 1 özel karakter (sembol)**
- **En az 6 karakter uzunluğunda**

Bu kontroller, kullanıcı hesabının güvenliğini artırmak için tasarlanmıştır. **CreateAsync** metodu bu kurallardan herhangi biri ihlal edilirse **başarısız olur** ve **result.Succeeded** özelliği **false** değerini döner.

public async Task<IActionResult> Logout()

- Oturum açmış kullanıcının oturumunu güvenli bir şekilde kapatır ve ana sayfaya yönlendirir.

```
adminController.cs  AdminLayout.cshtml  Contact.cshtml  EditBlog.cshtml  Register.cshtml  Comments.cshtml
blogy1
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <meta name="description" content="">
  <meta name="author" content="">

  <title>SB Admin 2 - Dashboard</title>

  <!-- Custom fonts for this template-->
  <link href="/vendor/fontawesome-free/css/all.min.css" rel="stylesheet" type="text/css">
  <link href="https://fonts.googleapis.com/css?family=Nunito:200,200i,300,300i,400,400i,600,600i,700,700i" rel="stylesheet">

  <!-- Custom styles for this template-->
  <link href="/css/sb-admin-2.min.css" rel="stylesheet">
</head>
<body id="page-top">
  <!-- Page Wrapper -->
  <div id="wrapper">
    <!-- Sidebar -->
    <ul class="navbar-nav bg-gradient-primary sidebar sidebar-dark accordion" id="accordionSidebar">
      <!-- Sidebar - Brand -->
      <a class="sidebar-brand d-flex align-items-center justify-content-center" href="/blogs/index">
        <div class="sidebar-brand-icon rotate-n-15">
          <i class="fas fa-laugh-wink"></i>
        </div>
      </a>
    </ul>
  </div>
</body>
</html>
```

- Adminlayoutta bunun işlemleri view kısmı yani shared altında
- **return RedirectToAction("Index", "Blogs");**
- **return RedirectToAction(...):** Bu komut, kullanıcıyı başka bir eyleme yönlendirir.
- **"Index":** Yönlendirilecek eylemin adıdır. Bu durumda, Index adlı eylem.
- **"Blogs":** Yönlendirilecek eylemin bulunduğu **kontrolcünün** adıdır.
- Bu komut, kullanıcının oturumu kapandıktan sonra **BlogsController** içindeki **Index** eylemine yönlendirileceğini belirtir.

public IActionResult Contact()

- İletişim formundan gelen mesajları listeler.

```

namespace blogv1.Models
{
    4 references
    public class Contact
    {
        0 references
        public int Id { get; set; }

        2 references
        public DateTime CreaatedAt { get; set; }

        1 reference
        public string Name { get; set; }
        1 reference
        public string Email { get; set; }

        1 reference
        public string Message { get; set; }
    }
}

```

```

@{
    Layout = "/Views/Shared/_AdminLayout.cshtml";
}
@model List<Contact>

<!-- Page Heading -->
<div class="d-sm-flex align-items-center justify-content-between mb-4">
    <h1 class="h3 mb-0 text-gray-800">Iletisimler </h1>
    <a href="#" class="d-none d-sm-inline-block btn btn-sm btn-primary shadow-sm">
        <i class="fas fa-download fa-sm text-white-50"></i> Rapor Olustur
    </a>
</div>
<!-- Basic Card Example -->
@foreach(var item in Model)
{
    <div class="card shadow mb-4">
        <div class="card-header py-3">
            <h6 class="m-0 font-weight-bold text-primary">@item.Name - @item.Email - @item.CreaatedAt</h6>
        </div>
        <div class="card-body">
            @item.Message
        </div>
    </div>
}

```

Boyle gosterir adminde

Kullanıcı tarafı da böyle görünür

```
ntroller.cs # Contact.cshtml _AdminLayout.cshtml Contact.cshtml Register.cshtml BlogDbContext.cs
<div class="mainbar">
  <div class="article">
    <h2><span>İletişim</span></h2>
    <div class="clr"></div>
    <p>You can find more of my free template designs at my website. For premium commercial designs, y</p>
  </div>
  <div class="article">
    <h2><span>Bize mail</span> gönder</h2>
    <div class="clr"></div>
    <form action="/Blogs/CreateContact" method="post" id="contactform">
      <ol>
        <li>
          <label for="name">İsim (zorunlu)</label>
          <input id="name" name="name" class="text" required />
        </li>
        <li>
          <label for="email">Email Adresi (zorunlu)</label>
          <input id="email" name="email" class="text" required />
        </li>
        <li>
          <label for="message">Mesajın</label>
          <textarea id="message" name="message" rows="8" cols="50"></textarea>
        </li>
        <li>
          <input type="image" name="imageField" id="imageField" src="/images/submit.gif" class="
          <div class="clr"></div>
        </li>
      </ol>
    </form>
  </div>
</div>
```

- **Neden CreateContact.cshtml Yok?**

Gönderdiğiniz HTML formunun action="/Blogs/CreateContact" adresine veri gönderdiğini belirtmişsiniz. Bu, formun verileri **sunucu tarafındaki bir eyleme (action) göndereceği** anlamına geliyor. Bu eylemin bir görevi de **gelen veriyi işlemek ve veritabanına kaydetmektir**.

Bu işleme ait bir View dosyasına ihtiyacınız yoktur, çünkü bu eylem bir View döndürmez. Bunun yerine, işlem başarıyla tamamlandıktan sonra kullanıcıyı başka bir sayfaya **yönlendirmesi** beklenir.

● BlogsController

İlk kısım gerekli olan veri tabanı bağlantıları yapılır ne kullanınılcaksa

public IActionResult Index()

- **var blogs = _context.Blogs.Where(x => x.Status == 1).ToList();**
Bu satır, veritabanından sadece Status (durum) değeri 1 olan (yani yayınlanmış) blog yazılarını çeker. Bu, taslak halindeki blogların ana sayfada görünmesini engeller.
- **return View(blogs);** Çektiği yayınlanmış blogların listesini, Index adlı sayfaya (View) gönderir ve kullanıcıya gösterir.

```
@model List<Blog>
<div class="mainbar">
  @foreach(var item in Model)
  {
    <div class="article">
      <h2><span>@item.Name</span></h2>
      <div class="clr"></div>
      <p><span> @item.PublishDate.ToString("MMMM d\\t\\h , yyyy 'at' h:mm tt")</span> @* Posted by <a href="#">Owner</a> *@ @* &nbsp; &nbsp; *@&nbsp; &nbsp; Etiketler
      <img src=@item.ImageUrl width="625" height="285" alt="" />
      @*
      @{
        string description = item.Description;
        string newdescription = description.Length > 50 ? description.Substring(0,250) + "...": description;
      }
      <p>@newdescription</p>
      <p class="spec"><a href="/Blogs/Details?id=@item.Id" class="rw">Devamı &raquo;</a>
      <a href="#" class="com" style="margin-left:10px;"><span>@item.CommentCount</span> Comments</a>
      @* <a href="#" class="com" style="margin-left:10px;"><span>@item.LikeCount</span> Like</a> *@
      <a href="#" class="com" style="margin-left:10px;"><span>@item.ViewCount</span> View</a>
      </p>
    </div>
  }
</div>
```

public IActionResult Details(int id)

```
public IActionResult Details(int id)
{
    var blog = _context.Blogs.Where(x => x.Id == id).FirstOrDefault(); //blogs tablousuna bagla
    //first or default uygun olanı veya default olanı döndürür

    blog.ViewCount += 1;
    _context.SaveChanges(); //view sayısını günceller

    var comment = _context.Comments.Where(x => x.BlogId == id).ToList();
    ViewBag.Comments = comment.ToList();
    //bu da bizim commenti döndürmek için mekanizmamız
    return View(blog);
}
```

- **return View(blog);** Satırının Anlamı
- Bu satır, eylemin son çıktısını belirtir ve kodun kritik bir parçasıdır.
- **return View(blog);** Bu komut, kontrolcünün işini bitirdiğini ve bir View döndüreceğini söyler. Parantez içindeki blog nesnesi, bu View'a bir **model** olarak gönderilir.

- Details cshtml de bu

- Details cshtml de bu

```
public IActionResult About() ve public IActionResult
Contact()
```

- Bu metodlar, sadece ilgili sayfaları (About.cshtml ve Contact.cshtml) görüntülemek için kullanılır. Veritabanıyla herhangi bir etkileşimleri yoktur.

public IActionResult CreateContact(Contact model)

Bu metot, iletişim formundan gelen verileri kaydeder.

- **model.CreaatedAt = DateTime.Now;** Mesajın oluşturulduğu tarihi ve saati belirler.
- **_context.Contacts.Add(model);** Mesajı veritabanına eklenmek üzere hazırlar.
- **_context.SaveChanges();** Değişiklikleri veritabanına kaydeder.
- **return RedirectToAction("Index");** Mesaj gönderildikten sonra kullanıcıyı ana sayfaya yönlendirir.

public IActionResult Support()

Destek sayfasını gösterir. Veritabanı işlemi içermez.

public IActionResult Login()

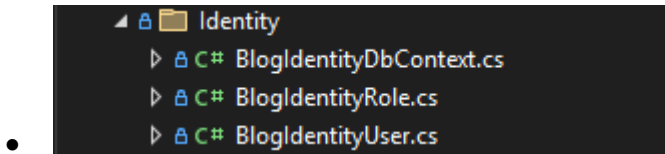
Kullanıcı giriş sayfasını gösterir.

[HttpPost] public async Task<IActionResult> Login(LoginViewModel model)

Bu metot, kullanıcı giriş bilgilerini kontrol eder ve oturum açma işlemini gerçekleştirir.

- **var user = await _userManager.FindByEmailAsync(model.Email);** Formdan gelen e-posta adresiyle eşleşen bir kullanıcıyı asenkron olarak bulur.
- **if(user == null):** Eğer kullanıcı bulunamazsa, tekrar giriş sayfasını gösterir.
- **var result = await _signInManager.PasswordSignInAsync(...):** Bulunan kullanıcının şifresini kontrol eder ve başarılıysa oturum açma işlemini gerçekleştirir.
- **if (result.Succeeded):** Giriş başarılı olursa, kullanıcıyı Admin denetleyicisindeki Index sayfasına yönlendirir.
- **else:** Giriş başarısız olursa, tekrar giriş sayfasını gösterir.

Identity



BlogIdentityDbContext

- `using Microsoft.AspNetCore.Identity.EntityFrameworkCore;` Bu satır, `IdentityDbContext` sınıfını kullanabilmek için gerekli olan kütüphaneyi içeri aktarıyor. Bu kütüphane, **Entity Framework Core** ile **ASP.NET Core Identity** arasındaki entegrasyonu sağlıyor.
- `using Microsoft.EntityFrameworkCore;` Bu satır, veritabanı işlemlerinin temelini oluşturan **Entity Framework Core** sınıflarını içeri aktarıyor.
- `public class BlogIdentityDbContext :`
`IdentityDbContext<BlogIdentityUser, BlogIdentityRole, string>` Bu satır, `BlogIdentityDbContext` adında bir sınıf tanımlıyor. Bu sınıf, `IdentityDbContext` sınıfından türediği için ASP.NET Core Identity'nin tüm işlevlerini miras alıyor.
- `<BlogIdentityUser, BlogIdentityRole, string>` Bu kısım, `IdentityDbContext`'in hangi türde kullanıcı, rol ve anahtar (primary key) kullanacağını belirliyor.
 - `BlogIdentityUser`: Uygulamanın **kullanıcı modeli** (Id, Email, PasswordHash vb. özellikleri içerir).
 - `BlogIdentityRole`: Uygulamanın **rol modeli** (Admin, User gibi rolleri tanımlar).
 - `string`: Kullanıcı ve rol nesnelerinin **benzersiz kimlik (ID) alanı** için kullanılacak veri tipini belirtiyor.
- `public`
`BlogIdentityDbContext(DbContextOptions<BlogIdentityDbContext> options):base(options) { ... }` Bu sınıfın **kurucu metodudur**.
- `DbContextOptions<BlogIdentityDbContext> options`: Bu parametre, veritabanı bağlantı bilgilerini ve diğer yapılandırma seçeneklerini içeriyor. Bu bilgiler genellikle uygulamanın `appsettings.json` dosyasından okunur.
- `:base(options)`: Gelen `options` parametresini, miras alınan `IdentityDbContext` sınıfının kurucu metoduna gönderiyor. Bu, ASP.NET Core Identity'nin veritabanına doğru şekilde bağlanmasını ve tabloları oluşturmalarını sağlıyor.
- **IdentityDbContext ile Ne Yaptın?**
- Normal bir `DbContext` sınıfı, senin belirlediğin modeller (Blog, Comment, vb.) için tablolar oluştururken, **IdentityDbContext** sınıfı standart veritabanı

tablolarına ek olarak, kullanıcı yönetimi için gerekli olan tabloları da otomatik olarak oluşturuyor. Bu tablolar şunları içerir:

- **AspNetUsers:** Kullanıcı bilgileri.
- **AspNetRoles:** Rol bilgileri.
- **AspNetUserRoles:** Hangi kullanıcının hangi role sahip olduğu bilgisi.
- **AspNetUserLogins:** Kullanıcıların oturum açma bilgileri (örneğin harici sağlayıcılar).
- **AspNetUserClaims:** Kullanıcılara ait ek bilgiler.
- Bu sayede, bu kod parçasıyla, kullanıcı kayıt, giriş, şifre yönetimi ve rol tabanlı yetkilendirme gibi karmaşık işlemleri kendi başına yapmana gerek kalmıyor; ASP.NET Core Identity'nin hazır altyapısını kullanabiliyorsun.

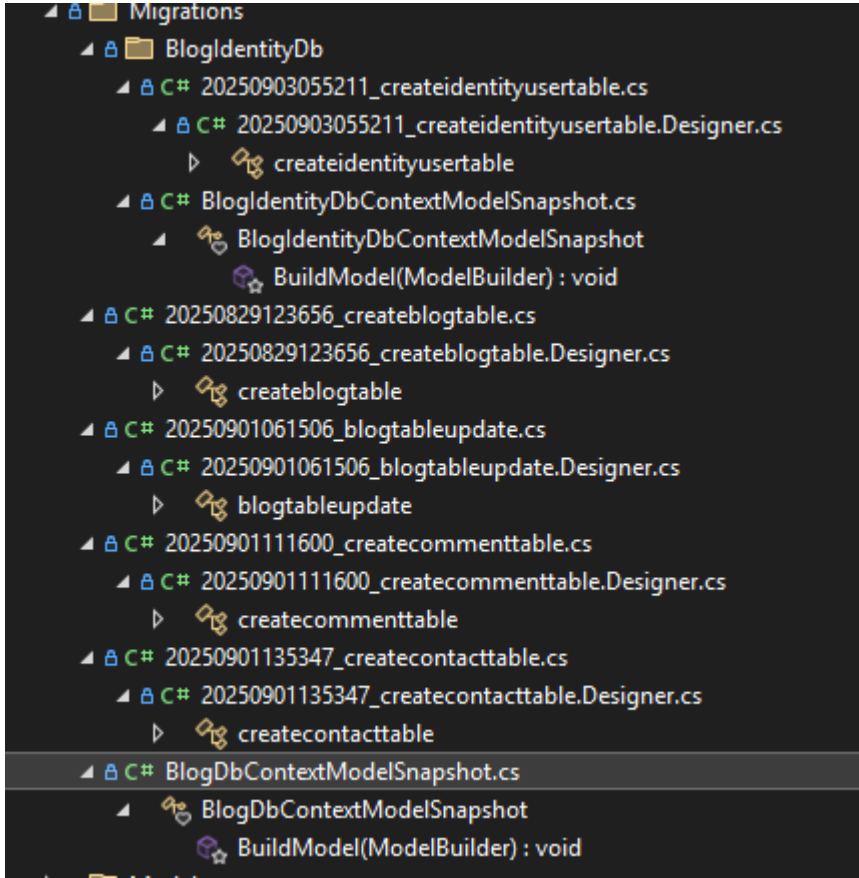
BlogIdentityRole

- Rol tanımı

BlogIdentityUser

- Name surname vs tanımlama

Migrations



- **Veritabanı Geçiş (Migration) Sistemi Detaylı Anlatım**

- Bu sistem, veritabanı yapınızdaki değişiklikleri yönetmenin modern ve hatasız bir yoludur. En basit ifadeyle, veritabanınızın bir nevi **tarihçesini** tutar.

- **1. Başlangıç Durumu**

- Başlangıçta veritabanınız yoktur. Sadece Blog, Comment, Contact gibi model sınıflarınız vardır. Bu modeller, veritabanınızın nasıl görünmesi gerektiği hakkında EF Core'a bilgi verir.

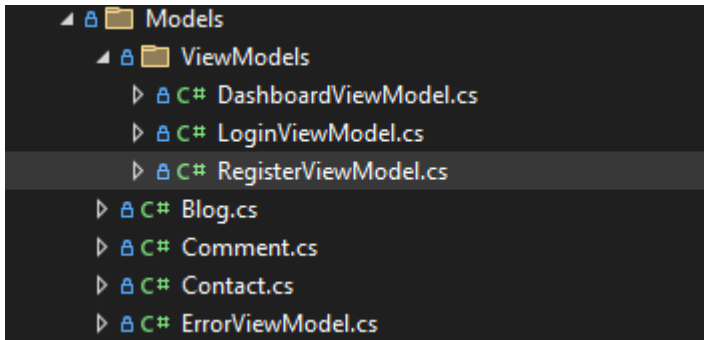
- **2. İlk Geçiş Oluşturma (Add-Migration)**

- Projenizde ilk kez Add-Migration komutunu çalıştırdığınızda (örneğin, Add-Migration CreateBlogTables), EF Core şunları yapar:
- **Modelleri tarar:** BlogDbContext içinde tanımladığınız tüm DbSet'leri inceler.
- **Geçiş dosyasını oluşturur:** Modellerinize bakarak, veritabanına bu tabloları (Blogs, Comments, Contacts) nasıl ekleyeceğini belirler. 2025..._CreateBlogTables.cs gibi yeni bir dosya oluşturur. Bu dosya, Up() ve Down() olmak üzere iki metot içerir:

- `Up()`: `CreateTable` komutlarını içerir. Bu komutlar, veritabanında yeni tabloları oluşturmak için kullanılır.
 - `Down()`: `DropTable` komutlarını içerir. Bu komutlar, değişiklikleri geri almak isterseniz tabloları siler.
- **Snapshot oluşturur:** Veritabanınızın o anki durumunun bir anlık görüntüsünü alır. Bu, sonraki geçişleri oluştururken temel olarak kullanılır.
- **3. Veritabanını Güncelleme (Update-Database)**
 - Bu komutu çalıştırdığınızda, EF Core şunları yapar:
 - **Bekleyen geçişleri kontrol eder:** Henüz veritabanına uygulanmamış geçiş dosyalarını bulur.
 - **Geçişleri uygular:** `Up()` metodunun içinde yer alan komutları sırasıyla veritabanında çalıştırır. Sonuç olarak, veritabanınızda `Blogs`, `Comments`, `Contacts` gibi tablolar oluşturulur.
 - **Geçiş kaydederek:** `__EFMigrationsHistory` adında özel bir tabloya bu geçişin uygulandığını kaydeder.
- **4. Mevcut Yapıya Değişiklik Yapma**
 - Diyelim ki `Blog` modelinize yeni bir `Category` sütunu eklemek istiyorsunuz.
 - **1. Adım (Kod Değişikliği):** `Blog.cs` model sınıfına `public string Category { get; set; }` gibi yeni bir özellik eklersiniz.
 - **2. Adım (Yeni Geçiş Oluşturma):** `Add-Migration AddCategoryToBlog` gibi yeni bir komut çalıştırırsınız. EF Core şunları yapar:
 - **Snapshot'u inceler:** Mevcut veritabanı yapınızın son durumuna bakar.
 - **Modeli inceler:** `Blog` modelindeki yeni `Category` özelliğini fark eder.
 - **Yeni geçiş dosyasını oluşturur:** Veritabanına bu sütunu eklemek için gereken `AddColumn` komutunu içeren bir geçiş dosyası oluşturur.
 - **3. Adım (Veritabanını Güncelleme):** `Update-Database` komutunu çalıştırırsınız. EF Core, veritabanında henüz uygulanmamış olan `AddCategoryToBlog` geçişini bulur ve `Blogs` tablosuna `Category` sütununu ekler.
- **Bu Sistemin Sağladığı Avantajlar**
 - **Sürüm Kontrolü:** Tıpkı `Git` gibi, veritabanı şemanızın tüm geçmişini kolayca görebilir ve yönetebilirsiniz.
 - **Kolay İşbirliği:** Bir ekip üyesi veritabanında bir değişiklik yaptığında, diğer üyeler `Update-Database` komutuyla kendi yerel veritabanlarını tek bir komutla en son sürüme güncelleyebilirler.

- **Ortamlar Arası Eşitlik:** Geliştirme, test ve canlı (prod) ortamlarındaki veritabanı yapılarının her zaman aynı olmasını sağlar.
- **Hata Kurtarma:** Bir geçiş işlemi sırasında hata oluşursa, `Down()` metodu sayesinde önceki stabil duruma kolayca dönebilirsiniz.
- Bu sistem, veritabanı yönetimini manuel SQL komutlarından kurtararak geliştirme sürecini çok daha güvenli ve pratik hale getirir.

Models



Modeller (Models Klasörü)

- Modeller, uygulamanızın **veritabanı tablolarına karşılık gelen** sınıflardır. Bunlara aynı zamanda **veri modelleri** veya **varlıklar (entities)** da denir.
- **Blog.cs:** Veritabanındaki **Blogs** tablosunu temsil eder. `Id`, `Title`, `Content`, `ViewCount` gibi özellikler içerir.
- **Comment.cs:** Veritabanındaki **Comments** tablosunu temsil eder. Yorumun içeriği, yazarı ve hangi bloga ait olduğu gibi bilgileri içerir.
- **Contact.cs:** Veritabanındaki **Contacts** tablosunu temsil eder. İletişim formundan gelen ad, e-posta ve mesaj gibi verileri tutar.
- **ErrorViewModel.cs:** Hata sayfalarını yönetmek için kullanılan bir modeldir.
- Bu model sınıfları, Entity Framework Core tarafından veritabanı tablolarına dönüştürülür ve veritabanı işlemlerini nesne yönelimli bir şekilde yapmanızı sağlar.

•

Görünüm Modelleri (ViewModels Klasörü)

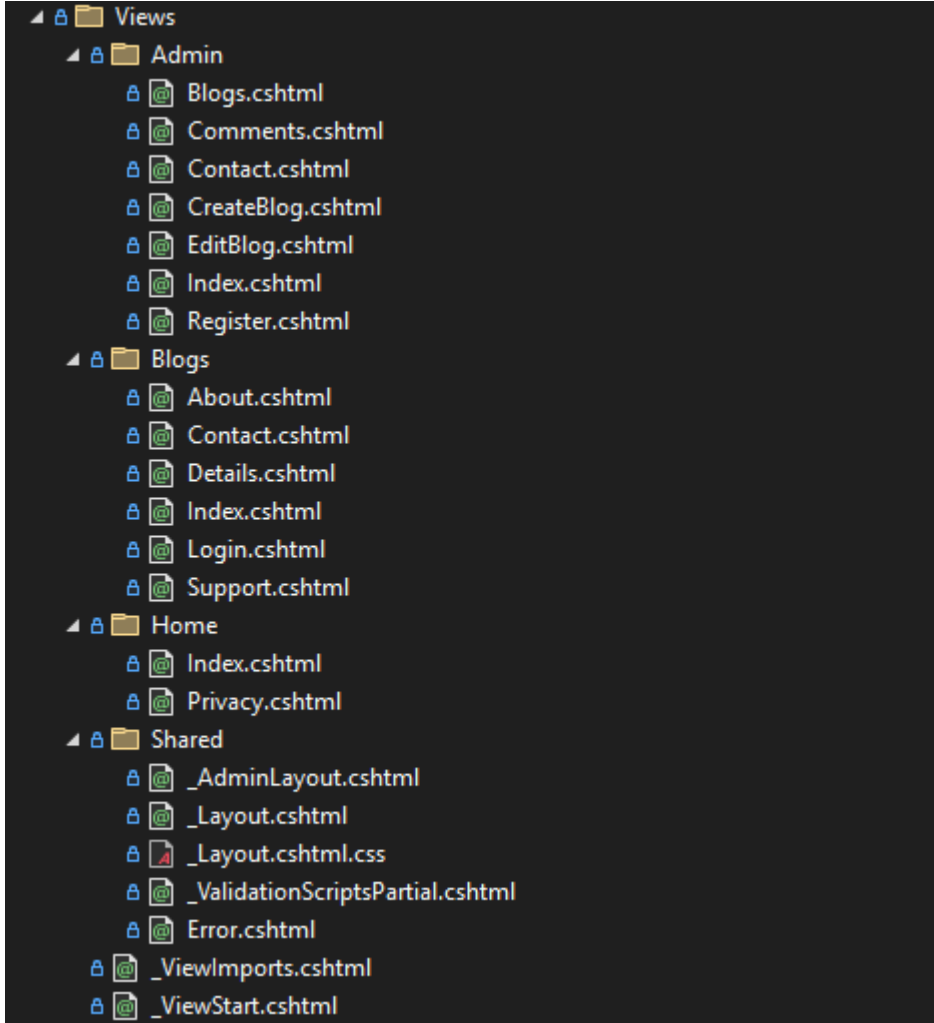
- Görünüm modelleri, **belirli bir sayfa (view) için özel olarak oluşturulmuş** sınıflardır. Veritabanı modellerinden farklı olarak, doğrudan veritabanı tablolarına karşılık gelmezler.

- **DashboardViewModel.cs:** Admin paneli ana sayfası (Index.cshtml) için oluşturulmuştur. Bu model, tek bir blog nesnesini değil, toplam blog sayısı, toplam görüntülenme sayısı gibi farklı verileri bir araya getirir.
- **LoginViewModel.cs:** Giriş sayfası için kullanılır. Genellikle Email ve Password gibi özellikleri içerir. Veritabanındaki User modeliyle birebir aynı olmak zorunda değildir, yalnızca giriş formu için gereken bilgileri tutar.
- **RegisterViewModel.cs:** Kayıt sayfası için kullanılır. Name, Surname, Email, Password ve RePassword gibi bilgileri bir arada tutar.

- **Model ve ViewModel Arasındaki Fark**

- **Model:** Veritabanı yapısını yansıtan, doğrudan veritabanı tablolarına karşılık gelen sınıflardır.
- **ViewModel:** Belirli bir sayfanın (view) ihtiyacı olan verileri taşımak için oluşturulan sınıflardır. Bu veriler birden fazla modelden, başka bir kaynaktan veya sadece o sayfaya özel olarak toplanabilir.
- Bu ayrım, uygulamanızın daha düzenli ve güvenli olmasını sağlar. Doğrudan veritabanı modellerini sayfaya göndermek yerine, sadece sayfanın ihtiyaç duyduğu verileri içeren bir ViewModel kullanmak **veri sızıntısını engeller** ve performansı artırır.

Views



MVC Mimarisinde Views Klasörünün Yeri

- MVC (Model-View-Controller) mimarisinde:
- **Model** (veri)
- **Controller** (iş mantığı)
- **View** (kullanıcı arayüzü)
- olarak üç ana bileşen bulunur. Views klasörü, bu mimarideki **View** bileşenine karşılık gelir. Bir Controller (örneğin BlogsController), iş mantığını tamamladıktan sonra bir View'e veri gönderir. View, bu veriyi alır ve bir web sayfası (.cshtml dosyası) olarak HTML çıktısı üretir.
-
- **Klasörlerin ve Dosyaların Amacı**
- Views klasörü, projenin düzenini korumak için genellikle denetleyici (controller) isimlerine göre alt klasörlere ayrılır.

- **Admin Klasörü**

- Bu klasör, **AdminController**'in kullandığı tüm sayfaları (view) içerir.
- `Blogs.cshtml`: Admin panelindeki blog listesi sayfası.
- `Comments.cshtml`: Yorumları yönetme sayfası.
- `Contact.cshtml`: İletişim mesajlarını yönetme sayfası.
- `CreateBlog.cshtml`: Yeni blog yazısı oluşturma formu.
- `EditBlog.cshtml`: Var olan bir blog yazısını düzenleme formu.
- `Index.cshtml`: Admin paneli ana sayfası (genellikle dashboard).
- `Register.cshtml`: Admin kullanıcı kaydı sayfası.

- **Blogs Klasörü**

- Bu klasör, **BlogsController**'in kullandığı herkese açık (public) sayfaları içerir.
- `About.cshtml`: Hakkımızda sayfası.
- `Contact.cshtml`: İletişim formu.
- `Details.cshtml`: Blog yazısının detay sayfası.
- `Index.cshtml`: Blog ana sayfası (yayınlanmış blogların listesi).
- `Login.cshtml`: Kullanıcı giriş sayfası.
- `Support.cshtml`: Destek sayfası.

- **Home Klasörü**

- **HomeController** tarafından kullanılan sayfaları içerir.
- `Index.cshtml`: Uygulamanın ana giriş sayfası.
- `Privacy.cshtml`: Gizlilik politikası sayfası.

- **Shared Klasörü**

- Bu klasör, birden fazla sayfada ortak olarak kullanılan dosyaları içerir.
- `_AdminLayout.cshtml` ve `_Layout.cshtml`: Uygulamanın farklı kısımları için ana sayfa düzenleridir. Header, footer, menü gibi ortak HTML yapılarını içerirler.
- `_Layout.cshtml.css`: `_Layout` dosyasına ait CSS stil dosyası.
- `_ValidationScriptsPartial.cshtml`: Formlarda client-side doğrulama (JavaScript ile) yapmak için gerekli scriptleri içerir.
- `Error.cshtml`: Uygulamada bir hata oluştuğunda gösterilen sayfa.
- `_ViewImports.cshtml`: View'lere otomatik olarak dahil edilecek using direktiflerini içerir.
- `_ViewStart.cshtml`: Tüm view'lerin başında çalışır ve hangi `_Layout` dosyasının kullanılacağını belirtir.

- Her bir `.cshtml` dosyası, HTML kodunun yanı sıra C# kodunu da içerebilir. Bu teknolojiye **Razor** adı verilir ve dinamik web sayfaları oluşturmayı mümkün kılar. Örneğin, bir `.cshtml` dosyasında, bir **@foreach** döngüsü kullanarak bir blog listesindeki her bir blogu ekrana basabilirsiniz.

Appsetting.json

- Bu dosya, uygulamanın çalışırken ihtiyaç duyduğu ayarları ve yapılandırma bilgilerini saklamak için kullanılır.
- `appsettings.json` dosyası, uygulamanın kodundan bağımsız olarak ayarları kolayca değiştirmenizi sağlar. Böylece, veritabanı bağlantı adresi, loglama seviyesi veya API anahtarları gibi bilgileri, kodunuzu yeniden derlemeye gerek kalmadan farklı ortamlara (geliştirme, test, canlı sunucu) göre ayarlayabilirsiniz.

•

İçeriğin Açıklaması

- **"ConnectionStrings"**: Bu bölüm, uygulamanın veritabanına bağlanmak için kullandığı bağlantı dizelerini (connection strings) içerir.
 - **"DefaultConnection"**: Bu, bağlantı dizesine verdiğiniz bir isimdir. Kodunuzda bu ismi kullanarak veritabanına kolayca bağlanabilirsiniz.
 - **Data Source =NB3\\SQLEXPRESS**: Bağlanılacak veritabanı sunucusunun adını belirtir.
 - **database=blogV1**: Bağlanılacak veritabanının adını belirtir.
 - **Integrated Security=True**: Windows kimlik doğrulamasını kullanarak veritabanına bağlanmayı sağlar. Yani bir kullanıcı adı ve şifreye gerek kalmaz.
 - **TrustServerCertificate=True**: Sunucu sertifikasına güvenileceğini belirtir, bu genellikle yerel geliştirme ortamlarında kullanılır.
- **"Logging"**: Bu bölüm, uygulamanın loglama ayarlarını kontrol eder. Loglama, uygulamanın çalışma zamanındaki olayları (bilgilendirme, uyarı, hata gibi) kaydetme işlemidir.
 - **"LogLevel"**: Logların hangi seviyede kaydedileceğini belirler.
 - **"Default": "Information"**: Uygulamanın genel olarak Information seviyesindeki logları ve daha yüksek seviyeleri (Warning, Error, vb.) kaydetmesini sağlar.
 - **"Microsoft.AspNetCore": "Warning"**: ASP.NET Core altyapısıyla ilgili logların sadece Warning seviyesinde ve daha yüksek seviyelerde kaydedilmesini sağlar. Bu, gereksiz bilgilendirme loglarının önüne geçerek log dosyasını daha temiz tutar.

- **"AllowedHosts"**: Bu bölüm, uygulamanın hangi sunucu adlarına (host) gelen isteklere yanıt vereceğini belirler.
 - **"*"**: Uygulamanın gelen tüm sunucu adlarına yanıt vermesini sağlar. Bu, geliştirme aşamasında yaygın olarak kullanılır. Canlı ortamda ise güvenlik için belirli bir alan adı veya IP adresi belirtilir.

Program.cs

- **İlk Bölüm: Servisleri Tanımlama**
- Bu kısım, uygulamanızın hangi araçları ve hizmetleri kullanacağını söylediğiniz yerdir.
- **var builder = WebApplication.CreateBuilder(args);**
 - Uygulamayı başlatmak için gereken bir "yapıcı" (builder) oluşturur. Bu, tüm ayarların toplandığı merkezdir.
- **builder.Services.AddControllersWithViews();**
 - Uygulamanın **MVC (Model-View-Controller)** mimarisini kullanacağını belirtir. Yani, denetleyiciler ve sayfalar (.cshtml) arasında bir bağ kurulur.
- **builder.Services.AddDbContext<BlogDbContext>();**
 - **BlogDbContext** sınıfını bir hizmet olarak tanımlar. Bu sayede, Controller içinde `_context` gibi bir değişkenle veritabanına kolayca erişim sağlarsın.
- **builder.Services.AddDbContext<BlogIdentityDbContext>(options => { ... });**
 - **ASP.NET Core Identity** için özel bir veritabanı bağlantısı kurar.
 - `{ ... }` içindeki kod, **appsettings.json** dosyasından "DefaultConnection" adlı bağlantı dizesini bulur ve bu bilgiyi kullanarak **BlogIdentityDbContext**'i bir SQL Server veritabanına bağlar.
- **builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme).AddCookie(options => { ... });**
 - Uygulamada **kimlik doğrulama** (authentication) servisini başlatır.
 - Oturumları yönetmek için **çerezleri (cookie)** kullanacağını belirtir.
 - **options.LoginPath = "Blogs/Index";**: Eğer bir kullanıcı yetkisi olmayan bir sayfaya erişmeye çalışırsa, onu Blogs/Index adresine yönlendirir.
- **builder.Services.AddIdentity<BlogIdentityUser, BlogIdentityRole>().AddEntityFrameworkStores<BlogIdentityDbContext>().AddDefaultTokenProviders();**

- **Identity** sisteminin kendisini yapılandırır.
- Hangi kullanıcı ve rol sınıflarını (`BlogIdentityUser`, `BlogIdentityRole`) kullanacağını belirtir.
- Kullanıcı ve rol verilerini yönetmek için **Entity Framework Core**'u kullanacağını söyler.
-
- **İkinci Bölüm: Uygulamanın Davranışlarını Tanımlama**
- Bu kısım, bir web isteği geldiğinde uygulamanın nasıl davranacağını adım adım belirler.
- **var app = builder.Build();**
 - Tanımlanan tüm servisler ve ayarlar ile birlikte çalışan bir uygulama nesnesi (`app`) oluşturur.
- **if (!app.Environment.IsDevelopment()) { ... }**
 - Eğer uygulama "geliştirme ortamında" değilse (yani canlı sunucudaysa), özel ayarların devreye girmesini sağlar.
 - **app.UseExceptionHandler("/Home/Error");** Canlı ortamda bir hata olursa, kullanıcıya genel bir hata sayfası gösterir.
 - **app.UseHsts();** Tarayıcıların siteye her zaman güvenli (HTTPS) bağlantı üzerinden erişmesini zorunlu kılar.
- **app.UseHttpsRedirection();**
 - Gelen tüm HTTP isteklerini otomatik olarak güvenli HTTPS isteklerine dönüştürür.
- **app.UseStaticFiles();**
 - CSS, JavaScript, resimler gibi statik dosyaların tarayıcıya sunulmasını sağlar.
- **app.UseRouting();**
 - Gelen isteğin URL adresine bakarak, hangi `Controller` metodunun çalışacağını belirlemeye hazırlanır.
- **app.UseAuthentication();**
 - Kullanıcının kimliğini doğrular. Örneğin, çerezlere bakarak kullanıcının kim olduğunu anlar.
- **app.UseAuthorization();**
 - Kullanıcının belirli bir sayfaya veya fonksiyona erişim yetkisinin olup olmadığını kontrol eder.
- **app.MapControllerRoute(...)**
 - URL adreslerinin nasıl okunacağını belirleyen bir rota (route) kuralı tanımlar.
 - **pattern: "{controller=Blogs}/{action=Index}/{id?}":** Varsayılan URL yapısının `site.com/Blogs/Index` gibi olacağını belirtir.

Eğer URL'de bir şey yazılmazsa, otomatik olarak Blogs denetleyicisindeki Index metodunu çalıştırır.

- **app.Run();**
 - Uygulamayı başlatır ve gelen web isteklerini dinlemeye başlar. Uygulamanın çalışması bu komutla başlar.

ÖZET

- Bizim elimizde temelinde 3 bileşen var model view controller.
- Önce modelde orda hangi değişkenleri kullanıcaksak onları tanımlarız. Bunlar aslında tabloların bir görünümüdür.
- Sonrasında biz backend tarafını yaparız controllerda yapılır. Blog için admin ve blog diye iki ayrı controller yapılır
- Sonrasında biz bu controllerda yazdıklarımızı model yapısını kullanarak view üzerinden görünümünü veririz.
- Görüntü kısmında her bir methodun ayrı bir görüntüsü ve klasik blog ve admin sayfası için görüntüler vardır. Bu görüntüleri internetten şemalar indirerek sağlayıp projeye gereken yerlerini dahil ederiz.

