**Asp.Net Core Wep API ve Asp. Net Core MVC ile Basit Sosyal Medya ICRUD işlemleri**

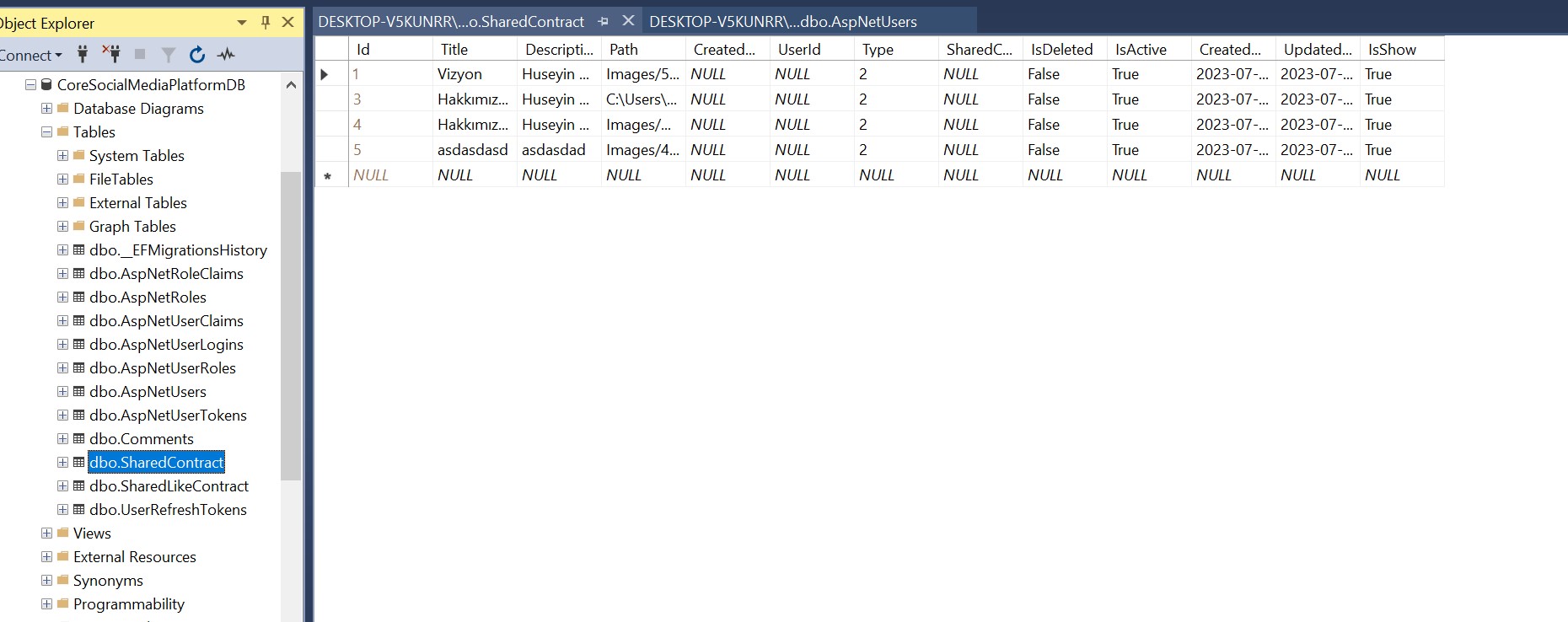
**Kullanılan Veri Tabanı:** MSSQL

**Kullanılan Yazılım Dili:** C#

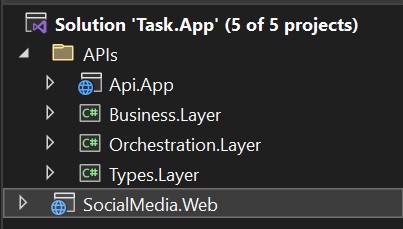
**Kullanılan Frontend Teknolojileri:** Html, Css, Bootstrap, Javascript, Jquery

**Proje Hakkında**

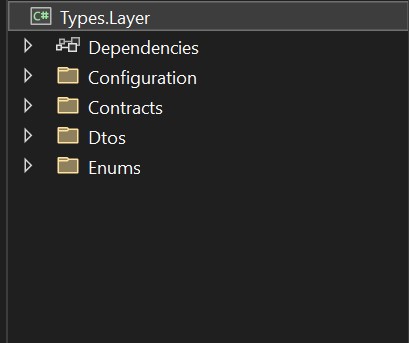
Asp.Net Core Web API projesi EntityFramework Code First yaklaşımı ile oluşturuldu. Bunun için öncelikle gerekli olan tablo yapısına uygun entityler Types katmanında oluşturuldu. Daha Sonra Business katmanına geçilerek burada IdentityDbContext bağlantısı oluşturuldu ve FluentAPI ile veri tabanında bulunacak olan tabloların sutunları uygun bir şekilde kodlandı. Veri tabanından veri çekme ve veri gönderme gibi işlemlerin bulunduğu Repository işlemleride bu katmanda gerçekleştirildi. UnitOfWork Patern i projeye dahil edilerek, gerçek business kodlarının bulunacağı Orchestration katmanına geçildi. API için gerekli olan Serviceler ve Token bazlı kimlik doğrulama için gerekli olan classlar bu katman içerisinde oluşturuldu. Orchestration katmanında bulunan methodlarEntity değil Dto alacak ve verecek şekilde düzenlendi. Bu katman tamamlandıktan sonra Program.cs kısmına gerekli olan Kimlik doğrulama kodları ve servisler eklendi ve Endpointler tasarlandı. Oluşturmuş olduğumuz Web API projesinin oluşturmuş olduğumuz MVC projesi üzerinden tüketimi sağlandı.



*Migration işlemi sonrası oluşan tablo yapısı*



*Proje katmanları*



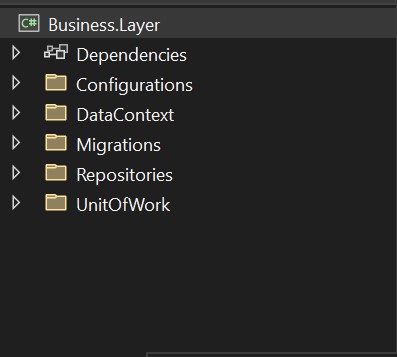
*Types Katmanı*

Bu katmanda bulanan Configuration klasörü içerisinde Client ve CustomTokenOption classlarını ekledim. Client classının görevi “Web API projemize hangi Uygulamalardan istek geliyor ve gelen uygulamalar benim hangi API larıma erişebilir” bilgilerini tutmaktır. CustomTokenOption ise kullanıcılardan gelen istekleri ve istekte bulunan kullanıcıların hangi API larımıza erişeceği bilgilerini tutar.

Contracts klasörü içerisinde Entity ve Interfacelerimiz bulunmaktadır. Bu Proje için AppUser, Shared, SharedLike, Comment ve UserRefreshToken olmak üzeri Toplam 5 Entity ve IdentityUser kütüphanesinin sağlamış olduğu tabloları kullandık.

Dto klasörü içerisinde Entitylerimizin dış dünyaya açacağı propertylerine özel olarak classlar oluşturduk. Bu classları diğer classlardan ayırmak için isimlendirmelerini yaparken sonlarına Dto ekledik.

Enums klasörü içerisinde proje içerisinde kullanmış olduğumuz Enumlara yer verdik. Bu proje için bir tane Enum kullandık oda EFileType. Bu enumı kullanma sebebimiz Eklenen gönderi bir video mu resim mi yoksa düz yazımı bunu belirlemekti.



*Business Katmanı*

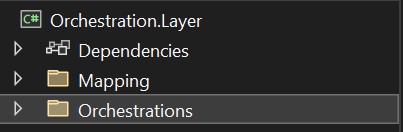
Business katmanı içerisinde bulunan Configurations klasörü içerisinde Entitylerimizin veri tabanındaki kısıtlarının, formatlarının ve index ayarları için gerekli işlemler gerçekleştirildi.

DataContext klasörü içerisinde veri tabanı bağlantısı gerçekleşti ve yazmış olduğumuz Configuration ların implementasyonu Reflection sayesinde gerçekleştirildi.

Migrations klasörü içerinde proje boyunca almış olduğumuz migrationlar bulunmaktadır.

Repositories klasörü içerinde veri tabanında yapılacak olan işlemlerin class ve methodları bulunmaktadır. Bu classlar için daha önce Contracts klasörü içerinde interfaceler oluşturmuştuk. Bu interface ve classları ise Web API nin program.cs kısmında service olarak geçerek constructor içerisinde interfaceler üzerinden classların kullanılmasını gevşek bağlı bir şekilde gerçekleştirmiş olacağız.

UnitOfWork klasörü içerisinde savechange işlemi için gerekli olan metodları yazdık. Savechange işlemini bazı yazılımcılar Repository katmanında gerçekleştirebilmektedir. Fakat bizim business kodlarımız Orchestration katmanındadır ve 1 metod içerisinde aynı anda birden fazla tabloya veri ekleyebilmekteyiz. Dolayısıyla bu işlemlerin herhangi birinde hata olduğunda işlemin gerçekleşmesini istemeyiz. Bundan dolayı kaydetme işlemini tek seferde gerçekleştirmek için UnitOfWork pattern ini kullandık.

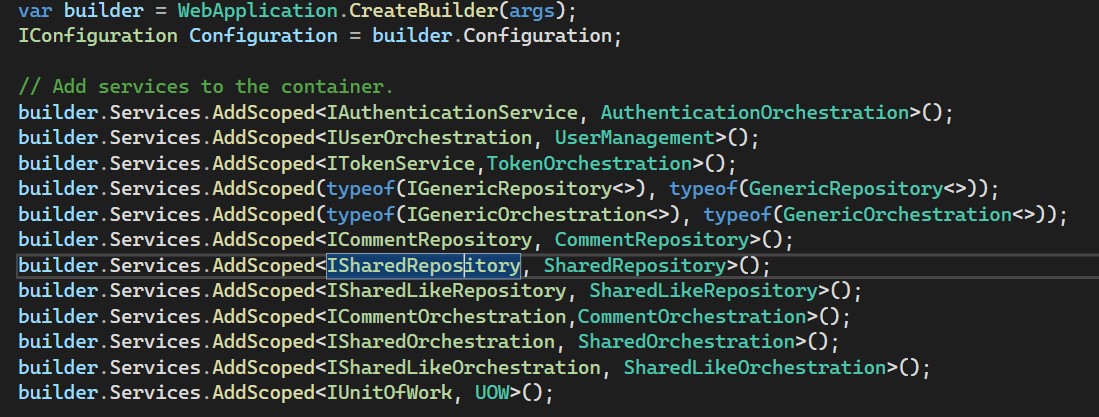


*Orchestration Katmanı*

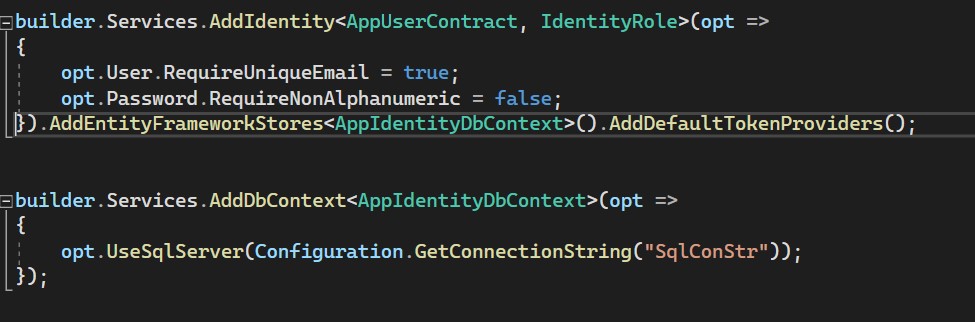
Bu katmanda bulunan Mapping klasörü içerisinde oluşturmuş olduğumuz Entity ve Dtoların birbirlerine otomatik maplenmesi için gerekli olan DtoMapper ve Bu classımızın bellekte gereksiz yer tutmasını engellemek için gerekli olan ObjectMapper metodu bulunmaktadır. Gereksiz yer tutmasını engellemek için Lazy classı kullanılmıştır.

Orchestration klasörü içerisinde business kodlarımızı bulunduran classlar ve token bazlı kimlik doğrulama için gerekli olan classlar ve metodları bulunmaktadır.

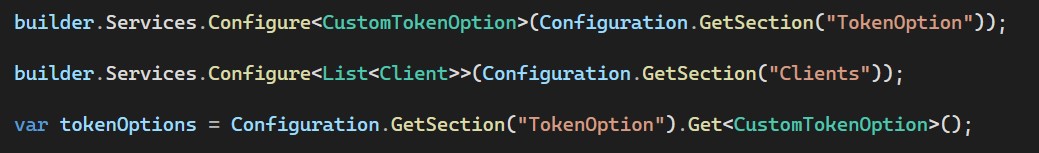
Ayrıca bu katmanda neler bulunabilir diye sorduğumda aklıma FluentValidation kütüphanesi gelmekte, bu kütüphane bize ne sağlar? diye soracak olur isek. FluentValidation ile oluşturmuş olduğumuz Validation işlemleri tamamen server bazlı işlem görür. FluentValidation kütüphanesi ise Client bazlı çalıştığından dolayı validation işlemlerinde server maliyetinden kurtulmamızı ve daha performanslı uygulamalar geliştirmemize olanak sağlar.



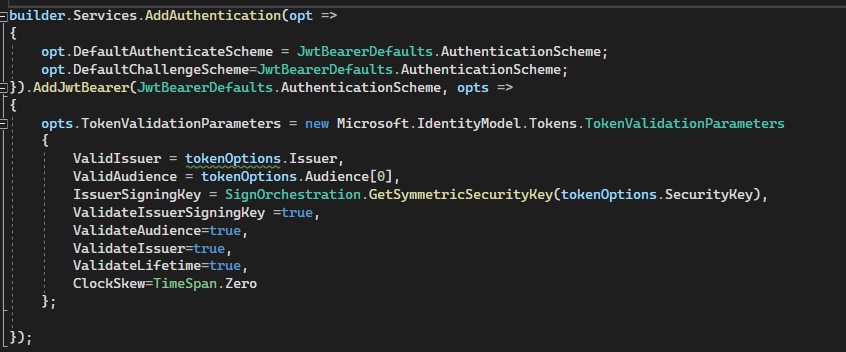
*Program.cs içerisinde tanımlanan servisler*



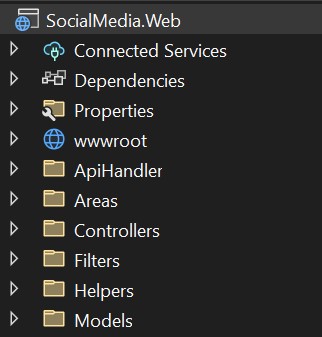
*IdentityUser için gerekli classlar ve AppIdentityDbContext classının service olarak eklenmesi*



*Option Pattern ile App.setting içerisine eklediğimiz verilerin otomatik maplenmesi*



*Authorize işlemi için gerekli kodlar*



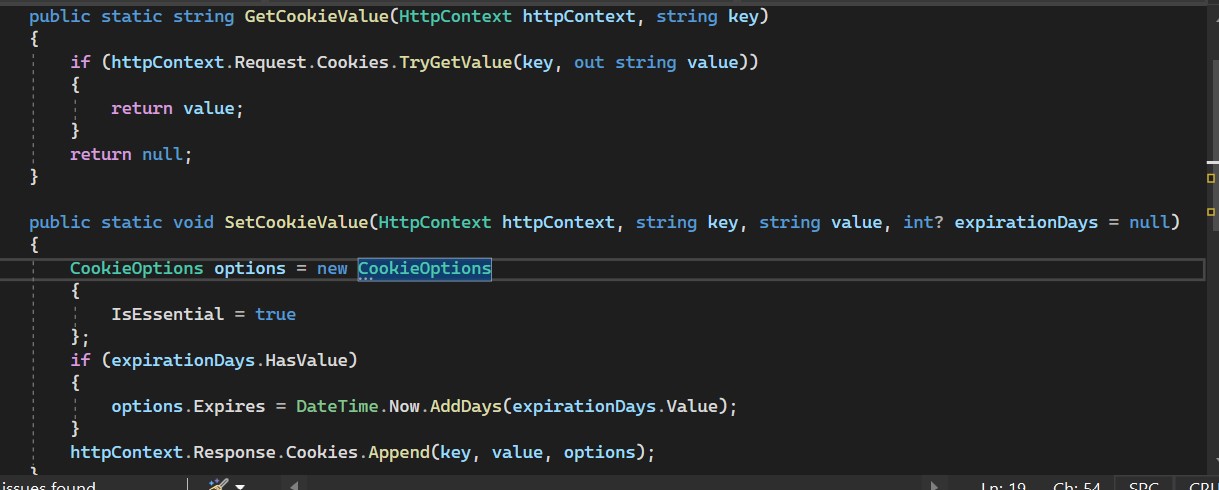
*Asp.Net MVC Proje Dosyaları*

MVC projemizde bulunan ApiHandler Klasörü içerisinde API ile haberleşmemizi sağlayan RestApiHandler sınıfını tutarız. Bu sınıf içerisinde bulunan generic metodlar sayesinde tek bir class kullanarak bütün endpointlere erişebilmekteyiz.

Areas Klasörü içerinde Manager kısmını oluşturuyoruz. Manager kısmını normalde sadece Admin görecek şekilde düzenleyebiliriz. Fakat bu projede bunu çok önemsemedim.

Filters klasörü içerisinde controller ve actionlarımızın üzerine yazabileceğimiz Attributeleri bulunduruyoruz. Bu proje içerisinde şuanlık 1 tane Filter kullanıyoruz. LoginFilter , giriş yapılıp yapılmadığını kontrol ediyor.

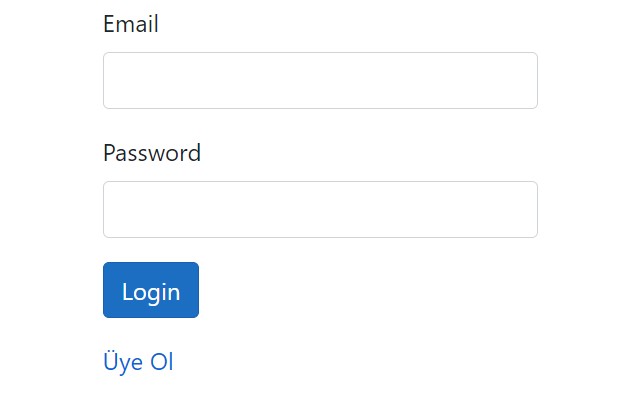
Helpers klasörü içerisinde projede kullanacağımız yardımcı classları ve metodlarını ekliyoruz. Bu proje içerisinde Helper olarak cookie okumak ve cookie ye yazmak için kullandığımız CookieHelper classını kullandık bu class içerisinde;



Metodları bulunmaktadır.

**Proje Sayfaları**

Proje geliştirilirken UI biraz daha geri planda tutularak geliştirme yapıldı.

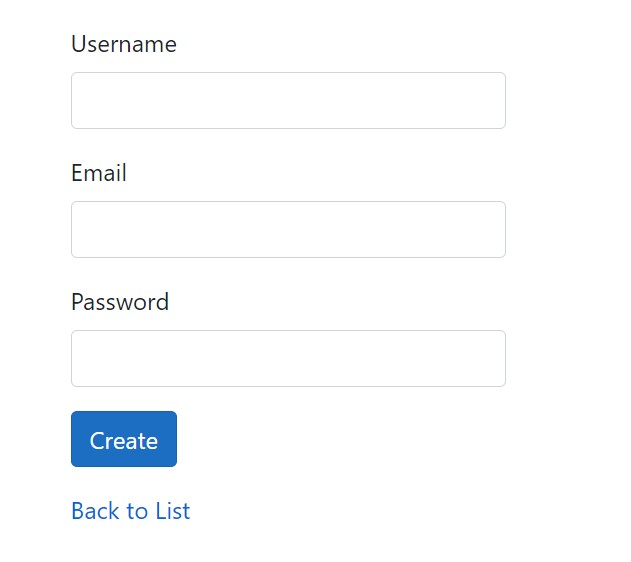


*Login Sayfası*

Login Sayfasında giriş yaptığımızda oluşturmuş olduğumuz API bizim için bir Token üretmektedir. Örnek Token ; eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c

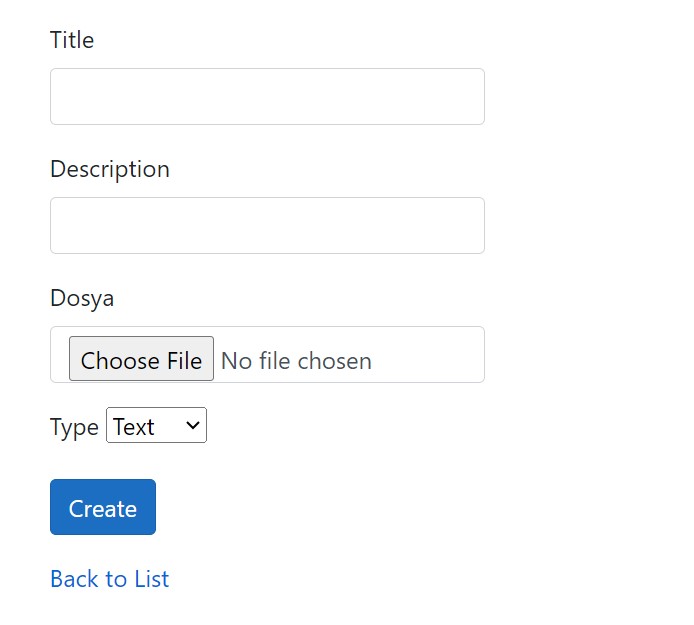
Jwt Token genelolarak 3 bölümden oluşur. İlk bölüm Header(başlık), ikincisi Payload(yük), üçüncü ve son olarakta signature(imza) bölümüdür. Bu konu oldukça derin bir konu o yüzden projeye dönmek daha mantıklı.

Login işleminden sonra üretilen token ile API ımızda bulunan yetki gerektiren endpointlere erişim sağlayabilmekteyiz.



*Üye Ol Sayfası*

Genellikle Web sitelerinde ilk üyelik aşamasında kullanıcıyı sıkmamak adına gerekli temel bilgiler istenir daha sonra isteğine göre bilgileri güncellemesi sağlanır. Bizde üye ol sayfamızı ona göre düzenledik.



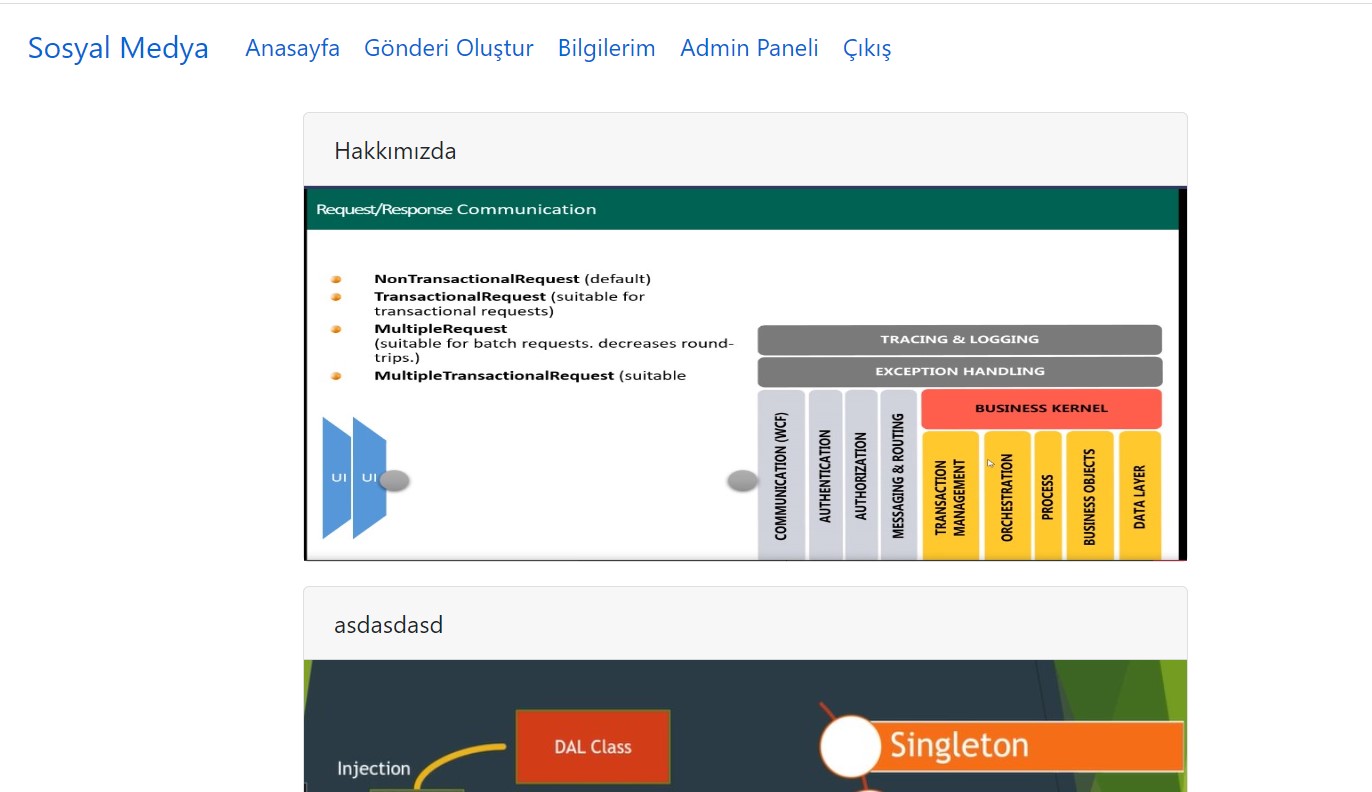
*Gönderi ekleme sayfası*

Bu sayfanın erişimi giriş yaptıktan sonra gerçekleşmektedir. Ayrıca gönderi eklemek içinde API a geçerli bir token göndermemiz gerekir.



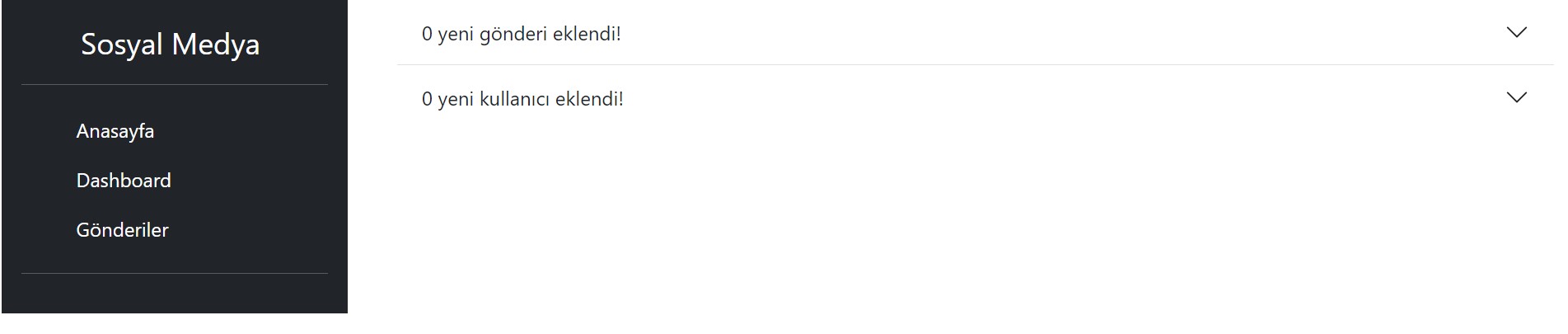
*Profilim Sayfası*

Bu sayfada solda bulunan form üzerinden kullanıcı bilgileri güncellenebilmektedir. Sağ tarafta ise kişinin gönderileri görünmektedir.



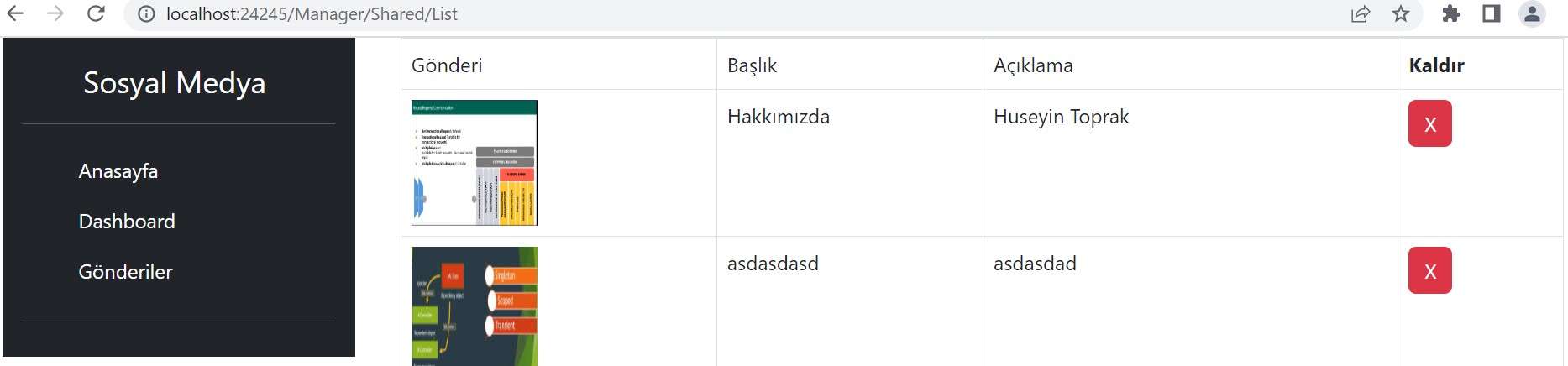
*Home/Index sayfası*

Bu sayfada bütün gönderiler listelenmektedir.



*Dashboard/Index sayfası*

Veri tabanında bulunan shared ve user tablolarında IsShow adında bir alanbulunmaktadır. Bu alanın amacı sisteme yeni bir gönderi eklendiğinde veya yeni bir user eklendiğinde admini haberdar etmektir. Bildiri hazırlamaktır. Bu bildiriler admin Dashboard a girdiği anda false durumundan true ya geçmektedir.



*Manager/Shared/List*

Bu sayfada bütün gönderiler listelenir ve admin uygunsuz gördüğü gönderileri silebilir.