



**T.C.**  
**ADANA ALPARSLAN TÜRKEŞ BİLİM VE**  
**TEKNOLOJİ ÜNİVERSİTESİ**  
**BİLGİSAYAR VE BİLİŞİM FAKÜLTESİ**  
**BİLGİSAYAR MÜHENDİSLİĞİ**

**STAJ RAPORU**

**Öğrenci Adı ve Soyadı : Anıl Cem Elemir**

## İçindekiler

1.	ADANA BÜYÜKŞEHİR BELEDİYESİ YAZILIM DEPARTMANI .....	1
2.	GİRİŞ .....	2
3.	RAPOR.....	3
3.1	N Katmanlı Mimari Taslağı.....	3
3.2	EntityLayer (Varlık Katmanı) .....	4
3.3	DataAccessLayer (Veri Erişim Katmanı) .....	6
3.4	CoreLayer (Çekirdek Katmanı).....	8
3.5	BusinessLayer(İş Mantığı Katmanı).....	9
3.6	UserInterface Katmanı (StajBlogSitesi) .....	11
4.	SONUÇ .....	28

## Şekiller

Şekil 1.1. Adana Büyükşehir Belediyesinin Logosu

Şekil 1. Adana Büyükşehir Belediyesinin Logosu ..... iii

Şekil 1. Adana Büyükşehir Belediyesinin Logosu ..... 1

Şekil 3.2.1 EntityLayer İçindeki Modeller

Şekil 3.2.2 Database Migration'ları.....

Şekil 3.2.3 Database Şeması.....

Şekil 3.3.1 DataAccessLayer Genel Bakış.....

Şekil 3.5.1 DataAccessLayer Genel Bakış.....

Şekil 3.5.2 Validasyon Sınıfları.....

Şekil 3.6.1 wwwroot Klasörü.....

Şekil 3.6.2 Hakkımızda Kısmı

Şekil 3.6.3 Bir Bloğun Detayları

Şekil 3.6.4 Yazarın Yazdığı Blogların Listesi

Şekil 3.6.5 Blog Ekleme Kısmı

Şekil 3.6.6 Blog Düzenleme Kısmı

Şekil 3.6.7 Yorum Kısmı

Şekil 3.6.8 İletişim Kısmı

Şekil 3.6.9 Yazar Profil Güncelleme Kısmı

Şekil 3.6.10 Mail Bültenine Abone Olma Kısmı

Şekil 3.6.11 Siteye Kayıt Olma Kısmı

Şekil 3.6.12 Siteye Giriş Yapma Kısmı

Şekil 3.6.13 404 Hata Sayfası

Şekil 3.6.14 Dashboard Ana Sayfa

Şekil 3.6.15 Tüm Bildirimlerin Listelendiği Kısım

Şekil 3.6.16 Bildirim Pop-Up'ı

Şekil 3.6.17 Mesaj Pop-Up'ı

## 1. ADANA BÜYÜKŞEHİR BELEDİYESİ YAZILIM DEPARTMANI

Staj yerim olan Adana Büyükşehir Belediyesi, Türkiye'nin en büyük ve köklü belediyelerinden biridir. Adana'nın tüm belediyeçilik hizmetlerini koordine eden bu kurum, kentsel gelişimden sosyal hizmetlere, altyapı projelerinden kültürel etkinliklere kadar geniş bir yelpazede hizmet sunmaktadır. Bu kapsamda, belediyenin dijital dönüşüm ve yenilikçi teknolojilerle donatılması, şehirde yaşayan vatandaşlara daha verimli hizmetler sunulması amacıyla, Yazılım Departmanı kilit bir rol üstlenmektedir.

Yazılım Departmanı, belediyeye bağlı tüm dijital platformların geliştirilmesi, yönetimi ve güncellenmesiyle ilgilenmektedir. Bu departmanda, modern yazılım geliştirme metodolojileri uygulanarak, vatandaşların aktif olarak kullandığı mobil ve web tabanlı uygulamalar geliştirilmektedir. Örneğin, belediyenin çeşitli web modülleri .NET framework, Java ve PHP gibi teknolojilerle kodlanmakta, mobil uygulamalar ise iOS için Swift, çok platformlu çözümler için Flutter kullanılarak geliştirilmektedir. Bu sayede, Adana halkına daha hızlı, güvenilir ve kullanıcı dostu dijital hizmetler sunulmaktadır.

Adana Büyükşehir Belediyesi Yazılım Departmanı, her yıl onlarca lise ve üniversite öğrencisini stajyer olarak kabul ederek, onlara gerçek dünyada uygulamalı deneyimler kazandırmaktadır. Bu staj programı, öğrencilere teorik bilgilerini pratikte uygulama şansı sunarken, aynı zamanda profesyonel bir iş ortamında çalışma deneyimi kazandırmaktadır.



Şekil 1.1. Adana Büyükşehir Belediyesinin Logosu

## 2. GİRİŞ

Bu rapor, Adana Büyükşehir Belediyesi Yazılım Departmanı'nda gerçekleştirdiğim staj sürecinde edindiğim deneyimleri ve öğrendiklerimi kapsamaktadır. Stajımın ilk gününde, işyerine gittiğimde bana neler ile ilgilendiğim ve ne üzerinde ilerlemek istediğim soruldu. Ben de .NET Framework üzerinden bir proje yapmak istediğimi belirttim. Ancak, bana .NET Core üzerinden ilerlememin daha uygun olacağı söylendi. Bunun sebepleri arasında .NET Core'un daha modern, platform bağımsız olması ve performans açısından daha verimli olması yer alıyordu. Ayrıca, .NET konusunda oldukça hakim mühendisler tarafından bana yardımcı olunabileceği belirtildi ve ben de bu teklifi kabul ettim.

Benden, staj süresince çalışacağım 20 iş gününü dolu dolu geçireceğim bir blog sitesi yapmam istendi. İlk haftada iş sağlığı ve güvenliği uzmanından bir konferans aldım. Bu konferans, işyerinde güvenli çalışma yöntemleri ve olası tehlikelerden nasıl kaçınılacağı konusunda önemli bilgiler sundu.

Daha sonra, bana n katmanlı mimari (n-tier architecture) öğretildi ve istenilen blog sitesi için gereken veritabanının nasıl oluşturulacağı anlatıldı. Bu proje kapsamında, veritabanı yönetimi, kullanıcı arayüzü tasarımı ve sunucu tarafı programlama gibi konularda çalışmalar yaptım. Ayrıca, profesyonel bir iş ortamında, işyerinin gereken kurallarına uygun nasıl davranılması gerektiğini ve karşılaşılan sorunların nasıl çözüldüğünü deneyimledim.

Staj sürecimde, bana belirli bir proje verildi ve bu proje üzerinde çalışırken departman çalışanlarının teknik bilgi ve tecrübelerinden yararlandım. Karşılaştığım sorunlarla ilgilenildi ve nasıl kaçınmam gerektiği söylendi. Proje süreci boyunca aldığım destek sayesinde, yazılım geliştirme sürecinin her aşamasını öğrenme ve uygulama fırsatım oldu. Proje tamamlandığında, çalışmamı sunarak geri bildirim aldım ve bu süreç, mesleki becerilerimi geliştirmemde önemli bir rol oynadı.

Birebir içinde olmasam bile, Java ile geliştirilen bir belediye modülünün yüksek mühendisler tarafından tasarım sürecine şahit oldum ve bu sürecin nasıl ilerlediğini gözlemledim. Birebir son kullanıcılar ile temasa geçildiğini ve genel son kullanıcı isteklerinin ne olduğunu, son kullanıcıların isteklerine nasıl bir karşı aksiyon alındığını, neler uygulandığını gördüm.

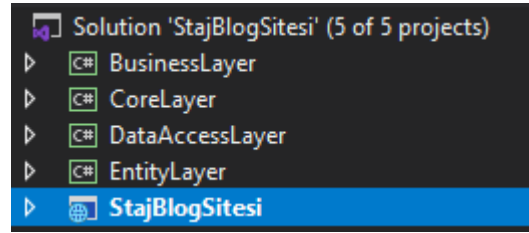
Sonuç olarak, bana çok önemli ölçüde katkı sağladığımı ve ileride işime yarayacak birikimler edindiğimi düşündüğüm bir deneyim oldu.

### 3. RAPOR

Staj dönemimin başında, Adana Büyükşehir Belediyesi Yazılım Departmanı'ndaki yetkili mühendis ile bir araya gelerek, staj sürecim boyunca nasıl bir proje yapacağımı ve hangi teknolojilerle çalışacağımı belirledik. Yapılan görüşmelerde, benim yazılım geliştirme alanındaki ilgi ve tecrübelerim göz önünde bulundurularak, hangi platform ve araçlarla çalışacağımıza birlikte karar verdik. Daha önce .NET Framework ile çeşitli web sitesi projeleri geliştirmiş olmam nedeniyle, yetkili mühendis bana .NET Core teknolojisini kullanmamı önerdi. .NET Core'un modern yazılım geliştirme süreçlerine daha uygun, çapraz platform desteği sunan, performans açısından daha verimli ve sürekli güncellenen bir teknoloji olması nedeniyle bu kararı aldık.

#### 3.1 N Katmanlı Mimari Taslağı

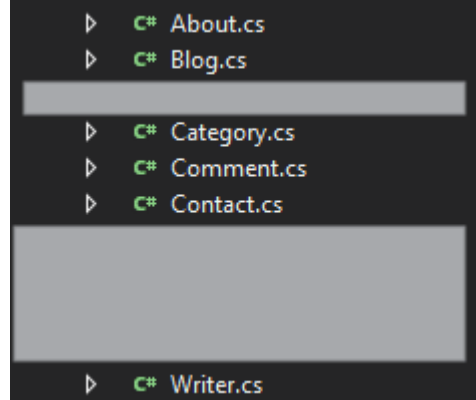
Benden sorumlu mühendisin yönlendirmesi dolayısıyla, daha önce yaptığım dağınık kodlama tarzımı bir kenara bırakıp, bana öğretilen N katmanlı mimariyle blog sistemi tasarlamaya başladım. Bu mimari yaklaşımda, projemi beş katmana ayırarak geliştirdim: BusinessLayer, CoreLayer, EntityLayer, DataAccessLayer ve UI Katmanı.



Şekil 3.1.1 Proje Katman İsimleri

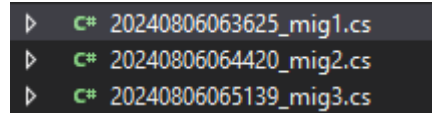
### 3.2 EntityLayer (Varlık Katmanı)

İlk olarak, EntityLayer'da projedeki veritabanı tablolarını ve bu tablolara karşılık gelen sınıfları tanımladım. Bu katman, veritabanı ile uygulamanın diğer katmanları arasındaki ilişkiyi kuran temel sınıfları içeriyordu. Ayrıca database'imi benim için de bir ilk olacak olan Code-First yöntemiyle oluşturacağım için ilk aşamada gereken tüm entity'lerimi buraya tanımladım. Bu entity'ler de About, Blog, Category, Comment, Contact ve Writer entityleriydi.



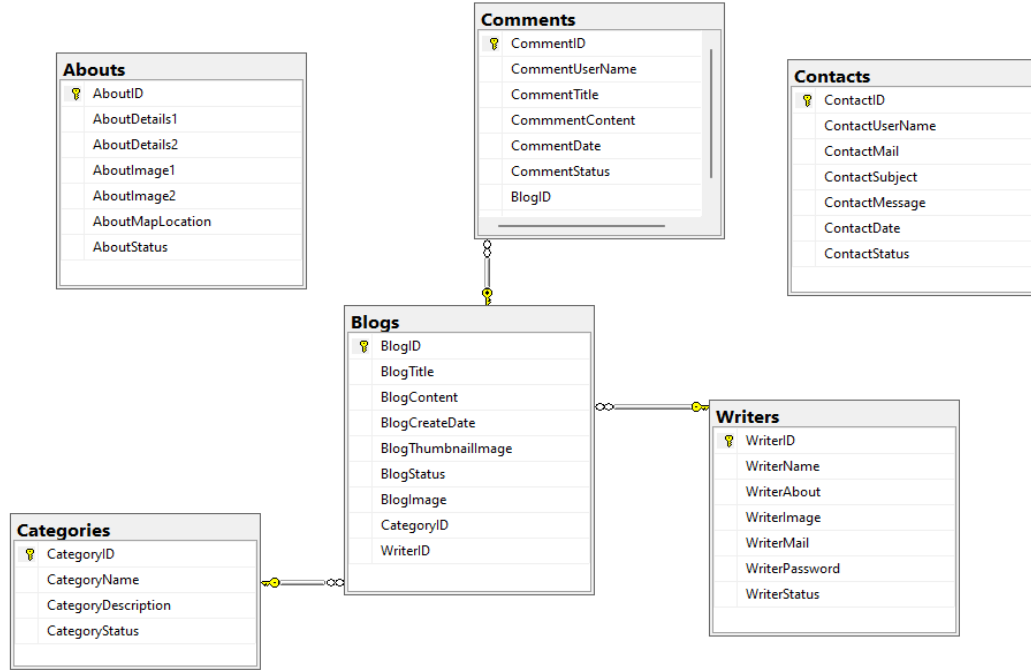
Şekil 3.2.1 EntityLayer İçindeki Modeller

Ardından, package manager üzerinden migration oluşturup database oluşturulmasını gerçekleştirdim.



Şekil 3.2.2 Database Migration'ları

İlk Etapta oluşturulan database ise şu şekildeydi;



Şekil 3.2.3 Database Şeması



### 3.3 DataAccessLayer (Veri Eriřim Katmanı)

Sonrasında, mimari tasarımda DataAccessLayer'da veritabanı işlemlerini yöneterek, uygulamanın veri katmanını izole ettim. Bu katmanda, Entity Framework Core kullanarak veritabanına erişim, CRUD (Create, Read, Update, Delete) işlemleri ve kompleks sorguların yazılması gibi işlemleri gerçekleřtirdim. DataAccessLayer, bu işlemleri izole ederek, diğerkatmanların doğrudan veritabanı ile etkileşimini engelleyip, daha modüler ve bakım yapılabilir bir yapı sağladı.

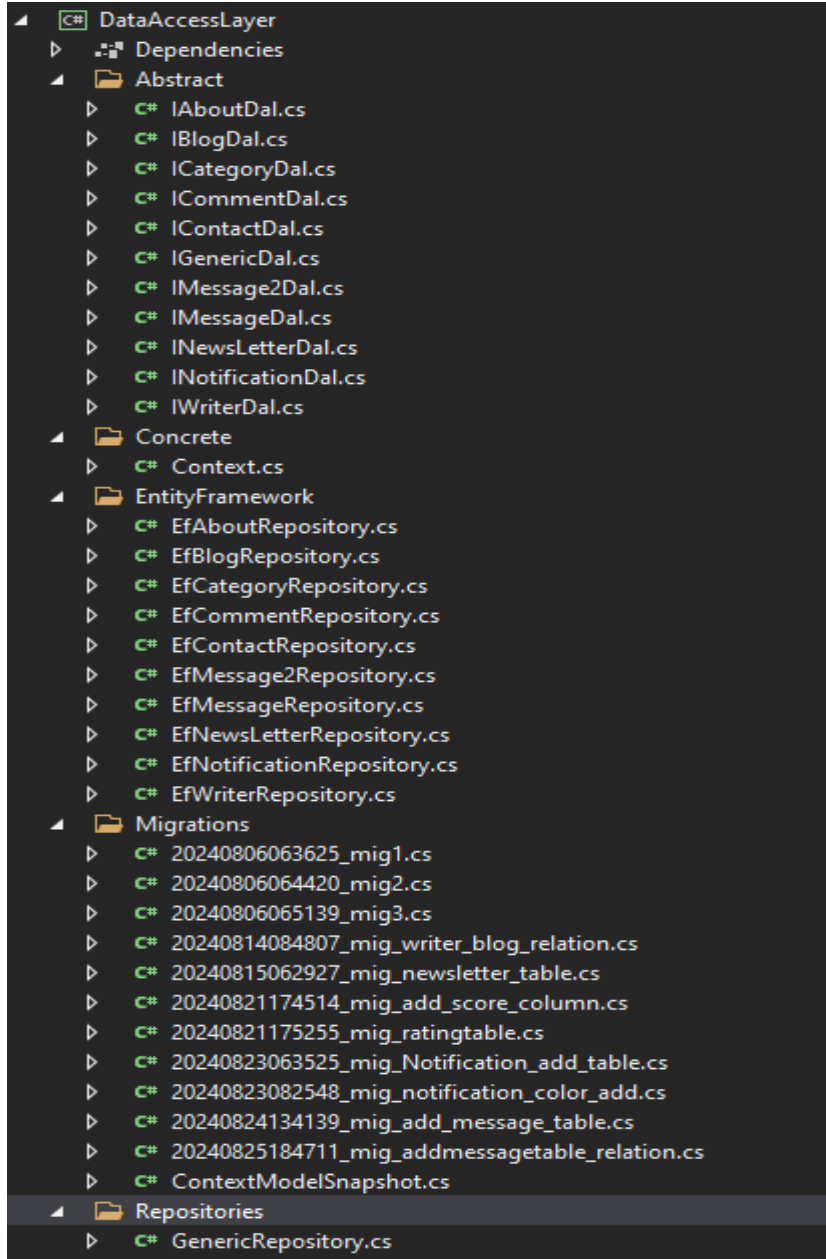
DataAccessLayer, belirli bir düzen içinde yapılandırıldı. Bu katman abstract ve concrete klasörlerine ayrıldı. Abstract klasöründe, her tabloya karşılık gelen I.....Dal isimli interface sınıfları bulunuyordu. Bu interface sınıfları, tüm tabloların ortak kullanacağı yöntemleri tanımlayan IGenericDal isimli bir interface'ten türetilmişti. IGenericDal, temel CRUD işlemleri gibi sık kullanılan yöntemlerin şablonlarını sundu. Bu sayede, her tablo için bu yöntemler tekrar yazılmak zorunda kalmadı.

Concrete klasöründe yer alan GenericRepository, IGenericDal interface'inde tanımlanan bu yöntemlerin gövdelerinin (body) yazıldığı yerdı. Bu yapı, tüm tablolar için tekrar eden kodların önüne geçerek, kodun bakımını ve okunabilirliğini artırdı. Ayrıca, her tabloya özgü ek yöntemler de ilgili Repository sınıfına eklenebildi, bu sayede esneklik sağlanırken, DRY (Don't Repeat Yourself) prensibi uygulanmış oldu. Repository sınıflarında, DbContext'e erişirken using yapısını kullandık; bu sayede, kaynak sızıntılarını önledik ve oluşturulan nesnelerin bellekten otomatik olarak serbest bırakılmasını sağladık. Dispose işlemi, bellek yönetimini daha etkili hale getirdi.

Concrete klasöründe ayrıca uygulamanın veritabanı bağlamını yöneten context sınıfı bulunuyordu. Bu sınıf, veritabanı bağlantı dizesini (connection string) ve her tablo için DbSet tanımlarını içeriyordu. Veritabanının oluşturulması ve yönetimi de bu context üzerinden yapıldı.

EntityFramework klasörü ise migrations işlemlerini yönetti ve veritabanındaki tablolar için Ef...Repository isimli sınıfları içeriyordu. Bu sınıflar, tablolarla ilgili CRUD işlemlerini kapsadı ve Entity Framework'ün sunduğu özellikleri kullanarak veri erişimini kolaylařtırdı. Ayrıca, Repositories klasöründen GenericRepository yapısı kullanılarak veri erişim işlemleri daha esnek ve yeniden kullanılabilir hale getirildi.

Bu yapı sayesinde, DataAccessLayer, veritabanı işlemlerini merkezi bir yerden yöneterek, uygulamanın farklı katmanlarında tekrar eden kod yazımını önledi, modülerlik ve bakım kolaylığı sağladı.



Şekil 3.3.1 DataAccessLayer Genel Bakış

### 3.4 CoreLayer (Çekirdek Katmanı)

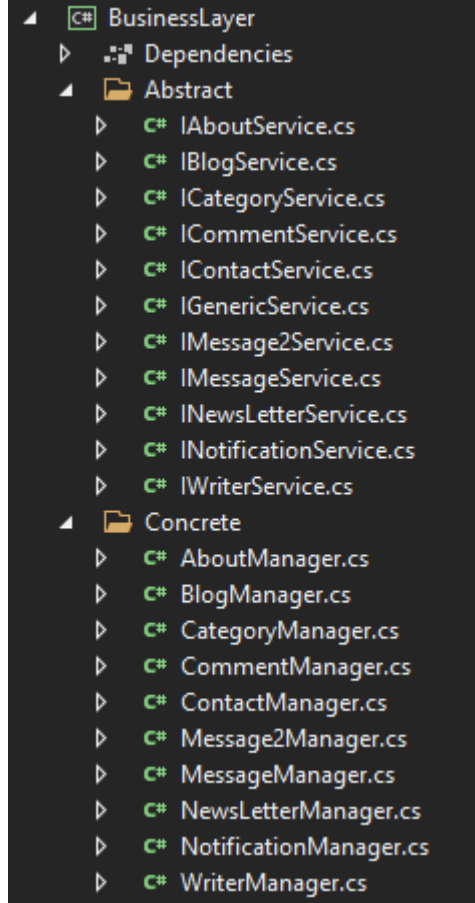
CoreLayer'ın yapısı kısaca özetlendi ve bu katmanla ilgili daha fazla detaya inmem için sürem yetmediği belirtildi. Aynı zamanda, CoreLayer hakkında yeterli bilgiye sahip olmadığım için bu bölüme daha fazla zaman ayırmam gerektiği vurgulandı. CoreLayer, projenin temel altyapısını oluşturur ve sıkça kullanılan yardımcı sınıfları, sabitleri, genişletme metotlarını (extension methods), ve genel uygulama hizmetlerini (common services) içerir. Bu katman, diğer katmanlar tarafından yaygın olarak kullanılan işlevleri sağlar, böylece kodun yeniden kullanılabilirliği artar ve projenin daha modüler bir yapıya kavuşması sağlanır.

Bu nedenle, CoreLayer'ın önemi ve yapısal özellikleri üzerine derinlemesine bilgi edinmem gerektiği dile getirildi. Bu kapsamda, daha sonra araştırmam için bana şu konular önerildi: Bağımlılık Enjeksiyonu (Dependency Injection), Yardımcı Sınıflar (Utility Classes), Genişletme Metotları (Extension Methods), ve Ortak Arayüzler (Common Interfaces). Bu konular, CoreLayer'ın işleyişini anlamak ve daha etkin bir şekilde uygulayabilmek için kritik öneme sahip.

Bu konuları detaylı bir şekilde araştırarak, CoreLayer'ı daha iyi anlayacak ve ilerleyen aşamalarda bu yapıyı projeye entegre etmek için çalışmalarına devam edeceğim. Bu süreçte, CoreLayer'ın projenin geneline nasıl katkı sağladığını ve diğer katmanlarla olan etkileşimini daha net bir şekilde kavrayarak, uygulamanın genel mimarisini güçlendirmeyi hedefliyorum.

### 3.5 BusinessLayer(İş Mantığı Katmanı)

Business katmanımızda, projenin iş mantığını yöneten yapıların oluşturulduğu bir düzen bulunur. Bu katman, diğer tüm katmanlar gibi abstract (soyut) ve concrete (somut) olarak iki ana klasöre ayrılmıştır.

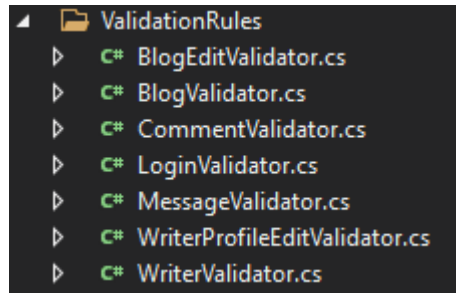


Şekil 3.5.1 DataAccessLayer Genel Bakış

Abstract klasöründe, projedeki servislerin (services) tanımlandığı interface yapıları bulunur. Bu interface yapıları, DRY (Don't Repeat Yourself) felsefesine uygun olarak, tekrar eden kodların önüne geçmek amacıyla GenericService olarak adlandırılan genel bir yapı içerisinde birleştirilmiştir. Bu sayede, projede kullanılan her servis için ortak olan işlemler tek bir yerde tanımlanmış ve kodun daha temiz, okunabilir ve bakımının kolay olması sağlanmıştır.

Concrete klasöründe ise, abstract klasörde tanımlanan servislerden türeyen manager sınıfları yer alır. Bu manager sınıfları, servislerin özelliklerini miras alarak projeye özel iş mantıklarını barındırır. Her bir manager sınıfı, ilgili servislerden miras alır ve constructor (yapıcı metot) aracılığıyla DataAccessLayer'dan nesneleri oluşturur. Bu nesneler, veritabanı ile doğrudan iletişime geçmek için kullanılır. Yani, veritabanına erişim, manager sınıfları içerisinde DataAccessLayer'daki DAL sınıflarıyla sağlanır. Bu yapı, katmanlar arası bağımlılığı azaltır ve iş mantığının veritabanı işlemlerinden soyutlanmasını sağlar. Ayrıca, bazı sınıflar için özel metotlar da bu manager sınıfları içerisinde tanımlanır, böylece projeye özgü iş ihtiyaçlarına göre ek işlevsellik sağlanır.

Ayrıca, Business katmanında ValidationRules adında bir klasör bulunur. Bu klasörde, validasyon sınıfları yer alır. Bu sınıflar, FluentValidation.Core kullanılarak özel validasyon kuralları oluşturur. Bu yaklaşım, HTML ve CSS'in sunduğu temel validasyon yöntemlerinin ötesine geçerek, uygulamanın gereksinimlerine daha uygun ve kompleks validasyon senaryoları geliştirmeyi sağlar. Bu yapı, validasyon kurallarını uygulamanın iş mantığından ayırarak, hem bakımını kolaylaştırır hem de kuralların merkezi bir yerden yönetilmesini sağlar.



**Şekil 3.5.2 Validasyon Sınıfları**

Bu özel validasyonlar, uygulamanın kullanıcı deneyimini iyileştirmek, veri bütünlüğünü korumak ve iş süreçlerine uygun doğrulamaları gerçekleştirmek amacıyla geliştirilmiştir. Sonuç olarak, bu yöntem, uygulamanın güvenilirliğini ve işlevselliğini artırır, kullanıcı hatalarını minimize eder ve daha güvenli bir uygulama ortamı sağlar.

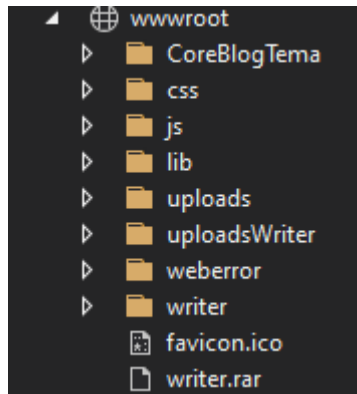
### 3.6 UserInterface Katmanı (StajBlogSitesi)

Projede kullanıcıya en yakın katman olan UI (User Interface) katmanına geçtiğimde, kullanıcıların blog sitesini nasıl deneyimleyeceğini tasarlamaya odaklandım. Bu katmanda, ASP.NET Core MVC (Model-View-Controller) mimarisini kullanarak, verilerin kullanıcıya sunulması ve kullanıcıdan gelen etkileşimlerin işlenmesini sağladım.

Ana sayfada, modern ve kullanıcı dostu bir görünüm elde etmek için MyBlogTema adlı temayı kullandım. Yazarlara özel dashboard bölümünde ise, işlevsellik ve görsellik açısından zengin Purple Dashboard temasını tercih ettim. UI tasarımında, Bootstrap gibi modern CSS frameworklerini kullanarak duyarlı ve estetik bir arayüz oluşturdum. Ayrıca, JavaScript ve jQuery ile form doğrulama ve kullanıcı girdilerinin kontrolü gibi dinamik işlemleri gerçekleştirdim.

UI katmanında yapılan çalışmalar kapsamında, ilk olarak projedeki tüm statik dosyaları barındıran wwwroot klasörünü inceledim. Bu klasörde, proje boyunca kullandığım iki tema da yer almaktadır: Ana sayfada kullandığım MyBlogTema ve yazarlara özel dashboard bölümünde tercih ettiğim Purple Dashboard. Bu temaları wwwroot altında bulundurmanın sebebi, projede kullanılan CSS, JavaScript, görseller ve diğer statik dosyalara kolay ve hızlı erişim sağlamak, aynı zamanda projenin performansını artırmaktır.

Ayrıca, wwwroot klasöründe uploadsWriter isimli bir klasör oluşturdum. Bu klasör, yazarların profil fotoğraflarını barındırmak için kullanılmaktadır. Her yazarın profil fotoğrafı, bu klasörde saklanmakta ve veritabanında yalnızca bu dosyalara giden yol bilgisi tutulmaktadır. Bunun yanı sıra, uploads klasöründe blog yazılarına eklenen resimler saklanmaktadır. Bu yaklaşım, hem veritabanının yükünü azaltmakta hem de dosya yönetimini daha düzenli ve sürdürülebilir hale getirmektedir.

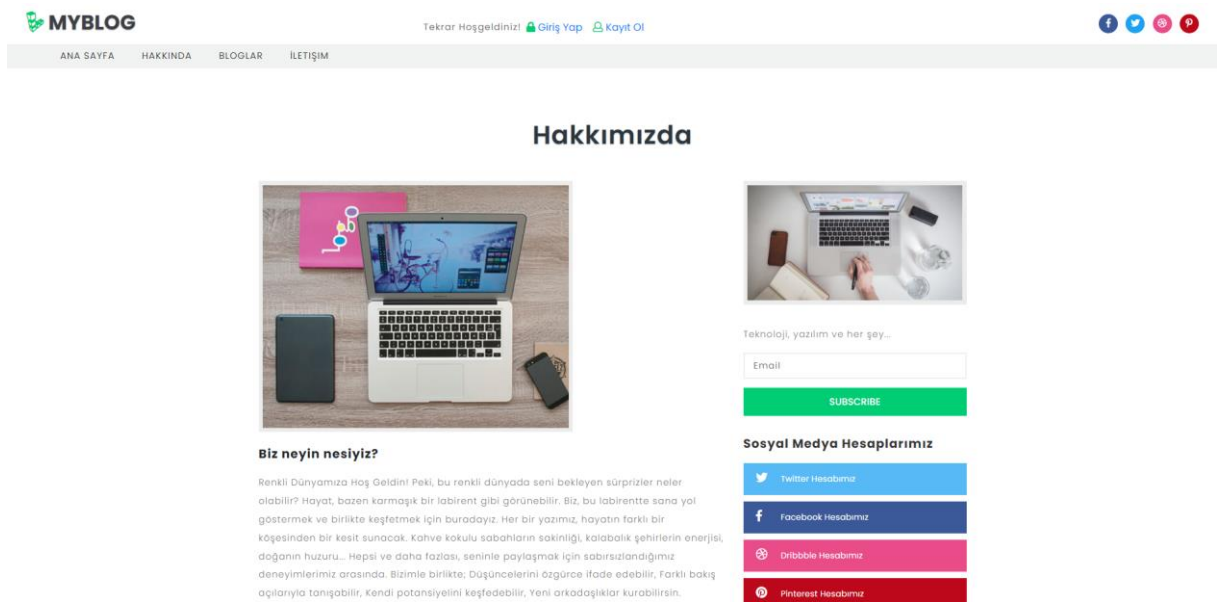


Şekil 3.6.1 wwwroot Klasörü

Controller kısmına geldiğimizde ise tüm iş burda biteceğinden dolayı burayı oldukça detaylı olacak, şu şekilde başlayacak olursam;

## AboutController

AboutController, projenin "Hakkımızda" bölümünü yönetir. Bu Controller, AboutManager üzerinden Hakkımızda bölümüne ait bilgileri veritabanından çeker ve kullanıcıya sunar. Index() metodu, bu bilgileri alarak View'a aktarır. İlk başta, veritabanı bağlantısı ile ilgili bazı hata mesajları aldım, özellikle NullPointerException gibi hatalarla karşılaştım. Bu hatalar, EfAboutRepository nesnesinin doğru şekilde başlatılmaması veya veritabanı bağlantısının eksik olması durumunda ortaya çıkıyordu. Sorunu çözmek için repository sınıfını doğru yapılandırdım ve veritabanı bağlantısını düzgün şekilde gerçekleştirdim. Bu düzeltmelerden sonra, Controller sorunsuz bir şekilde çalışmaya başladı.



Şekil 3.6.2 Hakkımızda Kısmı

## BlogController

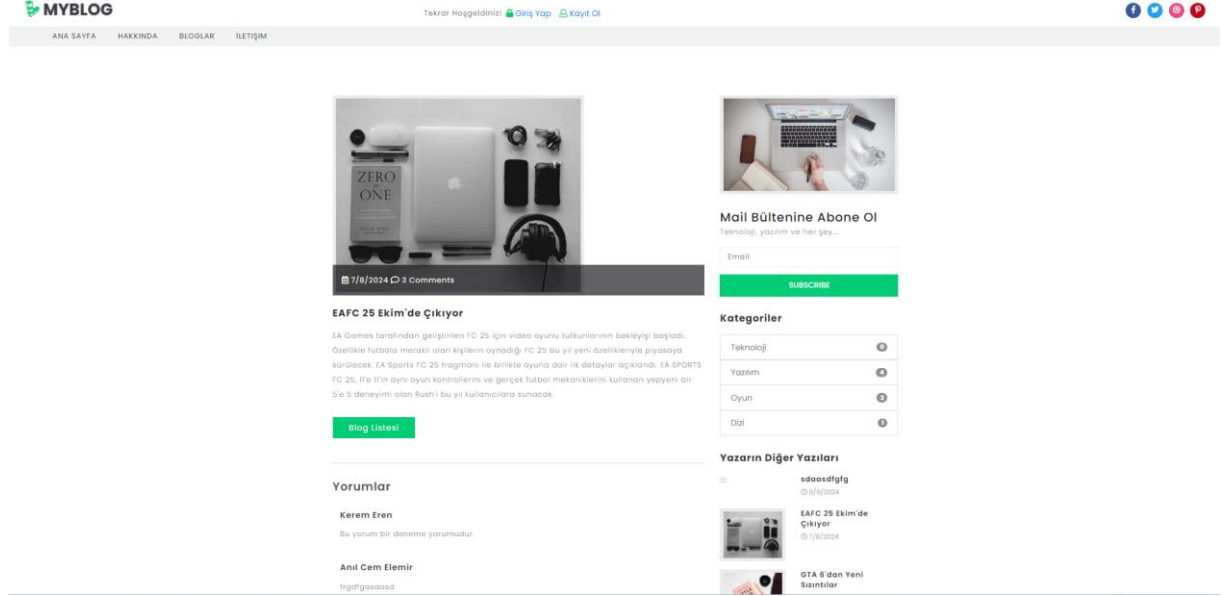
BlogController, blog yazılarıyla ilgili tüm işlemleri yönetir. Bu Controller, blogların listelenmesi, detaylarının gösterilmesi, yazı ekleme, düzenleme ve silme gibi işlevleri sağlar.

Index():

Tüm blog yazılarını kategorileriyle birlikte listeler. Veritabanından veri yüklenmesinde yavaşlık yaşanmıştır. Bu durumu, lazy loading kullanarak çözmüştüm.

BlogDetails(int id):

Belirtilen ID'ye sahip blogun detaylarını gösterir. ID'nin null olabileceği senaryoları göz ardı ettiğimde, ArgumentException gibi hatalarla karşılaşmıştım. Bu hataları gidermek için ID kontrolü ekleyerek ve null değerlerin handle edilmesini sağlayarak sorunu çözmüş oldum.



Şekil 3.6.3 Bir Blogun Detayları

BlogListByWriter():

Oturum açmış yazara ait blogları listeler. User.Identity.Name null geldiğinde kullanıcı yönlendirme hataları yaşanmıştır. Bu durum, kullanıcının oturum açmadığı durumlarda meydana geliyordu. Bu sorunu çözmek için oturum kontrolü ekleyerek ve kullanıcının oturum açmaması durumunda Login sayfasına yönlendirme yaptım.

#	Blog Başlığı	Oluşturma Tarihi	Kategori	Blog Durumu	Sil	Düzenle
2	EAFC 25 Ekim'de Çıkıyor	7/8/2024	Oyun	True	<button>Sil</button>	<button>Düzenle</button>
4	GTA 6'dan Yeni Sızıntılar	7/7/2024	Oyun	True	<button>Sil</button>	<button>Düzenle</button>
1050	sdaasdfg	9/9/2024	Yazılım	True	<button>Sil</button>	<button>Düzenle</button>

Yeni Blog Oluştur

Şekil 3.6.4 Yazarın Yazdığı Blogların Listesi

BlogAdd() [GET]:

Yeni bir blog yazısı ekleme formunu gösterir. Kategori seçeneklerini sağlamak için CategoryManager kullanır.



BlogAdd() [POST]:

Yeni bir blog yazısını veritabanına ekler. Blog resmi eklenirse, bu resim wwwroot/uploads klasörüne kaydedilir ve resim yolu veritabanında tutulur. Dosya yükleme sırasında, dosya yolunun yanlış yapılandırılması nedeniyle FileNotFoundException hatası aldım. Bu hatayı çözmek için dosya yolunu doğru şekilde belirledim ve dosya adlarının çakışmaması için benzersiz adlar kullanmaya başladım. Validation işlemleri için BlogValidator kullanır.

Şekil 3.6.5 Blog Ekleme Kısım

BlogDelete(int id):

Belirtilen ID'ye sahip blog yazısını siler. Silerken, ilişkili veritabanı kayıtlarının düzgün şekilde silinmemesiyle ilgili ForeignKeyConstraint hatasıyla karşılaştım. Bu durumu çözmek için, silme işlemi sırasında ilişkili verilerin de silinmesini sağladım.

EditBlog(int id) [GET]:

Düzenleme formunu doldurmak üzere seçilen blog yazısının bilgilerini getirir. Kategori seçeneklerini sağlamak için CategoryManager kullanır.

EditBlog(Blog b) [POST]:

Blog yazısını günceller ve veritabanına kaydeder. Yeni dosya yüklenirse, mevcut resmi siler ve yeni dosyayı yükler. Yeni dosya yüklenmemişse, mevcut resmi korur. Düzenleme işlemi sırasında, aynı verinin farklı kullanıcılar tarafından aynı anda güncellenmeye çalışılmasıyla ConcurrencyException hatası aldım. Bu hatayı çözmek için, kullanıcıya uyarı mesajı gösterip veriyi yeniden yüklemesini sağladım ve çakışmaları önledim. Validation işlemleri için BlogEditValidator kullanır.

Şekil 3.6.6 Blog Düzenleme Kısım

## CategoryController

CategoryController, blog kategorilerinin yönetiminden sorumludur. Index() metodu, tüm kategorileri listeler ve bu kategorileri kullanıcıya sunar. Kategorilerle ilgili ForeignKeyConstraint hatası aldım çünkü bazı bloglar, silinmiş kategorilere bağlıydı. Bu hatayı çözmek için, veritabanı ilişkilerini doğru şekilde yapılandırdım ve silinmiş kategorilere bağlı blogları ele alarak sorunu çözdüm.

## CommentController

CommentController, blog yazıları için yorum işlemlerini yönetir. Bu Controller, yorum ekleme, yorum listesi gösterme gibi işlevleri sağlar.

Index():

Anasayfa olarak işlev gören boş bir görünümü döner.

PartialAddComment() [GET]:

Yorum ekleme formunu içeren bir partial view döner. Kullanıcıların yorum ekleyebilmesi için formu sağlar.

PartialAddComment(Comment c, int id) [POST]:

Yeni bir yorum ekler ve belirli bir blog yazısına ilişkilendirir. Kullanıcı oturum açmışsa, kullanıcının adını otomatik olarak yorumun CommentUserName alanına ekler. Bu işlem sırasında, User.Identity.Name kullanılarak kullanıcının e-posta adresi alınır ve bu e-posta adresiyle eşleşen yazar

bilgisi Context kullanılarak elde edilir. Yazar bulunursa, CommentUserName alanı güncellenir. Yorumun tarihi CommentDate olarak ayarlanır ve yorumun aktif olduğunu belirten CommentStatus true olarak ayarlanır. Yorumun BlogID'si ilgili blog yazısının ID'siyle ilişkilendirilir. Yorum başarıyla eklendikten sonra, kullanıcıyı BlogDetails sayfasına yönlendirir.

CommentListByBlog(int id):

Belirli bir blog yazısına ait yorumları listeler. Yorumların listesi PartialView olarak döner, böylece ana sayfada yorumlar dinamik olarak yüklenebilir.

**Yorumlar**

**Kerem Eren**  
Bu yorum bir deneme yorumudur.

**Anıl Cem Elemir**  
frgdfgasdasd

**Anıl Cem Elemir**  
b

**Bir Yorum Yapın**

**Yorum Yap**

### Şekil 3.6.7 Yorum Kısım

## ContactController

ContactController, kullanıcılardan gelen iletişim mesajlarını yönetir.

Index() [GET]:

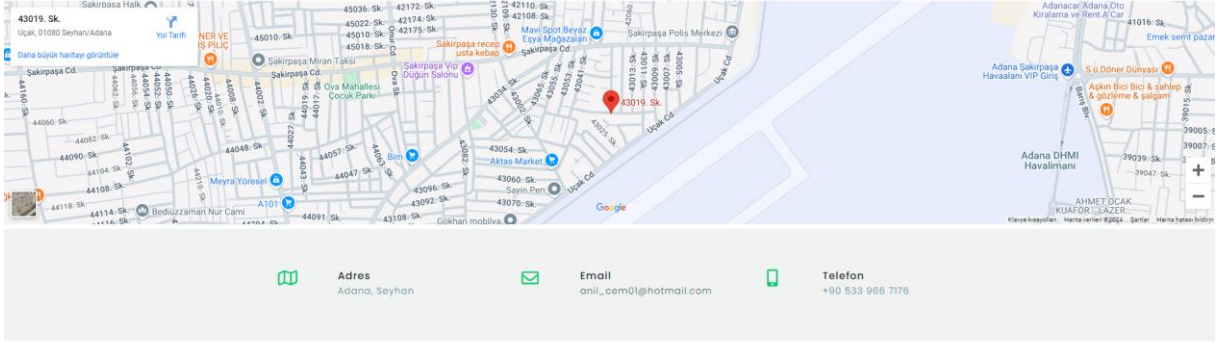
İletişim formunu gösterir.

Index(Contact c) [POST]:

Kullanıcının gönderdiği iletişim mesajını alır, tarih ve durum bilgilerini ekleyerek veritabanına kaydeder ve Blog sayfasına yönlendirir. Bu kısmı geliştirmek için bir admin paneline sahip olmam gerekiyordu ve benimde admin paneli için daha geniş zamana ihtiyaç duyduğumdan ötürü burası biraz sembolik olarak kalmış oldu, bu şekilde kalmış olan birkaç kısım daha var yeri gelince değineceğim.

## Bize Ulařın

DÜřÜNCELERİNİZİ ÖNEMSİYORUZ.



řekil 3.6.8 İletişim Kısmı

### WriterController

WriterController, yazarların profil yönetimi, kişisel bilgilerini güncelleme ve diğerk kişisel işlemlerini yönetmek için kullanılır. Projenin yazar odaklı yapısı nedeniyle bu Controller, projenin en kritik parçalarından biridir.

Index():

Boş bir View döner, genellikle test amaçlı olarak kullanılır.

WriterProfile():

Yazarın profilini görüntüler. İlk olarak profili doğru şekilde yükleyememe sorunlarıyla karşılaştım, özellikle veritabanında yazar bilgileri tam olarak eşleşmediğinde NullReferenceException hataları aldım. Bu durumu, profil yüklemesi id'lere dikkat edip düzgün id işlemleri yaparak çözdüm.

WriterMail():

Yazarın mail kutusunu görüntüler. Bu işlevin doğru çalışabilmesi için mail modülünü entegre ettim.

Test():

Test sayfasını döner ve genellikle arayüz veya işlem denemeleri için kullanılır

WriterNavbarPartial():

Yazarın kullandığı navigasyon çubuğunu içeren PartialView'ı döner. Projenin dinamik yapısı nedeniyle, bu PartialView'da doğru verilerin gösterilmemesi gibi hatalarla karşılaştım. Bu durumu, veri yüklemelerini doğru şekilde yapılandırarak çözdüm.

WriterFooterPartial():

Yazarın kullandığı alt kısım içeren PartialView'ı döner.

WriterEditProfile() [GET]:

Oturum açmış yazarın profil düzenleme formunu döner. Profili düzenlerken, kullanıcı verilerinin doğru şekilde yüklenmemesi ve formun eksik verilmesi gibi sorunlarla karşılaştım. Bu hataları, yükleme sırasında verilerin doğru şekilde doldurulmasını sağlayarak çözdüm.

WriterEditProfile(Writer w, IFormFile file) [POST]:

Yazarın profil bilgilerini ve yüklediği profil fotoğrafını alır, validasyon işlemlerini yapar ve günceller. Profil fotoğrafı varsa, wwwroot/uploadsWriter klasörüne kaydedilir ve dosya yolu veritabanında tutulur. Profil güncelleme işlemi sırasında dosya yükleme hatalarıyla karşılaştım, bazen upload ettiğim halde dosya gitmiyordu controller'a, bazen upload edildiği halde gözüküyordu. Bunları da breakpointler koyarak dosyayı formdan çekemediğimi görüp, kodda birkaç düzenleme ile çözdüm.

**Şekil 3.6.9 Yazar Profil Güncelleme Kısım**

ChangePassword() [GET]:

Kullanıcıların mevcut şifrelerini değiştirmesi için bir form döner.

ChangePassword(string CurrentPassword, string NewPassword, string ConfirmNewPassword) [POST]:

Kullanıcının şifresini değiştirme işlemi yapılır. Kullanıcının e-posta adresi üzerinden yazar bilgileri alınır. Mevcut şifre kontrol edilir; eğer yanlışsa kullanıcıya hata mesajı döner. Yeni şifre validasyonları gerçekleştirilir:

Şifre boş olmamalı.

En az 6 karakter uzunluğunda olmalı.

En az bir büyük harf, bir küçük harf, bir rakam ve bir özel karakter içermeli.

Yeni şifre ile tekrar şifresi aynı olmalıdır. Aksi takdirde hata mesajı gösterilir.

Eğer tüm validasyonlar geçilirse şifre güncellenir ve başarı mesajı gösterilir.

Bu validasyonları ayrı bir validator değilde controller içindeki kodlar ile validate etmemin sebebi burda şifre değiştirirken yapılması gereken kontrollerin de aynı zamanda burdan yapılması gerektiği için, iki yerde ayrı ayrı kullanıp kafamı karıştırmak yerine, hepsini aynı yerde yapıp toplu bir şekilde hataları tek seferde ekrana göndermekti.

WriterController, yazarların site içindeki deneyimini şekillendiren ve kişiselleştirilmiş işlemleri yöneten temel bir yapıdır. Profillerin düzenlenmesi, görsellerin yönetimi ve kişisel bilgilerin korunması gibi önemli işlevleri sorunsuz bir şekilde gerçekleştirmek için çeşitli hatalarla karşılaştım ve bu hataları çözmek için düzenlemeler yaptım. Projenin bu aşamasında, yazarların ihtiyaçlarını karşılamak ve güvenli bir kullanıcı deneyimi sunmak amacıyla, Controller'ı en iyi hale getirdim.

### NewsletterController

NewsletterController, projemde sonradan eklediğim bülten aboneliklerini yönetmek için kullanılan bir Controller'dır. Projemin gelişimi sırasında, kullanıcıların bültene abone olabilmeleri gerektiğini fark ettim ve bu ihtiyacı karşılamak için Newsletter tablosunu ve ilgili Controller'ı ekledim. Bir mail abonelik bülteni oluşturmadığım için burası yine sembolik kalan kısımlardan biri oldu.

SubscribeMail() [GET]:

Abonelik formunu gösterir.

SubscribeMail(Newsletter p) [POST]:

Kullanıcının abonelik bilgilerini alır, durumu aktif hale getirir ve veritabanına kaydeder.



#### Mail Bültenine Abone Ol

Teknoloji, yazılım ve her şey...

SUBSCRIBE

Şekil 3.6.10 Mail Bültenine Abone Olma Kısım

## RegisterController

RegisterController, kullanıcıların sisteme kayıt olmasını ve hesap onaylama süreçlerini yönetir. Kullanıcıdan alınan bilgiler doğrultusunda validasyon yapılır, kullanıcı veritabanına eklenir ve onay e-postası gönderilir.

Index() [GET] [AllowAnonymous]:

Kullanıcıların kayıt formunu görüntülemelerine olanak tanır. Kullanıcı henüz sisteme kayıt olmadığından anonim erişime izin verilir.

Index(Writer w, string confirmPassword) [POST] [AllowAnonymous]:

Kayıt formundan gelen verileri işler ve yeni kullanıcı kaydı oluşturur.

Adımlar:

Kullanıcı Var Mı Kontrolü: E-posta adresine göre veritabanında mevcut bir kullanıcı olup olmadığı kontrol edilir. Eğer e-posta zaten kayıtlıysa, bir hata mesajı döner ve kayıt işlemi durdurulur.

Şifre ve Validasyon Kontrolü: FluentValidation ile şifre ve diğer gerekli alanlar kontrol edilir. Şifre doğrulaması yapılır; şifreler eşleşmiyorsa kullanıcıya uyarı verilir.

Başarılı Kayıt: Eğer tüm validasyonlar geçilirse, yeni yazar durumu false olarak eklenir (yani onay bekler) ve varsayılan bir profil resmi atanır. Ardından, kullanıcıya hesap onaylama e-postası gönderilir.

SendConfirmationEmail(string userEmail):

Kayıt işlemi başarılı olduğunda, kullanıcıya hesap onaylama e-postası gönderilir.

Mail Gönderme Adımları:

MimeMessage kullanılarak e-posta oluşturulur.

SmtpClient kullanılarak SMTP sunucusuna bağlanılır ve e-posta gönderilir.

Mailtrap gibi bir sandbox ortamı kullanılarak, test amaçlı e-posta gönderimi yapılır.

**MYBLOG** Tekrar Hoşgeldiniz! [Giriş Yap](#) [Kayıt Ol](#)

ANA SAYFA HAKKINDA BLOGLAR İLETİŞİM

### Kayıt Ol


Adınız Soyadınız	E-Mail
<input type="text"/>	<input type="text"/>
Şifre	Şifreyi Onayla
<input type="password"/>	<input type="password"/>
<input type="button" value="Kayıt Ol"/>	

Kayıt ol tuşuna basarak, tüm şartları okuduğumu ve onlađığımı kabul ediyorum.

#### Hakkımızda

"Merhaba, dünyanın en renkli köşesinde buluştu! Biz, sadece bir blog değiliz; sıcak bir kahve eşliğinde sohbet ettiğimiz, fikir

#### Son Paylaşımlar



Batu bir maymundur.  
Batu bir maymundur.  
Batu bir m...

9/11/2024

#### Bültene Abone Ol

Teknoloji, yazılım ve her şey...

Şekil 3.6.11 Siteye Kayıt Olma Kısım

## LoginController

LoginController, kullanıcıların sisteme giriş yapmalarını, çıkış yapmalarını ve hesap onay süreçlerini yönetir. Kullanıcı bilgileri kontrol edilir, doğrulama yapılır ve giriş başarılı olduğunda kullanıcıya yetkilendirme yapılır.

Index() [GET] [AllowAnonymous]:

Kullanıcıların giriş formunu görüntülemelerine olanak tanır. Anonim erişime izin verilir, çünkü kullanıcılar henüz sisteme giriş yapmamıştır.

Index(Writer w, bool rememberMe) [POST] [AllowAnonymous]:

Giriş formundan gelen kullanıcı bilgilerini işler ve giriş işlemini gerçekleştirir.

Adımlar:

Validasyon Kontrolü: LoginValidator kullanılarak gelen bilgiler doğrulanır. Eğer validasyon hatası varsa, kullanıcıya geri döndürülür.

Veritabanı Kontrolü: Veritabanında (Context) girilen e-posta ve şifreye sahip bir kullanıcı olup olmadığı kontrol edilir.

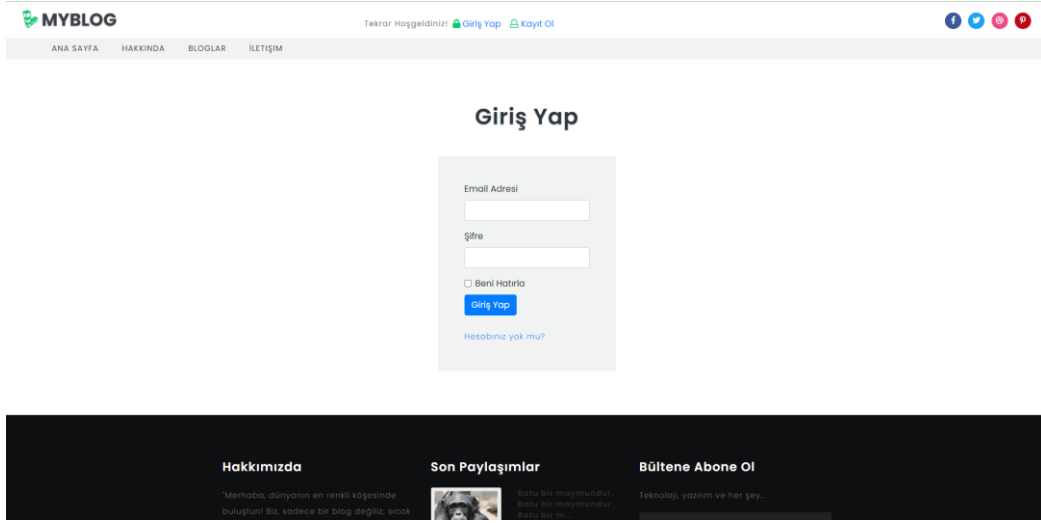
Giriş Başarılıysa: Kullanıcı bulunursa, kullanıcıya ait Claimler (E-posta ve ad bilgisi) oluşturulur ve kullanıcı sisteme giriş yapar. Eğer "Beni Hatırla" seçeneđi işaretlendiyse, oturum sürekli olur.

Giriş Başarısızsa: Kullanıcı bulunamazsa, aynı giriş sayfasına geri döndürülür.

Logout() [POST]:

Kullanıcının oturumu kapatmasını sağlar. HttpContext.SignOutAsync() fonksiyonu çağırılarak kullanıcının yetkileri temizlenir ve kullanıcı ana sayfaya yönlendirilir.





Şekil 3.6.12 Siteye Giriş Yapma Kısım

ConfirmEmail(string email) [AllowAnonymous]:

Kullanıcıya gönderilen e-postadaki onay bağlantısına tıklanınca, bu metod çağrılır.

Adımlar:

Veritabanında girilen e-posta ile eşleşen kullanıcı aranır.

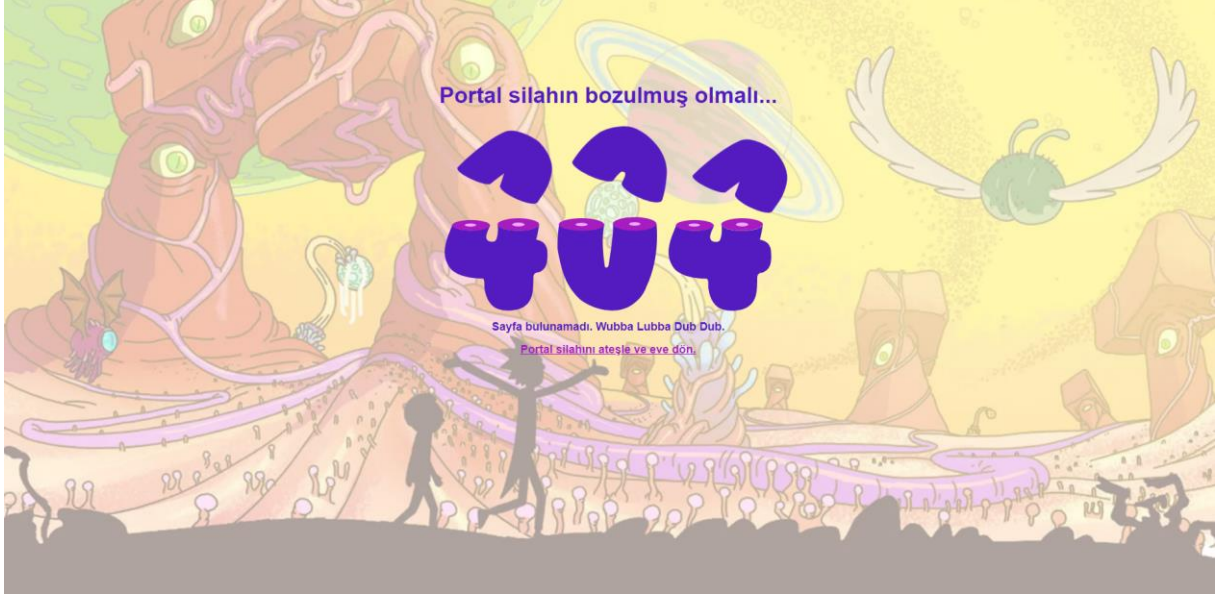
Eğer kullanıcı bulunursa ve henüz onaylanmamışsa, kullanıcının durumu WriterStatus = true olarak güncellenir ve e-posta onaylanmış olur.

Kullanıcıya e-posta başarıyla onaylandığına dair mesaj gösterilir ve giriş sayfasına yönlendirilir.

## ErrorPageController

ErrorPageController, hata sayfalarını yönetmek için sonradan eklenen bir Controller'dır. Projenin belirli aşamalarında, kullanıcıya özelleştirilmiş hata mesajları göstermek için bu Controller'ı ekledim. Bu sayede, kullanıcılar hatalarla karşılaştığında daha açıklayıcı ve yönlendirici mesajlar alabiliyor.

Error1(int code): Belirtilen hata koduna göre bir hata sayfası döner. Hata sayfasını yapılandırırken, özellikle 404 ve 500 gibi yaygın hata kodlarını ele alarak kullanıcıya daha anlamlı mesajlar sunmayı hedefledim. Ancak, hata mesajlarını doğru şekilde yakalamakta zorluk çektim ve Exception hatalarıyla karşılaştım. Bu hataları çözmek için, global hata yakalama mekanizmasını ekleyip özelleştirilmiş hata sayfalarını devreye soktum.



Şekil 3.6.13 404 Hata Sayfası

### Sonradan Eklemeler

Projenin bu temel işlevleri sağladıktan sonra, yazarların site içindeki aktivitelerini daha detaylı bir şekilde izleyebilecekleri ve yönetebilecekleri bir Dashboard oluşturma gereği duyduğum için veritabanı tablolarımda ve controllerlarımda yeni eklemeler yaptım.

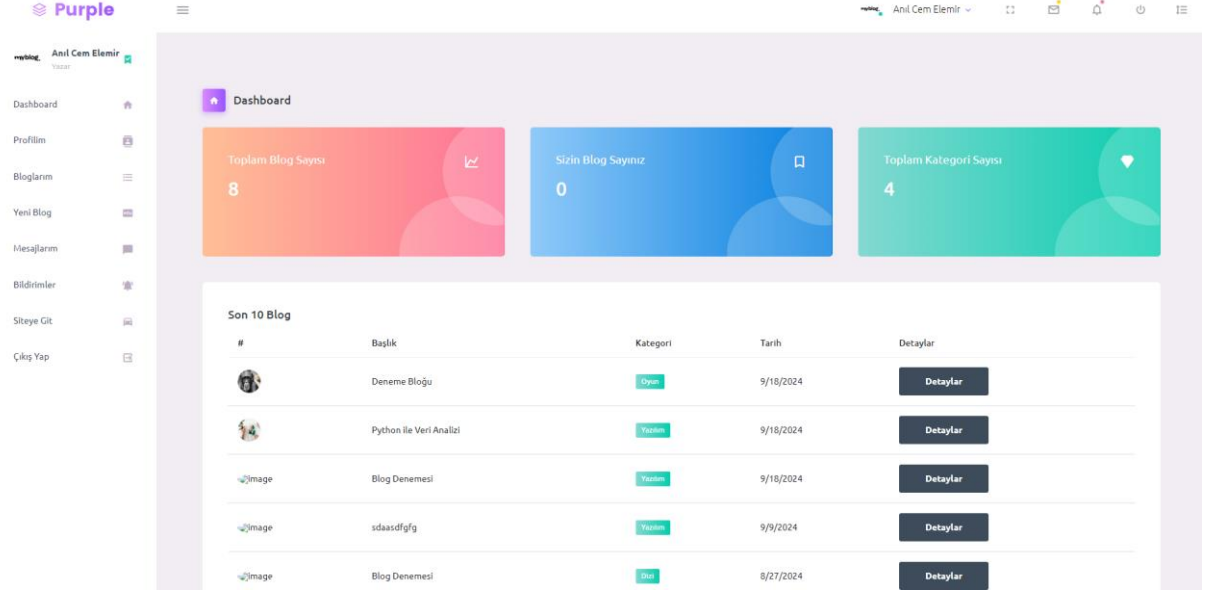
Yazarların blog performanslarını izleyebilecekleri bir yapı kurmak için, veritabanına BlogRating adlı yeni bir tablo ekledim. Bu tablo, her eklenen blog ve yapılan yorum için bir puanlama sistemi sunuyordu. Her yeni blog eklenmesi durumunda BlogRating tablosundaki ilgili değer otomatik olarak artırılacak şekilde bir trigger tanımladım. Ayrıca, her yorum yapıldığında blogun puanı artırılıyor ve yazarın toplam blog sayısı ile kategori sayısı güncelleniyordu. Bu SQL triggerlar sayesinde, yazarların içeriklerinin performansını anlık olarak izleyebilmeleri mümkün hale geldi. Bu yapı oluşturuldu fakat daha fazla zaman gerektiği için kullanılamadı.

Projemde yazarlar arasında doğrudan iletişim kurulmasını sağlamak için, veritabanı modelime Message2 adlı bir tablo ekledim. Bu tablo, iki farklı Writer nesnesine referans veren çift Foreign Key yapısıyla çalışıyordu. Message2 tablosunda, SenderID ve ReceiverID olarak tanımlanan iki foreign key bulunuyordu. Bu yapıyla, yazarlar hem mesaj gönderen hem de mesaj alan olarak aynı tabloda yer alabildiler. Bu çift foreign key yapısı, veri tabanındaki mesajlaşma sistemini esnek ve işlevsel hale getirdi. Migrationlar kullanarak bu yapıyı veritabanımda sorunsuz bir şekilde oluşturdum.

Bu değişikliklerin ardından projeye yeni controllerlar ekledim:

## DashboardController

Index(): Yazarların blog performanslarını ve genel istatistikleri görüntüler. ViewBag.TotalBlogCount, ViewBag.TotalWritersBlogCount, ve ViewBag.CategoryCount ile sistemdeki toplam blog sayısı, yazarın yazdığı blog sayısı ve kategori sayısı gösterilir. SQL triggerlar ile güncellenen verilerde başlangıçta uyumsuzluk yaşadım. Triggerların doğru çalışmasını sağlamak için veritabanı yapılandırmasını ve trigger kodlarını gözden geçirdim.



Şekil 3.6.14 Dashboard Ana Sayfa

## NotificationController

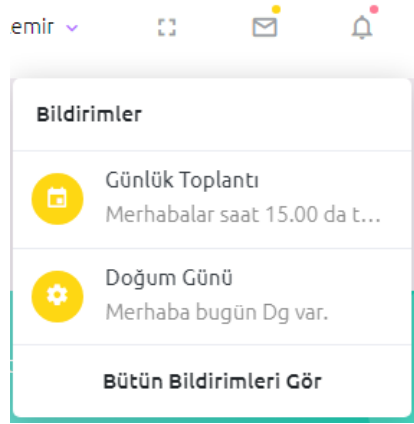
Bu kısımda admin panel ile çok daha efektif olacaktı, bildirimler admin panelinden manipüle edilebilecekti, fakat burası da görsel bi iyileştirme olarak kaldı.

Index(): Bir işlevi yok.

AllNotification(): Kişiye bildirimleri listeler.

#	Bildirim	Tarih
1	Merhabalar saat 15.00 da toplantı var.	8/23/2024
2	Merhaba bugün Dg var.	8/23/2024

Şekil 3.6.15 Tüm Bildirimlerin Listelendiği Kısım



**Şekil 3.6.16 Bildirim Pop-Up'ı**

### **MessageController**

MessageController, kullanıcıların mesaj kutusuna erişmesini, mesaj detaylarını görüntülemesini ve yeni mesaj göndermesini sağlayan işlevleri içerir. Bu işlevler, mesajlaşma özelliklerinin yönetilmesini sağlar.

Inbox() [AllowAnonymous]:

Kullanıcıların gelen mesajlarını görüntülemelerine olanak tanır.

Adımlar:

Oturum açan kullanıcının e-posta adresi User.Identity.Name ile alınır.

Veritabanında (Context) bu e-posta adresine sahip yazar (Writer) aranır.

Eğer yazar bulunursa, yazarın WriterID'si kullanılarak mesajlar (Message2) getirilir ve gelen kutusu listesi (Inbox) kullanıcıya görüntülenir.

Eğer yazar bulunamazsa, "Yazar bulunamadı" hatası döndürülür.

MessageDetails(int id) [AllowAnonymous]:

Kullanıcıların bir mesajın detaylarını görüntülemelerini sağlar.

Adımlar:

Mesajın ID'si ile ilgili mesaj mm.GetbyId(id) ile veritabanından getirilir.

Mesajın içeriği ve detayları kullanıcıya görüntülenir.

SendMessage() [HttpGet]:

Kullanıcıların yeni bir mesaj göndermesi için formu görüntüler.

SendMessage(Message2 message, string ReceiverName) [HttpPost]:

Kullanıcıların mesaj göndermesini sağlar.

Adımlar:

Oturum açan kullanıcının e-posta adresi alınır ve veritabanında (Context) bu e-postaya sahip yazar (Writer) bulunur. Bu yazar mesajın göndericisi (SenderID) olacaktır.

Alıcı yazar (Receiver) kullanıcı adıyla (WriterName) bulunur. Eğer alıcı bulunamazsa, hata mesajı görüntülenir.

Mesaj validasyonu MessageValidator ile yapılır. Eğer validasyon başarılı olursa:

Mesajın göndericisi ve alıcısının WriterID'leri belirlenir.

Mesajın gönderilme tarihi MessageDate olarak atanır.

Mesaj durumu aktif (MessageStatus = true) olarak ayarlanır.

Mesaj veritabanına eklenir ve kaydedilir.

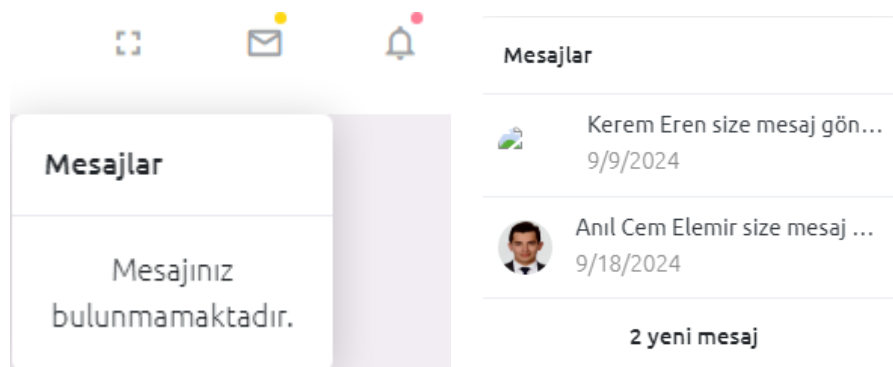
Eğer validasyon hatası varsa, hatalar kullanıcıya gösterilir.

Başarılı mesaj gönderme işleminden sonra kullanıcı gelen kutusuna (Inbox) yönlendirilir.

MessageValidator ile mesaj içeriğinin doğruluğu kontrol ediliyor, bu da mesajın geçerli olup olmadığını kontrol ediyor.

Kullanıcı giriş yapmış olmalı ki gönderici ve alıcı bilgileri doğru bir şekilde eşleştirilebilsin.

Mesaj gönderme ve alma işlemlerinde veritabanı ilişkileri doğru bir şekilde yönetiliyor ve mesajlar Message2 tablosuna kaydediliyor.



Şekil 3.6.17 Mesaj Pop-Up'ı

## **Görünüm ve Layout Yönetimi:**

Layout Kullanımı: Projede iki ana layout kullanılmıştır: WriterLayout yazarların kontrol paneli için, ve UserLayout ise ana sayfa için. İlk başta, Partial Views kullanılarak temel görünüm bileşenleri oluşturulmuştur. Ancak, proje sürecinde mühendislerden aldığım geri bildirimler doğrultusunda ve daha iyi yönetim için ViewComponents kullanımına geçilmiştir. Bu geçiş, görünüm bileşenlerinin daha modüler ve yeniden kullanılabilir olmasını sağlamıştır.

ViewComponents Kullanımı: Görünümler ve kullanıcı arayüzü bileşenlerinin yönetimini geliştirmek amacıyla ViewComponents kullanımı benimsenmiştir. Bu geçiş, belirli işlevlerin daha bağımsız ve etkili bir şekilde yönetilmesine olanak tanımış ve görünüm bileşenlerinin tekrar kullanılabilirliğini artırmıştır.

Görünümler ve Performans İyileştirmeleri: Görünümler, kullanıcı arayüzünü oluşturmak için çeşitli bileşenlerle desteklenmiştir. Özellikle, veri güncellemeleri ve kullanıcı etkileşimleri sırasında performans sorunları yaşanmıştır. Bu sorunlar, görünüm bileşenlerinin optimizasyonu ve performans iyileştirmeleri ile çözülmüştür. Ayrıca, ViewComponents ile ilgili yaşanan performans sorunları, bu bileşenlerin yeniden yapılandırılması ve sorgu optimizasyonları ile çözülmüştür.

## 4. SONUÇ

Bu proje, .NET Core teknolojisi kullanılarak kapsamlı bir blog yönetim sistemi geliştirmeyi hedeflemiştir. Proje, kullanıcıların blog içeriklerini yönetmeleri, yazarlar arasında iletişim kurmaları ve performans izleme işlevlerini kapsamaktadır. İşte projenin genel sonuçları:

### 1. Kullanıcı Yönetimi ve Giriş Sistemleri:

**Kullanıcı Kayıt ve Giriş:** Kullanıcıların sisteme kayıt olabilmesi ve giriş yapabilmesi sağlanmıştır. Kayıt ve giriş işlemleri, güvenli şifreleme yöntemleri ve form doğrulama ile desteklenmiştir.

**Giriş Sonrası Yönlendirme:** Kullanıcılar giriş yaptıktan sonra ana sayfaya yönlendirilmekte ve kullanıcı adı, 'Tekrar Hoşgeldiniz!' mesajı ile birlikte görüntülenmektedir. Ayrıca, giriş yaptıktan sonra 'Giriş Yap' ve 'Kayıt Ol' seçenekleri gizlenmektedir.

### 2. Blog Yönetimi:

**Blog Ekleme ve Düzenleme:** Yazarlar bloglarını ekleyebilir ve düzenleyebilir. Blog eklerken yazarın kimliği formdan alınmakta, bu da hard-coding yerine dinamik bir yapı sağlar.

**Blog Performans İzleme:** DashboardController ile yazarların blog performansları anlık olarak izlenebilir. SQL triggerlar, blog eklenme ve güncellenme işlemlerinde verilerin otomatik olarak **güncellenmesini sağlar**.

### 3. İletişim ve Bildirimler:

**Mesajlaşma:** MessageController ile yazarlar arasında mesajlaşma özelliği sunulmuştur. Mesajlar, gelen kutusunda sıralanabilir ve okunmuş durumları güncellenebilir.

**Bildirim Yönetimi:** NotificationController ile yazarlar önemli bildirimleri görüntüleyebilir ve yönetebilir. Bildirimler okunmuş ve okunmamış durumlarına göre sıralanır, ayrıca tarih ve önem derecesine göre filtrelenebilir.

### 4. Görünüm ve Layout Yönetimi:

**Layout Kullanımı:** İki ana layout kullanılmıştır: WriterLayout yazarların kontrol paneli için, ve UserLayout ise ana sayfa için. Başlangıçta Partial Views kullanılarak görünümüler oluşturulmuş, ancak daha sonra ViewComponents kullanımıyla bu yapılar daha modüler hale getirilmiştir.

**ViewComponents:** Bu geçiş, görünüm bileşenlerinin daha yeniden kullanılabilir ve etkili bir şekilde yönetilmesini sağlamıştır.

### 5. Veritabanı Yapıları ve Performans:

**Yeni Tablo ve Triggerlar:** Veritabanına BlogRating ve Message2 adlı yeni tablolar eklenmiş, SQL triggerlar ile blog performans verileri otomatik olarak güncellenmiştir. Yazarlar arasında mesajlaşma ve blog performansları izlenebilir hale getirilmiştir.

Veritabanı İyileştirmeleri: Performans sorunları yaşanan alanlarda, veri tabanı sorguları optimize edilmiştir.

## 6. Güvenlik:

AntiForgeryToken Kullanımı:

AntiForgeryToken eklentisi ile CSRF (Cross-Site Request Forgery) saldırılarına karşı koruma sağlanmıştır. Özellikle formlar üzerinde yapılan işlemler sırasında, bu token kullanılarak kullanıcıların güvenliği artırılmıştır. Tüm veri giriş formlarına [ValidateAntiForgeryToken] eklenmiştir, böylece form gönderim işlemleri doğrulanarak kötü amaçlı saldırılara karşı daha güvenli hale getirilmiştir.

HashPassword Kullanımı:

HashPassword kullanılarak kullanıcıların şifreleri güvenli bir şekilde saklanmıştır. Şifrelerin düz metin yerine hashlenmiş şekilde veritabanında tutulması, kullanıcı hesaplarının kötü niyetli saldırılara karşı korunmasına yardımcı olmuştur. Bu sayede, şifreler veritabanında okunabilir biçimde saklanmamış, tersine çevrilemeyecek şekilde kodlanmıştır.

Güvenlik Artırma Adımları:

Kayıt ve giriş işlemlerinde ekstra güvenlik sağlamak için form doğrulama süreçleri sıkılaştırılmıştır. FluentValidation ile form doğrulama kuralları kullanılarak, sadece güvenli ve doğru verilerin kabul edilmesi sağlanmıştır.

Kullanıcıların oturumları boyunca güvenliklerinin korunmasına yönelik ek önlemler alınmış ve bu önlemler sistemin genel güvenlik yapısını güçlendirmiştir.

Bu ek güvenlik önlemleri, projedeki kullanıcı verilerinin korunmasına ve saldırılara karşı dayanıklılığın artırılmasına yardımcı olmuştur. Proje, güvenlik standartlarına uygun olarak geliştirilmiş ve bu sayede kullanıcı verilerinin gizliliği sağlanmıştır.

Sonuç: Bu proje, kullanıcıların blog içeriklerini etkili bir şekilde yönetmelerini, performanslarını izlemelerini ve yazarlar arasında etkili iletişim sağlamalarını mümkün kılan kapsamlı bir sistem sunmaktadır. Yapılan geliştirmeler ve eklenen yeni özellikler, sistemin genel işlevselliğini ve kullanıcı deneyimini önemli ölçüde artırmıştır. Proje, karşılaşılan teknik zorlukları aşarak başarılı bir şekilde tamamlanmış ve gelecekteki geliştirmeler için sağlam bir temel oluşturulmuştur.