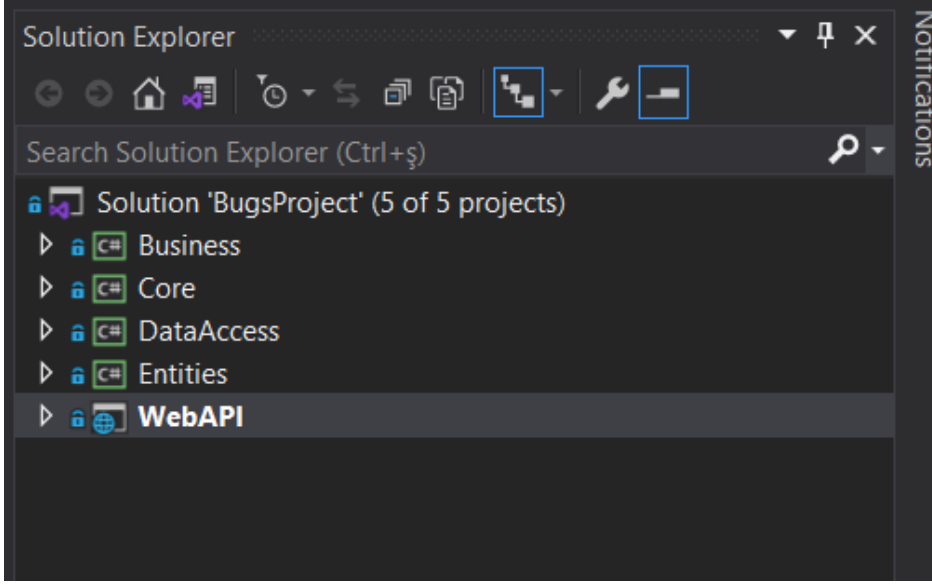


Geliştirici Dökümanı



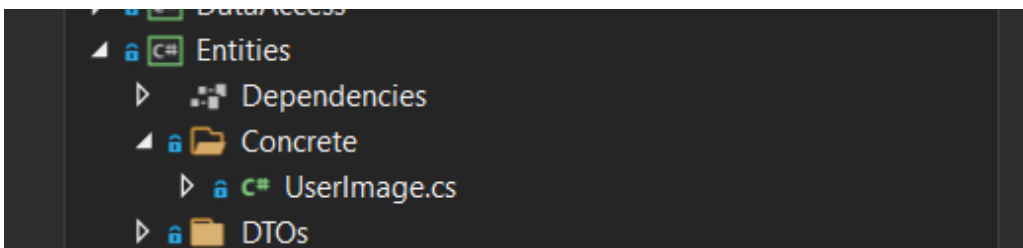
İlk olarak 5 tane katman oluşturduk. İlk katmanımız Entities katmanı, burada veri tabanımızdaki tabloları sınıf halinde tuttuk. İkinci oluşturduğumuz katman ise DataAccess bu katman ise veri tabanı işlemlerimizi yapacağımız katmandır, veriye erişim kodları yazılır. Veriye erişmek için farklı teknikler kullanılabileceği için de klasörlemeye gittik. EntityFramework kullandık. Bir sonraki katmanımız ise Core katmanı, burada her projede olabilecek kodları tutarız. Bir kez yazıp birçok projede kullanmamızı sağlar. Business katmanımız ise iş sınıfımızdır. Burada kurallarımızı belirler buradaki kurallara göre veri tabanına kayıt sağlarız. Son katmanımız ise WebAPI. Business katmanı ile Android, ios gibi farklı sistemlerle iletişim kuramaz o yüzden api, servis odaklı mimariler yazarız. Bunu da WebAPI katmanıyla sağlamış olduk.

Entities katmanından başlayacak olursak;



Concrete klasörü içerisinde veri tabanımıza karşılık gelecek sınıflar vardır.

DTOs klasörü ise data transfer object kısaltımıdır. Burda ise join yaptığımız sınıflarımız vardır.



IEntity adında interface oluşturduk. Veri tabanı tablolarına karşılık gelen sınıflara sen IEntitysin diyerek tek grupta toplamış olduk. Bu sayede referans yönetimi oluşturmayı amaçladık.

```
UserImage.cs  UserForLoginDto.cs  UserForRegisterDto.cs  dbo.UserImages [Design]
Entities
1  using Core.Entities;
2  using System;
3  using System.Collections.Generic;
4  using System.Text;
5
6  namespace Entities.Concrete
7  {
8      public class UserImage : IEntity
9      {
10         public int UserImageId { get; set; }
11         public int UserId { get; set; }
12         public string ImagePath { get; set; }
13         public DateTime UserImageDate { get; set; }
14     }
15 }
16
```

dbo.UserImages [Design]

Update Script File: dbo.UserImages.sql

	Name	Data Type	Allow Nulls	Default
	UserImageId	int	<input type="checkbox"/>	
	UserId	int	<input type="checkbox"/>	
	ImagePath	nvarchar(50)	<input type="checkbox"/>	
	UserImageDate	datetime	<input checked="" type="checkbox"/>	
			<input type="checkbox"/>	

Concrete altındaki UserImage.cs sınıfı veri tabanımızdaki dbo.UserImages tablosuna karşılık gelmektedir.

DTOs klasörü içerisine gelecek olursak da;

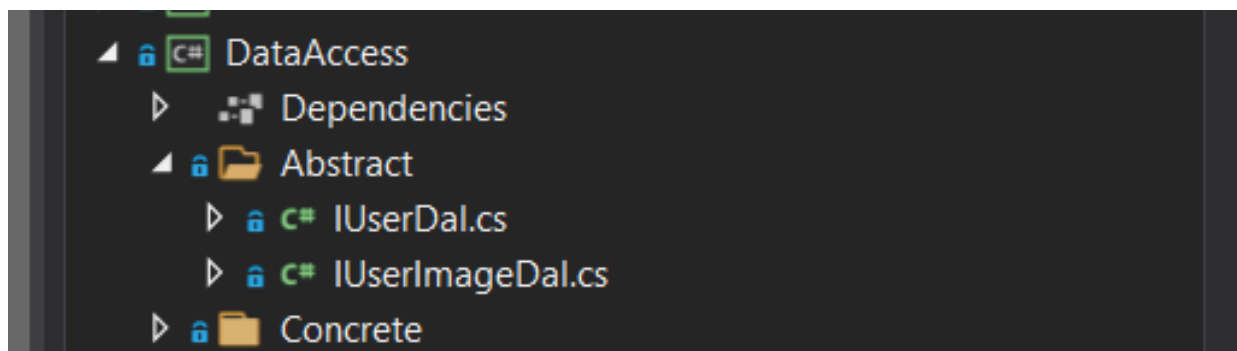
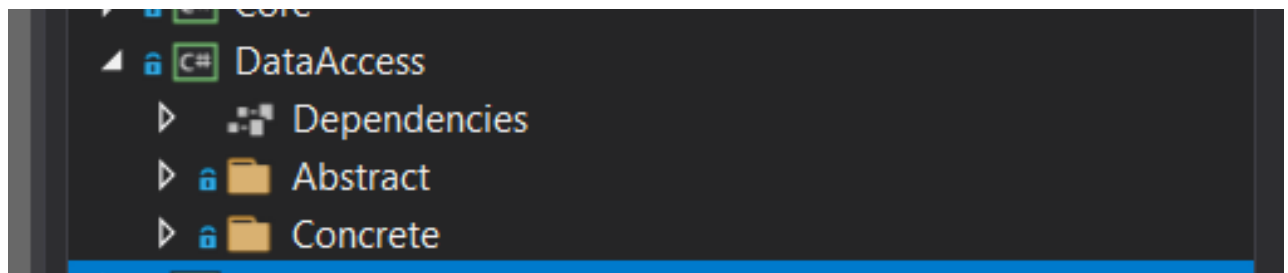
```
3
4  namespace Entities.DTOs
5  {
6      public class UserForLoginDto : IDto
7      {
8         public string Email { get; set; }
9         public string Password { get; set; }
10     }
11 }
12
```

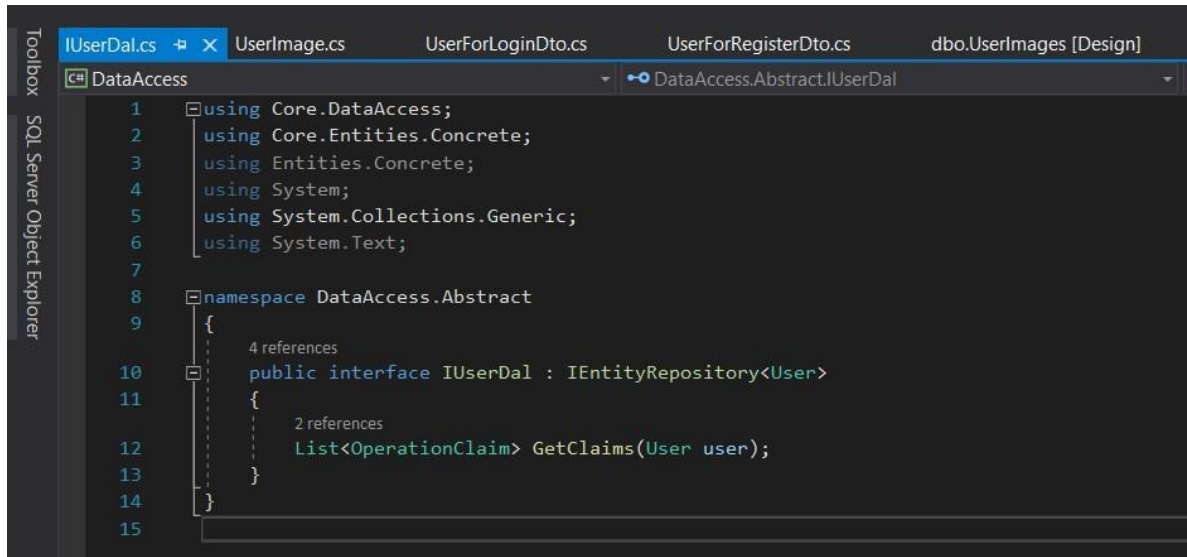
Sisteme giriş yapmak için kullanacağımız sınıftır.

```
8      {
9          3 references
10         public class UserForRegisterDto : IDto
11         {
12             2 references
13             public string Email { get; set; }
14             1 reference
15             public string Password { get; set; }
16             1 reference
17             public string FirstName { get; set; }
18             1 reference
19             public string LastName { get; set; }
20         }
21     }
```

Bu sınıfımız ise sisteme üye olmak için kullanacağımız sınıftır.

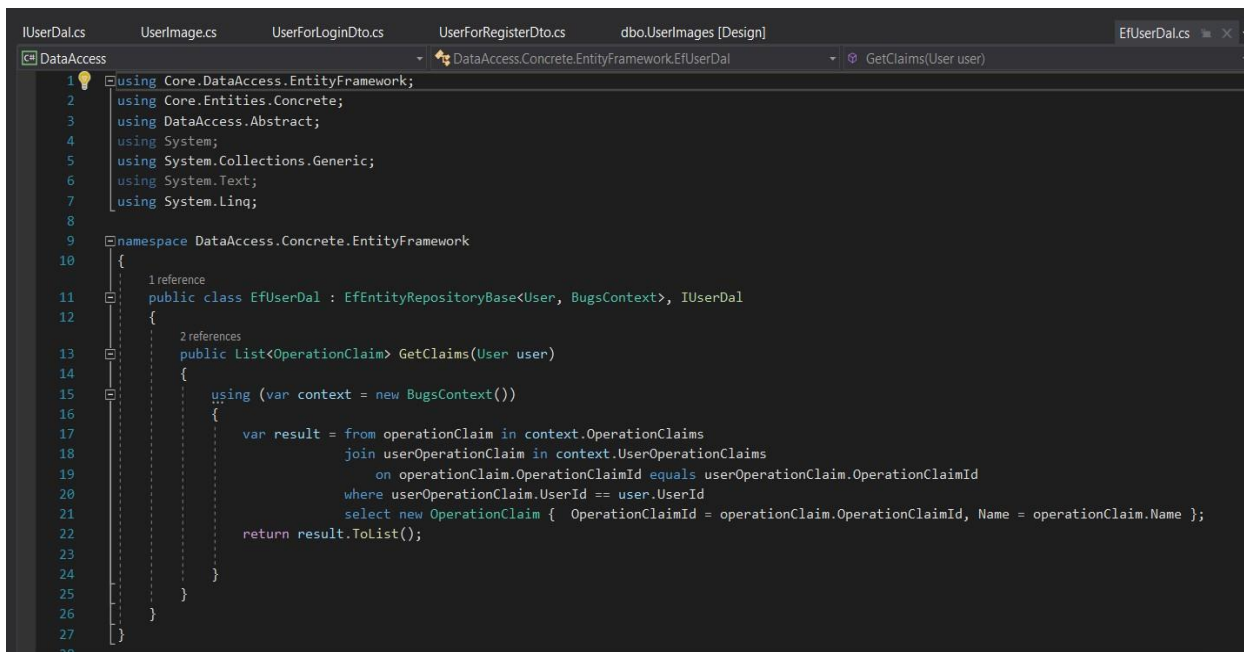
DataAccess katmanına gelecek olursak veri tabanı işlerimizi yapan katman demiştik. İçerisinde Abstract ve Concrete klasörü bulunduruyor. Soyut sınıflarımızı Abstract içerisinde somut sınıflarımızı ise Concrete klasöründe tuttuk.



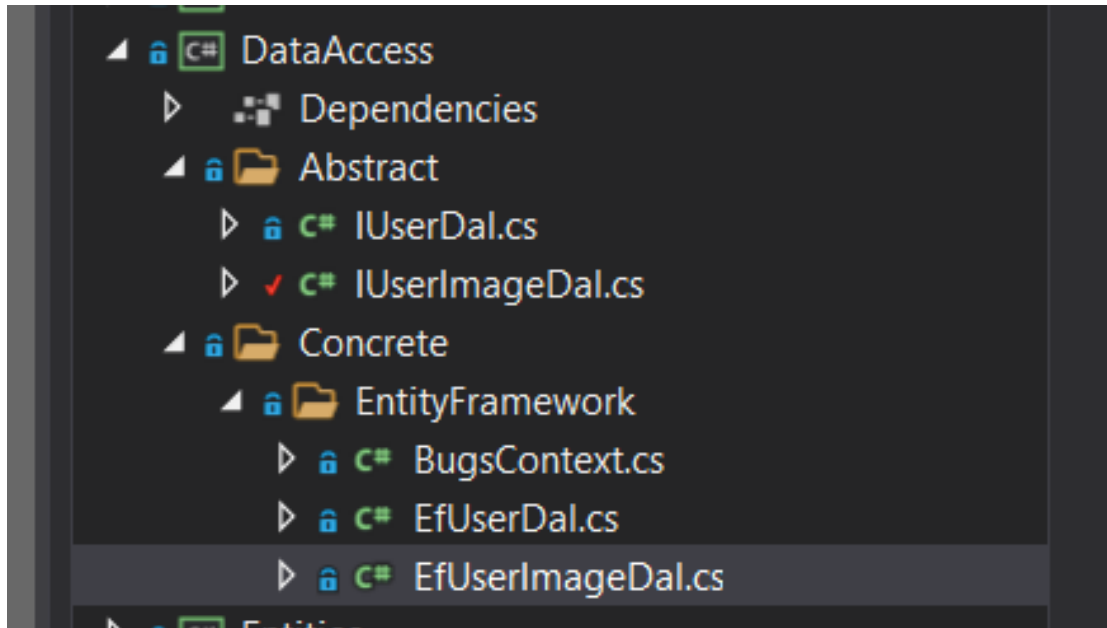


```
1 using Core.DataAccess;
2 using Core.Entities.Concrete;
3 using Entities.Concrete;
4 using System;
5 using System.Collections.Generic;
6 using System.Text;
7
8 namespace DataAccess.Abstract
9 {
10     4 references
11     public interface IUserDal : IEntityRepository<User>
12     {
13         2 references
14         List<OperationClaim> GetClaims(User user);
15     }
16 }
```

CRUD operasyonlarımız(Ekleme-Silme-Güncelleme-Listeleme) bizim tüm Dal(Data Access Layer) sınıflarımız için generic hale getirilip ortak kullanılabilir kodlar olduğu için IEntityRepository sınıfında generic halde oluşturduk. User sınıfı içinde implement ettik. Her Dal sınıfında kendi has metotları olabilir onu da bu sınıfta verdik. Bizim User yani kullanıcılarımız için yetkilerin getirilmesi metotunu yazdık. Bu metotların içerisini ise Concrete klasörü altında EntityFramework klasörü altında EfUserDal sınıfında doldurduk.

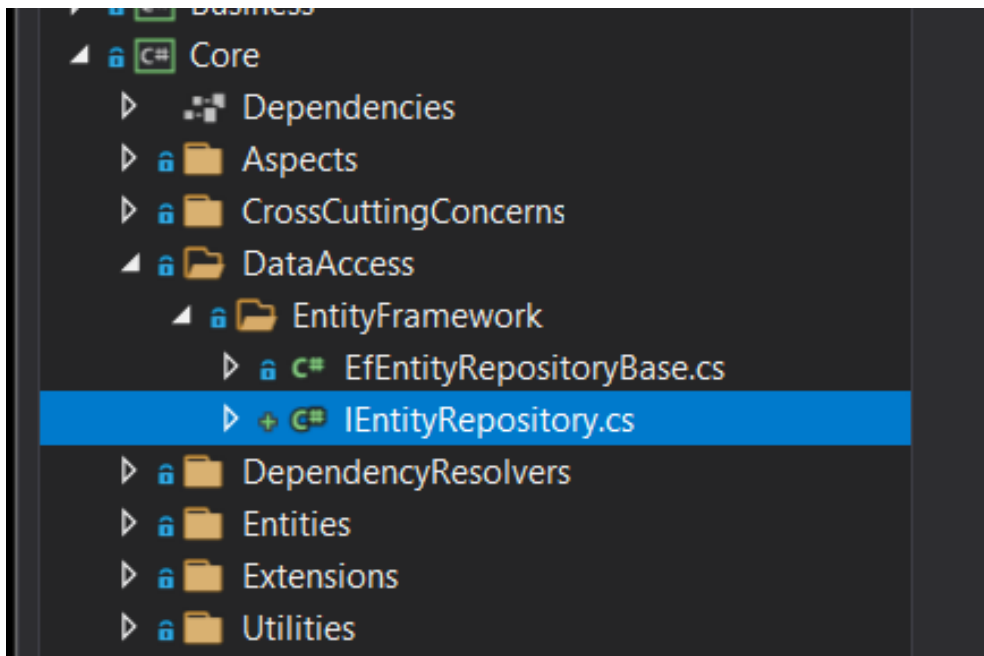


```
1 using Core.DataAccess.EntityFramework;
2 using Core.Entities.Concrete;
3 using DataAccess.Abstract;
4 using System;
5 using System.Collections.Generic;
6 using System.Text;
7 using System.Linq;
8
9 namespace DataAccess.Concrete.EntityFramework
10 {
11     1 reference
12     public class EfUserDal : EfEntityRepositoryBase<User, BugsContext>, IUserDal
13     {
14         2 references
15         public List<OperationClaim> GetClaims(User user)
16         {
17             using (var context = new BugsContext())
18             {
19                 var result = from operationClaim in context.OperationClaims
20                             join userOperationClaim in context.UserOperationClaims
21                             on operationClaim.OperationClaimId equals userOperationClaim.OperationClaimId
22                             where userOperationClaim.UserId == user.UserId
23                             select new OperationClaim { OperationClaimId = operationClaim.OperationClaimId, Name = operationClaim.Name };
24                 return result.ToList();
25             }
26         }
27     }
28 }
```



IUserImageDal.cs sınıfı kendine has bir metot içermediği için base olarak yazdığımız IEntityRepository sınıfını implement etti. Bu sınıfın somutu olan EfUserImageDal ise yine base olarak EfEntityRepositoryBase sınıfını implemente etti böylece generic olarak yazılmış olan veri tabanına ekleme kodlarına UserImage sınıfını kullanarak sahip oldu.

Bu bahsetmiş olduğumuz IEntityRepository ve EfEntityRepositoryBase generic yapıda olup diğer projelerde değişikliğe uğramadan kullanılabileceği için Core katmanına yazdık.



IEntityRepository.cs sınıfı ise;

```
1 using Core.Entities;
2 using System;
3 using System.Collections.Generic;
4 using System.Linq.Expressions;
5 using System.Text;
6
7 namespace Core.DataAccess
8 {
9     public interface IEntityRepository <T> where T : class, IEntity, new()
10     {
11         List<T> GetAll(Expression<Func<T, bool>> filter = null);
12         T Get(Expression<Func<T, bool>> filter);
13         void Add(T entity);
14         void Update(T entity);
15         void Delete(T entity);
16     }
17 }
18
```

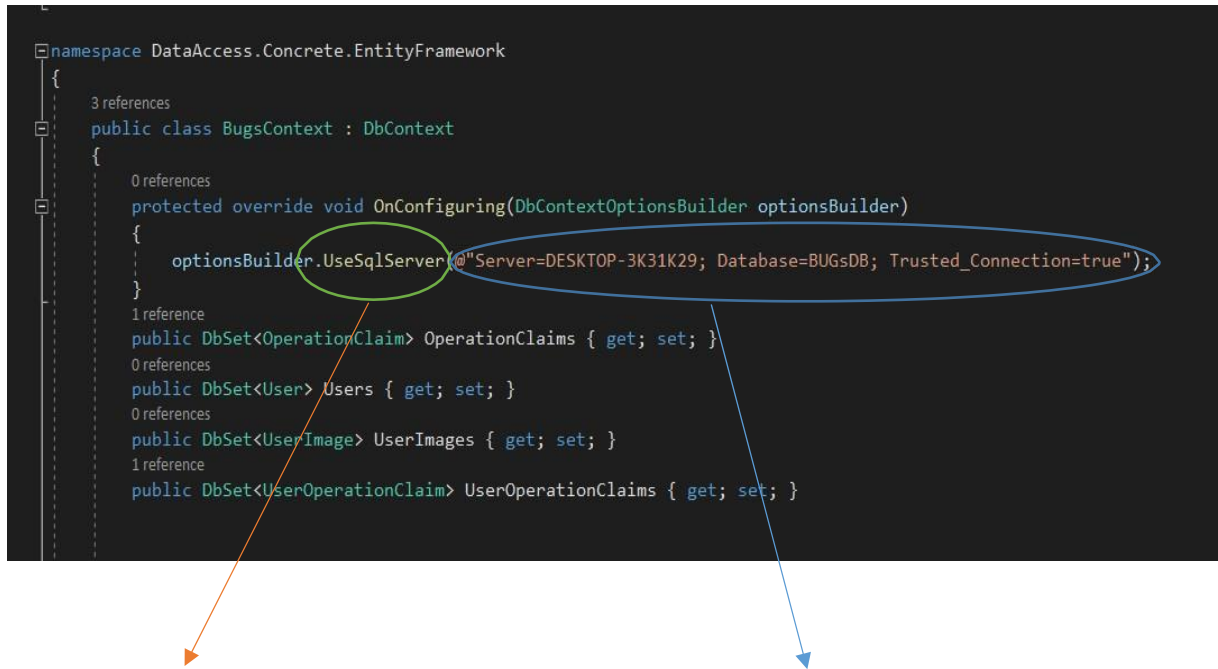
Bu yapıdadır. T türü için CRUD operasyonları içerir ama biz sadece veri tabanı tablolarımıza karşılık gelecek sınıflar için çalışmak istediğimiz için T ye filtre uyguladık.

Gelecek olan T bir class olmalı,

IEntity türünde (yani veri tabanı tablosuna karşılık gelecek bir sınıf) aynı zamanda instancesi oluşturulabilir yani new () lenebilir bir sınıf olsun dedik. Bu new () koşulunu ekleyerek IEntity interfacesinin T yerine kullanılabilmesini engellemiş olduk.

IEntityRepository interfacesindeki metotların içini ise EfEntityRepositoryBase sınıfında doldurduk. İsimlendirmedeki Ef(Entity framework e karşılık gelmektedir, standart isimlendirme yapılmaya özen gösterilmiştir.) EfEntityRespositoryBase sınıfından bahsedilmeden önce Context sınıfımızı anlatalım;

Context sınıfı genel anlamda veri tabanı işlemlerinin halledildiği sınıftır.



Sql server kullan dedik.

Veri tabanı yolumuz.

`public DbSet<User> Users { get; set; }` satırı ise User sınıfı Users tablosuna denk geliyor anlamına gelmektedir.

EfEntityRespositoryBase sınıfımız bizden bir veritabanı sınıfı ve bir context istiyor. Aynı zamanda da IEntityRespository i implemente ediyor. Yani orada oluşturduğumuz metotların bu sınıfta içerisinde TEntity(veri tabanı sınıfımız) ve Context ile doldurcaz. T için yanlış sınıf gönderilmemesi için yine şart koyduk.

```

namespace Core.DataAccess.EntityFramework
{
    2 references
    public class EfEntityRepositoryBase<TEntity, TContext> : IEntityRepository<TEntity>
        where TEntity : class, IEntity, new()
        where TContext : DbContext, new()
    {
        3 references
        public void Add(TEntity entity)
        {
            using (TContext context = new TContext())
            {
                var addedEntity = context.Entry(entity);
                addedEntity.State = EntityState.Added;
                context.SaveChanges();
            }
        }

        3 references
        public void Delete(TEntity entity)
        {
            using (TContext context = new TContext())
            {
                var deletedEntity = context.Entry(entity);
                deletedEntity.State = EntityState.Deleted;
                context.SaveChanges();
            }
        }
    }
}

```

Bu kod ile veri tabanına yerleşir. Veri tabanıyla

Üst satırlardaki işlemleri gerçekleştirir.

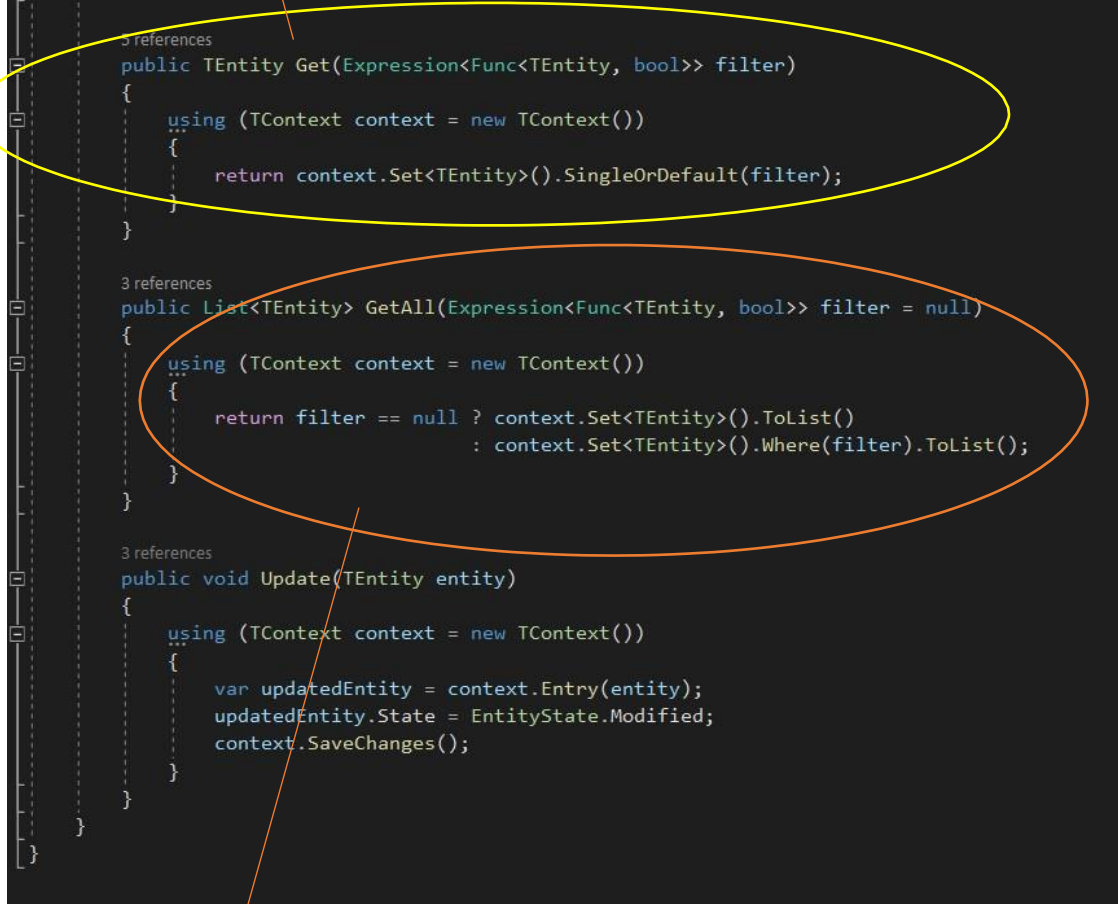
Veri tabanında ne yapılacağını söyler.(Ekleme işlemi yapar.)

Context hafızada büyük bir yer kapladığı içinde using bloğu içerisinde yazdık. Kullanıldıktan sonra garbage collector ile hafızadan silinir.

Silme için Deleted,

Güncelleme için Modified kullanarak bu işlemleri yaptık.

Bu kodla da Linq ile arama yapmayı sağlar. Gönderilen filtredeki elamanı bulur ve o datayı verir.

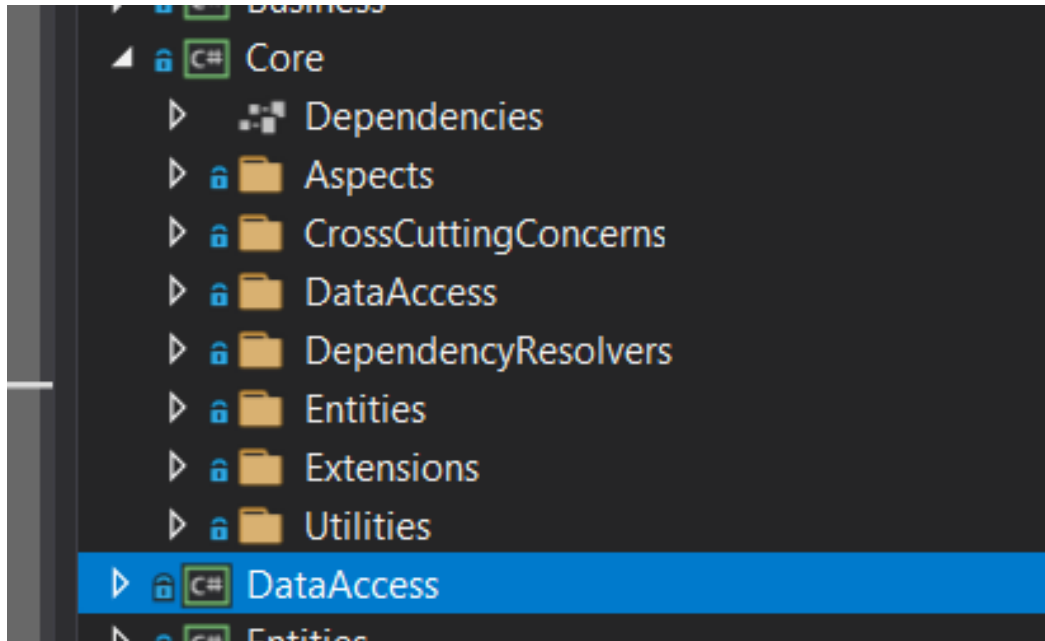


```
5 references
public TEntity Get(Expression<Func<TEntity, bool>> filter)
{
    using (TContext context = new TContext())
    {
        return context.Set<TEntity>().SingleOrDefault(filter);
    }
}

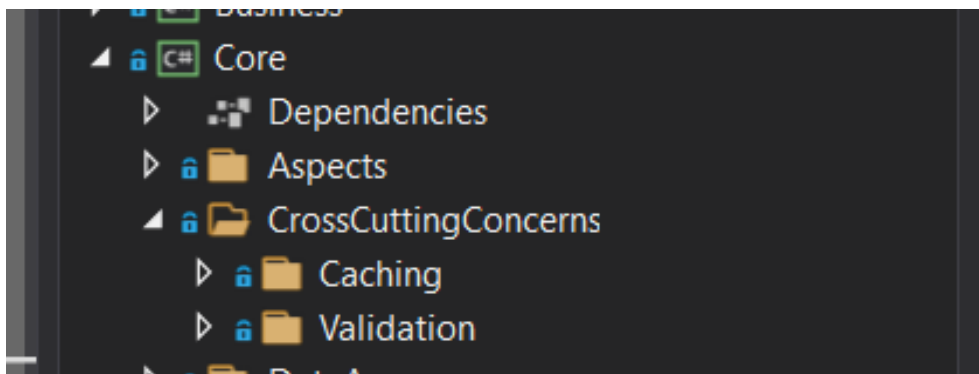
3 references
public List<TEntity> GetAll(Expression<Func<TEntity, bool>> filter = null)
{
    using (TContext context = new TContext())
    {
        return filter == null ? context.Set<TEntity>().ToList()
            : context.Set<TEntity>().Where(filter).ToList();
    }
}

3 references
public void Update(TEntity entity)
{
    using (TContext context = new TContext())
    {
        var updatedEntity = context.Entry(entity);
        updatedEntity.State = EntityState.Modified;
        context.SaveChanges();
    }
}
```

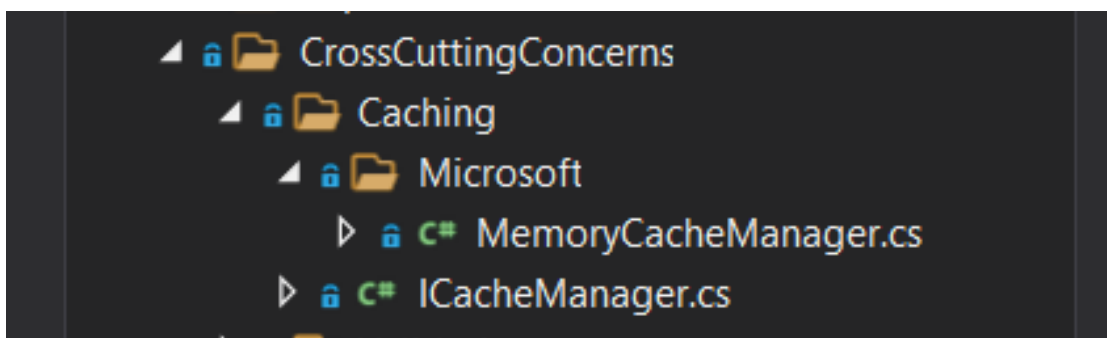
Bu kodla ise filtre verilirse filtreye göre listeleme verilmezse de tüm veriyi listeleme işlemi yapılmıştır.



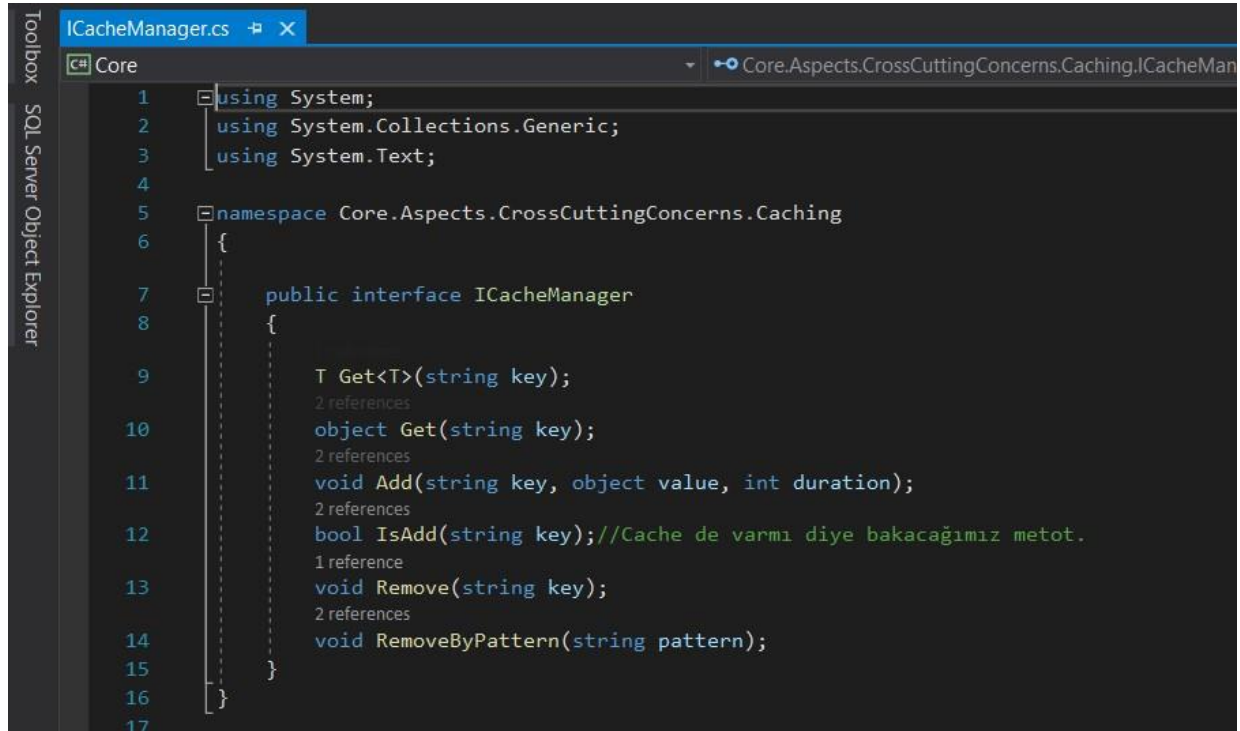
Core katmanı bizim projemizdeki evrensel katmandır. Diğer projeler için alıp değişiklik yapmadan kullanabilirsiniz. Klasörleri açıklamaya başlayacak olursak bizim CrossCuttingConcerns dediğimiz uygulamayı dikine kesen ilgi alanları olarak Türkçeye çevireceğimiz yapılarımız vardır. Bu yapılar Loglama, Cache, Transaction, Authorization(yetkilendirme), Validation olarak sayılabilir. Her katmanda bunları kullanabiliriz.



Cache ve Validation için ayrı klasörleme yaptık. Caching için ayrıntı verecek olursak;



Cache gelecek taleplerin daha hızlı sunulması için verileri geçici depolayan yapılardır. Verilere tekrar erişmek istendiğinde uygulamamız veriyi veri tabanından değil cache den alır. Bu da ölçeklenebilir, hızlı, performansı yüksek proje olmasını sağlar. Resim ekleme, güncelleme gibi durumlarda da cache in uçurulmasını isteriz.



```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace Core.Aspects.CrossCuttingConcerns.Caching
6 {
7     public interface ICacheManager
8     {
9         T Get<T>(string key);
10        object Get(string key);
11        void Add(string key, object value, int duration);
12        bool IsAdd(string key); //Cache de varmı diye bakacağımız metot.
13        void Remove(string key);
14        void RemoveByPattern(string pattern);
15    }
16 }
17
```

Biz cache e bir key yollayacağız ve o da keye karşılık gelen değeri bize verecek o yüzden parametre olarak key istedik.

Add() metodu cache e alır.

Duration ise geçerlilik süresidir.

IsAdd() metodu cache de varmı ona bakar.

RemoveByPattern(string pattern) içerisinde belirlediğimiz kelimelerin geçtiği metotlardan cache i uçurur. Çalışma anında bellekten siler.

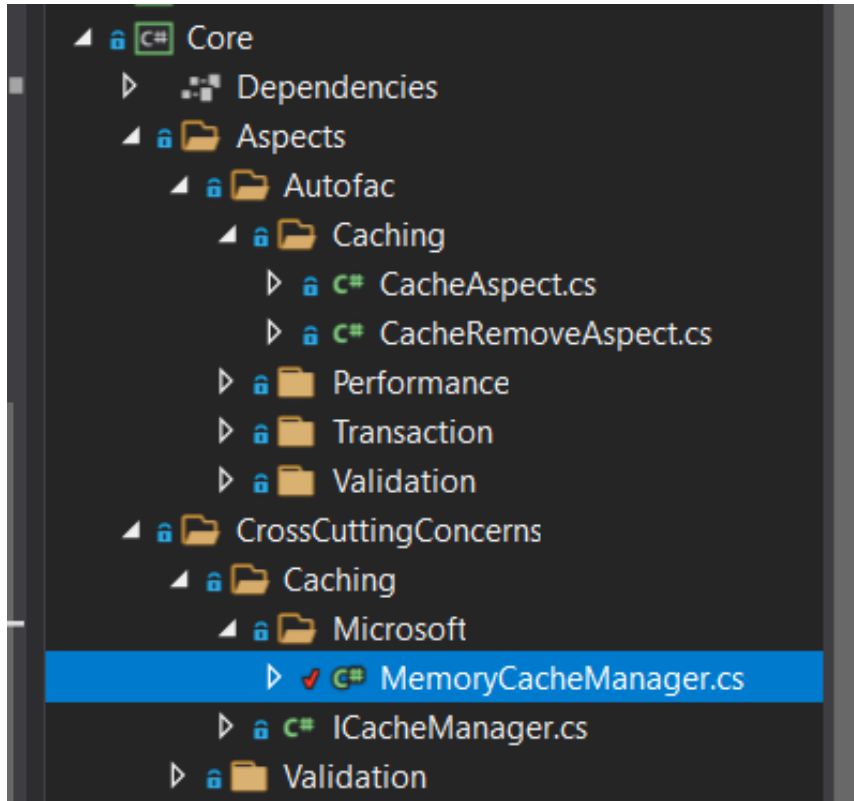
```
11 {
12     2 references
13     public class MemoryCacheManager : ICacheManager
14     {
15         IMemoryCache _memoryCache;
16         0 references
17         public MemoryCacheManager()
18         {
19             _memoryCache = ServiceTool.ServiceProvider.GetService<IMemoryCache>();
20
21         2 references
22         public void Add(string key, object value, int duration)
23         {
24             _memoryCache.Set(key, value, TimeSpan.FromMinutes(duration));
25
26         1 reference
27         public T Get<T>(string key)
28         {
29             return _memoryCache.Get<T>(key);
30
31         2 references
32         public object Get(string key)
33         {
34             return _memoryCache.Get(key);
35
36         2 references
37         public bool IsAdd(string key)
38         {
39             return _memoryCache.TryGetValue(key, out _);
40
41         1 reference
42         public void Remove(string key)
43         {
44             _memoryCache.Remove(key);
45     }
```

```
2 references
public void RemoveByPattern(string pattern)
{
    var cacheEntriesCollectionDefinition = typeof(MemoryCache).GetProperty("EntriesCollection",
        System.Reflection.BindingFlags.NonPublic | System.Reflection.BindingFlags.Instance);
    var cacheEntriesCollection = cacheEntriesCollectionDefinition.GetValue(_memoryCache) as dynamic;
    List<ICacheEntry> cacheCollectionValues = new List<ICacheEntry>();

    foreach (var cacheItem in cacheEntriesCollection)
    {
        ICacheEntry cacheItemValue = cacheItem.GetType().GetProperty("Value").GetValue(cacheItem, null);
        cacheCollectionValues.Add(cacheItemValue);
    }

    var regex = new Regex(pattern, RegexOptions.Singleline | RegexOptions.Compiled | RegexOptions.IgnoreCase);
    var keysToRemove = cacheCollectionValues.Where(d => regex.IsMatch(d.Key.ToString())).Select(d => d.Key).ToList();

    foreach (var key in keysToRemove)
    {
        _memoryCache.Remove(key);
    }
}
```



Aspects klasörü içerisinde de Caching klasörü bulunuyor. Bunu da cache işlemlerini attribute şeklinde yazabilmek için oluşturduk. Aspect klasörünü de açıklamamız gerekirse;

AOP (Aspect Oriented Programming) kullandık. Aop yazılım karmaşıklığını azaltmaya, modüleriteyi arttırmaya yarayan yaklaşım biçimidir. Cross Cutting Concerns gibi yapıları attribute halinde yazıp farklı modüllere ayırarak karmaşıklığı azaltmamızı sağladı.

```
MemoryCacheManager.cs*   ICacheManager.cs
Core
using Core.Interceptors;
using Core.Aspects.CrossCuttingConcerns.Caching;

namespace Core.Aspects.Autofac.Caching
{
    2 references
    public class CacheAspect : MethodInterception
    {
        15 private int _duration;
        16 private ICacheManager _cacheManager;

        1 reference
        public CacheAspect(int duration = 60)
        {
            _duration = duration;
            _cacheManager = ServiceTool.ServiceProvider.GetService<ICacheManager>();
        }

        3 references
        public override void Intercept(IInvocation invocation)
        {
            26 var methodName = string.Format($"{invocation.Method.ReflectedType.FullName}.{invocation.Method.Name}");
            27 var arguments = invocation.Arguments.ToList();
            28 var key = $"{methodName}({string.Join(",", arguments.Select(x => x?.ToString() ?? "<Null>"))})";
            29 if (_cacheManager.IsAdd(key))
            {
                30 invocation.ReturnValue = _cacheManager.Get(key);
                31 return;
            }
            32 invocation.Proceed();
            33 _cacheManager.Add(key, invocation.ReturnValue, _duration);
        }
    }
}
```

Constructorsdurationisminde parametrealıyor. Bu cachealma süresidir. Eğer parametre yollamazsak varsayılan olarak 60 dakika olarak belirledik. 60 dakika cache alırsın sonrasında uçurur.

Metotun parametresi varsa listeye ekler

Burda ise key oluşturduk.

if satırıyla da anahtarı alıyoruz. Cache bakıyoruz bu keye karşılık data var mı eğer varsa cache den çağırıyoruz. Yoksa veritabanından.

Uçurma işlemleri için kullanılacak.

```
CacheRemoveAspect.cs MemoryCacheManager.cs* ICacheManager.cs
Core
using Castle.DynamicProxy;
using Core.Aspects.CrossCuttingConcerns.Caching;
using Core.Interceptors;
using Core.Utilities.IoC;
using System;
using System.Collections.Generic;
using Microsoft.Extensions.DependencyInjection;
using System.Text;

namespace Core.Aspects.Autofac.Caching
{
    3 references
    public class CacheRemoveAspect : MethodInterception
    {
        private string _pattern;
        private ICacheManager _cacheManager;

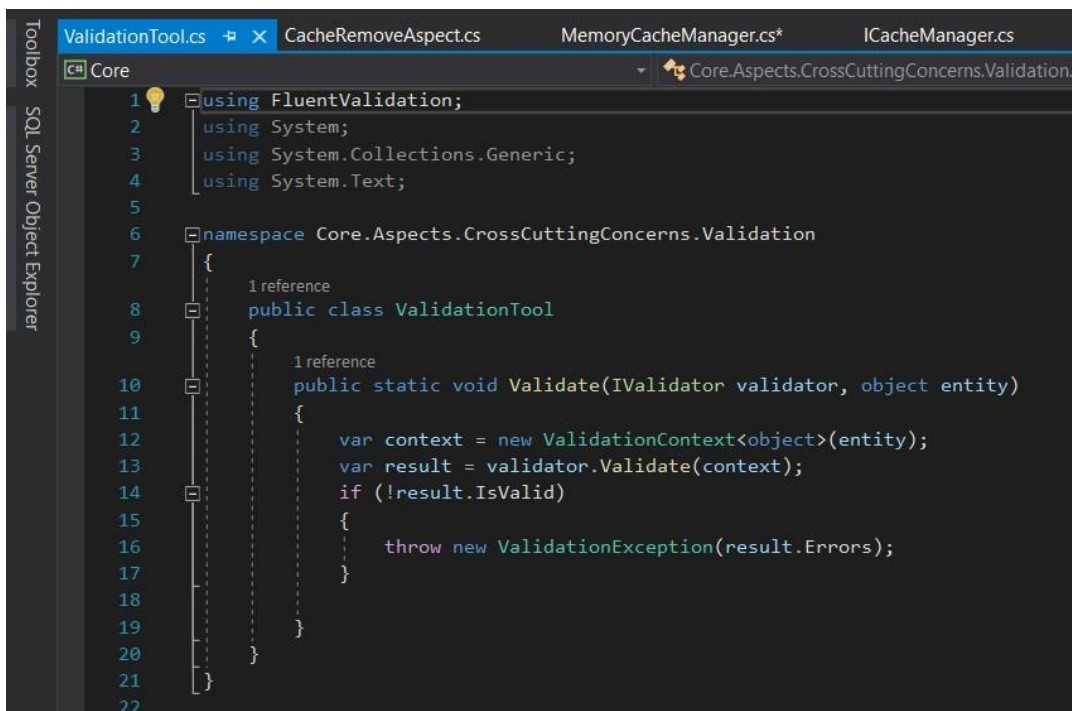
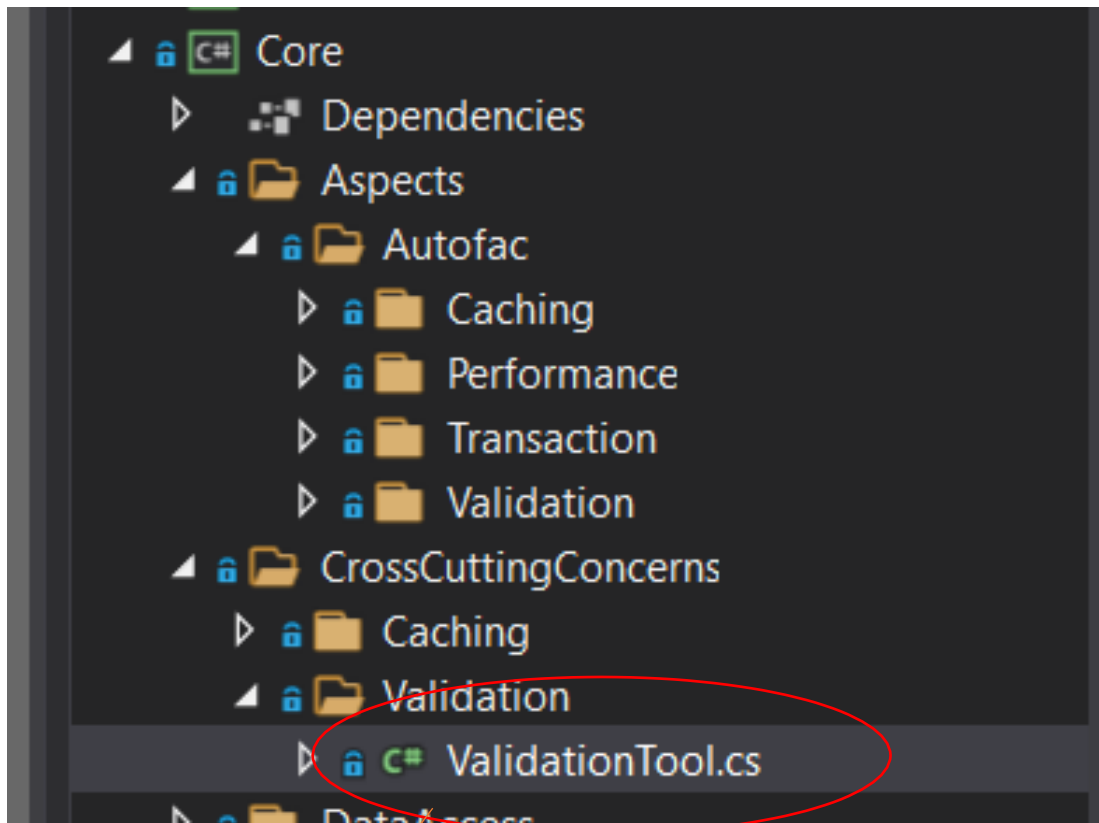
        2 references
        public CacheRemoveAspect(string pattern)
        {
            _pattern = pattern;
            _cacheManager = ServiceTool.ServiceProvider.GetService<ICacheManager>();
        }

        2 references
        protected override void OnSuccess(IInvocation invocation)
        {
            _cacheManager.RemoveByPattern(_pattern);
        }
    }
}
```

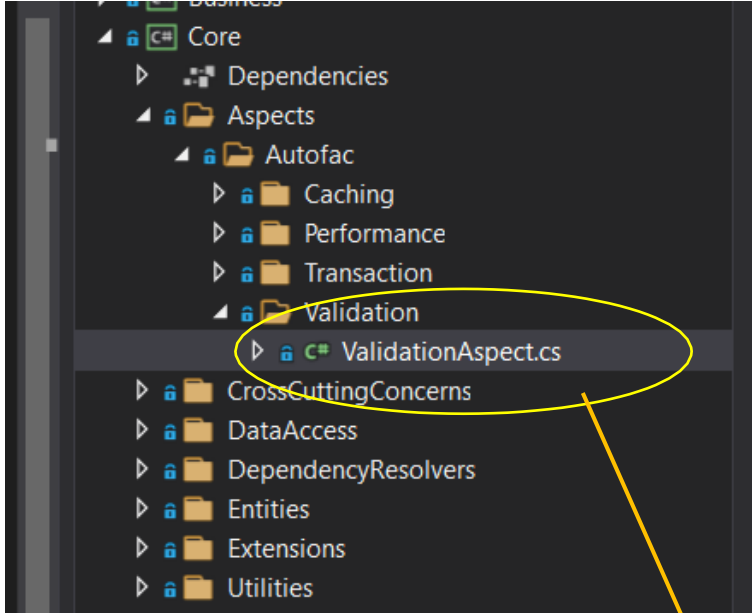
Pattern ismini verdiğimiz string bir ifade geldiğinde, o ifadenin geçtiği metotlardan uçurur. Mesela parametre olarak Get geldi. İçinde get bulunan bellekteki tüm keyleri iptal et.

IInvocation parametre olarak gelecek metotumuzdur. Bu kod bloğu metot başarılı olunca çalış demektir.

Bizim metotlarımız için doğrulama yapmamız gerekmektedir.



Bu sınıf sayesinde doğrulama yapılacak işlemi parametre olarak alır. Validate metodu sayesinde de doğrular. ValidationTool.Validate(validator,entity) şeklinde kullanım sağlar.



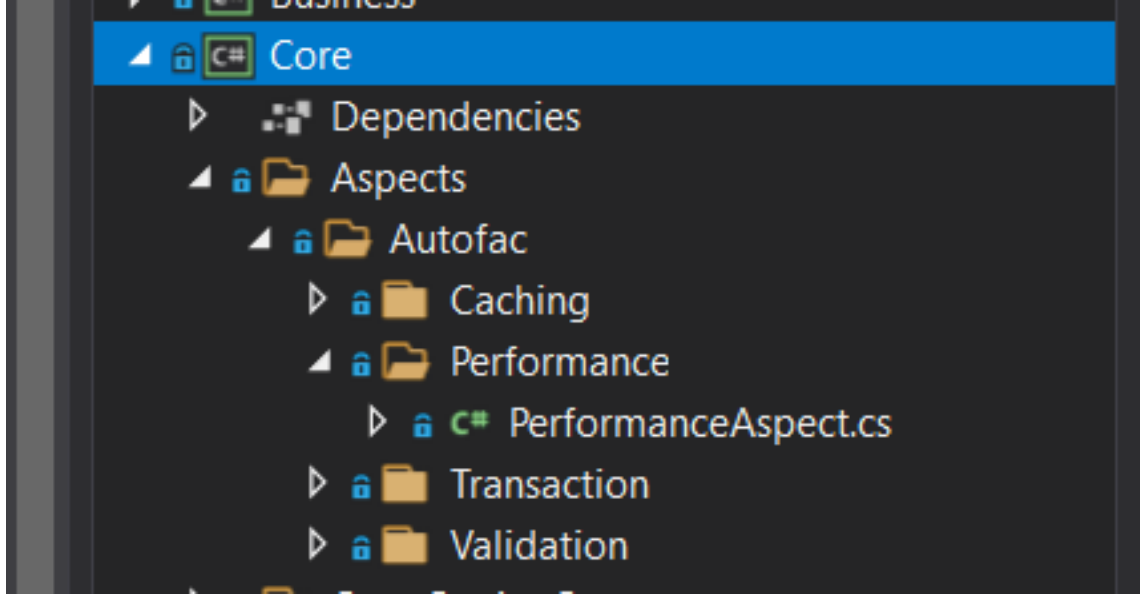
```
namespace Core.Aspects.Autofac.Validation
{
    1 reference
    public class ValidationAspect : MethodInterception
    {
        private Type _validatorType;
        0 references
        public ValidationAspect(Type validatorType)
        {
            if (!typeof(IValidator).IsAssignableFrom(validatorType))
            {
                throw new System.Exception("Bu bir doğrulama sınıfı değil");
            }

            _validatorType = validatorType;
        }
        4 references
        protected override void OnBefore(IInvocation invocation)
        {
            var validator = (IValidator)Activator.CreateInstance(_validatorType);
            var entityType = _validatorType.BaseType.GetGenericArguments()[0];
            var entities = invocation.Arguments.Where(t => t.GetType() == entityType);
            foreach (var entity in entities)
            {
                ValidationTool.Validate(validator, entity);
            }
        }
    }
}
```

Bu sınıf sayesinde de doğrulama metotlarımızı Attribute halinde yazabiliyoruz. Doğrulama işlemleri OnBefore yani metotların başına yazılacağı için OnBefore metotunun içerisi dolduruldu. OnBefore

alıřacak metodu parametre olarak alır nce doęrulama iřlemine yapıp daha sonra parametre olarak aldıęı metodu alıřtırır.

Core klasr altında Performance klasr ise sistemimizdede performans zafiyeti olunca bizi uyarması iin yazdıđ. Timer kullanarak metodun nnde kronometre bařlattık metod bitiminde durdurduk. Geen sreyi bulduk. Bu sre bizim belirledięimiz alıřma sresini gemiřse sistemde performans zafiyeti olduęu anlamına gelmektedir. Aspect olarak yazdıęımız iinde Attribute halinde yazabiliriz. Metodun nne [PerformanceAspect(5)] řeklinde yazabilmemizi saęladık.

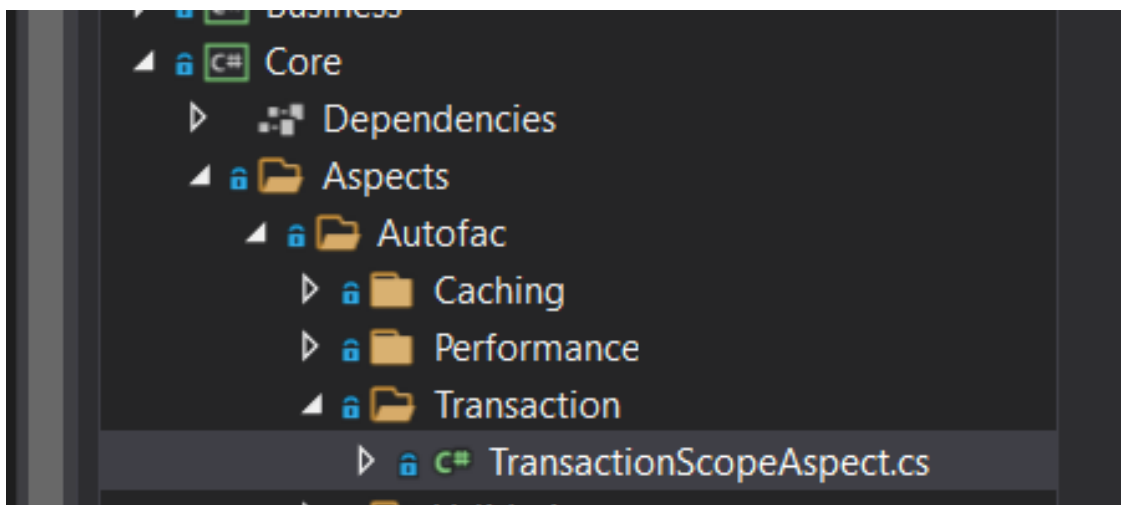


```

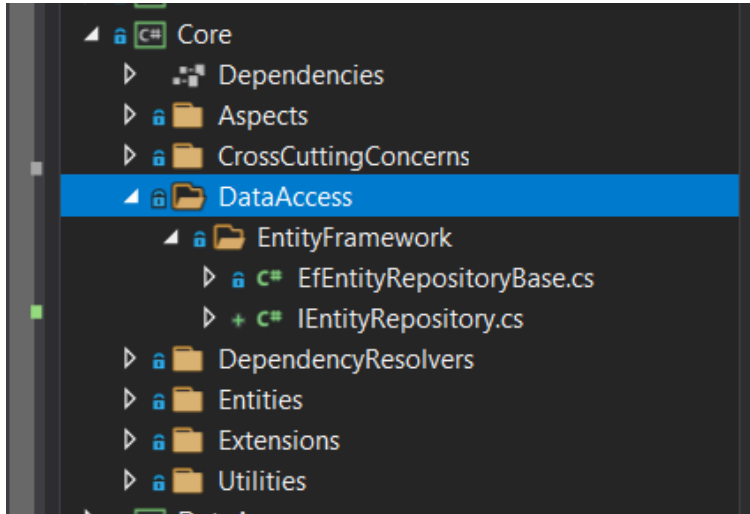
9
10 namespace Core.Aspects.Autofac.Performance
11 {
12     2 references
13     public class PerformanceAspect : MethodInterception
14     {
15         private int _interval;
16         private Stopwatch _stopwatch;
17
18         1 reference
19         public PerformanceAspect(int interval)
20         {
21             _interval = interval;
22             _stopwatch = ServiceTool.ServiceProvider.GetService<Stopwatch>();
23         }
24
25         4 references
26         protected override void OnBefore(IInvocation invocation)
27         {
28             _stopwatch.Start();
29         }
30
31         2 references
32         protected override void OnAfter(IInvocation invocation)
33         {
34             if (_stopwatch.Elapsed.TotalSeconds > _interval)
35             {
36                 Debug.WriteLine($"Performance : {invocation.Method.DeclaringType.FullName}." +
37                     $"{invocation.Method.Name}-->{_stopwatch.Elapsed.TotalSeconds}");
38             }
39             _stopwatch.Reset();
40         }
41     }
42 }

```

Core katmanında AOP mantığıyla Attribute halinde yazmak için oluşturduğumuz bir başka yapı da Transaction. Uygulamamızda tutarlılığı korumak için yapılır. Veritabanımızda yapılan işlemlerin her birisi bizim için transactiondur. Bunu bir örnekle açıklamak gerekirse; banka hesabımda 100tl var 10 tl para gönderme işlemi yapacağım. Benim hesabımdan 10tl düşüp, gönderdiğim hesapta 10tl yükselmesi lazım. Benim hesabımdan 10tl düşükten sonra gönderdiğim hesabın 10tl artması sırasında hata alınırsa benim hesabıma iade edilmesi lazım, işlemi iptal etmesi gerekir. Sistem güvenliğimiz ve tutarlılığımız için bu yapıyı oluşturduk.



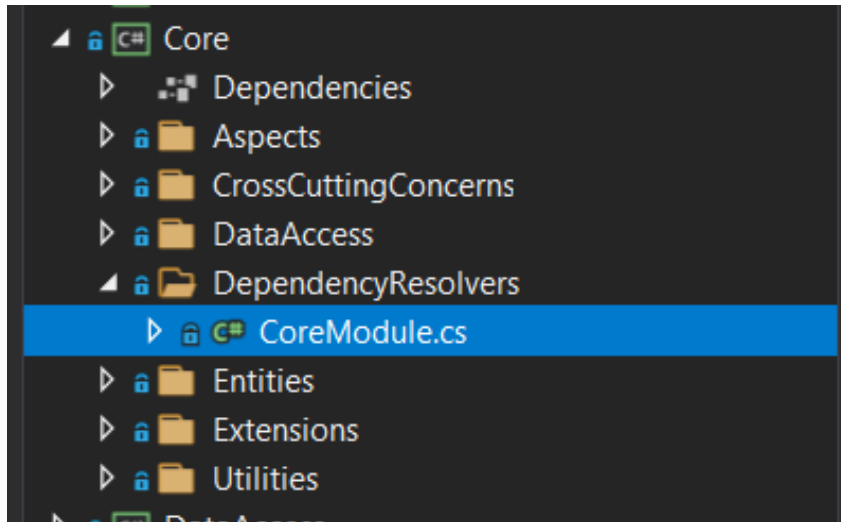
Core katmanındaki DataAccess ise bizim diğer projelerimizde kod değiştirmeden veri tabanı işlemleri yapabileceğimiz klasördür.



Bu sınıfımız generic yapıda CRUD operasyonlarını içeren interfacedir. Bu metotların içini ise EFEntityRepositoryBase içerisinde doldurduk. Yukarıda daha ayrıntılı açıklanmıştır.

```
6
7 namespace Core.DataAccess
8 {
9     public interface IEntityRepository <T> where T : class, IEntity, new()
10    {
11        List<T> GetAll(Expression<Func<T, bool>> filter = null);
12        T Get(Expression<Func<T, bool>> filter);
13        void Add(T entity);
14        void Update(T entity);
15        void Delete(T entity);
16    }
17 }
18
```

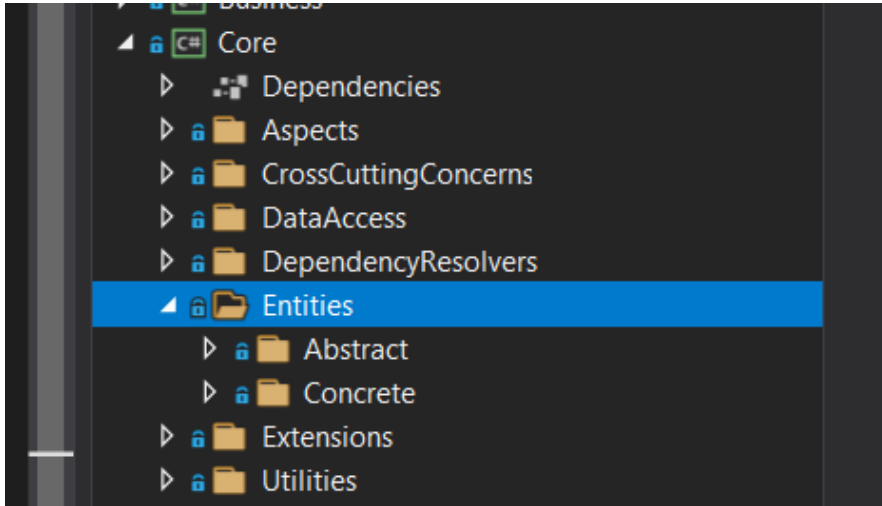
Tüm projelerde kullanabileceğimiz injection yapılabilecek yapı oluşturmak içinde DependencyResolvers(BağımlılıkÇözümleyici) içerisinde CoreModule sınıfını oluşturduk. Evrensel bağımlılıklarımızı buraya yazdık.



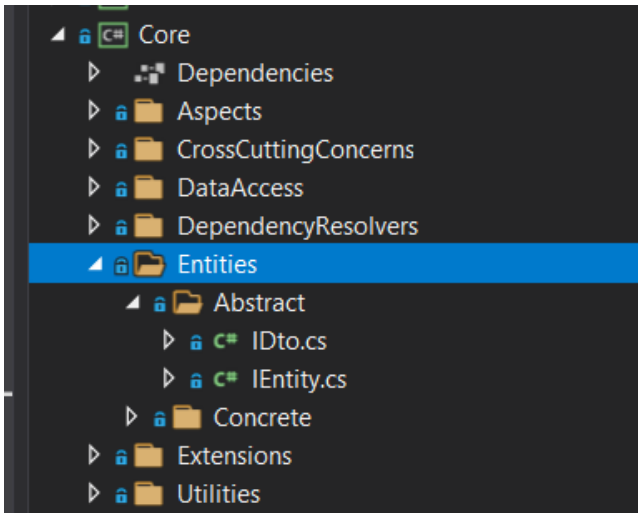
```
11 namespace Core.DependencyResolvers
12 {
13     1 reference
14     public class CoreModule : ICoreModule
15     {
16         2 references
17         public void Load(IServiceCollection serviceCollection)
18         {
19             serviceCollection.AddMemoryCache();
20             serviceCollection.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();
21             serviceCollection.AddSingleton<ICacheManager, MemoryCacheManager>();
22             serviceCollection.AddSingleton<Stopwatch>();
23         }
24     }
25 }
```

Biri bizden HttpContextAccessor isterse
HttpContextAccessor ver. Bizim yerimize
new() le, instance üret dedik.

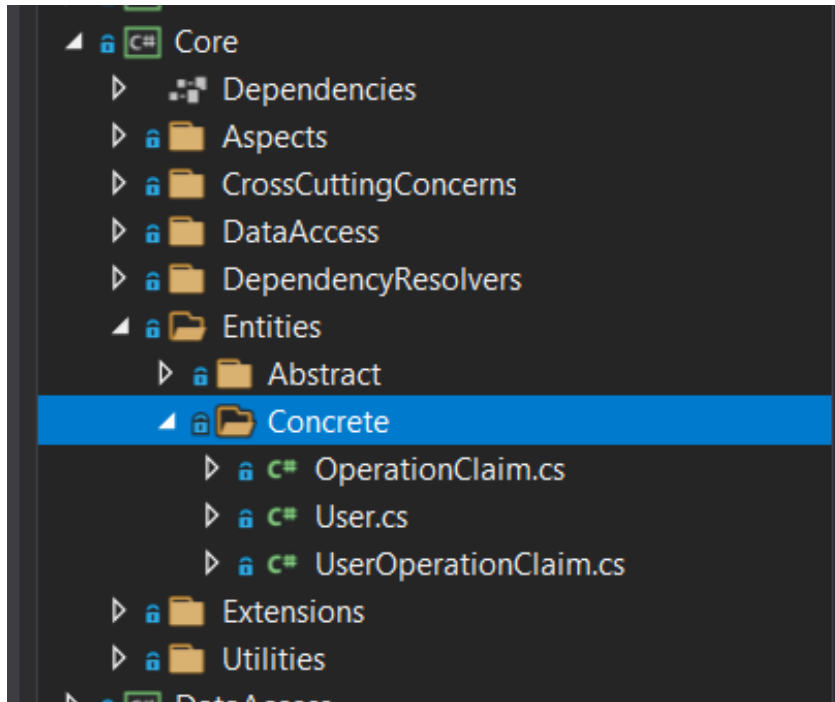
Yine aynı şekilde instance üretir,
bellekte referansını oluşturur.



Bizim her projemizde kullanabileceğimiz veri tabanı sınıflarımızı Entities klasöründe yazdık. Abstract klasörü içerisinde iki sınıfımız var.



Bu iki sınıfta amacı veri tabanı sınıflarını interfacerler sayesinde ortak çatıda toplayabilmek için. Bu sayede demiş olduk ki; IEntity sınıfı implement eden sınıflar veri tabanı nesnesi(sınıftır). Aynı şekilde IDto sınıfı implemente eden sınıflarda join işlemi yapılmış veri tabanı sınıfıdır.



OperationClaim sınıfımız; bizim sistemimizde işlem yapmak için gerekli olacak yetkileri tutacak sınıftır.(admin, user yetkisi gibi. Operayon bazında yetkilendirme de yapabiliriz) Aşağıda ekran resmi olan veri tabanı tablomuza denk gelmektedir.

	Name	Data Type	Allow Nulls	Default
	OperationClaimId	int	<input type="checkbox"/>	
	Name	varchar(50)	<input type="checkbox"/>	
			<input type="checkbox"/>	

```

7 public class User : IEntity
8 {
9     2 references
    public int UserId { get; set; }
10     3 references
    public string FirstName { get; set; }
11     3 references
    public string LastName { get; set; }
12     4 references
    public string Email { get; set; }
13     3 references
    public byte[] PasswordSalt { get; set; }
14     2 references
    public byte[] PasswordHash { get; set; }
15     1 reference
    public bool Status { get; set; }
16 }
17 }

```

Sistemimizdeki kullanıcılar için gerekli olan bilgileri tutan sınıftır. Aşağıdaki veri tabanı tablomuza karşılık gelmektedir.


dbo.Users [Design] ➔ X User.cs dbo.OperationClaims [Design]				
Update Script File: dbo.Users.sql				
	Name	Data Type	Allow Nulls	Default
	UserId	int	<input type="checkbox"/>	
	FirstName	nvarchar(50)	<input type="checkbox"/>	
	LastName	nvarchar(50)	<input type="checkbox"/>	
	Email	nvarchar(50)	<input type="checkbox"/>	
	PasswordSalt	varbinary(500)	<input type="checkbox"/>	
	PasswordHash	varbinary(500)	<input type="checkbox"/>	
	Status	bit	<input type="checkbox"/>	
			<input type="checkbox"/>	

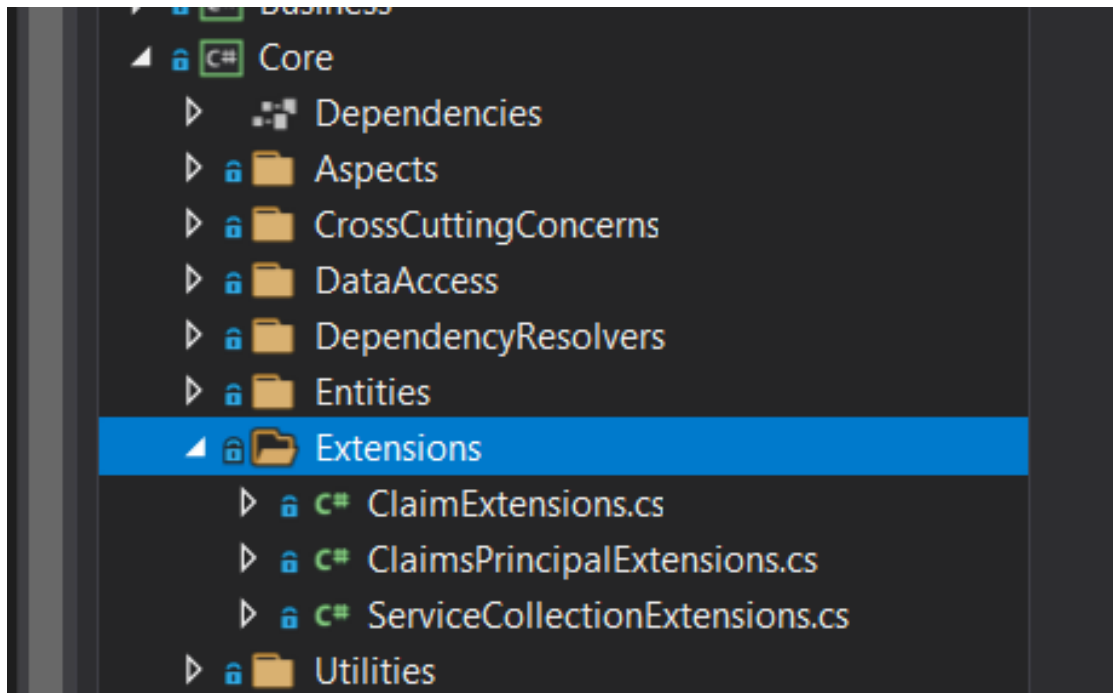

```

1 reference
public class UserOperationClaim : IEntity
{
    0 references
    public int UserOperationClaimId { get; set; }
    1 reference
    public int UserId { get; set; }
    1 reference
    public int OperationClaimId { get; set; }
}

```

UserId ve OperationClaimId lerini tuttuğumuz sınıftır. 1 id sine sahip kişinin 1id sindeki yetkiye sahiptir yapısını oluşturduk. Aşağıdaki veri tabanı tablomuza karşılık gelmektedir.

dbo.UserOperationClaims [Design] X UserOperationClaim.cs				
Update Script File: dbo.UserOperationClaims.sql				
	Name	Data Type	Allow Nulls	Default
	 UserOperationClaimId	int	<input type="checkbox"/>	
	UserId	int	<input type="checkbox"/>	
	OperationClaimId	int	<input type="checkbox"/>	
			<input type="checkbox"/>	



Extensions klasörünü de hazırda var olan sınıflara kendi metotlarımızı da eklemek, yani hazırda yazılmış sınıfı genişletmek için oluşturduk.

```
0 references
public static class ClaimExtensions
{
    1 reference
    public static void AddEmail(this ICollection<Claim> claims, string email)
    {
        claims.Add(new Claim(JwtRegisteredClaimNames.Email, email));
    }

    1 reference
    public static void AddName(this ICollection<Claim> claims, string name)
    {
        claims.Add(new Claim(ClaimTypes.Name, name));
    }

    1 reference
    public static void AddNameIdentifier(this ICollection<Claim> claims, string nameIdentifier)
    {
        claims.Add(new Claim(ClaimTypes.NameIdentifier, nameIdentifier));
    }

    1 reference
    public static void AddRoles(this ICollection<Claim> claims, string[] roles)
    {
        roles.ToList().ForEach(role => claims.Add(new Claim(ClaimTypes.Role, role))); //Her rolü claime ekler
    }
}
```

Hazırda yazılmış olan Claim sınıfına kendi metotlarımızı ekledik.

```

namespace Core.Extensions
{
    0 references
    public static class ClaimsPrincipalExtensions
    {
        1 reference
        public static List<string> Claims(this ClaimsPrincipal claimsPrincipal, string claimType)
        {
            var result = claimsPrincipal?.FindAll(claimType)?.Select(x => x.Value).ToList();
            return result;
        }

        1 reference
        public static List<string> ClaimRoles(this ClaimsPrincipal claimsPrincipal)
        {
            return claimsPrincipal?.Claims(ClaimTypes.Role);
        }
    }
}

```

Clean rolünü verir.

İlgili claim type a göre birinin claimlerini ararken

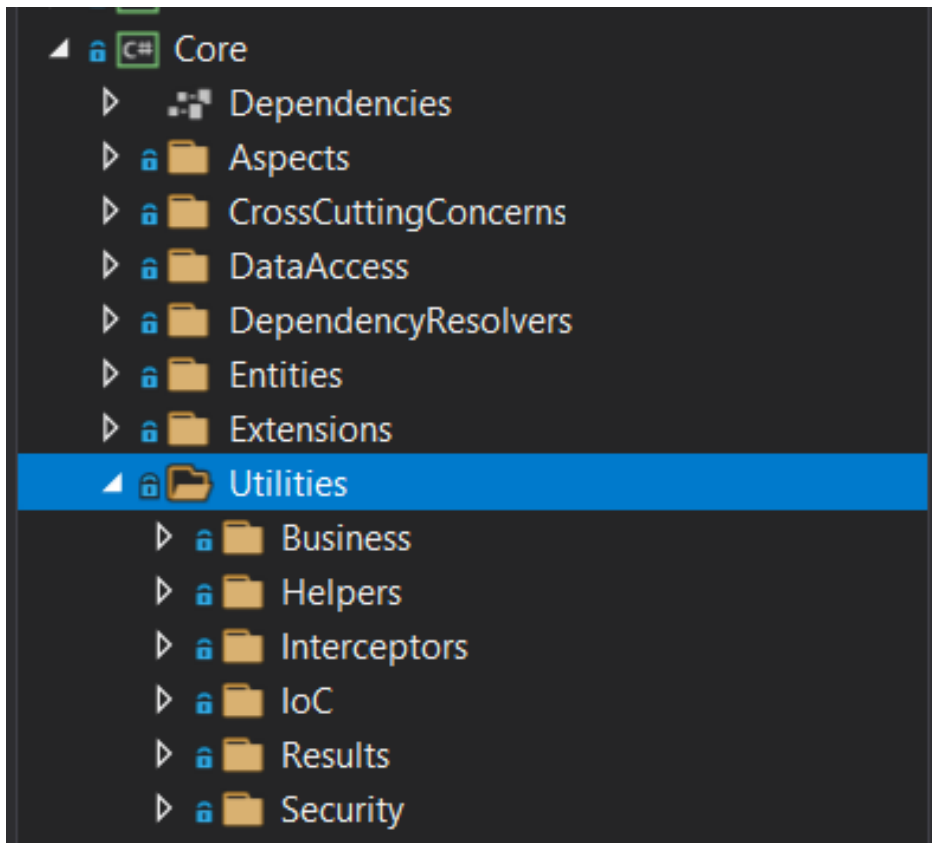
```

0 references
public static class ServiceCollectionExtensions
{
    1 reference
    public static IServiceCollection AddDependencyResolvers
    (this IServiceCollection serviceCollection, ICoreModule[] modules)
    {
        foreach (var module in modules)
        {
            module.Load(serviceCollection);
        }
        return ServiceTool.Create(serviceCollection);
    }
}

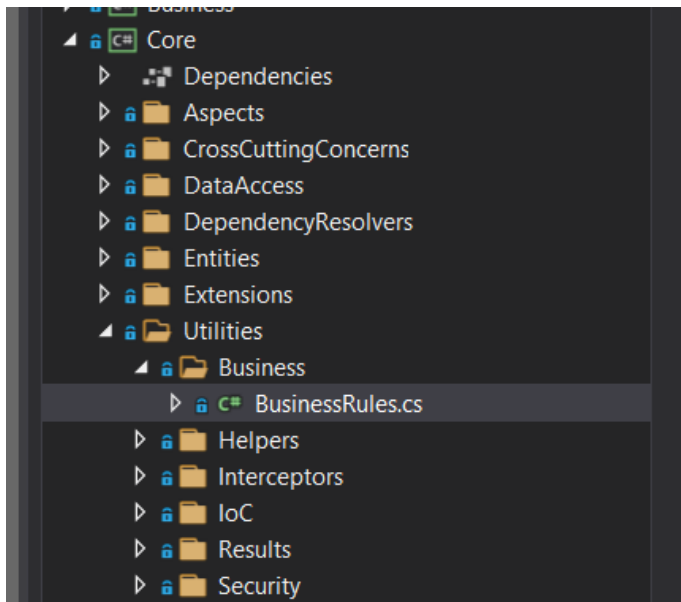
```

Interfcelerimizin servis karşılığını verir. Injection oluşturmamıza olanak sağlar.

Core katmanımızın son klasörü ise Utilities klasörü;



Bizim projemiz için gerekli araçları ise buraya yazdık.

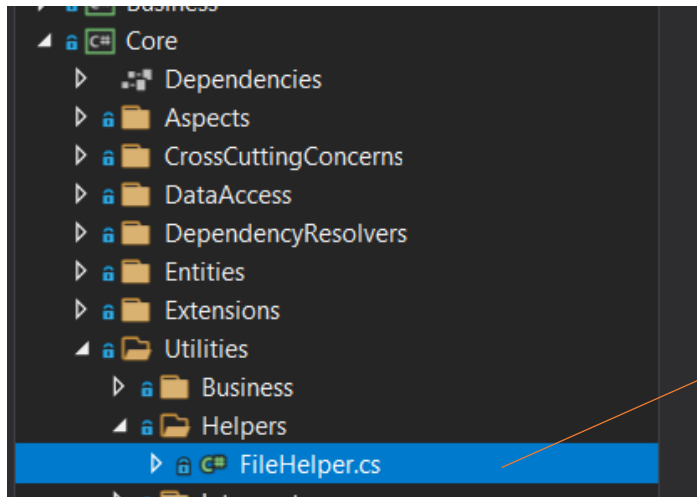


```

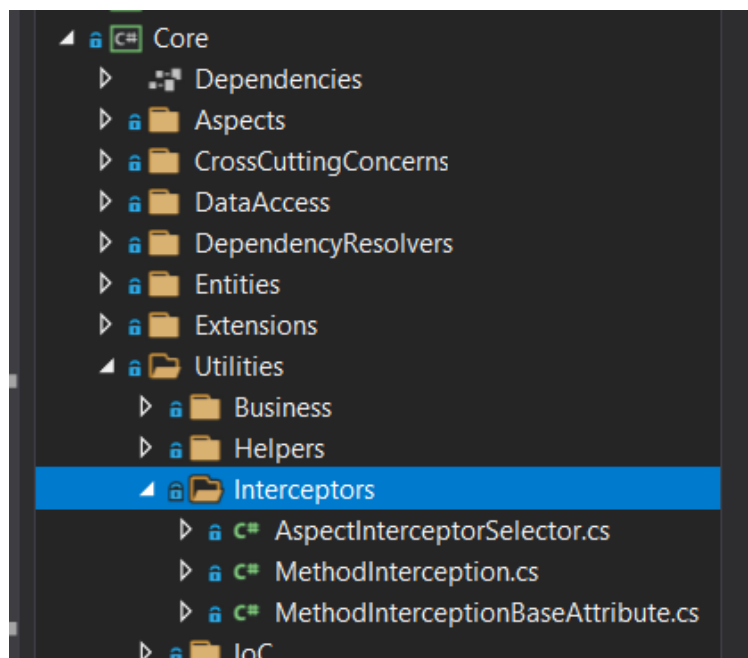
1reference
public class BusinessRules
{
    1reference
    public static IResult Run(params IResult[] logics)
    {
        foreach (var logic in logics)
        {
            if (!logic.Success)
            {
                return logic;
            }
        }
        return null;
    }
}

```

İş kurallarımız arttıkça if-else ler karışıyor sistem karmaşıklığı artmaktaydı bizim tüm iş metotlarımız IResult döndürdüğü için de iş motoru yazdık. Bu metot bize BusinessRules.Run(metot) şeklinde kullanım sundu. Metotta işlemimiz başarısızsa geriye değer döndürüp başarısız olduğunu söylüyoruz. Başarılıysa olduğu gibi çalışmaya devam ediyor. Params parametresi kullanarak da istediğimiz sayıda parametre göndermek istedik. Sonuç olarak bu kod bloğuyla tüm metotlarımızı buraya yolladık, iş kurallarına uymayan varsa da hata döndürdük.

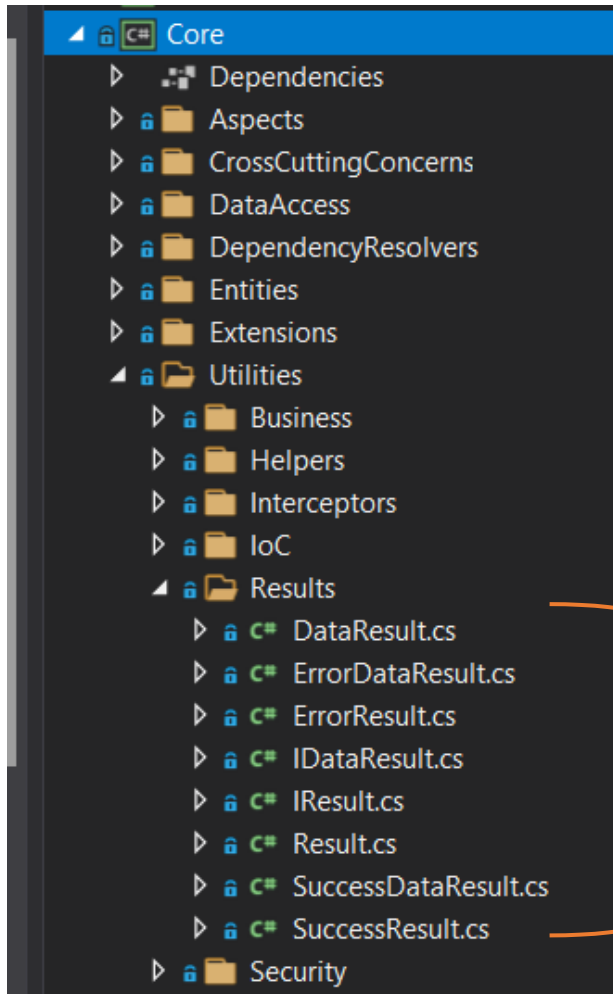


Resim ekleme-
silme- güncelleme
gibi
işlemleri
vaktiğimiz sınıf

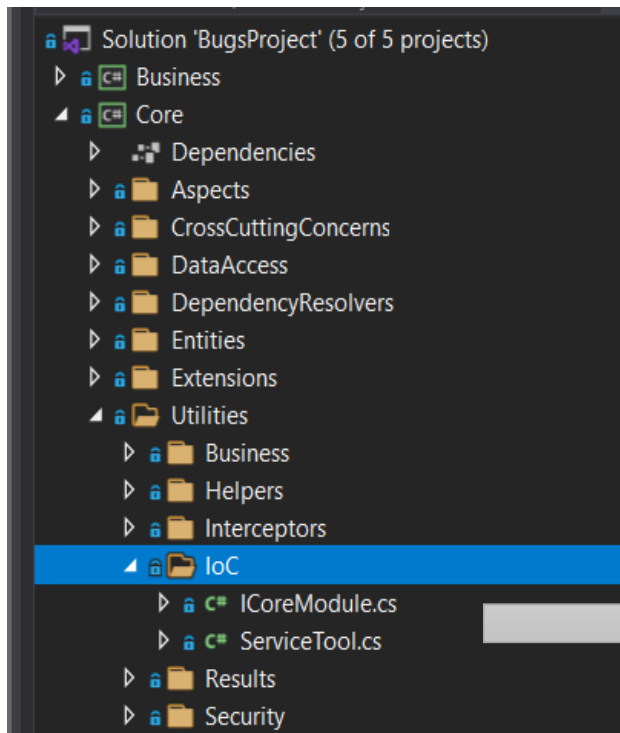


```
namespace Core.Interceptors
{
    1 reference
    public class AspectInterceptorSelector : IInterceptorSelector
    {
        0 references
        public IInterceptor[] SelectInterceptors(Type type, MethodInfo method, IInterceptor[] interceptors)
        {
            var classAttributes = type.GetCustomAttributes<MethodInterceptionBaseAttribute>
                (true).ToList();
            var methodAttributes = type.GetMethod(method.Name)
                .GetCustomAttributes<MethodInterceptionBaseAttribute>(true);
            classAttributes.AddRange(methodAttributes);

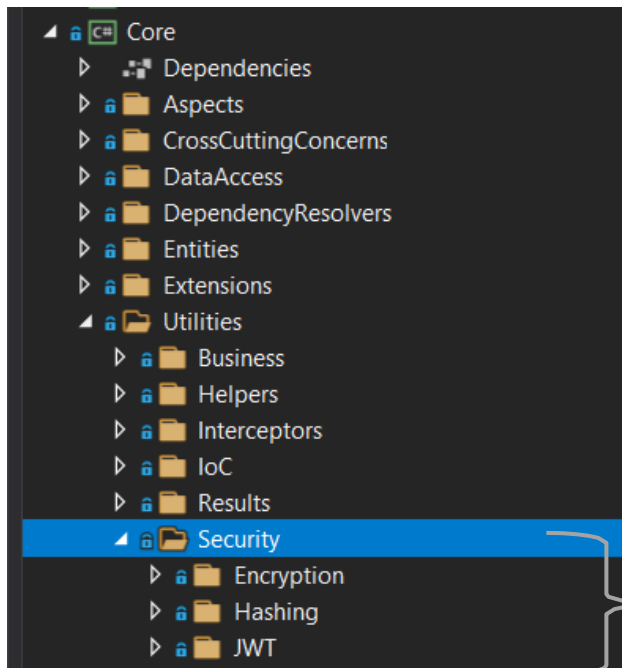
            return classAttributes.OrderBy(x => x.Priority).ToArray();
        }
    }
}
```



Return ile sadece tek bir değer döndürebiliriz. Birden çok değer döndürmek için bu yapıyı oluşturduk. Burada oluşturduğumuz sınıflarla Success (metodun başarılı olup- olmadığı bilgisi), message (durum mesajı) veya



IoC ise bizim oluşturmuş olduğumuz sınıfları kullanırken bellekten yer ayırır, new()ler, instance üretir.



Kullanıcıların üye olurken girdiği şifreleri veri tabanımızda olduğu gibi saklamak yerine Sha512 algoritmasına göre hash ve salt işlemleri yaparak saklarız o yüzden bu sınıfları oluşturduk.

```
namespace Core.Utilities.Security.JWT
{
    4 references
    public class TokenOptions
    {
        2 references
        public string Audience { get; set; }
        2 references
        public string Issuer { get; set; }
        1 reference
        public int AccessTokenExpiration { get; set; }
        2 references
        public string SecurityKey { get; set; }
    }
}
```

Appsettingste okuduğumuz değerleri buraya atayacağız.


```

namespace Core.Utilities.Security.JWT
{
    2 references
    public class JwtHelper : ITokenHelper
    {
        2 references
        public IConfiguration Configuration { get; } //API mizdeki appsettingsini okumamızı sağlar
        private TokenOptions _tokenOptions; //Okuduğumuz değerleri TokenOptions nesnesine atmamızı sağlar.
        private DateTime _accessTokenExpiration; //Token ne zaman geçersizleşecek.
        0 references
        public JwtHelper(IConfiguration configuration)
        {
            Configuration = configuration;
            _tokenOptions = Configuration.GetSection("TokenOptions").Get<TokenOptions>();
            //appsettingteki Tokenoptions bölümünü al ve TokenOptions sınıfını kullanarak eşle diyor.
        }
        2 references
        public AccessToken CreateToken(User user, List<OperationClaim> operationClaims) //Kullanıcı için token oluşturur.
        {
            _accessTokenExpiration = DateTime.Now.AddMinutes(_tokenOptions.AccessTokenExpiration); //Şimdiye 10 dakika ekler.
            var securityKey = SecurityKeyHelper.CreateSecurityKey(_tokenOptions.SecurityKey);
            var signingCredentials = SigningCredentialsHelper.CreateSigningCredentials(securityKey);
            var jwt = CreateJwtSecurityToken(_tokenOptions, user, signingCredentials, operationClaims);
            var jwtSecurityTokenHandler = new JwtSecurityTokenHandler();
            var token = jwtSecurityTokenHandler.WriteToken(jwt);

            return new AccessToken
            {
                Token = token,
                Expiration = _accessTokenExpiration
            };
        }
    }
}

```

JWT oluşturmak için gerekli tüm bilgiler.

JWT oluşturur bu

```

1 reference
public JwtSecurityToken CreateJwtSecurityToken(TokenOptions tokenOptions, User user, //JWT oluşturur.
    SigningCredentials signingCredentials, List<OperationClaim> operationClaims)
{
    var jwt = new JwtSecurityToken(
        issuer: tokenOptions.Issuer,
        audience: tokenOptions.Audience,
        expires: _accessTokenExpiration,
        notBefore: DateTime.Now,
        claims: SetClaims(user, operationClaims),
        signingCredentials: signingCredentials
    );
    return jwt;
}

1 reference
private IEnumerable<Claim> SetClaims(User user, List<OperationClaim> operationClaims)
{
    var claims = new List<Claim>();
    claims.AddNameIdentifier(user.UserId.ToString());
    claims.AddEmail(user.Email);
    claims.AddName($"{user.FirstName} {user.LastName}"); // $ iki stringi yan yana yazmamızı sağlar.
    claims.AddRoles(operationClaims.Select(c => c.Name).ToArray());

    return claims;
}

```

Şu andan önceki bir değer

Oluşturulan jwt yi döndürür.

Rol ekler.

User bilgisi ve Claimlerini alarak claim listesi oluşturur.

Verilen şifrenin Hash ve Saltını oluşturur. Her kullanıcı için farklı key üretilir.

```
2 references
public class HashingHelper
{
    1 reference
    public static void CreatePasswordHash(string password, out byte[] passwordHash, out byte[] passwordSalt )
    {
        using (var hmac= new System.Security.Cryptography.HMACSHA512())
        {
            passwordSalt = hmac.Key;
            passwordHash = hmac.ComputeHash(Encoding.UTF8.GetBytes(password));
        }
    }

    1 reference
    public static bool VerifyPasswordHash(string password, byte[] passwordHash, byte[] passwordSalt)
    {
        using (var hmac = new System.Security.Cryptography.HMACSHA512(passwordSalt))
        {
            var computedHash = hmac.ComputeHash(Encoding.UTF8.GetBytes(password));
            for (int i = 0; i < computedHash.Length; i++)
            {
                if (computedHash[i] != passwordHash[i])
                {
                    return false;
                }
            }
            return true;
        }
    }
}
```

Doğrulama yapacağım
IZ
kevi y979cağız

Kullanıcımızın sonradan
girmiş olduğu parolayı
hashlememizi sağlar.

VerifyPasswordHash metodu sisteme sonradan girmek isteyen kişinin şifresinin hashiyle veri kaynağımızdaki ilgili keye göre hashin eşleşip eşleşmediğini kontrol eder.

Business katmanı bizim iş katmanımızdır. İş kurallarımıza göre Data Access ile veri tabanı işlemleri yapar.

```
namespace Core.Utilities.Security.Encryption
{
    2 references
    public class SecurityKeyHelper
    {
        2 references
        public static SecurityKey CreateSecurityKey(string securityKey)
        {
            return new SymmetricSecurityKey(Encoding.UTF8.GetBytes(securityKey));
        }
    }
}
```

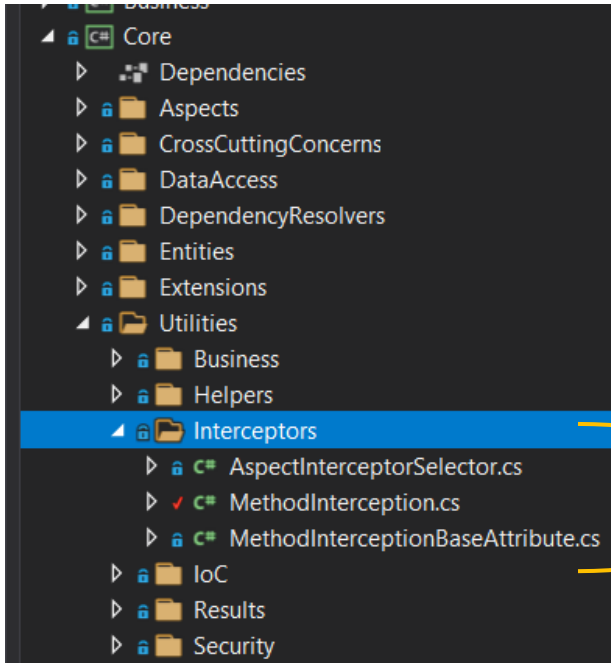
Parametre olarak security key alıp byte array karşılığı olan Securitykey üreten metottur.

```

namespace Core.Utilities.Security.Encryption
{
    1 reference
    public class SigningCredentialsHelper
    {
        1 reference
        public static SigningCredentials CreateSigningCredentials(SecurityKey securityKey)
        {
            return new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha512Signature);
        }
    }
}

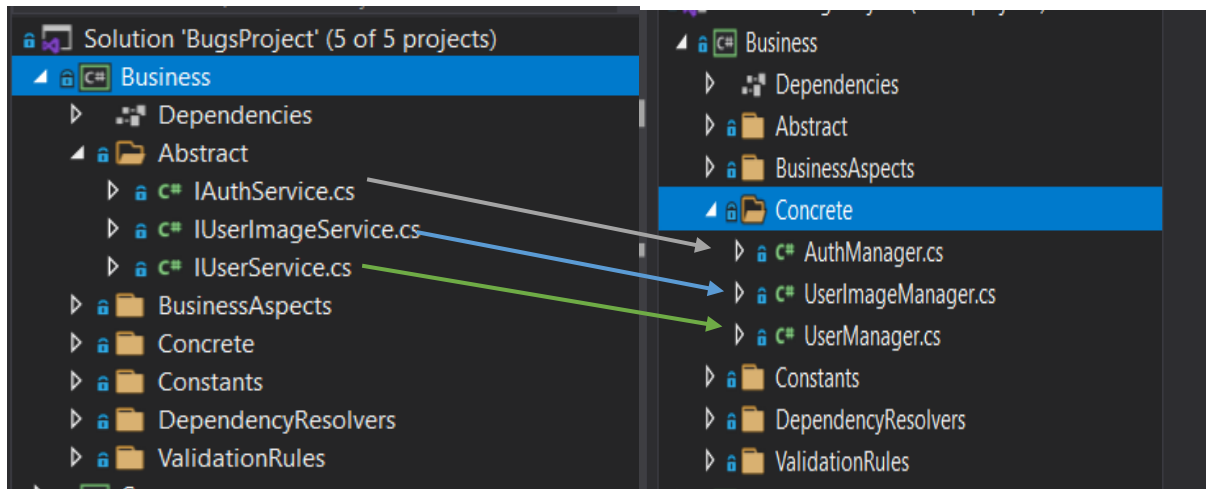
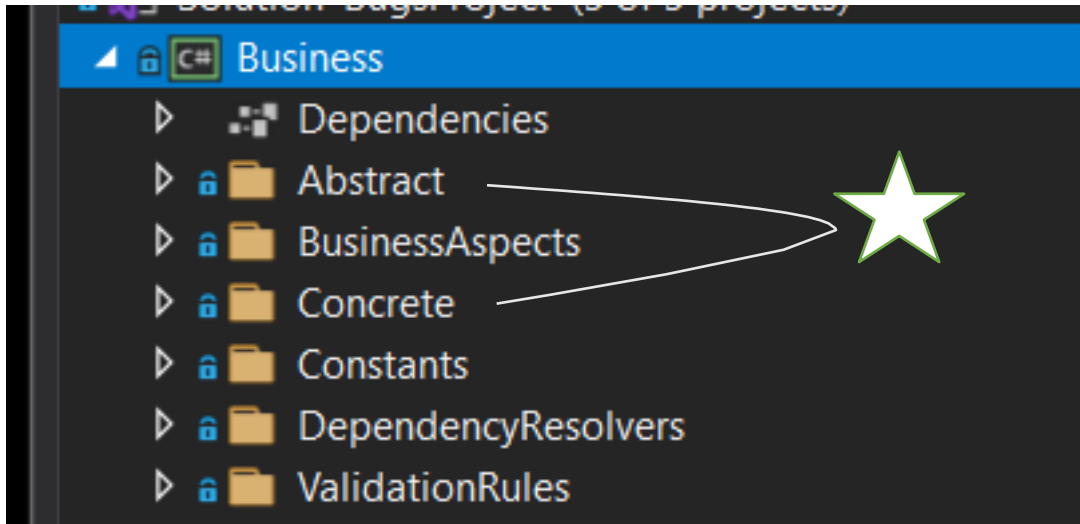
```

JWT sistemi yönettiğimiz için bu metotla da diyoruz ki; güvenlik anahtarın ve şifreleme algoritman bu.

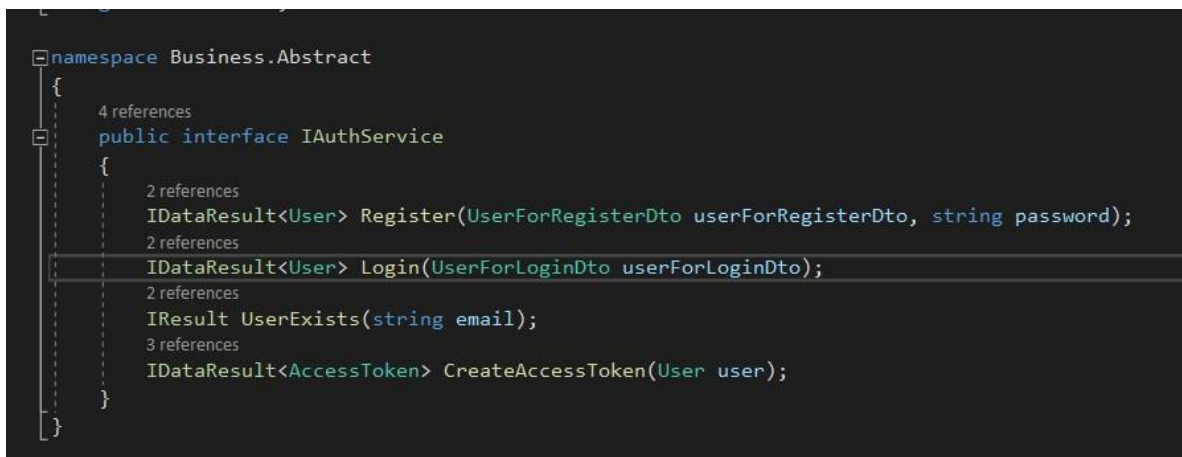


Metotlarımız için Attribute yapısı oluşturmuştuk. Metot başarılı olduğunda, metodun başında, sonunda, hata verdiğinde çalışabilmesi için gerekli olan varıy bu

Business katmanında bulunan Abstract ve Concrete klasörünü birlikte ele alalım çünkü Abstract klasörü içerisinde interfaceler yardımcıyla sadece metot tanımlamaları yapıp Concrete klasörüyle de içleri doldurulmuştur.



IAuthService Register, Login, UserExists, CreateAccessToken metotlarına sahiptir. Sırasıyla bu metotlar; sisteme üye olma, sisteme giriş yapma, Sistemden çıkış yapmak ve token oluşturmak için yazılmıştır.



Bu metotların içeri ise AuthManager sınıfında doldurulmuştur.

Register metotunda kullanıcımızın sisteme üye olması için gerekli bilgiler alınmıştır. Şifrelerimizi veri tabanında olduğu gibi tutmayacağımız (hash ve saltlanacağı için) HashingHelper sınıfındaki CreatePasswordHash metodu kullanılmıştır. Sonuç olarak kullanıcıdan Email, Firstname, Lastname ve

password olarak sisteme üye olarak ekledik. Status durumu true, şifreleri ise salt ve hashli olarak eklenmiş oldu.

```
2 references
public class AuthManager : IAuthService
{
    private IUserService _userService;
    private ITokenHelper _tokenHelper;

    0 references
    public AuthManager(IUserService userService, ITokenHelper tokenHelper)
    {
        _userService = userService;
        _tokenHelper = tokenHelper;
    }

    2 references
    public IActionResult Register(UserForRegisterDto userForRegisterDto, string password)
    {
        byte[] passwordHash, passwordSalt;
        HashingHelper.CreatePasswordHash(password, out passwordHash, out passwordSalt);
        var user = new User
        {
            Email = userForRegisterDto.Email,
            FirstName = userForRegisterDto.FirstName,
            LastName = userForRegisterDto.LastName,
            PasswordHash = passwordHash,
            PasswordSalt = passwordSalt,
            Status = true
        };
        _userService.Add(user);
        return new SuccessDataResult<User>(user, Message.UserRegistered);
    }
}
```

Sistemimiz giriş yapması içinde email ve şifre istedik. Sonrasında Email null mu onu kontrol ettik eğer null ise kullanıcı bulunamadı hatası verdirdik. Daha sonrada şifre doğrulaması yaptık Eğer şifre yanlışsa tekrar hata verildi. Email ve şifre doğruysa sisteme girişe izin verildi.

```
2 references
public IActionResult Login(UserForLoginDto userForLoginDto)
{
    var userToCheck = _userService.GetByMail(userForLoginDto.Email);
    if (userToCheck == null)
    {
        return new ErrorDataResult<User>(Message.UserNotFound);
    }

    if (!HashingHelper.VerifyPasswordHash(userForLoginDto.Password, userToCheck.PasswordHash, userToCheck.PasswordSalt))
    {
        return new ErrorDataResult<User>(Message.PasswordError);
    }

    return new SuccessDataResult<User>(userToCheck, Message.SuccessfulLogin);
}

2 references
public IActionResult UserExists(string email)
{
    if (_userService.GetByMail(email) != null)
    {
        return new ErrorResult(Message.UserAlreadyExists);
    }
    return new SuccessResult();
}
```

CreateAccessToken metodu ise bir kullanıcıyı parametre olarak alıp o kullanıcının tokenini üretir.


```

3 references
public IActionResult CreateAccessToken(User user)
{
    var claims = _userService.GetClaims(user);
    var accessToken = _tokenHelper.CreateToken(user, claims);
    return new SuccessDataResult<AccessToken>(accessToken, Message.AccessTokenCreated);
}
}

```

IUserService sınıfı ise kullanıcılarımızın resim eklemesi-silmesi-güncellemesi-resimleri listeleyebilmesi için oluşturulmuş Service sınıfıdır.

```

{
    4 references
    public interface IUserImageService
    {
        2 references
        IActionResult Add(IFormFile file, UserImage userImage);

        2 references
        IActionResult Delete(UserImage userImage);

        2 references
        IActionResult Update(IFormFile file, UserImage userImage);

        3 references
        IActionResult<List<UserImage>> GetAll(Expression<Func<UserImage, bool>> filter = null);

        4 references
        IActionResult<UserImage> GetById(int id);
    }
}

```

SecuredOperation("user,admin") attribute ıyla yetkilendirme yaptık. User veya admin yetkisine sahip olmayanlar add metotunu kullanamayacak.

CacheRemoveAspect attribute ıyla da içerisinde IUserImageService.Get geçiren cachedeki verileri uçurduk.

TransactionScopeAspect ile de tutarlılığı sağlamayı amaçladık.

C# da bulunan IFormFile sınıfını kullanarak resim ekleme ve silme işlemlerini yaptık.

```

2 references
public class UserManager : IUserImageService
{
    IUserImageDal _userImageDal;

    0 references
    public UserManager(IUserImageDal userImageDal)
    {
        _userImageDal = userImageDal;
    }
    [SecuredOperation("user,admin")]
    [CacheRemoveAspect("IUserImageService.Get")]
    [TransactionScopeAspect]
    2 references
    public IActionResult Add(IFormFile file, UserImage userImage)
    {
        userImage.ImagePath = FileHelper.AddAsync(file);
        userImage.UserImageDate = DateTime.Now;
        _userImageDal.Add(userImage);
        return new SuccessResult(Message.PostAdded);
    }

    2 references
    public IActionResult Delete(UserImage userImage)
    {
        var oldpath = Path.GetFullPath(Path.Combine(AppContext.BaseDirectory, "..\\..\\..\\wwwroot")) +
            _userImageDal.Get(p => p.UserId == userImage.UserId).ImagePath;
        IActionResult result = BusinessRules.Run(FileHelper.DeleteAsync(oldpath));

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

        _userImageDal.Delete(userImage);
        return new SuccessResult(Message.PostDeleted);
    }
    [CacheAspect]

```

CacheAspect ile cache aldık.

PerformanceAspect(5) attribute sıyyla da eğer metotun çalışması 5 dakikadan uzun sürerse performans zafiyeti vardır bize haber ver demiş olduk.

GetAll metoduyla da tüm resimleri

listeledik. GetById ile verilen id deki

resimi getirdik.

```

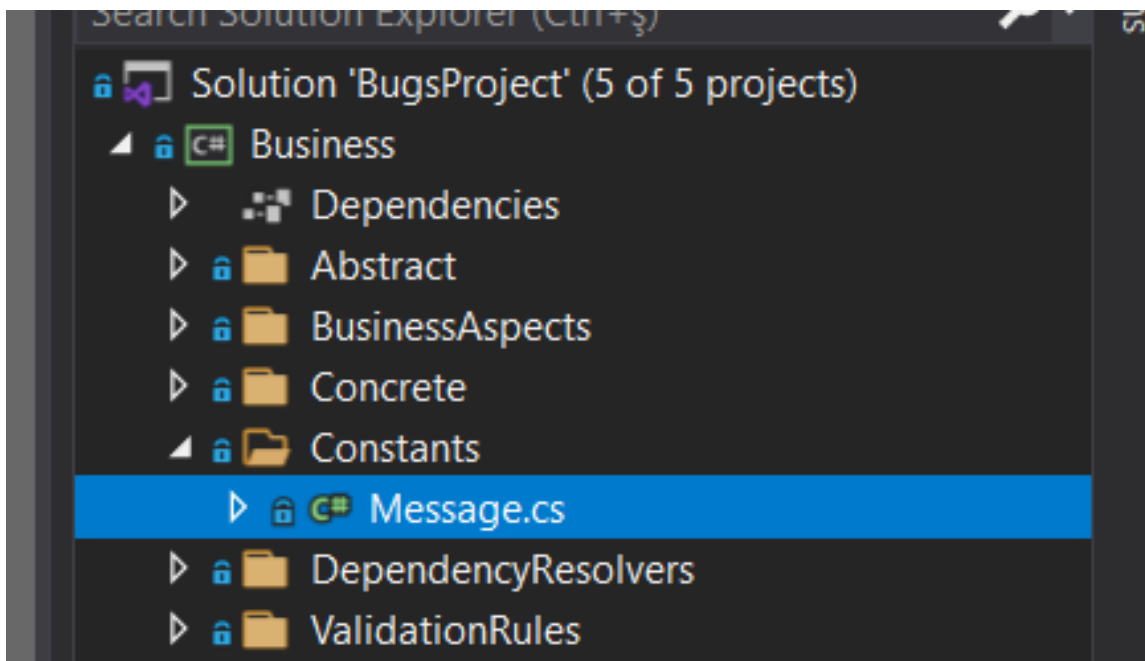
    [CacheAspect]
    [PerformanceAspect(5)]
    3 references
    public IActionResult<List<UserImage>> GetAll(Expression<Func<UserImage, bool>> filter = null)
    {
        return new SuccessDataResult<List<UserImage>>(_userImageDal.GetAll(filter));
    }

    4 references
    public IActionResult<UserImage> GetById(int id)
    {
        return new SuccessDataResult<UserImage>(_userImageDal.Get(I => I.UserId == id));
    }

    [CacheRemoveAspect("IUserService.Get")]
    2 references
    public IActionResult Update(IFormFile file, UserImage userImage)
    {
        var oldpath = Path.GetFullPath(Path.Combine(AppContext.BaseDirectory, "..\\..\\..\\wwwroot"))
        + _userImageDal.Get(p => p.UserId == userImage.UserId).ImagePath;
        userImage.ImagePath = FileHelper.UpdateAsync(oldpath, file);
        userImage.UserImageDate = DateTime.Now;
        _userImageDal.Update(userImage);
        return new SuccessResult(Message.PostUpdated);
    }
}

```

CacheRemoveAspect("IUserService.Get") attribute ıyla da bu metod çalıştığında cache de IUserImageService.Get metodu içeren cache i uçurur. Metotumuz ise resim silme işlemleri yapar.



Constants klasörü ise sabitlerimizi koyduğumuz klasördür.


```

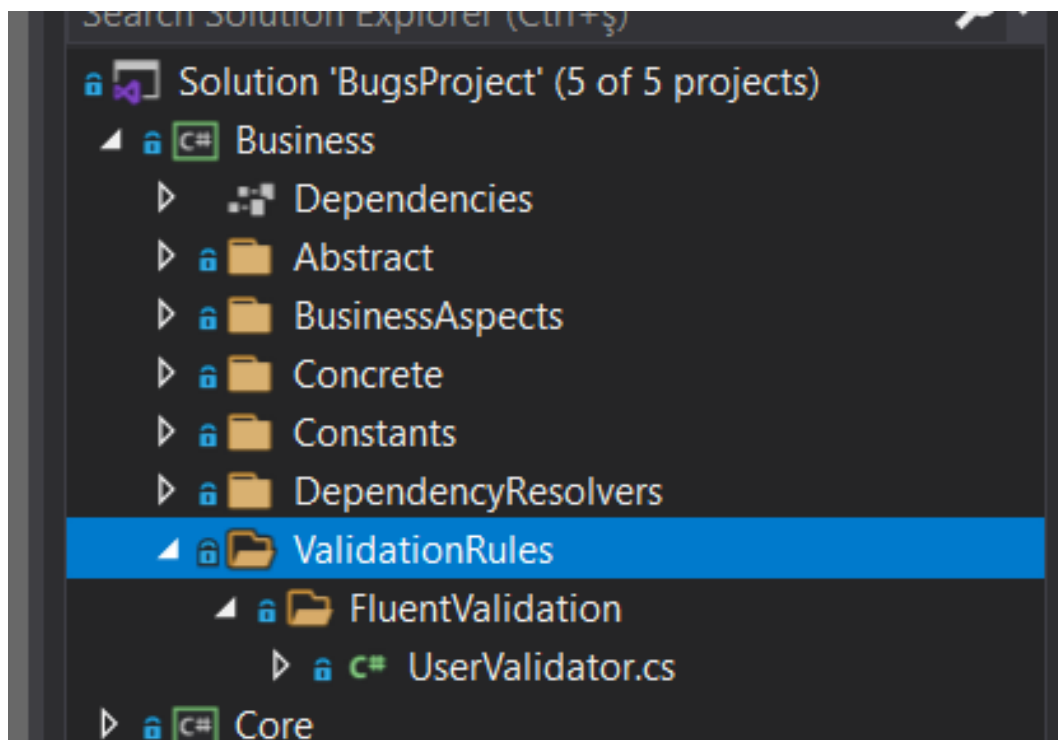
namespace Business.Constants
{
    14 references
    public class Message
    {
        public static string PostAdded = "Gönderi Eklendi";
        public static string PostDeleted = "Gönderi Silindi";
        public static string PostUpdated = "Gönderi Güncellendi";
        public static string PostsListed = "Gönderiler Listelendi";

        public static string UserAdded = "Kullanıcı Eklendi";
        public static string UserDeleted = "Kullanıcı Silindi";
        public static string UserUpdated = "Kullanıcı Güncellendi";
        public static string UserListed = "Kullanıcı Listelendi";

        public static string UserNameInvalid = "Kullanıcı adı geçersiz";
        public static string MaintenanceTime = "Sistem Bakımda";
        public static string UserNameAlreadyExists = "Bu isimde zaten başka bir kullanıcı var.";
        public static string AuthorizationDenied = "Yetkiniz yok.";
        public static string UserNotFound = "Kullanıcı bulunamadı";
        public static string PasswordError = "Şifre hatalı";
        public static string SuccessfulLogin = "Sisteme giriş başarılı";
        public static string UserAlreadyExists = "Bu kullanıcı zaten mevcut";
        public static string UserRegistered = "Kullanıcı başarıyla kaydedildi";
        public static string AccessTokenCreated = "Access token başarıyla oluşturuldu";
    }
}

```

Metotlarımızın çalışması sonucu verdiği mesajları tuttuğumuz sınıftır.



ValidationRules ise bizim doğrulama kurallarımızın olduğu sınıftır. Fluent Validation kütüphanesi kullandığımız içinde onu ayrı bir klasöre yazdık çünkü ilerde farklı bir kütüphane kullanmak isteyebiliriz.

```

namespace Business.ValidationRules.FluentValidation
{
    1 reference
    public class UserValidator : AbstractValidator<User>
    {
        0 references
        public UserValidator()
        {
            RuleFor(p => p.FirstName).NotEmpty();
            RuleFor(p => p.LastName).NotEmpty();
            RuleFor(p => p.Email).NotEmpty();
            RuleFor(p => p.Email).EmailAddress();
            RuleFor(p => p.FirstName).MinimumLength(2);
        }
    }
}

```

Kurallar yazdık.

Firstname alanı boş

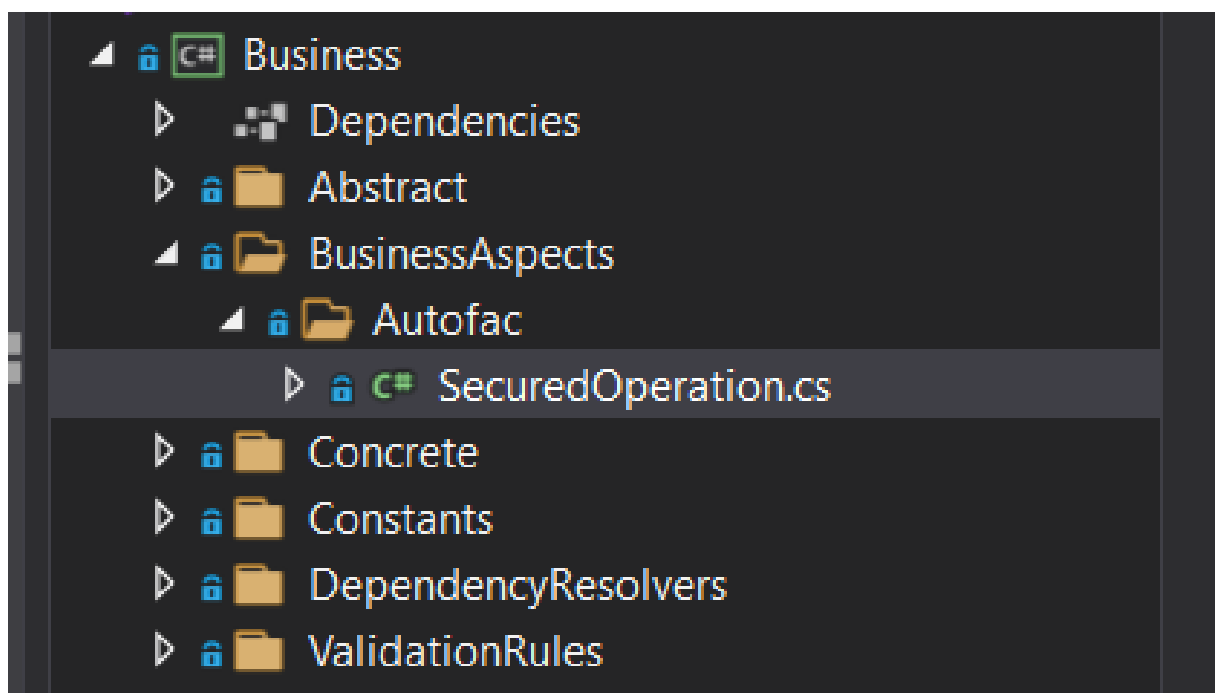
geçilemez. Lastname alanı

boş geçilemez. Email alanı

boş geçilemez.

Email email kurallarına uygun olsun.

Firstname uzunluğu 2 karakterden uzun olsun.

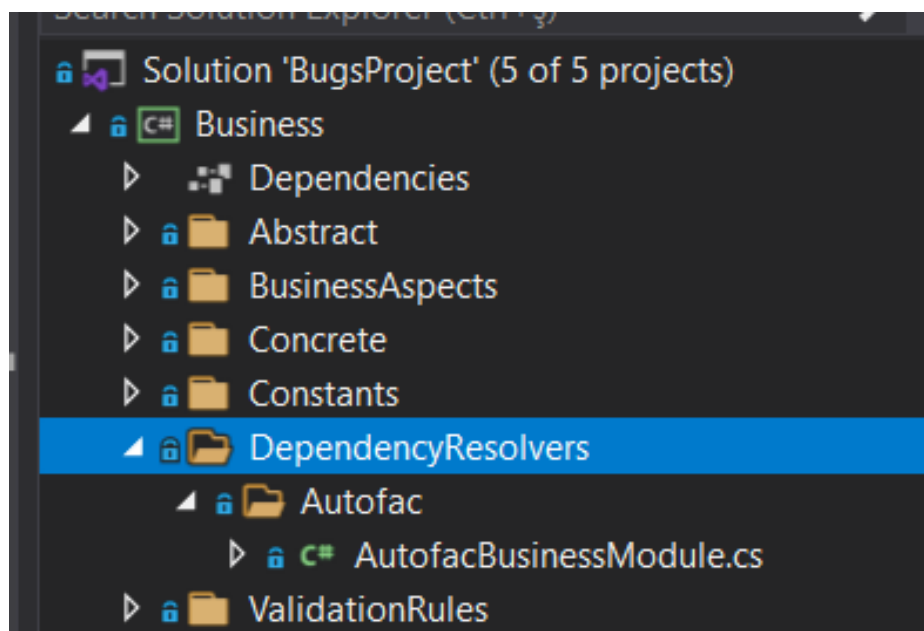


SecuredOperation("user,admin") şeklinde yetkilendirme yapmak için kullanmıştık. Split metodu ile , lerden itibaren ayırma yapar. Yetkilendirme işlemlerimizde metottan önce çalışacağı için OnBefore metotunu doldurduk.

```
namespace Business.BusinessAspects.Autofac
{
    2 references
    public class SecuredOperation : MethodInterception
    {
        private string[] _roles;
        private IHttpContextAccessor _httpContextAccessor;

        1 reference
        public SecuredOperation(string roles)
        {
            _roles = roles.Split(',');
            _httpContextAccessor = ServiceTool.ServiceProvider.GetService<IHttpContextAccessor>();
        }

        4 references
        protected override void OnBefore(IInvocation invocation)
        {
            var roleClaims = _httpContextAccessor.HttpContext.User.ClaimRoles();
            foreach (var role in _roles)
            {
                if (roleClaims.Contains(role))
                {
                    return;
                }
            }
            throw new Exception(Message.AuthorizationDenied);
        }
    }
}
```



IoC alt yapısı sağlayıp, AOP desteği verdiği için Autofac kullandık. Load metodu içerisinde instance üretmeyi yani new() leme bir diğer deyişle de bellekte yer ayırmaya yarar. SingleInstance de tek referans(instance) oluşturur. Build ise konfigrasyonu sağlar.

```
1 reference
public class AutofacBusinessModule : Module
{
    0 references
    protected override void Load(ContainerBuilder builder)
    {
        builder.RegisterType<UserManager>().As<IUserService>();
        builder.RegisterType<EfUserDal>().As<IUserDal>();

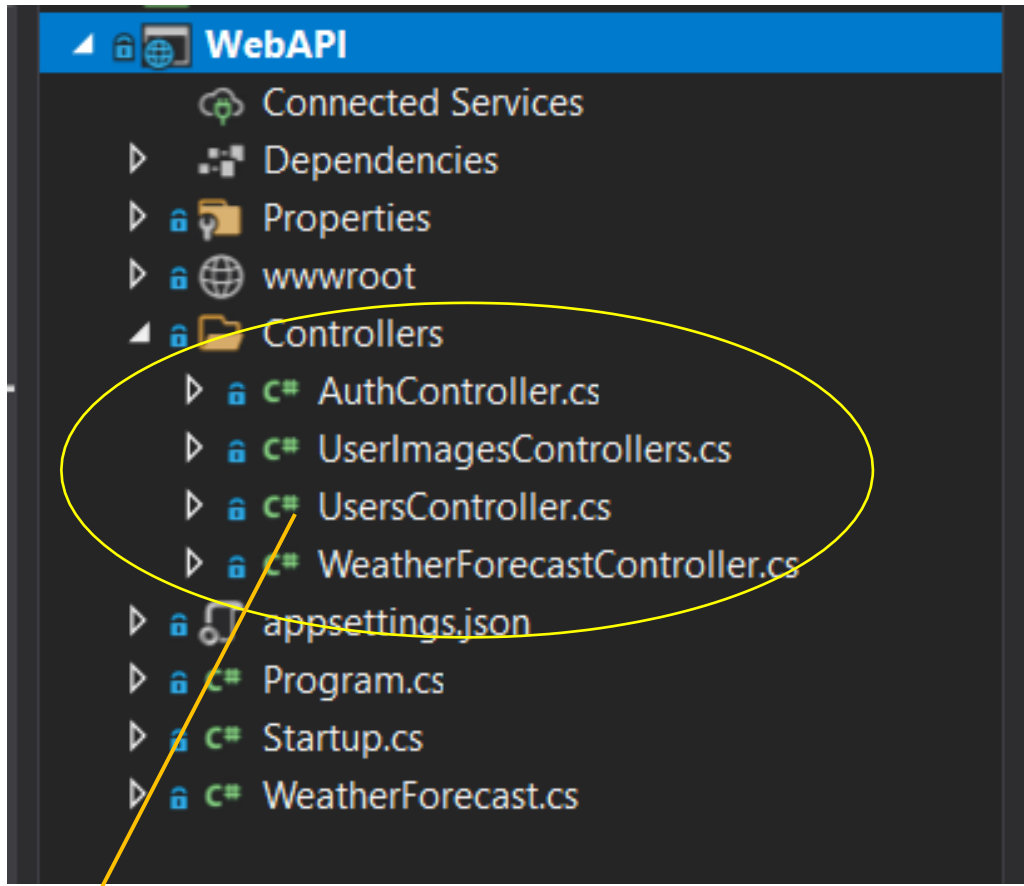
        builder.RegisterType<UserImageManager>().As<IUserImageService>().SingleInstance();
        builder.RegisterType<EfUserImageDal>().As<IUserImageDal>().SingleInstance();

        builder.RegisterType<AuthManager>().As<IAuthService>();
        builder.RegisterType<JwtHelper>().As<ITokenHelper>();

        var assembly = System.Reflection.Assembly.GetExecutingAssembly();

        builder.RegisterAssemblyTypes(assembly).AsImplementedInterfaces()
            .EnableInterfaceInterceptors(new ProxyGenerationOptions()
            {
                Selector = new AspectInterceptorSelector()
            }).SingleInstance();
    }
}
```

WebAPI katmanımız ise bizim projemizin dış dünyaya açıldığı kısımdır. API karşımıza standartlarla çıkar. Restful mimarisiyle json ile çıkar. Business ile farklı sistemler anlaşamaz(ios, android gibi). Bu yüzden farklı sistemlerin iletişim kurması için API (service) kullanılır. Özetle veri transferinde API kullanılır.



Gelen tüm istekleri Controller karşılar. Resim ekleme isteği-sisteme giriş yapma isteği-sisteme kayıt olma isteği gibi. Restful mimariler http protokolü üzerinden gelir.

```

namespace WebAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 reference
    public class AuthController : ControllerBase
    {
        private IAuthService _authService;

        0 references
        public AuthController(IAuthService authService)
        {
            _authService = authService;
        }

        [HttpPost("login")]
        0 references
        public ActionResult Login(UserForLoginDto userForLoginDto)
        {
            var userToLogin = _authService.Login(userForLoginDto);
            if (!userToLogin.Success)
            {
                return BadRequest(userToLogin.Message);
            }

            var result = _authService.CreateAccessToken(userToLogin.Data);
            if (result.Success)
            {
                return Ok(result.Data);
            }

            return BadRequest(result.Message);
        }
    }
}

```

Sistemimize gelen Giriş yapma isteğini karşılar. Aynı zamanda token oluşturur.

```

[HttpPost("register")]
0 references
public ActionResult Register(UserForRegisterDto userForRegisterDto)
{
    var userExists = _authService.UserExists(userForRegisterDto.Email);
    if (!userExists.Success)
    {
        return BadRequest(userExists.Message);
    }

    var registerResult = _authService.Register(userForRegisterDto, userForRegisterDto.Password);
    var result = _authService.CreateAccessToken(registerResult.Data);
    if (result.Success)
    {
        return Ok(result.Data);
    }

    return BadRequest(result.Message);
}
}

```

Sistemimize gelen üye olma isteğini karşılar. Kullanıcıya token verir ve bundan sonraki resim ekleme işlemleri o token aracılığıyla sağlanır.

```

{
    [Route("api/[controller]")]
    [ApiController]
    1 reference
    public class UserImagesController : ControllerBase
    {
        IUserImageService _userImageService;

        0 references
        public UserImagesController(IUserImageService userImageService)
        {
            _userImageService = userImageService;
        }

        [HttpPost("add")]
        0 references
        public IActionResult Add([FromForm(Name = "Image")] IFormFile file, [FromForm] UserImage userImage)
        {
            var result = _userImageService.Add(file, userImage);
            if (result.Success)
            {
                return Ok(result);
            }
            return BadRequest(result);
        }

        [HttpPost("update")]
        0 references
        public IActionResult Update([FromForm(Name = "Image")] IFormFile file, [FromForm(Name = "Id")] int Id)
        {
            var userImages = _userImageService.GetById(Id).Data;
            var result = _userImageService.Update(file, userImages);
            if (result.Success)
            {
                return Ok(result);
            }
            return BadRequest(result);
        }
    }
}

```

Business katmanımızdaki service

Resim ekleme isteğini karşılar. Eğer resim ekleme başarılıysa 200 mesajı, hata verir eklenemezse de 400lü mesaj verilir.

Resim güncelleme isteğini karşılar. Eğer resim güncelleme başarılıysa 200 mesajı, hata verir güncellenemezse de 400lü mesaj verilir.


```

[HttpPost("delete")]

public IActionResult Delete([FromForm(Name = "Id")] int Id)
{
    var forDelete = _userService.GetById(Id).Data;
    var result = _userService.Delete(forDelete);
    if (result.Success)
    {
        return Ok(result);
    }
    return BadRequest(result);
}

```

Resim silme isteğini karşılar.

```

[HttpGet("getall")]

public IActionResult GetAll()
{
    var result = _userService.GetAll();
    if (result.Success)
    {
        return Ok(result);
    }
    return BadRequest(result);
}

```

Resimleri listeleme isteğini karşılar. Ana sayfamızda resimleri bu metotla ekranda gösterdik.

```

[HttpGet("getbyid")]
0 references
public IActionResult GetById(int id)
{
    var result = _userService.GetById(id);
    if (result.Success)
    {
        return Ok(result);
    }
    return BadRequest(result);
}

```

Id si verilen resmi getirir.

```

[HttpGet("getbyuser")]
0 references
public IActionResult GetUser(int id)
{
    var result = _userService.GetAll(I => I.UserId == id);
    if (result.Success)
    {
        return Ok(result);
    }
    return BadRequest(result);
}

```

Id si verilen kullanıcının resimlerini getirir.


```

namespace WebAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]

    public class UsersController : ControllerBase
    {
        IUserService _userService;

        public UsersController(IUserService userService)
        {
            _userService = userService;
        }

        [HttpGet("getall")]

        public IActionResult GetAll()
        {
            var result = _userService.GetAll();
            if (result.Success)
            {
                return Ok(result);
            }
            return BadRequest(result);
        }

        [HttpDelete("delete")]

        public IActionResult Delete(User user)
        {
            var result = _userService.Delete(user);
            if (result.Success)
            {
                return Ok(result);
            }
            return BadRequest(result);
        }
    }
}

```

Tüm kullanıcıların listeleme isteğine karşılık gelir

Kullanıcı üyeliği silmek istediğinde kullanılır.

```

[HttpPost("add")]

public IActionResult Add(User user)
{
    var result = _userService.Add(user);
    if (result.Success)
    {
        return Ok(result);
    }
    return BadRequest(result);
}

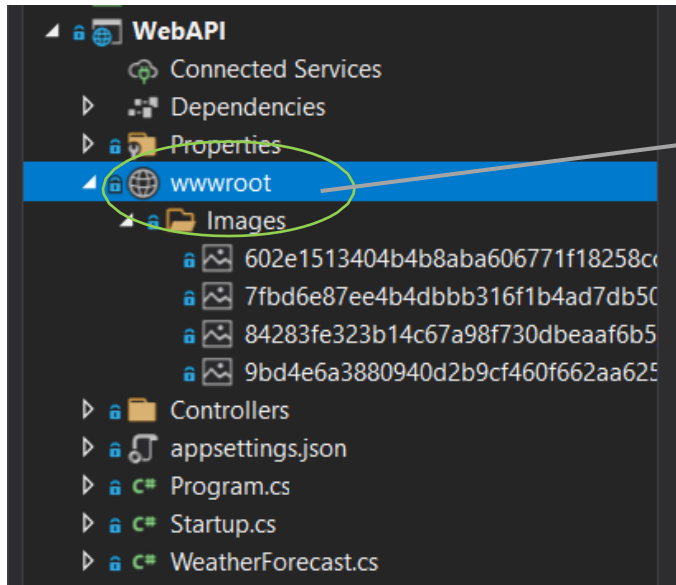
[HttpPut("update")]

public IActionResult Update(User user)
{
    var result = _userService.Update(user);
    if (result.Success)
    {
        return Ok(result);
    }
    return BadRequest(result);
}

```

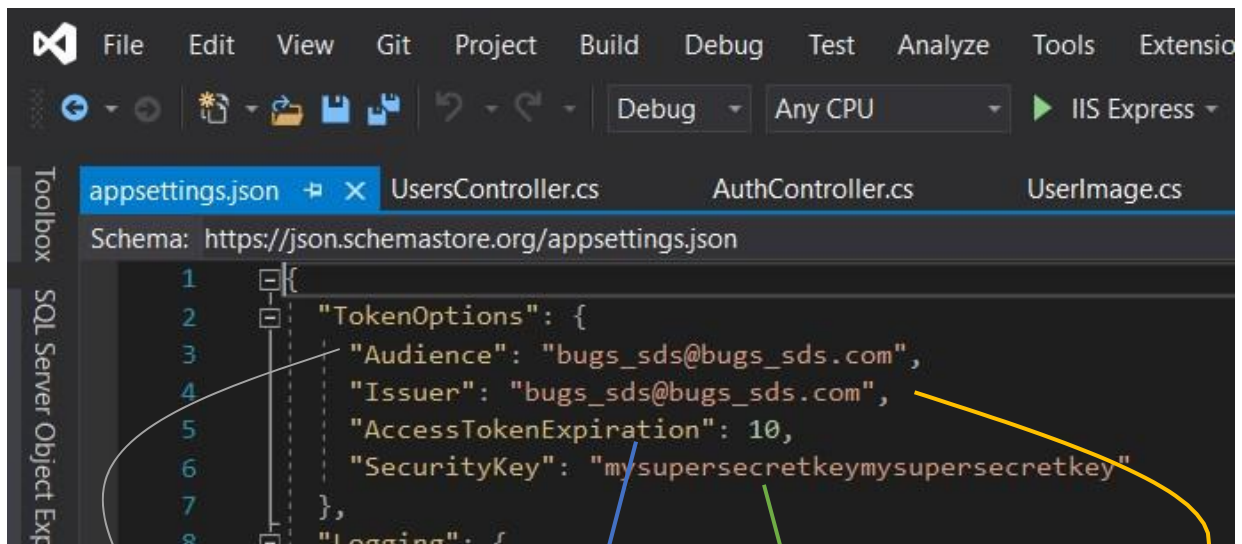
Kullanıcı eklenir

Kullanıcı güncellemek için kullanılır.



Resimlerimizi
burada
tuttuk

Appsettings.json dosyası ise WebAPI dosyamızın ayarlarını, özelliklerini konfigure ettiğimiz yerdir. Json web token konfigrasyonunu burda yapmış olduk. JWT(Json Web Token)imizin olmazsa olmaz alanlarını verdik.



Kitle demektir.
Projemizin web adresini

Tokenimizin
dakika cinsinden
geçerlilik
süresidir

Kitleyi

Bu tokeni oluştururken asp
.netin kullanacağı
anahtarın adı. Türkçe
karakter
kullanılmamalı

Server konfigrasyonunun yapıldığı yer. Yayın ortamımız.

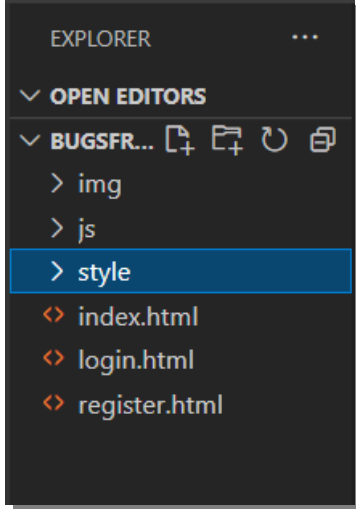


```
namespace WebAPI
{
    0 references
    public class Program
    {
        0 references
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        1 reference
        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .UseServiceProviderFactory(new AutofacServiceProviderFactory())
                .ConfigureContainer<ContainerBuilder>(builder =>
                {
                    builder.RegisterModule(new AutofacBusinessModule());
                })
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                })
            );
    }
}
```

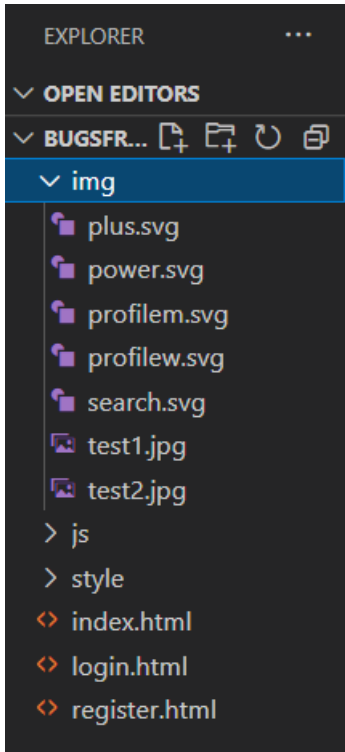
WebAPI deki program.cs de ise Autofac altyapısı kurduk. Bu metot sayesinde servis fabrikası olarak Autofac kullan dedik. C# da bulunan IoC alt yapısı yerine Autofac kullandık çünkü hem AOP hem de IoC alt yapısı sunmaktadır.

Frontend



Üç tane html sayfası bir tane görseli yani dizaynı belirlediğimiz bir style klasörümüz bulunuyor, bir tane javascript kodlarının bulunduğu bir dosya klasörü ve son olarak da hazır resimleri kullanmak için resim yüklediğimiz bir klasörümüz vardır.

İmg yani resim klasöründen başlarsak;

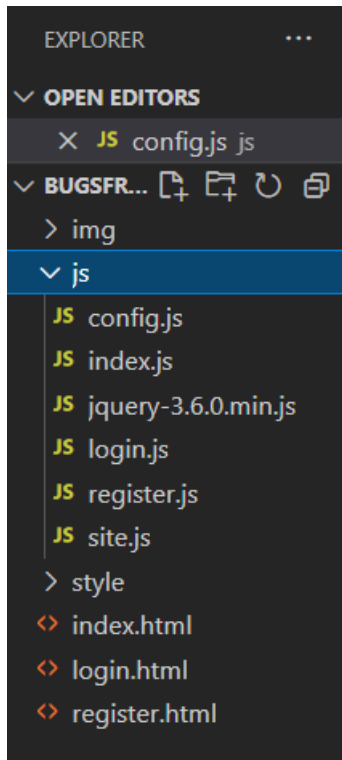


İmg klasöründe web sayfamızda kullandığımız kadın ve erkek profil görselleri, arama butonu görseli, resim ekleme buton görseli ve kapatma buton görseli bulunmaktadır. Yani web sitemizde kullandığımız tüm görseller burada bulunmaktadır.

İmg klasöründeki resimlerin eklenmesine dair örnek kod aşağıdaki gibidir;

```
profilem.svg × plus.svg
img > profilem.svg
1 5 9.422-18.197 9.818-19.121139.998 12.663c19.926 6.309 33.313 24.59 33.313 45.491v15.498c-39.094 40.293-89.713 65.219-144.728 71.612z"/></g></svg>
```

Js klasörüyle devam edersek;



Js dosyasında ilk olarak config.js dosyasıyla başlarsak;

```
JS config.js ×
js > JS config.js > ...
1 let ServerIp="https://localhost:44349";
2
```

Yukarıda ki ServerIp, göndereceğimiz gideceğimiz adresi tanımlar.

Yukarıda ki kodun ardından bir sonraki kodumuz empty metodunun yazılı olduğu kod olacaktır. Empty metodunda metnin boş mu olduğu kontrolü sağlanır.

```
JS config.js X
js > JS config.js > ...
3
4 function empty(e) {
5     switch (e.trim()) {
6         case "":
7         case 0:
8         case "0":
9         case null:
10        case false:
11        case typeof(e) == "undefined":
12            return true;
13        default:
14            return false;
15    }
16 }
17
```

Sonra ki kod profil resmi ile ilgili kod. Profil resmi ekleme işlemimiz olmadığından dolayı img klasörüne eklediğimiz kadın ve erkek figürlü görseller kullanıcıların profil resimlerine atanır.

```
JS config.js X
js > JS config.js > ...
17
18 function getRandomInt(max) {
19     return Math.floor(Math.random() * max);
20 }
21
```

```
JS config.js X
js > JS config.js > ...
40     const response = await fetch(url, {
41         method: 'POST', // *GET, POST, PUT, DELETE, etc.
42         headers: {
43             'Content-Type': 'application/json'
44             // 'Content-Type': 'application/x-www-form-urlencoded',
45         },
46         body: JSON.stringify(data) // body data type must match "Content-Type" header
47     });
48     return response.json(); // parses JSON response into native JavaScript objects
49 }
```

Yukarıda ki fetch GET, POST, PUT, DELETE gibi metotları getirebilecek konumdadır.

Son olarak json fonksiyonundan bahsederseniz burada kullanılan ajax fonksiyonu hazır fonksiyon olarak kullanılmıştır.

```
JS config.js X
js > JS config.js > ...
52     function json(url) {
53         var json = null;
54         $.ajax({
55             'async': false,
56             'global': false,
57             'url': url,
58             'dataType': "json",
59             'success': function(data) {
60                 json = data;
61             }
62         });
63         return json.data;
64     }
65
66     let userList = json(ServerIp + "/api/users/getall");
67 }
```

Js klasöründeki index.js dosyasına gelecek olursak;

İlk olarak add fonksiyonumuz vardır bu giriş işlemidir. Tıklandığında bizi anasayfamız olan index.html e bizi yönlendirmektedir.

```
JS index.js  X
js > JS index.js > ...
1  function add() {
2    | document.getElementById("upload").click();
3  }
4
```

logout fonksiyonumuz bulunmaktadır bu bir çıkış metodudur. False döndüğünde çıkış yapar.

```
JS index.js  X
js > JS index.js > ...
5  function logout() {
6    | sessionStorage.setItem("login", "false");
7    | window.location.href = "index.html";
8  }
```

Doğruluk sorgusunun yapılacağı metot ise;

```
JS index.js  X
js > JS index.js > submitForm
10  if (sessionStorage.getItem("login") == "true") {
11    | if (sessionStorage.getItem("userId") == null) {
12    |   | for (index in userList) {
13    |   |   | if (userList[index].email == sessionStorage.getItem("mail")) {
14    |   |   |   | sessionStorage.setItem("userId", userList[index].userId);
15    |   |   |   }
16    |   |   }
17    |   }
18  } else {
19    | window.location.href = "login.html";
20  }
21
```



```
JS index.js X
js > JS index.js > submitForm > success
22 function submitForm() {
23     var fd = new FormData();
24     var files = $("#upload")[0].files;
25 }
```

Burada form data dan bir dosya türetiyoruz.

```
JS index.js X
js > JS index.js > submitForm > success
27 if (files.length > 0) {
28     fd.append("Image", files[0]);
29     fd.append("userId", sessionStorage.getItem("userId"));
30 }
```

Yukarıda fotoğrafın yüklenmesi için id özelliğini kullanıyoruz.

```
JS index.js X
js > JS index.js > document.addEventListener("DOMContentLoaded") callback > $.getJSON() callback
31 $.ajax({
32     url: ServerIp + "/api/userimages/add",
33     type: "post",
34     data: fd,
35     contentType: false,
36     processData: false,
37     success: function (response) {
38         if (response != 0) {
39             window.location.reload();
40         } else {
41             alert("dosya yüklenemedi.");
42         }
43     },
44 });
45 } else {
46     alert("Lütfen bir dosya seçin.");
47 }
48 }
```

Yukarıda ajax fonksiyonunu kullanmaktayız. serverIp adresiyle dosya işlemleri gerçekleşir. İşlemi response gönderirsek sayfanın yanıtını bekliyoruz dosya yüklenemediyse dosya yüklenemedi hatası alacağız. Eğer bir dosya seçilmediyse lütfen bir dosya seçin hatası alacağız.

```
JS index.js X
js > JS index.js > document.addEventListener("DOMContentLoaded") callback > $.getJSON() callback
107 document.getElementById("search").onclick = () => {
108     var results = "";
109     let query = document.getElementById("query").value.trim();
110     if (!empty(query)) {
111         for (key in userList) {
112             fullName = userList[key].firstName + " " + userList[key].lastName;
113             let pp = null;
114             if (fullName.toLowerCase().search(query) != -1) {
115
116                 for (index in profileImages) {
117                     if (profileImages[index].id == userList[key].userId) {
118                         pp = profileImages[index].url;
119                         break;
120                     }
121                 }
122             }
123         }
124     }
125 }
```

Yukarıda ki search kısmı bizim arkadaş arama butonumuz olacak. Ara butonuna bastığımızda olacak işlemlerin kodları yazılmıştır. Girilen isimdeki boşluklar öncelikle trim ile giderilir ardından ad ile soyad birleştirilerek arama işlemi gerçekleşir.

Js klasöründeki login.js dosyasına gelecek olursak;

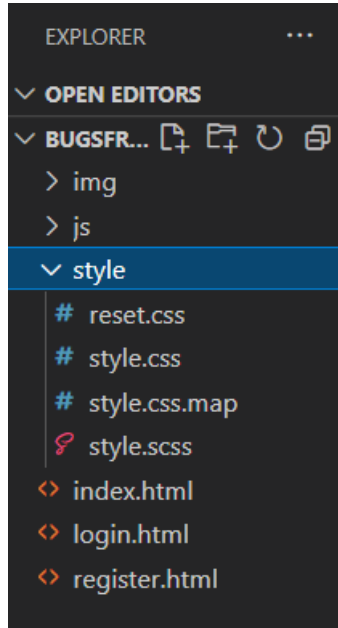
```
JS login.js X
js > JS login.js > ...
1 if(sessionStorage.getItem("login")==true){
2
3     window.location.href = "index.html";
4 }
5 document.addEventListener("DOMContentLoaded", function(){
6     let email = document.getElementById("email");
7     let password = document.getElementById("pass");
8     let loginBtn = document.getElementById("loginBtn");
9     loginBtn.onclick = ()=>{
10
11         let _data = {
12             email: email.value,
13             password: password.value
14         }
15         postData(ServerIp + "/api/auth/login", _data)
16             .then(data => {
17                 if (data.token != "" && data.expiration != "") {
18                     sessionStorage.setItem("login", "true");
19                     sessionStorage.setItem("token", data.token);
20                     sessionStorage.setItem("mail", _data.email);
21                     window.location.href = "index.html";
22                 }
23             });
24 }
```

Yukarıda ki login.js dosyasında giriş kontrolleri yapılmaktadır. Email ve password girilir login butonuna basıldıktan sonra doğrulama işlemi gerçekleştirilir. serverIp adresi ile de işlem gerçekleştirilmektedir.

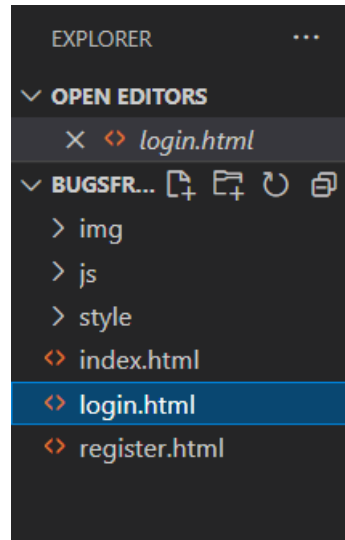
Js klasöründeki register.js dosyasına geçecek olursak;

```
JS register.js X
js > JS register.js > ...
1  if(sessionStorage.getItem("login")==true){
2      window.location.href = "index.html";
3  }
4  document.addEventListener("DOMContentLoaded", function(){
5      document.getElementById("registerBtn").onclick = ()=>{
6          let name = document.getElementById("name");
7          let lastname = document.getElementById("lastname");
8          let mail = document.getElementById("mail");
9          let pass = document.getElementById("pass");
10         let passCorrection = document.getElementById("passCorrection");
11         if(pass.value == passCorrection.value)
12         {
13             let _data = {
14                 firstName:name.value,
15                 lastName:lastname.value,
16                 email: mail.value,
17                 password: pass.value
18             }
19             postData(ServerIp + "/api/auth/register", _data)
20                 .then(data => {
21                     if (data.token != "" && data.expiration != "") {
22                         sessionStorage.setItem("login", "true");
23                         sessionStorage.setItem("token", data.token);
24                         sessionStorage.setItem("mail", _data.email);
25                         window.location.href = "index.html";
26                     }
27                 });
28         }
29     }
30     else{
31         alert("şifreler uyuşmuyor.");
32     }
33 }
34 }
35 });
```

Yukarıda ki register.js kodları eğer üye değilsek üye olmamız için sağlanacak işlemleri barındırmaktadır. Kodun ilk kısmında üye olmak için gereken bilgiler istenmektedir ardından ServerIp adresi ile işlem gerçekleştirilir.



Js dosyasının ardından style dosyasına geçerse burada görselliğin css kodlarıyla düzenlendiğini görmekteyiz. Dört dosya barındıran style klasörümüzde çoğu kodlar renk tonu, şekil, desen ve benzeri kodlar olduğu için ayrıntılı girmemekle beraber yapımının gerekliliği açıktır.



Klasörlerin ardından html kodlarına geçerse ilk olarak login.html'i ele alalım;

```
login.html X
login.html > html > body
1  <!DOCTYPE html>
2  <html lang="tr">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <link rel="stylesheet" href="style/style.css">
8      <script src="js/jquery-3.6.0.min.js"></script>
9      <script src="js/config.js"></script>
10     <script src="js/login.js"></script>
11     <title>Giriş</title>
12 </head>
```

Yukarıdaki login.html kodunda Türkçe karakter desteği, css kodu kullanılacağı gibi komutlar verilmiştir.

```
login.html X
login.html > html
13 <body>
14     <div class="login noselect">
15         <div class="input">
16             <span class="label">Email</span>
17             <input id="email" type="text" placeholder="Email adresi">
18         </div>
19         <div class="input">
20             <span class="label">Şifre</span>
21             <input id="pass" type="password" placeholder="Şifre">
22         </div>
23         <div class="input">
24             <button id="loginBtn" class="login-btn">Giriş</button>
25         </div>
26         <div class="input">
27             <span class="link">Hesabın yok mu? <a href="register.html">Kayıt Ol</a></span>
28         </div>
29     </div>
30 </body>
31 </html>
```

Yukarıda ki kodda girişin html kodları bulunmaktadır. Email., password, alımı ardından buton gösterimi eğer üyelik yoksa üye yapılması için bizi register ekranına aktaracak link bulunmaktadır.

Login.html'in ardından register.html'e geçerseniz;

```
<> register.html X
<> register.html > ...
1  <!DOCTYPE html>
2  <html lang="tr">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <link rel="stylesheet" href="style/style.css">
8      <script src="js/jquery-3.6.0.min.js"></script>
9      <script src="js/config.js"></script>
10     <script src="js/register.js"></script>
11     <title>Kayıt Ol</title>
12 </head>
```

Yukarıda ki koda Türkçe karakter desteği, css kodlarının kullanımı gibi bazı kurallar verilmiştir.

```
<> register.html X
<> register.html > ...
13 <body>
14     <div class="login noselect">
15         <div class="input">
16             <span class="label">İsim</span>
17             <input id="name" type="text" placeholder="İsim">
18         </div>
19         <div class="input">
20             <span class="label">Soyisim</span>
21             <input id="lastname" type="text" placeholder="Soyisim">
22         </div>
23         <div class="input">
24             <span class="label">Email</span>
25             <input id="mail" type="text" placeholder="Email adresi">
26         </div>
27         <div class="input">
28             <span class="label">Şifre</span>
29             <input id="pass" type="password" placeholder="Şifre">
30         </div>
31         <div class="input">
32             <span class="label">Şifre Onay</span>
33             <input id="passCorrection" type="password" placeholder="Şifre (Tekrar)">
34         </div>
35         <div class="input">
36             <button id="registerBtn" class="login-btn">Kayıt Ol</button>
37         </div>
38         <div class="input">
39             <span class="link">Zaten hesabın var mı? <a href="login.html">Giriş Yap</a></span>
40         </div>
41     </div>
42 </body>
43 </html>
```

Yukarıda ki kodda üyeliği olmayan birinin üye olması için alınması gereken bilgilerin html kodları verilmiştir.

Register.html'in ardından index.html'e geçerseniz;

```
<> index.html 1 X
<> index.html > html > body > div.container.noselect > div#posts.right
1  <!DOCTYPE html>
2  <html lang="tr">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta http-equiv="X-UA-Compatible" content="IE=edge">
7      <meta name="viewport" content="width=device-width, initial-scale=1.0">
8      <link rel="stylesheet" href="style/style.css">
9      <script src="js/jquery-3.6.0.min.js"></script>
10     <script src="js/config.js"></script>
11     <script src="js/index.js"></script>
12     <title>Ana Sayfa</title>
13 </head>
14
```

Yukarıdaki index.html kodunda Türkçe karakter desteği, css kodu kullanılacağı gibi komutlar verilmiştir.

Ardından son olarak index.html'in kodlarına bakacak olursak anasayfamızı temsil etmektedir. Anasayfamızda sol kısmında yani bir arama butonu ve arama texti bulunmaktadır. Arama işlemleri sol tarafta gerçekleşmektedir. Sağ tarafında ise resim ekleme butonu ve üyelerimizin paylaştığı fotoğrafların sergilendiği bir alan bulunmaktadır.

```
<> index.html 1 X
<> index.html > html
14
15 <body>
16
17   <div class="container noselect">
18     <div class="left">
19       <div class="search">
20         <input id="query" type="text" placeholder="Ara...">
21         <a href="#" id="search"></a>
22       </div>
23       <div id="results" class="result">
24
25       </div>
26     </div>
27     <div id="posts" class="right">
28       <div class="topmenu">
29
30         <input onchange="submitForm();" type="file" id="upload" accept="image/*">
31
32         <a onclick="add();" id="add" class="add"></a>
33         <a onclick="delete();" id="delete" class="delete"></a>
34         <a onclick="logout();" id="logout" class="logout"></a>
35       </div>
36
37     </div>
38   </div>
39 </body>
40
41 </html>
```