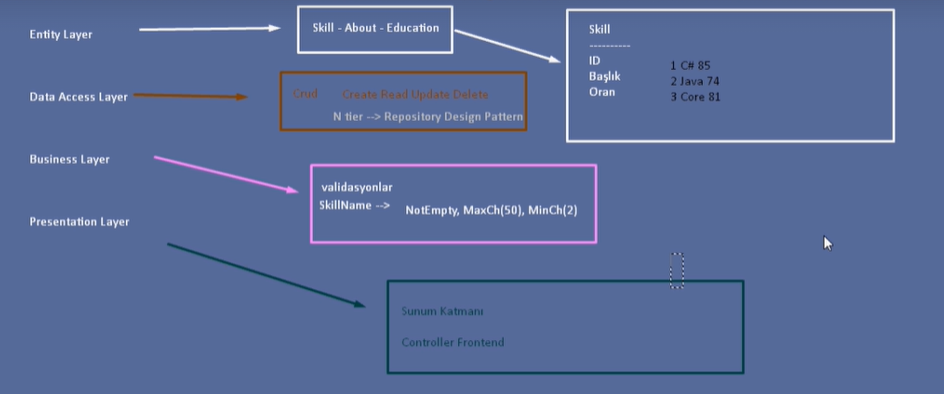
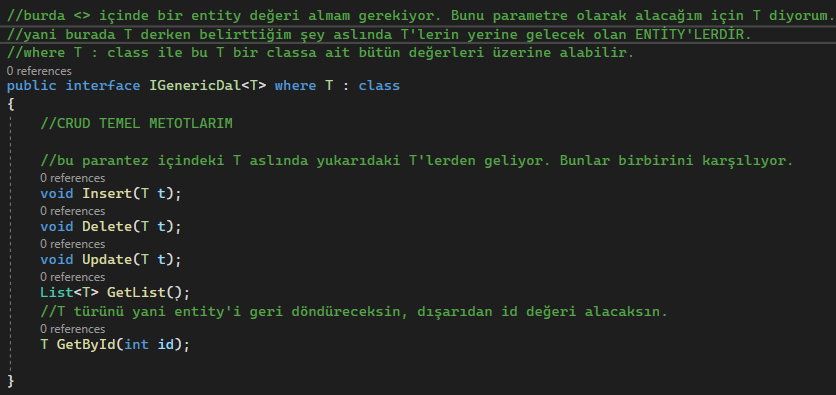
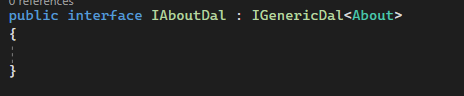
KATMANLI MİMARİYE BAKIŞ

* 4 adet katman bulunacak;
* 1-) Entity Layer: Bizim varlıklarımızı tutacak. Önreğin Skill-About-Education. Bu varlıkların kendi içerisinde propları olacak. Mesela Skill’in bir Id’si, Başlığı, Oranı olabilir. Sitemizde dinamik olarak tutmak istediğimiz bilgileri Entity olarak tanımlayacağız.
* 2-)Data Access Layer: Burada CRUD operasyonlarına ait metotlar hazırlanacak. Mimari olarak N tier , geliştirme yaklaşımı pattern olarak Repository Design Pattern kullanılacak.
* 3-) Businnes Layer: Burası validationların yazıldığı katman olacak.
* 4-) Presentation Layer: Sunum katmanımızdır. Controller tarafındaki kodlar ve Frontend tarafındaki tasarım kodları.
* 
* Projemiz boyunca 2 tane temel klasörle çalışalacak;
* 1-) Concrete: Genellikle classlarla çalışıcaz.
* 2-) Abstract: Genellikle interfacelerle çalışıcaz.
* Entitiyler oluşturulduktan sonra projeme eklemem gereken 4 adet paket var bunları entity, uı ve dataccess ve businnes yani tüm katmanlarıma kuracağım;
* 1-) EFCore
* 2-) Sql
* 3-) Tool
* 4-) Design
* Entity katmanımda tüm entity’lerimin Id kolonlarıma [Key] attibute’u tanımlayacağım ki bunun primary key olduğunu belirteceğim.
* Katmanlar arası referans işleminde katmanlar birbiri arasında referans alması gerekiyor;
* Entity katmanı Data Access katmanı içerisinde yer alacak ve Business katmanı içerisinde yer alacak ve UI katmanı içerisinde yer alacak yani Entity katmanı bütün katmanlar içerisinde yer alacak. Ancak herhangi bir katman entity katmanı içerisinde yer almayacak.
* DataAccess katmanı Business katmanı içerisinde yer alacak ve UI katmanı içerisinde yer alacak.
* Businnes katmanı sadece UI katmanı içerisinde yer alacak.
* 

MİGRATİON VE REFERANS İŞLEMLERİ

* DataAccess Katmanına Concrete klasörü eklenir. İçerisine Context sınıfımız eklenir.
* Context Classı: Bizim veritabanı yapılandırmamız içerisindeki bağlantı stringini tutan ve veritabanına yansıtılacak olan tabloları tutan classımızdır. Tablolarımız dbster olarak bu classa eklenirler dbset olarak verdiğimiz isimler bizim için tablo isimleri olacaktır. Ayrıca context sınıfımız DbContext sınıfından kalıtılmalıdır. Sql bağlantı stringimiz OnConfiguring metotu içerisinde yapılır.
* Migration: Biz projemizi geliştirirken EfCore’un CodeFirst yaklaşımını kullanıyoruz. CodeFirst yaklaşımında VS üzerinde oluşturulan sınıflar sql’E birer tablo olarak yansıtılacak. Bağlantı adresi ise sql’e bir veritabanı olarak yansıtılacak bu entitylerin içersindeki proplar ise sql’e tablo sütunu olarak yansıtılacak. Bu yansıtma işlemini migration üzerinden gerçekleşitiriz.
* Migration oluştururken default project olarak DataAccess katmanı seçilmeli çünkü context sınıfı burda. Mutlaka context sınıfının bağlı bulunduğu katmanı seçmem gerekiyor.
* Migration oluşturduktan sorna değişiklerin db tarafına aktarılması için update-database yapıyoruz.

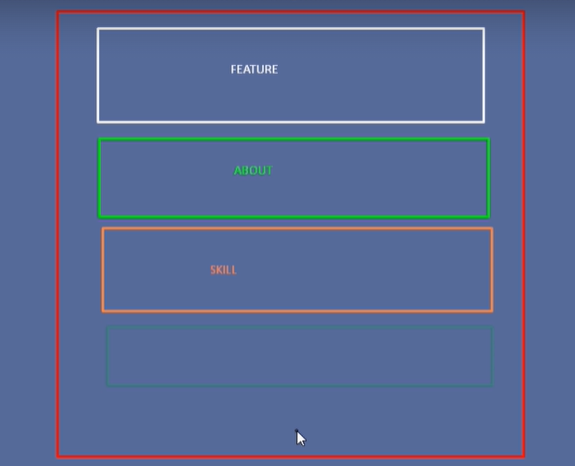
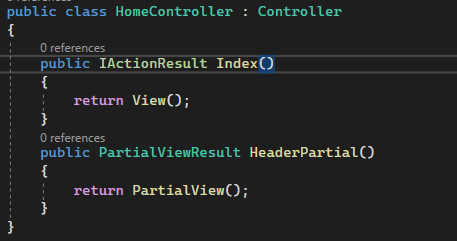
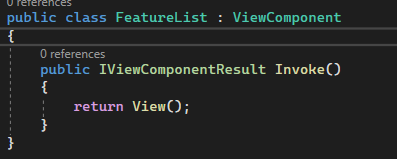
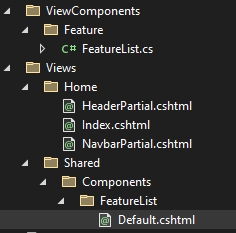
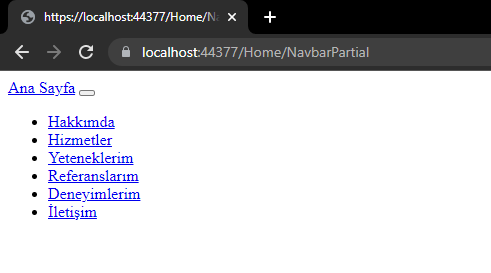
İNTERFACELER

* DataAccess katmanı içerisine Abstract klasörü ekliyorum bu benim interfacelerimi tutacak. Bu interfaceler ne olacak;
* Benim entitylerimin her birinin kendi içerisinde CRUD metotları olacak. Nedir bu CRUD metotlar, Ekle, Listele, Sil, Güncelle, Ara, Getir gibi. Temelde baktığımız zaman entitylerin herbiri için temel CRUD operasyonları yazılacak. Bu CRUD metotların içini doldurmadan önce birer imzaları olacak. Bu imzaların içleri Concrete klasöründe bulunan normal classlarda doldurulacak. Eğer direkt olarak inteface kullanmasak direkt olarak classı kullansak proje büyüdükçe projenin yönetimini zorlaştırmış olururuz ve mimariye uygun kod geliştirmeyiz amacımız proje büyüdükçe projenin yönetimini kolaylaştırmak.
* Baktığımız zaman tüm entitylerim için aynı crud işlemlerimi defalarca tekrar tekrar yapıyorum metotlar aynı farklı olan tek şeyler entityler bu kullanım yanlıştır. Ben bunu Generic hale getirirsem her bir entity için tek tek bütün metotları tanımlamak yerine tepede bir metot tanımlayıp bu metot içerisinden generic olarak diğerlerini türetirim.
* Generic yapıda olay şu: Ben her bir entity’im için ayrı ayrı metot tanımlamak yerine tepede bir yapı kurayım bu yapı sayesinde diğerlerinin tamamını bundan türeteyim.
* 
* GENERİC YAPININ KULLANIMI
* 
* Abstract Tarafında generic classta olarak metot imzalarımızı attıktan sonra entitylerimin crud işlemleri dışında kendilerine has metotlarının imzalarını taşayacak olan entitylerin özel oalrak abstract sınıflarını oluşturup bu sınıfları IgenericDal’dan kalıtıyoruz ve <> içeirsinde ilgili entity’in ismini vererek bu class bu entity için var diyoruz. Bu işlemlerden sonra Generic bir Repository classı oluşturuyoruz ve bu classı IgenericDal’dan kalıtıp metotları implemente ederiz ve bu imzası atılmış metotlarımızın içeirsini dolduruyoruz.
* Daha sonra entitylerimin özel kendine has olan metotları olabilir CRUD İşlemler dışında. Bu ihtiyacı karşılamak için DataAccess Katmanıma EntityFrameWork klasörünü ekliyorum. Her bir Entity için özel olarak class oluşturuyorum ve bu classları GenericRepository ‘Den kalıtarak crud metotları kazanıyorum özel olarak oluşturmuş olduğum örneğin IskillDal’dan da kalıtarak kendine has olarak oluşturulmuş metot imzalarınıda buraya geçirip o özel metotlarıda burda dolduruyorum.

GENERİC SERVİCES

* Business katmanında Abstract adında ve Concrete adında ve ValidationRules adında klasör ekliyorum.
* Service kullanmanın olayı şöyle repositylere içinde bulunan metotlara data gönderiyor. Datayı göndermeden önce belli validaiton işlemlerine tabi tutuluyor data ve belirli mapping işlemlerine tabi tutuluyor bunlardan sorna ortaya çıkan istediğim gibi olan data repository’e gönderiliyor ve repository’de uygulamam ve db arasındaki ilişkiyi gerçekleştiriyor. Bu yüzden benim Service interfacelerimi yazıp Manager classlarımı yazdıktan sonra bu manager classları içinde dependency injection ile repositorylerimi ele almam gerekli işlemleri uygulayıp daha sonra orataya çıkan datayı repositorylere göndermem gerekiyor. Yani şöyle özetleyebiliriz Kullanıcı create işlemi yapacak diyelim businnes katmanıyla iletişimde olur businnes katmanındaki İlgili manager’daki metotu kullnarak bu işlemi yapar. Manager’daki create matotuna datayı gönderir o create metotu ilgili validation işlemlirini dataya uygular ilgili dtoyu veya modeli veya modelview’ı entity’e mapping eder çevirir ve sonra bunu repositoryde bulunan create metotuna yollar. İstediğin şekilde gelen data repositoryin create metotuna girerek db ye bu datayı gönderir ve create işlemi başarılı bir şekilde gerçekleşir.

TEMANIN EKLENMESİ VE DEFAULT KISMI

* Wwwroot dosaysının içine image dosyalarını temaları , bootstrap vb dosyalarımızı atabiliriz. Bu dosya dışarıya açıktır.
* Partial View : sayfayı belli bölümlere bölmek için kullanılır. Yani bir nevi sayfaları parçalara ayrımak gibi. Bir bütün halinde verilen yapıyı parçalara ayırıyorum böylece yönetim çok daha kolaylaşacak hemde ana olan sayfamda kodlarım azalmış olacak. Bu işi İlgili Controllerda tanımlıyorum önce sonra cshtml sayfalarını oluşturuyorum.
* 
* 
* OLUŞTURDUĞUM PARTİALVİEW’İ ANA SAYFAMDA ÇAĞIRMAK
* Aynı klasör içinde olduğu için direkt oalrak “HeaderPartial” olarak çağırdık farklı bir konumda olsaydı ~/ şeklinde o konuma çıkmamız gerekirdi aman dikkat.
* 
* View Component: Bunu kullanmak için UI katmanıma ViewComponents adında bir klasör oluşturuyorum. View Component View bileşenlerini tutacak. Bizim her bir alanımız için ayrı bir View Component tanımlayabiliriz. Daha sonra oluşturduğum classı ViewComponent classından kalıtıyorum. Aslında controller’a benziyor ControllerDan değilde bu classtan kalıtılıyor o yüzden tıpkı controller’a yaklaştığımız gibi yaklaşıcaz buraya. IViewComponentResult tan bir metot yazıyoruz ve bu metotlar hep Invoke olarak tanımlanıyor.
* 
* Daha sonra View klasörü içine Shared klasörü ekliyorum ve bunun içine Components klasörünü ekliyorum. Daha sonra bunun içinede oluşturuduğumuz ViewComponent classının ismiyle aynı olan bir klasör daha ekliyorum. Daha sonra bu klasörün içine ekleyecğimiz partialview’ın ismi Default olmak zorunda.
* 
* Daha sonra bu Componenti ana sayfamda kullanmak için;
* 
* Buradaki fark şöyle denilebilir. PartialView kullandığımız zaman link kısmına PartialView’ın ismini yazdığımızda sadece PartialView’ın olduğu sayfaya gitmek mümkün ama Componentte böyle bir şey yok erişim engellemiş oluyoruz. Amaç Direkt olarak erişimi engellemek.
* ViewComponentte aşağıdaki olayı yapamayız yani;
* 
* View Component veya Partial Viewleri bir sayfada birden fazla tablo bilgisi göstereceksek kullanması mantıklı tarafta kalabiliyor. Sonuçta o başka bir cshtml sayfası oluyor o sayfaya farkllı bir model bağlayabiliyoruz. Yalnız View Componentte sadece 1 tane Invoke metot yazabiliyoruz aman dikkat Get, Post içeren metotlu işlerimizi ViewComponent ile yapmak mantıklı değil 2 tane Invoke metotu yazamayacağımız için Create Update gibi post metot lazım olan işlemlerde PartialView kullanmak daha mantıklı. Ancak bu seferde PartialWiev’in View’ine yönlendirme oluyor bunu düzeltmek için Ajax kullanıcaz.
* Unutmayalım ViewComponentlerin Invoke metotları async Task olamazlar. Bu yüzden InvokeAsync kullanmamız gerekir. Invoke yerine.

FLUENT VALİDATİON

* Fluent Validation kullanmak için hem businnes hemde UI katmanına FluentValidation.AspNetCore paketini kurmam gerekiyor.
* Ekleme, Güncelleme, Listeleme, Silme işlemi yaparken benim istediğim şartların sağlanıp sağlanmadığının kontrolünü yapacağım bir kütüphanedir.
* Kullanmak için Business katmanında ValidationRules klasörüne ilgili işlemin isminde bir class oluşturuyorum örnek olarak PortfolioValidator. Daha sonra bu classı AbstractValidator<İlgili Entity ismi> classından kalıtıyorum. Daha sonra ctor içinde RuleFor metotu içine gerekli validation işlemlerimi yapıyorum. Daha sonra ilgili comtroller içerisinde PorfolioValidator yani oluşturduğumuz classın isminde bir nesne üretiyoruz. Daha sorna aşağıdaki gibi bir kullanım yapıp geçerliliği kontrol ediyoruz.

PortfolioValidator validationRules = new PortfolioValidator();

* ValidationResult result = validationRules.Validate(portfolio);

if (result.IsValid)

{

//sorun yok

portfolioManager.TAdd(portfolio);

return RedirectToAction("Index");

}

else

{

foreach (var item in result.Errors)

{

//böylece validator classındaki ilgili property adına göre ilgili hata mesajı gelecek.

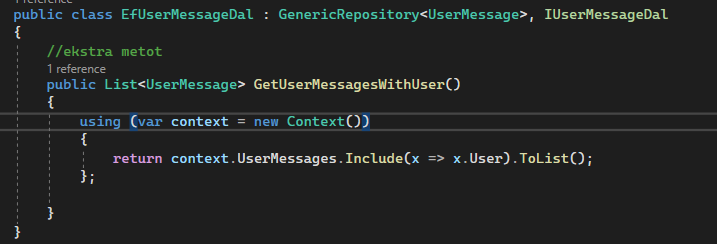
ModelState.AddModelError(item.PropertyName, item.ErrorMessage);

}

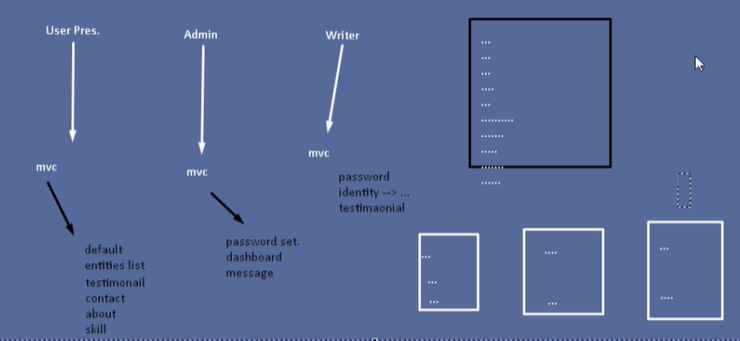
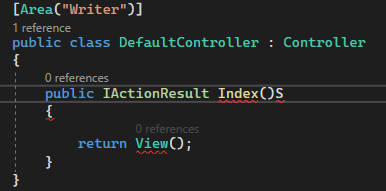
}

* return View();
* Daha sonra controllerda işimiz biter ilgili cshtml sayfasına gideriz. VE HATALARI ORDA GÖSTERİRİZ. SUMMARY VEYA TEK TEK OLACAK ŞEKİLDE.
* BURDA ŞUNU ANLAMIŞ OLDUM EĞER MODEL KULLANMAZSAK AYRI OLARAK. ATIYORUM EKLEME SAYYFASI İÇİN AYRI BİR MODEL YAPMAZSAK EKLEME İŞLEMİNDE TABLONUN TÜM ALANLARINI DOLDURMAMIZ GEREKİYOR YOKSA VALİDATİON KURALLARINA TAKILIYOR EKLEME YAPAMIYORUZ.

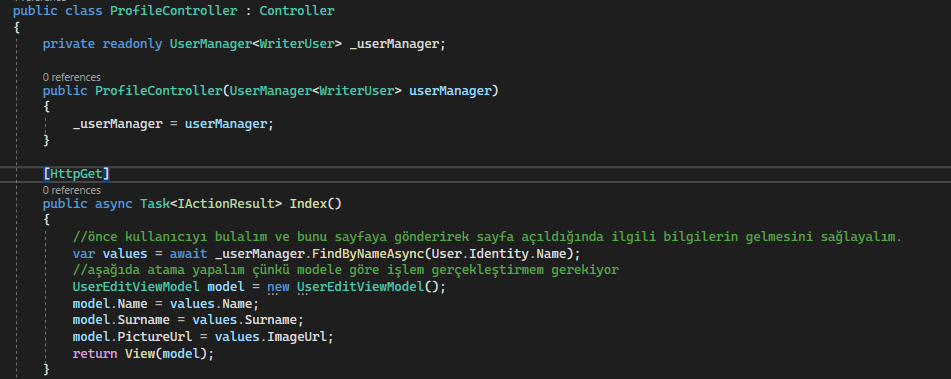
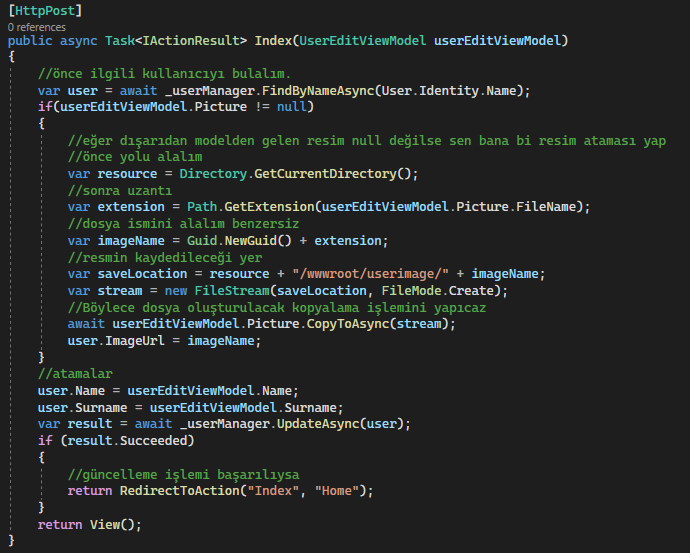
USER İŞLEMLERİ

* Mesela mesaj atma işlemi için user işlemlerine ihtiyaç duyabiliriz. Birisi mesaj atmak istediği zaman login olmalı hesabı yoksa register olmalı ancak oturum açtıktan sonra mesaj atma işlemini gerçekleştirebilir.
* Bu yapı için User tablosuna ihtiyacım olacak. Birde UserMessage tablosuna ihtiyacım olacak. Bir kullanıcının birden fazla mesajı olabilir ilişkisi kurulur.
* İlişkili tablolarda bir tabloya ait değeri diğer tarafta çağırmaya çalıştığımız zaman core yapısı buna müsade etmiyor. Include diye metot var bunu kullanmamız gerekiyor.
* Bakınız aşağıda UserMessage tablosunu model olarak kullanan bir cshtml sayfasında bu tabloyla ilişki olan User tablosunun Name alanını getirmeye çalıştığımda error veriyor. Böyle bir durumda Include metot kullanmamız gerekiyor.
* 
* Include metotu kullanıldığı zaman bu hata ortadan kalkacak böylece birleştirme işlemi gerçekleştirilmiş olacak. Böylece bir cshtml tablosu içinde birbiriyle ilişkili olan birden fazla tabloyu kullanabiliyoruz aslında. Bir tablo üzerinden diğer tabloyu kullanıyoruz.
* Bakınız burda bir önemli nokta daha var ınclude metotunda x dediğimiz zaman User tablosunun orda çıkma sebebi ilişki kurarken navigation prop kullanmış olmamız o olmasaydı burda x.User şeklinde User tablosunu çekemezdik.
* 
* Eğer belli bir alanın belli bir yerden sonra uzunluktan sonra gözükmemesini yerine ... gözükmesini istiyorsak kullanılabilecek bir yöntem aşağıdadır;
* 

AREALAR

* UI katmanımızda varsayılan olarka gelen Mvc yapısı yani Controller,Model,View yapıları dışında biz ayrı bir alanda yeni bir Mvc yapısı kurmak istersek Yeni bir area oluşturabiliriz. Yani Arealar sayesinde aynı katmanda ayrı bir çalışma alanı oluşturabiliyoruz kendimize.
* Düşünelim ki projenin birdan fazla arayüz alanı var. Baktığımız zaman hepsinin kendi içlerinde ayrı ayrı mvc gereklilikleri olacak. Ama bunların mvc gereklilikleri controller bazında değişkenlik gerektirecek. Eğer bu yapıyı kullanmazsak bir controller içinde bir çok controller kullanmış oluruz aslında.
* 
* Area kullandığımızda her alan için ayrı controller olduğu için okunma derecesi kolaylaşır daha anlaşılır bir yapı elde edilir.
* Area içlerindeki controller’ların hangi area ile çalıştığını belirtmek için aşağıdaki kullanımı yapmamız gerekiyor parantez içine area ismini yazıyoruz çünkü birden fazla areamız olabilir;
* 
* Areamızdaki controllar’ların view’ini oluştururken bu viewlar illa areanın içindeki layoutları kullanmak zorunda değiller. En dıştaki shared klasörlerini kullanabiliyorlar. Bu önemli bir nokta. Yani katmandaki tüm layoutları görebilir kullanabilirler. Ancak ViewImportu tekrar model klasörünün içine kurmamız lazım onu kullanamıyor kendi Model klasörünün içinde kendi ViewImportu olması lazım.
* Ben area kullandığımda otomatik olarak ScaffoldingReadMe.txt adında bir dosya oluştu. Bu dosya içindeki diyorki aşağıdaki ayarlamaları startup tarafında endpoint kısmına bu ayarlamayı ekleemen lazım diyor yoksa areamız çalışmaz route olayı olmadığı için sayfayı bulamıyor.

ASP NET CORE IDENTİTY

* Identity kütüphanesini projeye dahil etmek için UI katmanına ve DataAccess katmanına ve Entity katmanına Microsoft.AspNetCore.Identity paketini ve Microsoft.AspNetCore.Identity.EntityFrameworkCore paketini kuruyorum.
* Başarılı olunursa Identity paketi bizim db tarafımıza bazı tabloları otomatik olarak göndericek.
* Daha sonra mevcut context’ime gidiyorum ve diyorum ki sen artık DbContext sınıfından değil IdentityDbContext sınıfından kalıtılacaksın. Burada şunu diyelim IdentityDbContext temelinde DbContext’ten kalıtılıyor zaten. Daha sonra migration oluşturup update-database ile db tarafına gönderiyorum. Identity kütüphanesi bize belli başlı tabloları otomatik olarak db tarafına gönderiyor. Bu sayede Identity kütüphanesini başarılı bir şekilde projemize dahil etmiş oluyoruz.
* Daha sonra kullanıcı kayıt işlemi için bir model oluşturuyorum kullanıcı kayıt olurken hangi alanlara ihtiyacım varsa bu modelin içine yazacağım. Daha sonra dilersek validationlar yazabiliriz bu modele.
* Daha sonra kullanıcılarımızı karşılayacak olan bir classa ihtiyacımız var entity katmanı tarafında. Bu oluşturuğum entity’i IdentityUser’dan kalıtıyorum.
* Daha sonra birde role tablosunu karşılayacak olan bir classa ihtiyacımız var entity katmanı tarafında. Bu oluşturduğum entity’i IdentityRole’Dan kalıtıyorum.
* Daha sonra contextime giderek IdentityDbContext<> içine önce oluşturduğum karşılayacak olan user tablosunu sonra role tablosunu sonra ise tkey bilgisini veriyorum. Yani IdentityContext’te diyorum ki sen kendi user role tabloların yerine bu tablolar ile ayağa kalk user ve role tablosunu bu oluşturduğum tablolar olarak ata. Daha sonra migration oluşturarak bu değişikliği db tarafına gönderiyorum.
* Daha sonra ilgili controller’ımda dependency injection ile UserManager<> classını alıyorum ve bunun içine Tuser olarak entity’de tanımladığım classı vericem Ayrıca güncelleme işlemi içinde bu class kullanılır.
* Hata almamak için startup tarafında ConfigureServices metotu içinde services.AddIdentity<Tuser,TRole>().AddEntityFrameworkStrores<Context>() şeklinde register işlemlerini yapmayı unutmayalım. Ayrıca aynı yerde services.AddDbContext<Context>() şeklinde contexti register etmeyide unutmayalım.
* AŞAĞIDA IDENTİTY KÜTÜPHANESİ İLE GÜNCELLEME İŞLEMİ BULUNMAKTA HEMDE UPLOADLI
* 
* 
* Giriş yapma işlemi için ise giriş yaparken lazım olacak propların bulunduğu login modele ihtiyacımız var.
* Daha sonra ilgili controller içerisinde dependency injection olarak SignInManager<TUser> sınıfından bir örneğe ihtiyacımız var yani <> içine daha önce oluşturduğum User entity’ini veriyorum contexte verdiğim. Daha sonra ilgili kontroller içerisinde singIn metotumu yapıp giriş işlemimi yapıyorum;

if (ModelState.IsValid)

{

//login işlemi yapılabilir

var result = await \_signInManager.PasswordSignInAsync(userLoginModel.Username, userLoginModel.Password, true, true);

//eğer login işlemi başarılı olduysa

if (result.Succeeded)

{

return RedirectToAction("Index", "Default");

}

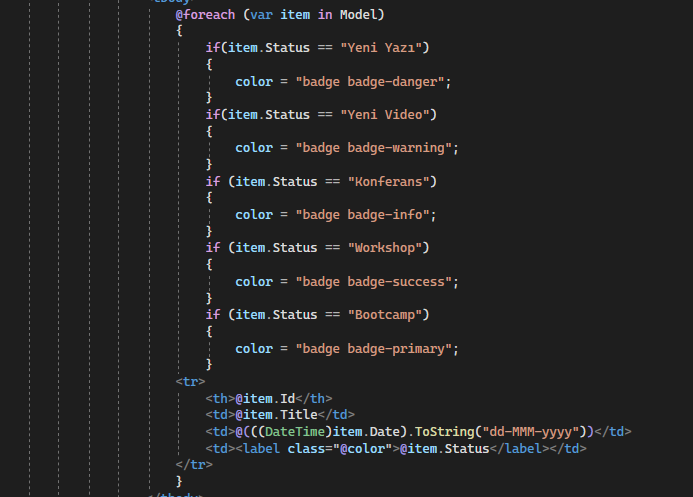
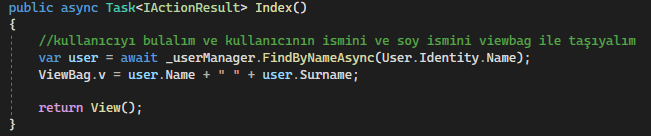
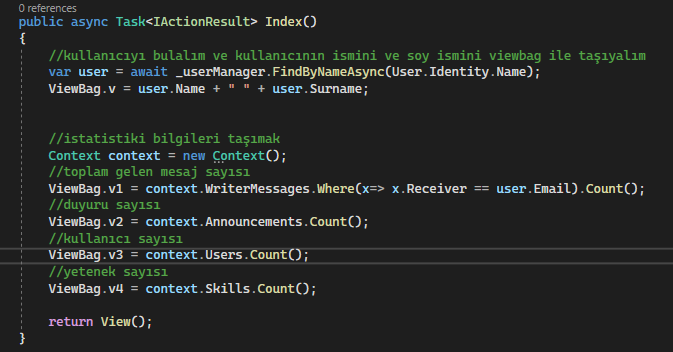
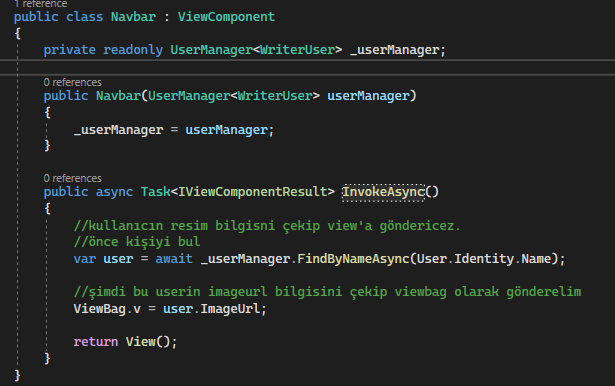
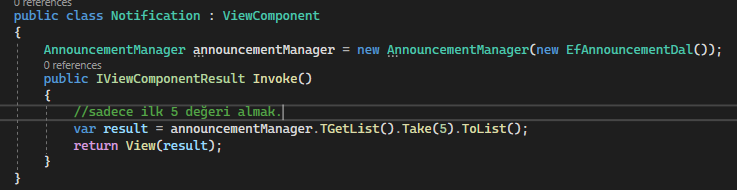
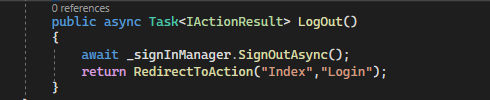
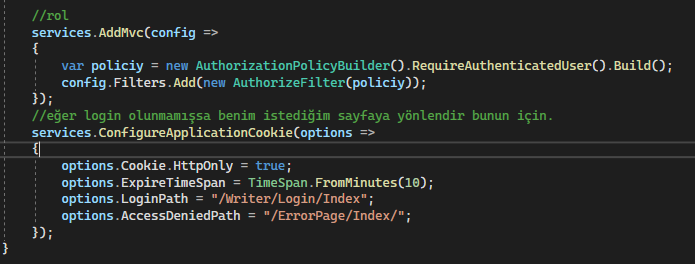
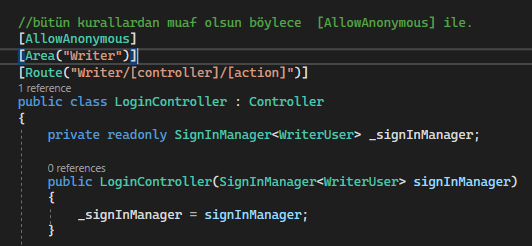
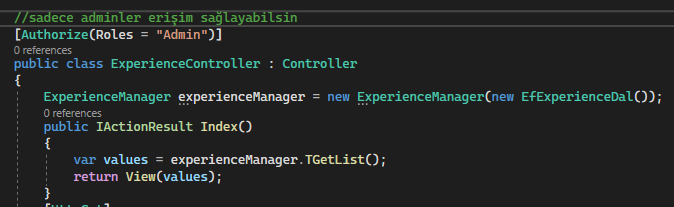
else

{

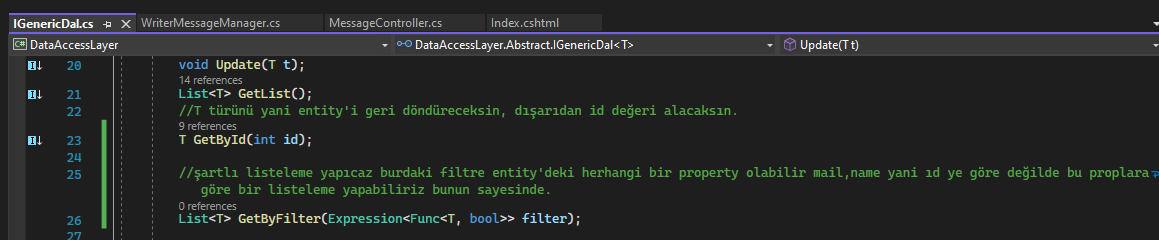
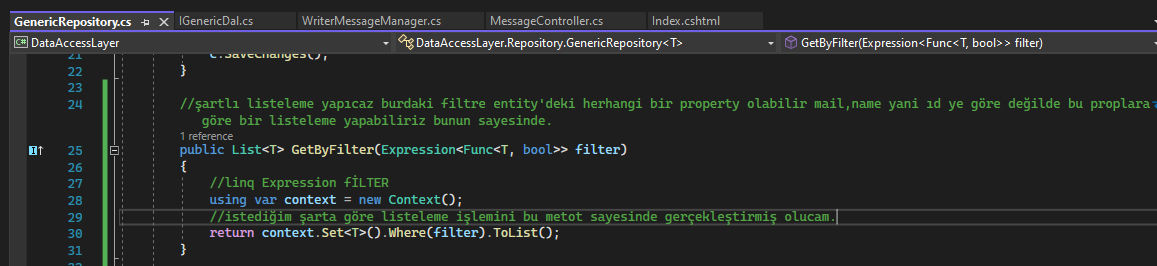
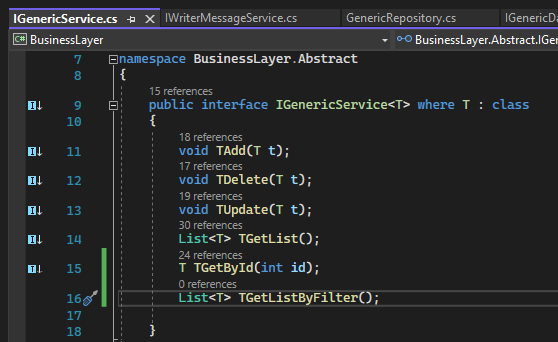
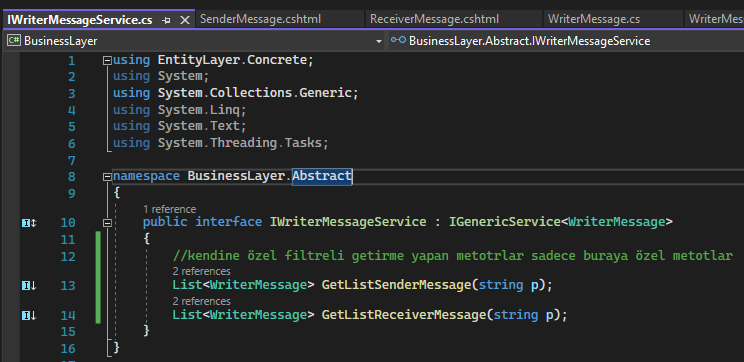
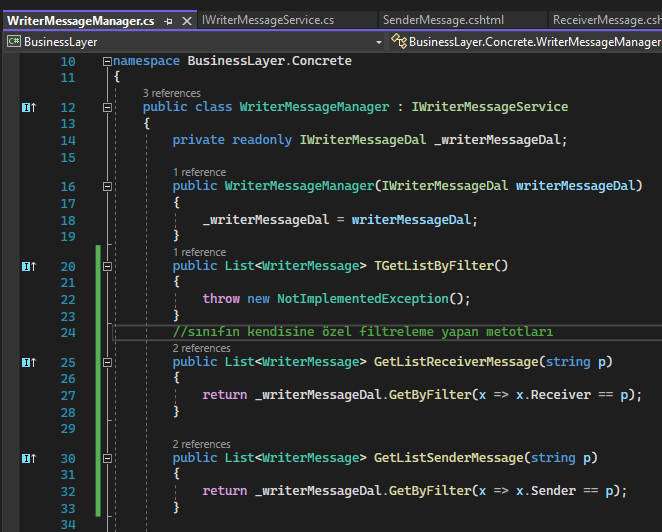
ModelState.AddModelError("", "Hatalı kullanıcı adı veya şifre");

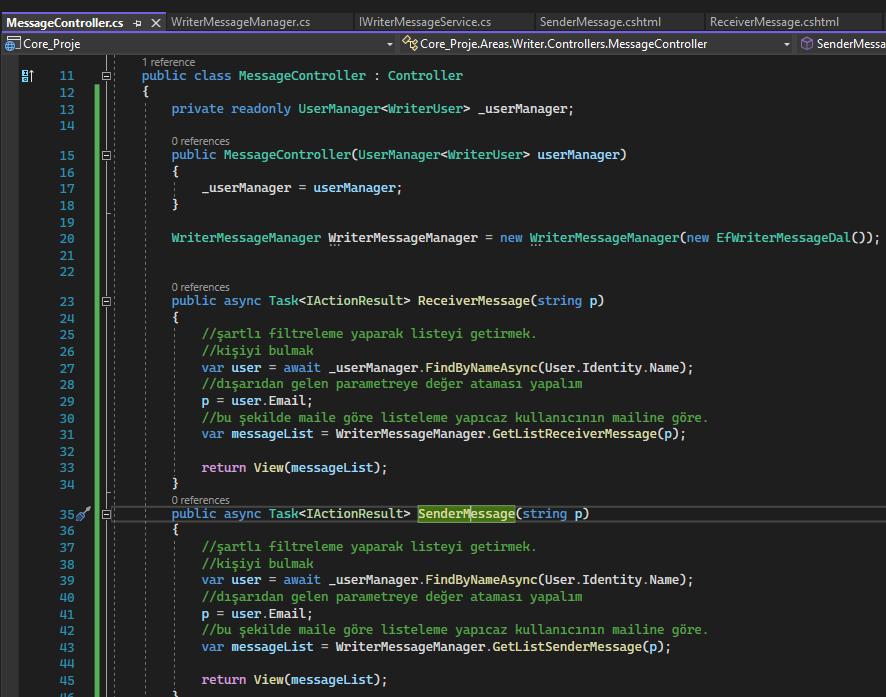
}

}

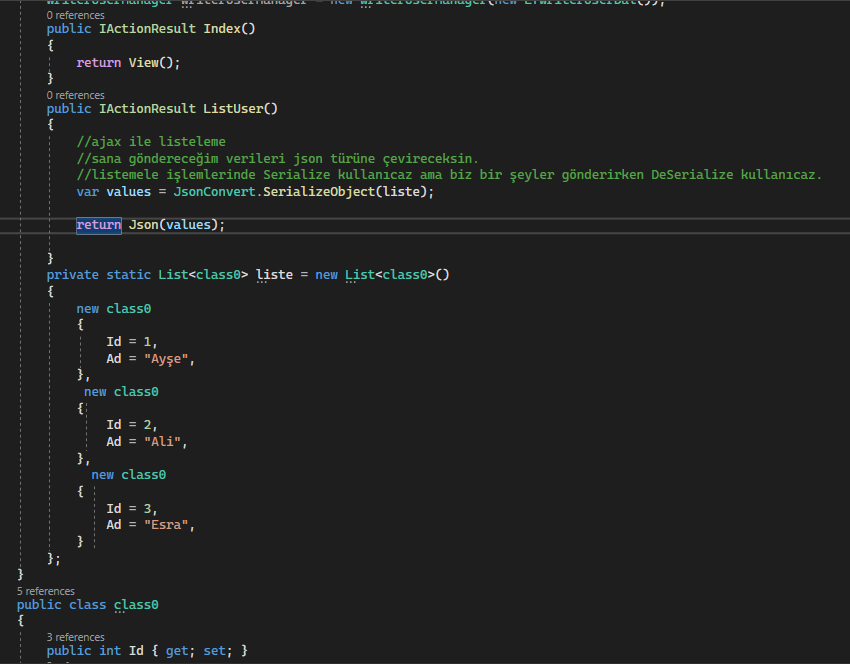
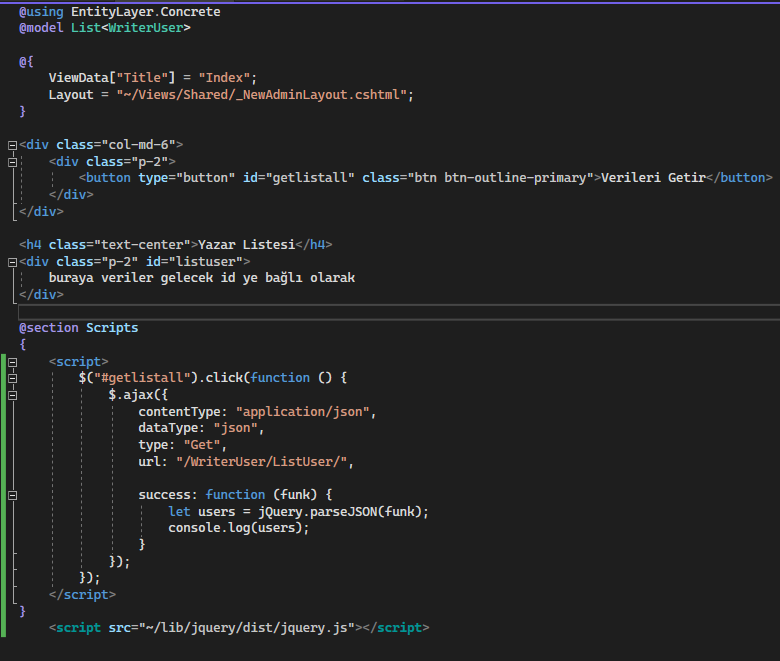
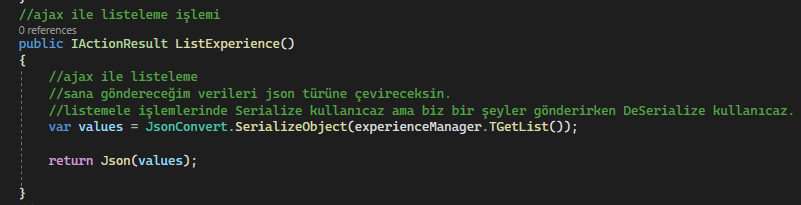
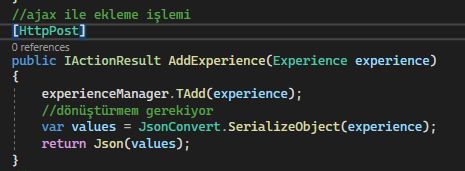
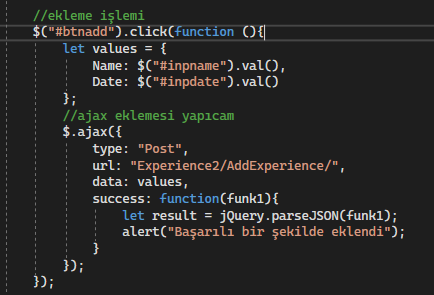
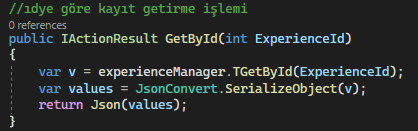
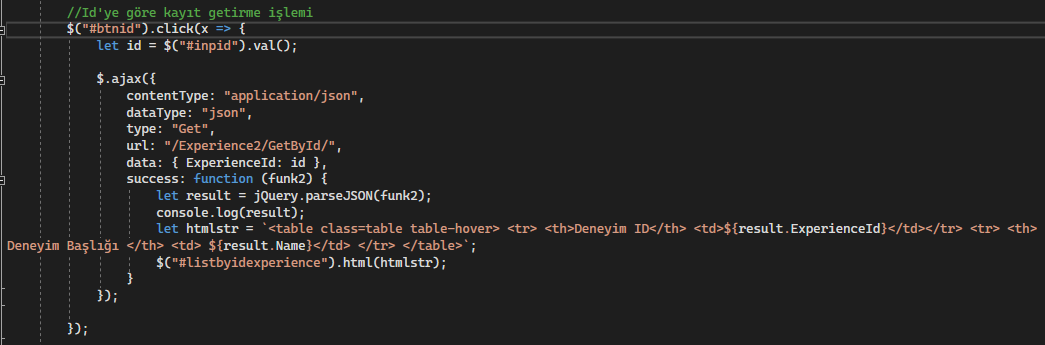
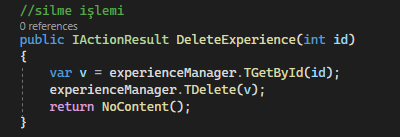
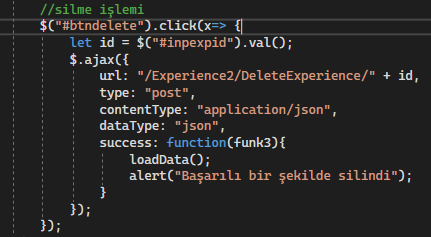
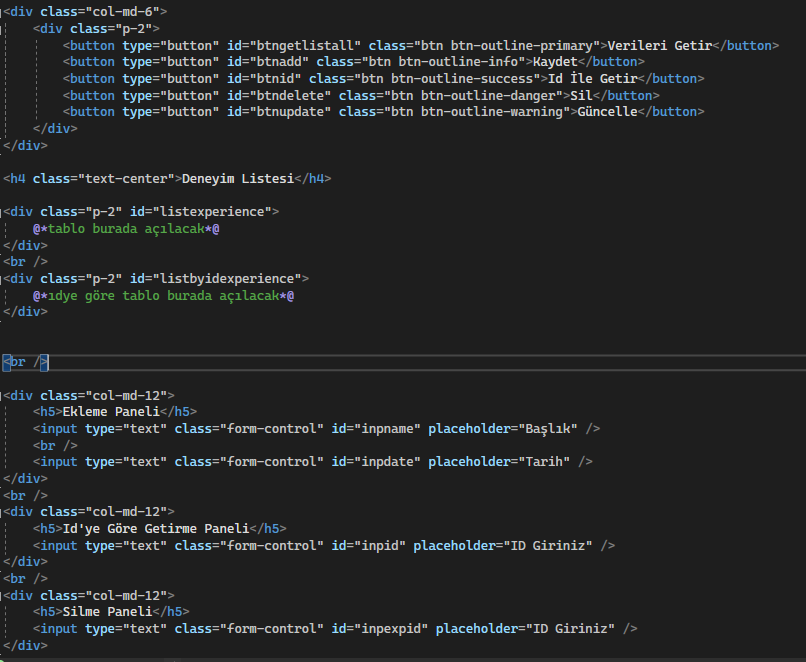
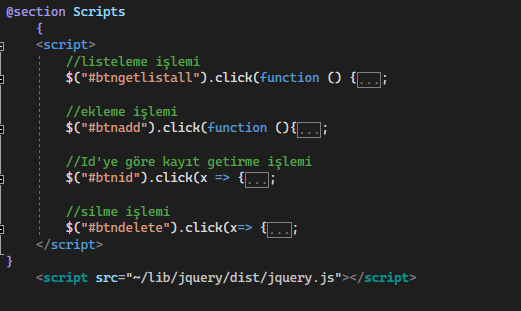
* return View();
* Yalnız bunları yaptıktan sonra veya önce startup tarafına aşağıdaki kodları routingten sonra eklemem gerkeiyor eğer eklemezsem bir sayfanın görünmesi için giriş yapılması gerek lazım dedim örneğin [Authorize] attribute’unu kullnarak ilgili action metot üstünde. Sisteme giriş yapmış olsam bile beni o sayfaya almaz o yüzden bunları startup tarafında kullanmayı unutmuyorum.
* Tarih DateTime formatını aşağıdaki resimdeki gibi ayarlayabiliriz ve dinamik olarak renklendirme yapmak için bir örnekte var içerisinde;
* 
* Kullanıcının bilgilerini View’a taşımak adına controller tarafında yapılmış bir fikir örneği;
* 
* Bazı istatistiki bilgileri View tarafına controllerdan taşımak için basit örnek aşağıda bulunuyor geliştiriebilir tabi bunlar;
* 
* Kullanıcın resim bilgisini View’a göndermek ile ilgili bir örnek aşağıda bulunuyor;
* 
* Take metotu sayesinde verinin sadece ilk 5 kayıdını liste olarak almak aşağıda;
* 
* ÇIKIŞ YAPMAK AŞAĞIDA IDENTİTY’DE;
* 
* IDENTİTYDE ROLLEME YAPMAK
* Rolleme yapmak için AspNetRoles tablosunu kullanıyorum.
* Daha sonra kullanılara rol ataması için AspNetUserRoles tablosunu kullanıyorum.
* Rolleme kullanmak için startup tarafında ve bazı controllerlarımda aşağıdaki ayarlamaları yaparım;
* 
* Aşağıdaki kısım login kısmı olacağı ve sisteme giriş yapmamış kullanıcıların göreceği ilk sayfa olacağı için tüm engellerden kurallardan muaf hale getiriyorum.
* 
* 

APİDEN BİLGİ ÇEKMEK

* Aşağıda hava durumu apisi çekilip controller içinde viewbagi le view sayfasına aktarılıyor;
* 
* ŞARTLI LİSTELEME YAPMAK FİLTER METOTU SAYESİNDE AŞAĞIDA BULUNUYOR;
* BURDA EXPESSİON KULLANMA SEBEBİBİM LAMBA EXPREESSİON İLE X=> X. ŞEKLİNDE TABLONUN DEĞİŞKENLERİNE ERİŞEBİLMEK BİR FUNC DELEGATE ARACILIĞIYLA. SONRA YAZDIĞIMIZ ŞART TRUE VEYA FALSE DÖNDÜRECEK ONDAN BOOL YAZIYORUZ.
* 
* 
* 
* 
* 



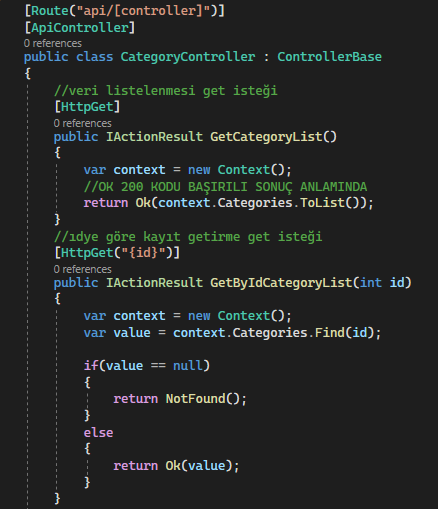
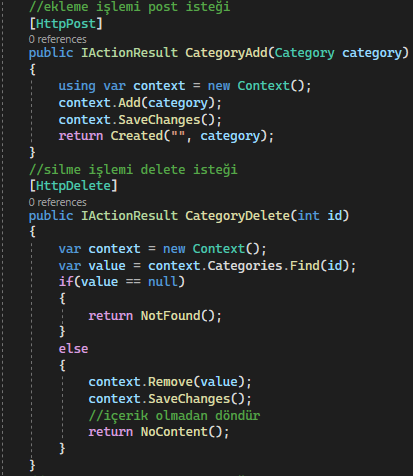
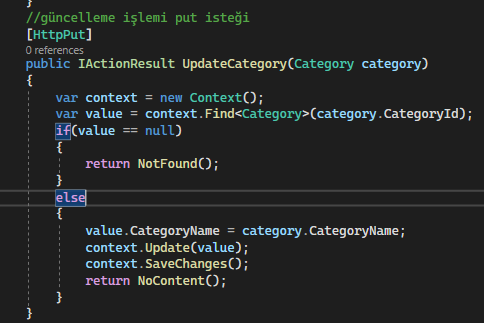
AJAX

* Ajax sayfada post işlemi gerçekleşirken sayfa yenilenmeden işlem yapılmasını sağlıyor.
* Ajaxta json türünde onun üzerinden çalışacağız.
* Ajax ile statik bir veriyi listeleme işlemi aşağıda bulunuyor;
* 
* 
* AJAX İLE VERİTABANINDAN VERİYİ LİSTELEME YANİ DİNAMİK VERİ İLE LİSTELEME
* 
* AJAX İLE EKLEME İŞLEMİ
* 
* AJAX İLE ID ‘YE GÖRE VERİ GETİRME İŞLEMİ
* 
* 
* AJAX İLE SİLME İŞLEMİ
* 
* 
* HEPSİ İÇİNCSHTML DE ORTAK OLAN NOKTA;
* 
* 

HATA SAYFALARINI ÖZELLEŞTİRMEK İÇİN STARTUP TARAFINDA;



APİLER

* Apilerle amacım: Uyguluma sadece bir web arayüzünde değil bir mobil cihazda veya benzeri başka bir cihaz üzerinden de almış olduğum verileri veya oraya göndereceğim verileri işlem gerçekleştirebilemmi sağlıyor. Yani bağımlılığımı azaltıp başka cihazlardan veri transferi veya başka cihazlara veri transferi gerçekleştirmek istiyorum. Kullanılabilecek yer olarak bir e ticaret sitesinin size sağladığı api sayesinde satışlarınızı kendi panelinize çekebilirsiniz.
* Visual studio ile gelen Swagger adında bir eklenti var apilerde. Apilerim üzerinde Temel CRUD işlemler başta olmak üzere bir takım işlemleri gerçekleştirmemi sağlayan bir eklentidir. Burası JSON tipinde verilerle işlem gerçekleştiriyor.
* 
* 
* 
* Using anahtar sözcüğü otomatik olarak Dispose metodunu uygular. Dispose ile öğe kullanıldıktan hemen sonra bellekten atılır. Dispose işleminin gerçekleşmesi için IDisposable arayüzü ile kontrat imzalaması gerekir. DbContext sınıfı da IDisposable arayüzü ile kontrat imzaladığı için using anahtar sözcüğü ile çağrılarak bir nevi kaynakları serbest bırakma işini garbage collectorden devralıp, iş biter bitmez yıkıcı metodun tetiklenmesini sağlıyoruz. Çoğu zaman garbage collector yıkıcı metodun ne zaman tetikleneceğini kestiremez.