23 HAZİRAN 2021

ÖDEV PROJE

PROGRAM TANITIM DÖKÜMANTASYONU

ZÜLKÜF EMRE YILMAZ

GITHUB/ZEYLMZ

İçindekiler

Projeyi Tanımaya Başlayalım

Projenin Amacı

Bize verilen Türkçe ve İtalyanca veri içeren Json dosyalarını sisteme import etmek, kullanıcıların milletlerine göre bu verileri sunmaktır. Türk ve İtalyan kullanıcılarımıza ayrı ayrı veri desteği sağlamaktayız.

Projenin İşleyişi

Dışarıdan gelen Türkçe ve İtalyanca verileri sistem içerisine çekebiliyoruz. Bu verileri veri tabanına kaydedebiliyoruz, kayıtlı veriler üstünde filtreleme tarzı işlemler yapabiliyoruz, toplu halde verileri güncelleme ve silme işlemleri yapabiliyoruz aynı zamanda amacımız olan konum bulma konuma göre veri gösterme gibi işlemleri de sistemimizde kullanabiliyoruz.

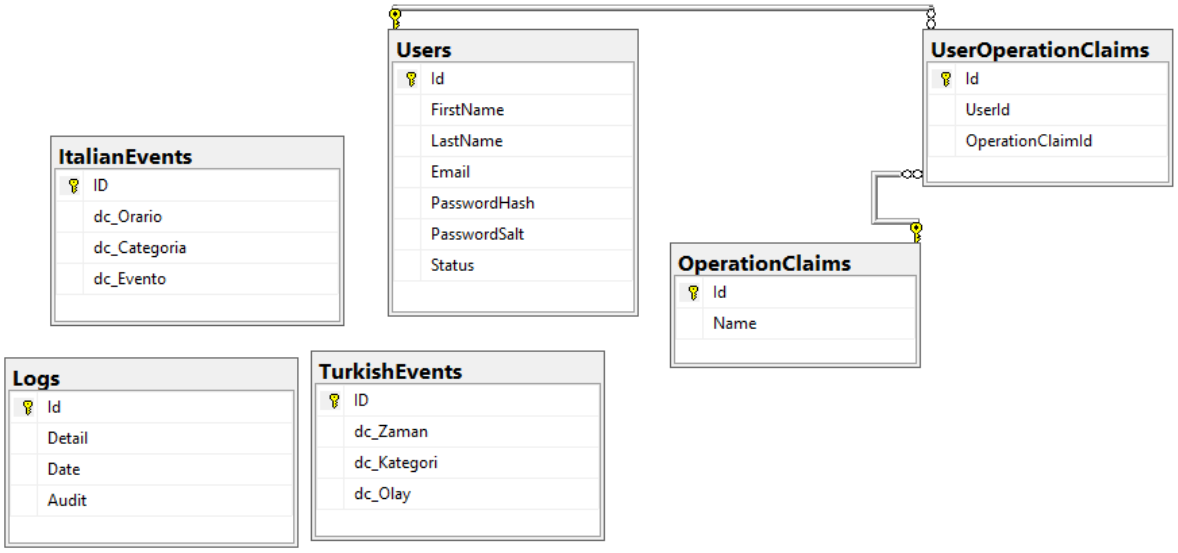
Projede Kullanılan Teknolojiler ve Araçlar

- Asp .Net Core (İstenilen)
- Entity Framework Core (İstenilen)
- MSSql Database (İstenilen)
- Autofac
- FluentValidation
- Log4Net
- IpData

Projede Kullanılan Yapılar

- Request Localization (İstenilen)
- Authentication (İstenilen)
- Cache InMemory (İstenilen)
- Exception Handler (İstenilen)
- ObjectMapping (İstenilen)
- Validation (İstenilen)
- Code Complexity (İstenilen)
- Code Readability (İstenilen)
- Multi-Tenancy (İstenilen)
- Logging
- Performance

Projenin Veri Tabanını Tanıyalım

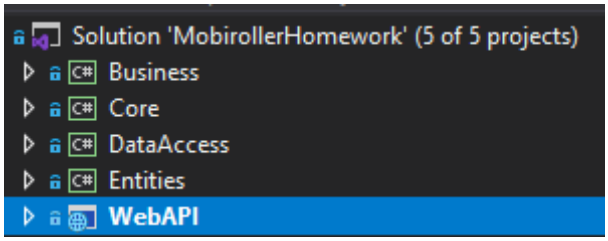


Tablolar yukarıdaki görseldeki gibidir.

- **Json** üzerinden çekilen verilerin tabloları **ItalianEvents** ve **TurkishEvents** olarak ayrılmıştır.
- **Logs** sistemde **Logging** kullandığımız işlemlerin hareket kaydını tutmaktadır. Aynı zamanda **Exception** kayıtlarını da bu alanda tutabiliriz.
- **Users** kullanıcılarımızın verilerini tuttuğumuz bölüm.
- **OperationClaims** sistemde eklemek istediğimiz rolleri ve yetkileri belirlediğimiz tablodur.
- **UserOperationClaims** kullanıcılara atanan yetkileri göstermektedir. Kullanıcının yetkisi var mı? Tarzı sorgular bu alanda gerçekleşir.

Projenin Katmanlarını Tanıyalım

Projeyi elimden geldiğince SOLID prensiplere uygun olarak geliştirmeye özen gösterdim. Aynı zamanda katmanlı mimari kullanarak nesneye yönelimli olarak geliştirmeye çalıştım.



- **Business:** İş katmanımız tüm kuralları doğrulamaları servisleri bulundurduğumuz alandır. Sistem içerisine eklenecek olan API – MVC vb. tüm Frameworkler buradan kuralları alırlar. Referans aldığı katmanlar (Core – DataAccess – Entities).
- **Core:** Sistemin beyni diyebiliriz veya sistemin Framework tabanı da diyebiliriz. Sistem içerisinde kullanılan ve her projede kullanılabilecek düzeyde genel extensions, helpers, tools tarzı işlemleri bünyesinde barındırır. Referans Aldıkları (Yok)

- **Data Access:** Verilere eriştiğimiz kaydettiğimiz, sildiğimiz ve güncellediğimiz bölümdür. ORM teknolojilerini kullandığımız ana katmandır. Referans Aldıkları (Core – Entities)
- **Entities:** Veri tabanı içerisinde bulundurduğumuz tabloları nesne, obje haline getirdiğimiz alandır. Tek kişi varlıklarımızı belirtmektir. Referans Aldıkları (Core)
- **WebAPI:** Mevcut projemizin ara yüzü diyebiliriz. Dışarıya aktaracağımız Business içeriklerini çağırdığımız ve kullanıcıya sunduğumuz katmandır. Referans Aldıkları (Business – Core – Entities)

Entities Katmanını Tanıyalım

Veri tabanı tablolarımızı nesneleştirdiğimiz alandır. Bu alanda tablo nesneleri dışında DTO gibi kavramları da burada oluşturuyoruz.

Concrete

Somut olan nesnelerimizi barındırdığımız alandır. Veri tabanında bizzat karşılığı olan tabloları veya bize gerekli olacak varlıkların somut hallerinin yazıldığı bölümdür.

Sistemimize örneğin Almanca verileri de eklediğimizi düşünelim. Veri tabanında Almanca verilerle ilgili oluşturulacak tablonun nesnesini buraya açacağız.

Geliştirici Notu: Bu sistemi geliştirmeye başladığımız ilk alandır.

ItalianEvent – Turkish Event

Veri tabanında bulunan ItalianEvents ve TurkishEvents tablolarına karşılık gelmektedir. IEntity implementasyonuna sahiptirler. Bu tip varlık nesnelerinin tek özelliği nesne olmalarıdır.

DTOs

Data Transfer Object olarak geçen Veri aktarım nesnelerini burada tanımlıyoruz.

Geliştirici Notu: Sadece DTO türündeki nesneleri burada oluşturunuz.

UserForLoginDto – UserForRegisterDto

Bu tablolarda da göreceğiniz üzere biz veri tabanımızda string olarak password değerini tutmamaktayız. Ancak kullanıcıyı kaydederken veya giriş yaptırırken bu password değerine ihriyacımız var. Encyription ve Hashing işlemleri için gerekli. Dolasıyla bu varlıkları kullanıyoruz. Her iki varlığın ortak özelliği IDto implementasyonuna sahip olmasıdır.

DataAccess Katmanını Tanıyalım

Veri tabanına bağlantı yaptığımız ORM teknolojilerini kullandığımız katmandır. Bu katmanın özelliği Business tarafından istenen Data olaylarını gerçekleştirip ona istediği Datayı vermektedir.

Abstract

Bu katmanın soyut nesnelerini buraya yazmaktayız. Veri erişim katmanlarımızın ana operasyonlarını burada tutuyoruz. Concrete klasörü içerisindeki Teknolojilere dayalı oluşturacağımız sınıflara bu operasyonları implemente ediyoruz. Dolasıyla bizim operasyonel süreçlerimiz asla değişmiyor.

IUserDal – IItalianEventDal – IturkishEventDal

Bu objelerin ortak özelliği Entities katmanında oluşturduğumuz varlıklarla ilişkilerinin olması ve IEntityRepository interfacesini inherit etmeleridir. Her Dal nesnesinde kullanılan Ekle – Sil – Güncelle tarzı verilerin hepsini IEntitiyRepository üzerinde tanımlanmıştır. Dolasıyla bu dosyalara sadece kendilerine özel operasyonları yazıyoruz. Örneğin IUserDal üzerinde olan GetClaims operasyonu gibi.

Concrete

ORM teknolojilerinin işlemlerini burada yapmaktayız. Burada oluşturulan teknolojilere bağlı sınıflar kendilerine ait olan soyutu implemente eder.

EntityFramework

Burada kullandığımız teknolojiye ait kodları yazdığımız bir alan. Bu projede bu teknolojiden yararlandık.

Context

Veritabanlarımızı eşleştirdiğimiz tanımladığımız bölümdür. Bu bölüme oldukça dikkat edilmesi gerekmektedir.

EfItalianEventDal – EfTurkishEventDal – EfUserDal

Bu nesnelerin hepsi EfEntityRepositoryBase sınıfını inherit etmiştir. Temel tüm işlemler EfEntityRepositoryBase sınıfı içerisindedir. Aynı zamanda kendilerine ait soyut sınıfı implemente ederler.

Örneğin EfUserDal içerisinde kendisine özel olan GetClaim içeriğinin tanımlanması gibi. Ortak olanlar EfEntityRepositoryBase üzerinden geliyor. O sınıfa bağlı içerikler bu gibi nesnelerin soyutundan geliyor.

Business Katmanını Tanıyalım

Bu katman bizim tüm iş süreçlerimizi kurallarımızı doğrulamalarımızı yaptığımız alandır. Tüm operasyonları buradan yönetiyoruz. Son kullanıcıyla Veri arasındaki katmanımızdır.

Abstract

Servislerimizi bu alanda oluşturuyoruz. Burada oluşturduğumuz servisin somut karşılığını yazmamız gerekmektedir. Aynı zamanda buradaki servislerin isimlerine göre API veya MVC katmanlarında Controller sınıflarını oluşturuyoruz.

IAuthService – IItalianEventService – ITurkishEventService – IUserService

- **IAuthService:** Auth işlemleriyle alakalı operasyonları belirlediğimiz alandır.
- **IItalianEventService:** İtalyanca olaylarla alakalı operasyonları belirlediğimiz alandır.
- **ITurkishEventService:** Türkçe olaylarla alakalı operasyonları belirlediğimiz alandır.
- **IUserService:** Auth işlemleri sırasında Claim çekmek kullanıcı eklemek ve aynı kullanıcıdan bir tane daha var mı gibi operasyonları belirlediğimiz alandır.

Concrete

Servislerimizin içindeki operasyonları implemente edeceğimiz Manager sınıflarını burada oluşturuyoruz. Tüm operasyonların kodları burada yazılmaktadır. İş katmanının kalbi niteliğinde bir alandır.

AuthManager – ItalianEventManager – TurkishEventManager – UserManager

Buradaki katmanlarda iş kurallarımızı, caching, validate, logging gibi tüm yapıları burada belirtiyoruz. Bu sınıf içerisindeki kuralları yazarken BusinessRules adlı araçtan faydalanabilirsiniz.

Business Aspects

Sadece mevcut projemizle alakalı olan Aspectleri tuttuğumuz alandır. Tüm projelerde kullanılabilecek düzeydeki Aspectler Core katmanındadır.

Autofac > SecuredOperation

Attiribute tarzı işlemlerde ücretsiz olması ve kullanım rahatlığı açısından Autofac teknolojisini kullanıyorum.

Autofac klasörü içerisinde SecuredOperation adlı Aspectimiz bizlerin Manager üzerindeki operasyonlara şu yetkideki kişiler hariç kimse kullanamaz diyebilmemizi sağlar.

```
[SecuredOperation("admin")]
2 references | zeylmz, 1 day ago | 1 author, 2 changes
public IResult UpdatingDataJson()
{
    List<ItalianEvent> italianEvent = JsonHelper<ItalianEvent>.LoadJson(url);
    _italianEventDal.RemoveRange(italianEvent);
    return new SuccessResult(Messages.UpdatedDataFromSourceSuccessful);
}
```

Constant > Messages

Business operasyonel işlemleri sırasında döneceğimiz mesajları buradan dönebiliriz. Çoklu dil destekli programlar yapacaksak eğer. Burası oldukça işimize yarayacaktır.

DependencyResolvers

Bağımlılık çözümleyicilerimizi burada belirtiriz. Sürekli her fonksiyon öncesi ve sonrası new() derdinden bizi kurtarıyor.

Autofac > AutofacBusinessModule

Microsoft'unda kendisine ait bağımlılık çözümleyicileri olsa da ben Aspect oluşturmak içinde Autofac tercih ettiğimden dolayı Autofac kullanıyorum. Burada bize avantajı İnstense bir kere yaratmak ve o yaratılan instense'i sürekli kullanmamızı sağlar.

Örneğin her servisi manageri kullanırken sürekli newlemek yerine tek seferde burada belirtiyoruz ve bu instense derdinden bizi kurtarıyor.

AutofacBusinessModule konusunda projemiz içerisine ilerleyen zamanlarda yeni bir Service – Manager – Dal gibi eklemeler yapılırsa burada bunları tanımlamamız gerekmektedir.

```
public class AutofacBusinessModule : Module
{
    0 references | zeylmz, 1 day ago | 1 author, 4 changes
    protected override void Load(ContainerBuilder builder)
    {
        builder.RegisterType<TurkishEventManager>().As<ITurkishEventService>();
        builder.RegisterType<EfTurkishEventDal>().As<ITurkishEventDal>();

        builder.RegisterType<ItalianEventManager>().As<IItalianEventService>();
        builder.RegisterType<EfItalianEventDal>().As<IItalianEventDal>();
```

ValidationRules

Doğrulama işlemlerini uygulayacağımız teknolojileri burada barındırıyoruz. Tüm doğrulama işlemlerinin kalbi burasıdır.

FluentValidation

Çoklu dil desteği, kolay kullanımı ve çokça dökümantasyonu olmasından kaynaklı bu teknolojiyi kullanmaktayım. Bu klasör altında klasör adındaki teknolojiye odaklı validate kodlarını yazıyoruz.

ItalianEventValidator – TurkishEventValidator

Burada hem İtalyanca hem de Türkçe veriler için doğrulama ekledim. Örneğin veritabanına eklenecek veri boş olamaz gibi. Burada eklediğimiz kuralları Manager içerisinde operasyonlarda kullanıyoruz. Ancak çok yoğun kod karmaşası yaratabileceğinden bu sınıfları aspect ile kullanmaktayım.

```
[SecuredOperation("admin,data.add")]
[ValidationAspect(typeof(ItalianEventValidator))]
[CacheRemoveAspect("ITurkishEventService.Get")]
2 references | zeyilmz, 1 day ago | 1 author, 3 changes
public IResult Update(ItalianEvent italianEvent)
{
    _italianEventDal.Update(italianEvent);
    return new SuccessResult(Messages.UpdatedDataFromDatabaseSuccessful);
}
```

Core Katmanını Tanıyalım

Burası bizim tüm projelerimizde ortak olarak kullanabileceğimiz tek bir projeye bağımlılığı olmayan bir katmandır. Bu katmanda birçok operasyonun temeli yazılmaktadır. Burası için projenin kalbi diyebiliriz.

Aspects

Tüm projelerde ortak olarak kullanılacak Aspectleri teknolojileriyle birlikte tanımladığımız bölümdür. Business Tarafından çağrılacak Attribute isimleri vardır.

Autofac

Caching

Business katmanın Cache işlemlerini kullanabilmesi için yazılan Aspectleri içermektedir.

- **CacheAspect:** Cache ‘e ekleme işlemleri için kullanılır.
- **CacheRemoveAspect:** Cache ‘den çıkarma işlemleri için kullanılır.

Exception

Exception işlemlerinin kaydını tutmamızdaki kilit aspect buradadır. Özel olarak Business üzerinde çağırarak da kullanabiliriz. Veya arka planda Interceptors olarak dosyaya kayıt tutmaktadır.

Logging

Log4Net altyapısıyla sisteme eklediğimiz Log işlemini Business tarafından çağırmamıza olanak tanıyan aspect ‘i içerir.

Performance

Çalışma süresinden endişe ettiğimiz. Acaba yavaş mı çalışıyor dediğimiz kısımlarda süre hesabı yapıp Debug kısmına yazan, istenilirse loglayan veya mail atan bir aspecttir. Business tarafından ölçüm yapılacak methot üzerinde çağrılır.

Validation

Business içerisinde belirttiğimiz Validate işlemlerini uzunca kodlar yazıp if else kullanmadan sadece operasyon üzerinde çağırarak hangi validator olduğunu belirttiğimiz ve direk çalışmaya başlayan bir aspect. Business tarafında temiz kod yapısına sahip olmamızı sağlıyor.

CrossCuttingConcerns

Caching

Cache işlemlerinin beyni burası diyebiliriz. Hangi cache sistemini kullanacaksak teknolojisiyle birlikte kodladığımız bölüm. Burada yaptığımız teknoloji değişimleri için aspect üzerinde değişiklik yapmamıza gerek yoktur. Ben Microsoft’un içinde gelen cache sistemini kullandım.

Ancak burada Redis gibi yapılarıda rahatça kullanabiliriz. Sadece ICacheManager'i yeni teknolojide implemente edip CoreModule üstünde instancesini almamız yeterli olacaktır.

Logging

Log4Net altyapısıyla hazırladığım bir alan. Bu alanda farklı loglama teknolojileri belirtebilirsiniz. Log detaylarını kendinize göre düzenleyebilirsiniz. Ancak burada yaptığınız değişikler eğer yeni bir teknoloji değil, Log4Net üzerinden olacaksa, WebAPI bölümünde bulunan log4net.config üzerinden de gerekli ayarlamaları yapmanız gerekmektedir. Bu ayarlamalarla ilgili detaylı dokümantasyona Log4Net web sitesinden ulaşabilirsiniz.

Validation

Business içerisinde sürekli if yazmaktan bizi kurtaran bir modül. Aspect tarafından bir validator type belirterek kullanıyoruz.

DataAccess

Normal DataAccess katmanından farkı bu bölüm tüm projelerde kullanılabilecek düzeyde ortak olan sistemleri belirlediğimiz alandır.

Abstract > IEntityRepository<T>

Tüm veri operasyonlarında ortak olan yapıları içerisinde barındırmaktadır. Generic bir yapıdadır. Programcıyı korumak için sadece varlık nesneleriyle çalışabilmesi için koşullar verilmiştir. Tüm projelerinizde ortak olacağını düşündüğünüz tüm operasyonları buraya yazmanız gerekmektedir.

Farklı bir teknolojiye geçecek dahi olsak temel operasyonlarımızın belli olmasını sağlamaktadır.

Concrete > EntityFramework > EfEntityRepositoryBase<TEntity, TContext>

Ortak olan operasyonların detaylarını EntityFramework teknolojisi için yazdığımız bölümdür. Burada IEntityRepository implementasyonu sayesinde ana operasyonlarımızı çekip kendisine ait teknolojiyle ortak operasyonların kodlarını işleyişini yazdığımız alandır. Bir veritabanı ve bir varlık türü vermemizi istemektedir. Bunlar üzerinden işlem yapmaktadır.

DependencyResolvers > CoreModule

IServiceCollection ile instance almamız gereken durumlarda bu eklentiyi kullanıyoruz. Tüm projelerimizde ortak instance yapıları burada bulunmaktadır.

Entities

Normal Entities katmanından bir farkı yoktur. Tüm projelerde ortak olan yapıları bünyesinde içermektedir.

Abstract > IEntity – IDto

Burada belirttiğimiz bu interfaceler kod okunabilirliğini arttırması amacı güderek oluşturulmuştur.

Ancak DataAccess tabanında istenen varlık nesneleri gibi durumlarda class olacak IEntity olacak ve new'lenebilir olacak şeklinde tanımlama yaptığımızda sadece Entities içindeki varlıkları kullanabilmesini diğer varlıklarda hata almasını sağlıyoruz.

Concrete

Bu alanda her projemizde kullanacağımız User – OperationClaim gibi varlıkları tutuyoruz. Aynı zamanda Core katmanı içerisinde bulunan Hashing gibi yapıların projeye bağımlı olmasını engelliyoruz.

Extensions

Mevcut yapıları genişletmek kendi sistemimize uygun hale getirmek için kullandığımız bölümdür. Sistemde yazacağınız ve tüm projelerde kullanabilecek düzeyde olan tüm genişletme işlemleri burada belirtiyoruz.

ClaimExtensions – ClaimsPrincipalExtensions

Kullanıcının mailine o yetkiyi eklemek, ID'ye göre tanıma yapmak gibi gibi işlemler için System.Security.Claims 'i genişlettiğimiz alandır. Her kullanımda elle ekleme yapmamak için tek seferde burada genişletme işlemi yaptık.

Aynı zamanda listelemeleri claimleri çekmeyi vs. de tek seferde belirlemek bizim yapımıza uygun hale getirmek için bu genişletmeleri oluşturduk

Exception > ErrorDetails > ExceptionMiddleware > ExceptionMiddlewareExtensions

- **ErrorDetails:** Kullanıcıya göstereceğimiz hataların varlık yapısıdır.
- **ExceptionMiddleware:** Ufak bir middleware yazmamız gerekti. Kullanıcı hata oluştuğu zaman bir çok sistemsel detayı görebiliyordu. Bunun önüne geçmek için bu middleware'i yazdık. Programın Exception Handler sisteminin temeli burası.
- **ExceptionMiddlewareExtensions:** ApplicationBuilder içerisinde yazdığımız middleware'i eklemek için oluşturduğumuz bir bölümdür.

ServiceCollectionExtensions

Demin belirttiğimiz CoreModule'ın sistem üzerinde tanınabilmesi açısından oluşturduğumuz bir genişletmedir. Microsoft.Extensions.DependencyInjection.IServiceCollection yapısını genişlettiğimiz kendimize ait ICoreModule türündeki sınıfları tanımlatabildiğimiz bölümdür.

Utilities

Bu alanda çeşitli yapıları tutuyoruz. İşimize yaran araçlar, güvenlik olayları, interceptorlar, olumlu olumsuz sonuç dönmemizi kolaylaştıran Result yapısı gibi tüm çeşitli içerikleri bu alan altında yazıyoruz.

Business > BusinessRules

Bu aracımız bizlerin sürekli eğer şöyleyse bu olsun böyleyse şu olsun dediğimiz işlemleri tek kontrol etmemize olanak tanıyan bir yapıdır. İş kurallarının düzenli durması ve kontrol edilmesini kolaylaştırmaktadır. IResult türündeki tüm metotları çalıştırabilmektedir.

```
[SecuredOperation("admin,data.add")]
[ValidationAspect(typeof(TurkishEventValidator))]
[CacheRemoveAspect("ITurkishEventService.Get")]
2 references | zeylmz, 1 day ago | 1 author, 6 changes
public IResult Add(TurkishEvent turkishEvent)
{
    IResult result = BusinessRules.Run
        (
            DataCountExceeded()
        );

    if (result != null)
    {
        return result;
    }

    _turkishEventDal.Add(turkishEvent);
    return new SuccessResult(Messages.AddedDataFromDatabaseSuccessfull);
}
```

Helpers > IpDataAPIHelper – JsonHelper

Bu alanda proje içerisinde kullanacağımız ancak projeye bağımlı olmayan araçları belirtiyoruz. Örneğin resim eklemek dosya eklemek gibi bir işlem yapmak istediğimiz zaman o aracı burada yazacağız.

- **IpDataAPIHelper:** Bu ipdata.co üzerinden kullandığım api servisinin kodlarını içermektedir. IpInfoData varlığında istediğimiz verileri yazarak işlem yapabilmekteyiz.
- **JsonHelper:** Bize verilen bir json dosyasının url'sini yazdıktan sonra Liste objesi haline getiren bir yapıdır. Bu araç sayesinde url üzerinden gelen verileri kullanıcıya gösterebiliyor ve db'ye kaydedebiliyoruz.

Interceptors

Derleme zamanında çalışacak araya girip işlem yapabilecek Aspect işlemlerinin ana base'i burasıdır. İşlem sonrası öncesi hata olunca gibi gibi araya girmemizi gerektirecek her durumun base'i burasıdır.

IoC

Core katmanı içerisinde bulunan DependencyResolvers bölümünün kalbidir. Buradaki amaç objelerin instance'larının kontolünü sağlamak. Bağımlılıkları en aza indirgemektir.

Messages

Her projemizde ortak olarak gösterilmesini planladığımız mesajları burada belirtiyoruz.

Results

Olumlu veya olumsuz sonuç döndürmek istediğimiz durumlarda bu yapıyı kullanıyoruz.

Security

Bu alanda sistemimizin güvenliğiyle alakalı araçları belirtiyoruz. Encryption Hashing gibi kavramların kalbi burasıdır. Aynı zamanda bu alanda Json Web Token sisteminin altyapısı bulunmaktadır.

WebAPI

Bu katman bizim mevcut projenin arayüz katmanı diyebiliriz. Burada business içerisinde oluşturduğumuz service ve manager yapılarının controllerlarını yaptığımız Backend işlemlerini Apiyi kullanacak kişilere sunduğumuz alandır. Proje geliştirmeye başladığımız zaman müdahale edeceğimiz son alan burasıdır.

Properties

Api projesi oluşturulduğunda otomatik olarak oluşan bir alandır. Burada başlangıç sayfasını url bilgisini vs. değiştirebildiğimiz bölümdür.

Controllers

Bu alan kullanıcıların erişebildiği api dökümantasyonu içerisinde bulunan işlemlerin yazıldığı bölümdür.

- **AuthController:** Bu alan bizlere login ve register işlemlerinin kullanıcılar tarafından yapılabilmesini sağlamaktadır. Business katmanı içerisinde bulunan IAuthService ile ortak çalışmaktadır.
- **EventsController:** RequestLocalization sisteminin çalıştığı bölüm. Kullanıcının ülkesine göre bilgi verdiğimiz alan.
- **ItalianEventsController:** İtalyanca veriler üzerinde işlem yaptığımız ekleme çıkarma güncelleme gibi tüm işlemleri kullanıcının yapmasını buradan sağlamaktayız.
- **TurkishEventsController:** Türkçe veriler üzerinde kullanıcıların işlem yapmasına olanak tanıdığımız bölümdür.

Appsettings.Json

Buranın içerisinde JWT sistemi için TokenOptions diye bir bölüm vardır. Buradan dakika cinsinden tokenin geçerlilik süresini gibi bilgileri ayarlabilirsiniz.

Log4Net.config

Log sisteminin çalışabilmesi için gerekli ayarlama ve tanılama işlemlerini yaptığımız bölümdür. Bu alandan daha detaylı bilgi istiyorsanız daha detaylı ayarlamalar yapmanız gerekmektedir. Log4Net dökümantasyonlarında tüm bilgiye detaylıca ulaşabilirsiniz.

Program.cs

Burasıda otomatik olarak oluşuyor proje başlangıcında ancak proje başlangıcına ek bu bölümde Autofac sistemini tanımlamış bulunmaktayım.

Startup.cs

Exception, Authorization gibi proje içerisinde çalışacak durumların gerekli ayarlamalarını buradan yapıyoruz.

Aynı zamanda içeride belirtilen AddCors ve UseCors gibi içeriklere Apinizi kullanacak ipleri belirtmelisinizi. Aksi taktirde karşı taraf Cors hatası alacaktır.

Geliştirmek İçin İzlenecek Yollar

Burada örnek olarak elimize İngilizce verilerin ulaştığını ve bunları da sisteme dahil etmek istediğimizi belirtelim. Onun üzerinden izleyeceğimiz yolları adım adım izleyelim.

Birinci Adım: Tablo oluşturmak – Varlık oluşturmak

Öncelikle gelen veriye göre veri tabanı üzerinde bir tablo oluşturunuz. Bu tabloyu Entities bölümünde Concrete içerisine varlık olarak oluşturalım ve IEntity implemantasyonuna sahip olduğundan emin olalım. Aksi takdirde sistem bunu bir veri tabanı nesnesi olarak görmeyecektir.

Örnek tablo ismi: EnglishEvents

Örnek varlık ismi: EnglishEvent

İkinci Adım: Veri erişim katmanı – Context'e dahil etmek

Bu bölümde DAL işlemlerini yapacağız. Ancak öncelikle sistemimizin bu tabloya erişebilmesi için DataAccess -> Concrete -> EntitiyFramework -> Contexts içerisinde bulunan veri tabanı nesnemizin context halini açarak oraya ekleme yapmalıyız. Örnek aşağıdadır.

```
0 references | zeyimz, 3 days ago | 1 author, 1 change
public DbSet<TurkishEvent> TurkishEvents { get; set; }
```

Ardından DataAccess içerisinde Abstract klasörüne DAL operasyonlarını tanımlayacağımız soyut nesnemizi oluşturalım. Bu oluşturduğumuz soyut nesnenin IEntityRepository<EnglishEvent> şeklinde miras aldığından emin olalım.

Sonrasında Concrete içerisinde bulunan EntityFramework klasörüne bir class oluşturalım. Ve bu class'ın EfEntitiyRepositoryBase<EnglishEvent, MyHomeworkContext> gibi bir yapıda olduğuna emin olalım. Aynı zamanda soyut nesnemizi de implemente edelim.

Örnek Soyut İsimlendirme: IEnglishEventDal

Örnek Somut İsimlendirme: EfEnglishEventDal

Üçüncü Adım: Service ve Manager sınıflarını oluşturmak

Geldik Business operasyonlarına burada yapmamız gereken öncelikle Business katmanı Abstract klasörü içerisine bir service oluşturalım. Bu servis içerisine gerekli operasyonlarımızı yazalım. Operasyonları yazarken ITurkishEventService veya IItalianEventService içerisindeki operasyonları kılavuz olarak alabilirsiniz.

Service oluşturulduktan sonra Concrete klasörü içerisine Manager sınıfımıza da oluşturalım. Bu sınıfı oluşturduktan sonra Servisi implemente etmeyi unutmayalım.

Bir ctor yardımıyla IEnglishEventDal dosyamızı buraya dahil edelim. Operasyonları yazmaya başlayalım. Operasyonları yazarken diğer dillerin Manager sınıflarını kılavuz olarak alabilirsiniz.

Örnek Soyut İsimlendirme: IEnglishEventService

Örnek Somut İsimlendirme: EnglishEventManager

Dördüncü Adım: Controller oluşturmak ve RequestLocalization işlemine dahil etmek

Bu adım son aşamamız olacaktır. WepAPI katmanında Controllers klasörü içerisine bir controller ekleyelim. Bu kontroller ismi örnek olarak EnglishEventsController olabilir.

Buraya kullanıcıların erişmesini istediğimiz operasyonları yazalım örnek olarak diğer dillerin controllerlarına bakabilirsiniz.

EventsController üzerine eklemek için servisi ctor içerisin eklemeniz gerekmektedir. Aynı şekilde GetEvents operasyonu içerisine aşağıdaki örnekteki gibi bir kod yazabilirsiniz.

```
if (clientIp.country_name == "United Kingdom" || clientIp.country_name == "United States" || clientIp.country_name == "Canada")
{
    var englishResult = _englishEventService.ReadJson();
    if (englishResult.Success)
    {
        return Ok(englishResult);
    }
    return BadRequest(englishResult);
}
```

Ancak İngilizce global bir dil olduğundan buraya şart olarak Türk kullanıcısıysa Türkçe diye bir şart oluşturup geriye kalan tüm ülkelere İngilizce veri gösterilebilir.

Ardından eklediğiniz yeni dili dokümantasyonlara ekleyerek işlemi tamamlayabilirsiniz.

Okuduğunuz için teşekkürler.

Github/zeylmz