**Katmanlar**

**Abc.Core:** Framework katmanı. Bizim için önemli olan bu katman bu projede bu katmanı geliştireceğiz diğer katmanları test için kullanacağız. Örn Cache işlemleri Logging işlemlerini yapacağız. Performans aspect leri yazacağız. Örn bir metodun çalışması 5 saniyeyi geçerse mil atacak.

**Abc.Northwind.Entities:** Veritabanındaki tabloların ve complex tiplerin bulunduğu katman. Proje boyunca kullnacağımız nesneler burada olacak örn: Sepet nesnesi gibi.

**Abc.Northwind.DataAccess:** Veritabanı CRUD işlemlerinin bulunduğu katmandır. Sadece veriye erişim için kullanacağız.

**Abc.Northwind.DataAccess.Tests:** DataAccess test projesi.

**Abc.Northwind.Business:** İş katmanı. Sadece iş kodlarını yazmak için kullanılır.

**Abc.Northwind.Business.Tests** Business birim testi katmanımız.

**Abc.Northwind.MVCWebUI:** MVC IU katmanı.

**Abc.Northwind.MvcWebUI.Tests:** MVC UI test katmanı.

**Abc.Northwind.WebApi:** Web api katmanımız.

**Abc.Northwind.WcfLibrary:** Wcf libray katmanı.

**2. Gün**

**Validation Aspect**

**Aspect Oriented Programming (AOP) kesişen ilgiler (Cross-Cutting)**

Business katmanında ValidationRules klasörü altında FluentValidation adında bir klasör oluşturup nesneleri orada tanımlayıp validation uyguladım. Yine bu katmana Fluentvalidation u ekledim.

Validation işlemini Business de yapmamızın sebebi. İster mobil, ister web forms istersek de mvc kullanalım Business her işlemde kullanılıyor olması.

public class ProductValidator : AbstractValidator<Product>

{

public ProductValidator()

{

RuleFor(p => p.ProductName).NotEmpty();

RuleFor(p => p.CategoryId).NotEmpty();

RuleFor(p => p.UnitPrice).NotEmpty();

RuleFor(p => p.UnitPrice).GreaterThan(0);

RuleFor(p => p.UnitPrice).GreaterThan(10).When(p => p.CategoryId == 1);// category id 1 ise fiyat 10 dan büyük olmalı

RuleFor(p => p.UnitsInStock).NotEmpty();

RuleFor(p => p.UnitsInStock).GreaterThan((short)0);

// RuleFor(p => p.ProductName).Must(StartWithA); // custom validator. A ile başlıyorsa

}

//private bool StartWithA(string arg)

//{

// return arg.StartsWith("A");

//}

}

Bunu SOLID e uygun kullanabilmek için Aspect den faydalanıp attribute yazmam gerekir.

Core katmanına gidip CrossCuttingConcerns adında bir klasör ekliyorum. Onun altında Validation ve bunun altına da FluentValidation adında klasör oluşturuyorum ValidationTool.cs adında bir validator classı yazıyorum.

public class ValidationTool

{

public static void FluentValiate(IValidator validator, object entity)

{

var result = validator.Validate(entity);

if (result.Errors.Count > 0)

{

throw new ValidationException(result.Errors);

}

}

}

Şimdi bu validation için AOP Aspect Oriaented Progrraming yapacağız. Bunu yapmak için Postsharp’ın 4.2.17 sürümünü kullanacağız. İşimiz sadece aspect yazmak oluğu için güncellemeler işimize yaramayacaktır.

<https://www.postsharp.net/downloads/postsharp-4.2/v4.2.17>

VS kapatıp kurulum yapmalıyız. Seçeneklerden Free i seçebiliriz.

Core katmanına gelip aspectlerimiz için Aspects adında bir klsör oluşturalım onun altınada Postsharp adında bir klasör oluşturup bir class ekliyorum. FluentValidationAspect.cs. Nuget den Postsharp yazıp versiyon olarak 4.2.17 i seçiyorum.

ProductManager.cs e gidip Add metodunun başına validationu bir attribute olarak verebiliriz.

[FluentValidationAspect(typeof(ProductValidator))]

public void Add(Product product)

{

\_productDal.Add(product);

}

**Business Product Test**

[TestClass]

public class ProductManagerTests

{

[ExpectedException(typeof(ValidationException))] // Metodun testten geçebilmesi için bu exceptionu fırlatması gerekir.

[TestMethod]

public void ProductValidationCheck()

{

// mock veriler oluşturuyorum. Burada veritabanına bağlanmaya gerek yok

//Nuget den Moq yüklenmeli.

Mock<IProductDal> productDalMock = new Mock<IProductDal>();

Mock<IQueryableRepository<Product>> repositoryMock = new Mock<IQueryableRepository<Product>>();

ProductManager productManager = new ProductManager(productDalMock.Object, repositoryMock.Object);

productManager.Add(new Product());

}

}

**Cache Yönetimi**

Core de CrossCuttingConcern klasöründe Caching diye bir klasör oluşturuyoruz. ICacheManager Adında bir interface ekliyoruz. MemoryCache i kullanacağız ama interface güdümlü gittiğimiz için istediğimiz zaman başka bir cache mekanizması ekleyebileceğiz.

public interface ICacheManager

{

T Get<T>(string key);

void Add(string key, object data, int cacheTime);

bool IsAdd(string key);

void Remove(string key);

void RemoveByPattern(string pattern);

void Clear();

}

MemoryCacheManager sınıfını IcacheManager den türetip cache kodlarını yazıyoruz.

Şimdi Core katmanındaki aspect içerisinde postsharp klasöründe Caching klasörü oluşturuyoruz . CacheAspect adında bir class ekliyoruz Aspect i burada yazacağız MemoryCacheManager de kaynak kodlar var.

Örn kullanım

[CacheAspect(typeof(MemoryCacheManager))] // aspect cache manager

public List<Product> GetAll()

{

return \_productDal.GetList();

}

**Cache Remove Aspect**

Cache klasörüne CacheRemoveAspect.cs adında bir class ekliyorum.

[Serializable]

public class CacheRemoveAspect : OnMethodBoundaryAspect

{

private readonly string \_pattern;

public CacheRemoveAspect()

{

}

public CacheRemoveAspect(string pattern)

{

\_pattern = pattern;

}

public CacheRemoveAspect(Type typeManager)

{

\_pattern = $"{typeManager.ReflectedType.Namespace}.{typeManager.ReflectedType.Name}.\*";

}

public override void OnSuccess(MethodExecutionArgs args)

{

var cacheManager = new MemoryCacheManager();

cacheManager.RemoveByPattern(string.IsNullOrEmpty(\_pattern)

? string.Format("{0}.{1}.\*", args.Method.ReflectedType.Namespace, args.Method.ReflectedType.Name)

: \_pattern);

}

}

Örn kullanım: Update Delete ve Remove metodlarına ekledim.

[TransactionScopeAspect]

[CacheRemoveAspect]

public void Update(Product product)

{

\_productDal.Update(product);

}

**Transaction Aspect**

Core Aspect klasörü içerisinde Postshartp altında Transaction adında bir klasör oluştrup TransactionScopeAspect adında bir class ekliiyoruz.

[Serializable]

public class TransactionScopeAspect : OnMethodBoundaryAspect

{

public TransactionScopeAspect()

{

}

public TransactionScopeAspect(TransactionScopeOption option)

{

\_option = option;

}

private TransactionScopeOption \_option;

public override void OnEntry(MethodExecutionArgs args)

{

args.MethodExecutionTag = new TransactionScope(\_option);

}

public override void OnExit(MethodExecutionArgs args)

{

((TransactionScope)args.MethodExecutionTag).Dispose();

}

public override void OnSuccess(MethodExecutionArgs args)

{

((TransactionScope)args.MethodExecutionTag).Complete();

}

}

Örn Kullanım:

[FluentValidationAspect(typeof(ProductValidator))]

[TransactionScopeAspect]

public void Add(Product product)

{

\_productDal.Add(product);

}

Not: Eğer Oracle veritabanı kullanırsak Oracle Services for Microsoft Transaction Server kurmak gerekir.

**NHibernate Entegrasyonu**

Core katmanında DataAccess klasöründeki NHibernate içerisine IEntityRepository adında bir interface ekliyoruz. Nuget den nhibernate ve Fluentnhibernate paketlerini ekliyoruz. NHibernate mappingleri xml ile yapar. Bu sayede classlar ile yapacağız.

NHibernate klasöründe NHibernateHelper adında bir class ekliyoruz.Burada bu sınıfı abstract yapıp IDisposable dan türetip NhiberNate özel kodları yazdık.

**3.Gün**

**Performans Aspect Yöntemi**

Abc.Core.Aspects. Postsharp. Klasörü altında Performance adında bir klasör ekliyorum ve içerisine PerformanceCounterAspect adında bir class ekleyip postsharp in OnMethodBoundaryAspect dan türetiyoruz. StopWatch ile basit bir kronometre mantığı ile metodun çalışma zamanını öğreneceğiz. Bunun çalışma anında oluşturulmasını istediğimiz için Serileştirmememiz gerekir. NonSerialized gelecek.

[Serializable]

public class PerformanceCounterAspect : OnMethodBoundaryAspect

{

private int \_interval; // saniye tutacağız örneğin bir metodun çalışma süresi 5 saniyeyi geçerse mail atsın gibi işlemler için

[NonSerialized]

private Stopwatch \_stopwatch; //NonSerialized runtime de devreye girer. Seriable ise compile time da.

public override void RuntimeInitialize(MethodBase method) // çalışma anında devreye girmesini istediğimiz için RuntimeInitialize

{

\_stopwatch = Activator.CreateInstance<Stopwatch>();

base.RuntimeInitialize(method);

}

public override void OnEntry(MethodExecutionArgs args)

{

\_stopwatch.Start();

base.OnEntry(args);

}

public override void OnExit(MethodExecutionArgs args)

{

\_stopwatch.Stop();

if (\_stopwatch.Elapsed.TotalSeconds > \_interval)

{

Debug.WriteLine("Performance : {0}.{1}-->{2}", args.Method.DeclaringType.FullName, args.Method.Name, \_stopwatch.Elapsed.TotalSeconds);

}

base.OnExit(args);

}

}

Örneğin productManager GetAll metodunda deneyelim.

[PerformanceCounterAspect(5)]

public List<Product> GetByCategory(int categoryId)

{

return \_productDal.GetList(p => p.CategoryId == categoryId);

}

**Önemli:**

Business içindeki Properties altında AssemblyInfo.cs sınıfına gidip kodu metod seviyesinden asemmbly seviyesinde çalışmasını sağlıyoruz

[assembly:PerformanceCounterAspect(5,AttributeTargetTypes = "Abc.Northwind.Business.Concrete.\*Manager")] //**TODO: Assembly seviyesinde kullandık. metodların çalışması 3 saniyeyi geçerse**

Abc.Northwind.Business.\* dersek Business deki bütün metolarda çalışır.

**Logging:**

Loglama da bir CrossCuttingConcerns olduğu için core katmanındaki klasöre gidip Logging adında bir klasör oluşturuyoruz.

MVC UI katmanına gidiyoruz Log4Net.config adında bir config dosyası açıyoruz. Burada gerekli ayarları yapacağız.

Şimdi LogAspect i yazalım Core katmanında ki Aspect klasörüne Logging klasörü oluşturup LogAspect.cs sınıfı ekliyoruz.

ProductManager de deneyelim.

[FluentValidationAspect(typeof(ProductValidator))]

[TransactionScopeAspect]

[CacheRemoveAspect]

[LogAspect(typeof(FileLogger))]

public void Add(Product product)

{

\_productDal.Add(product);

// \_productDal.Add(new Product());

}

Bu arada bunuda Performans aspect i gibi Assembly den ekleyebiliriz.

Web config e gidip log4net için hazırladığımz config dosyasının yolun vermemiz gerekir.

<add key="log4net.Config" value="log4net.config"/>

<add key="log4net.watch" value="false"/>

**WCF Entegrasyonu**

ProductService Adında bir class oluşturup IProductService den türetiyoruz. IProductService içerisine WCF kuralına dayalı veriyi serileştirmek için [ServiceContract] ve [OperationContract] ekliyorum

MVC Web projesinde Global Asax da bir modle tanımlamıştık.

ControllerBuilder.Current.SetControllerFactory(new NinjectControllerFactory(new BusinessModule()));

Şimdi buraya ServiceModule de ekleyeceğiz ve Web config de Soap dersek ServiceModule devreye girip soapa gidecek.

Abc.Core.Utilities.Common klasörüne gidip WffProxy adında bir class tanımlayalım.

public class WcfProxy<T>

{

public static T CreateChannel()

{

string baseAddress = ConfigurationManager.AppSettings["ServiceAddress"];

string address = String.Format(baseAddress, typeof(T).Name.Substring(1));

var binding = new BasicHttpBinding();

var channel = new ChannelFactory<T>(binding, address);

return channel.CreateChannel();

}

}

MVC Web Config de bir key ekliyoruz

<add key="ServiceAddress" value="http://localhost:50105/{0}.svc"/>

T burada bir channel üretir.

Abc.Northwind.Business.DependencyResolves.Ninject yoluna gidip daha önce oluşturduğumuz BusinessModule yanında birde ServiceModule oluşturuyoruz.

public class ServiceModule : NinjectModule

{

public override void Load()

{

Bind<IProductService>().ToConstant(WcfProxy<IProductService>.CreateChannel());

}

}

Biri bizden IProductService isterse instance sini oluşturur. Sağ tıklayıp add referance dememe gerek kalmaz.

MVC global.asax da BusinessModule ServiceModule() yazarsak böylece veriler WCF service den gelecektir.

//ControllerBuilder.Current.SetControllerFactory(new NinjectControllerFactory(new BusinessModule()));

ControllerBuilder.Current.SetControllerFactory(new NinjectControllerFactory(new ServiceModule()));

**Web API Entegrasyonu**

Web api projesinde Controller klasöründe ProductsController adında Web API 2 empty controlleri ekliyoruz

public class ProductsController : ApiController

{

private IProductService \_productService;

public ProductsController(IProductService productService)

{

\_productService = productService;

}

public HttpResponseMessage Get()

{

HttpResponseMessage message = Request.CreateResponse(HttpStatusCode.OK,\_productService.GetAll());

return message;

}

}

NhiberNate ve EntityFramework u de ekliyorum. Yine nuget den WebApiContrib.IoC.Ninject i yüklüyorum. ControllerFactory işlemi yapıştık mvc de bu onun web api implementasyonu. Ninject.MVC5 paketini de yüklüyoruz. Bu paket bize app\_start altında bir NinjectWebCommon.cs oluşturacak burada RegisterServices altında BusinessModule mizi veriyoruz.

private static void RegisterServices(IKernel kernel)

{

kernel.Load(new BusinessModule());

}

CreateKernel içerisine de NinjectResolver veriyoruz.

GlobalConfiguration.Configuration.DependencyResolver=new NinjectResolver(kernel);

**4.Gün**

Identity Menagment Server IDMS i araştır. Bir Identity Server kurup token bazlı rol yönetimi yapılabilir.

**Claim Base Auth()** Tüm katmanlar için geçerli bir yöntemdir.

Güvenlik ile ilgili çalışmaları business katmanında yapacağız. Business katmanında BusinessAspect diye bir klasör oluşturuyorum ve içerisine SecureOperationAspect adında bir class oluşturup postsharp’in OnMethodBoundaryAspect implemente ediyoruz. Burada eğer bir operasyon işlemi yapılıyorsa örneğin Ekleme işleminde izninin olup olmadığını kontrol edecek işlemi business de yapmalıyız. Ama örn menüyü göremesin istiyorsak bunu ilgili UI katmanında yapmak gerekir.

[Serializable]

public class SecureOperationAspect : OnMethodBoundaryAspect

{

private string \_operation;

public SecureOperationAspect(string operation)

{

\_operation = operation;

}

public override void OnEntry(MethodExecutionArgs args)

{

// identity den al

//string currentUser = "engin";

string currentUser = Thread.CurrentPrincipal.Identity.Name; // veya identity den gelebilir

//db den al

List<OperationClaim> claims = new List<OperationClaim>

{

new OperationClaim{OperationName="Product.Read",UserName="engin"},

new OperationClaim{OperationName="Product.Write",UserName="engin"},

new OperationClaim{OperationName="Category.Read",UserName="engin"},

new OperationClaim{OperationName="Category.Write",UserName="engin"},

new OperationClaim{OperationName="Customer.Read",UserName="engin"}

};

if (!claims.Any(c => c.UserName == currentUser && c.OperationName == \_operation))

{

throw new SecurityException("You are not authorized");

}

}

}

Business de DomainModels diye bir klaör oluşturuyorum içerisine OperationClaim.cs adında bir sınıf ekliyorum.Burada sadece propertyler olacak.

public class OperationClaim

{

public string OperationName { get; internal set; }

public string UserName { get; internal set; }

}

MVC katmanında test etmek için Core katmanındaki CrossCuttingConcers klasörüne Security adında bir klasör ve onun altına da Web adında klasör ekliyorum. Identity adında bir class ekleyip IIdentity den implemente ediyoruz.

public class Identity: IIdentity

{

public Guid Id { get; set; } // IIdentity den ilk 4 propery gelir gerisini ihtiyaca göre biz yazıyoruz.

public string Name { get; set; }

public string AuthenticationType { get; set; }

public bool IsAuthenticated { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Password { get; set; }

public string Email { get; set; }

public string[] Roles { get; set; }

}

Aynı klasöre AuthenticationHelper adında bir class ekliyorum ve yine aynı klasöre SecurityUtilities adında bir class daha ekliyorum.

MVC Global asax a gidip kodları ekliyoruz.

public override void Init()

{

PostAuthenticateRequest += MvcApplication\_PostAuthenticateRequest;

base.Init();

}

void MvcApplication\_PostAuthenticateRequest(object sender, EventArgs e)

{

try

{

var authCookie = HttpContext.Current.Request.Cookies[FormsAuthentication.FormsCookieName];

if (authCookie == null) return;

var encTicket = authCookie.Value;

if (String.IsNullOrEmpty(encTicket)) return;

var ticket = FormsAuthentication.Decrypt(encTicket);

var securityUtilities = new SecurityUtilities();

var identity = securityUtilities.FormsAuthTicketToIdentity(ticket);

var prin = new GenericPrincipal(identity, identity.Roles);

HttpContext.Current.User = prin;

Thread.CurrentPrincipal = prin;

}

catch (Exception)

{

}

}

Şimdi MVC controllere gidip bir LoginController ekliyoruz.

public string Index()

{

AuthenticationHelper.CreateAuthCookie(

new Guid(),

"engin",

"engin@gmail.com",

DateTime.Now.AddMinutes(15),

new[] { "Admin" },

false,

"Engin", "Demiroğ", "12345");

return "now authenticated";

}

Tabi kişiyi veritabanından sorgulatıp bu bilgilerin geldiğini varsayıyorum.

**Exception Aspect Yazımı**

Core katmanında Aspects klasörü içerisinde Postsharp’a Exeption adında bir klasör ekledim ve içerisine ExceptionAspect adında bir class ekledim.

[Serializable]

public class ExceptionAspect : OnExceptionAspect

{

[NonSerialized]

private LoggerService \_loggerService;

private Type \_loggerType;

public ExceptionAspect(Type loggerType)

{

\_loggerType = loggerType;

}

public override void RuntimeInitialize(MethodBase method)

{

if (\_loggerType.BaseType != typeof(LoggerService))

{

throw new System.Exception("Wrong Logger Type");

}

\_loggerService = (LoggerService)Activator.CreateInstance(\_loggerType);

base.RuntimeInitialize(method);

}

public override void OnException(MethodExecutionArgs args)

{

//if (\_loggerService != null)

//{

// \_loggerService.Error(args.Exception);

//}

\_loggerService?.Error(args.Exception);

}

}

Business içerisinde assembly infoya gelip Exception u assembly seviyesinde ekleyelim.

[assembly: ExceptionAspect(typeof(DatabaseLogger))] //**TODO: Exception handler**

**Arama Örneği**

Abc.Core.Utilities.Search altında.

**Specification Pattern**

**İş Kuralları Yönetimi**

Örneğin bir ürün sadece içecek kategorisinde olmalı. Buna bener iş kurallarını yazmak için Specification Pattern den yararlanmalıyız.

**DevFramework.Core.Utilities.SpecificationBase:** İmplemente edilen yer.

**DevFramework.Business.Specs.ProductSpecs** de iş kuralları var.

**DevFramework.Core.CrossCuttingConcern.ExceptionHandling.Exceptions** döneceği Business exception kodları