



Codevist

Yazılım Geliştirme ve Kodlama Standartları

Temmuz 2016

codeVist

Başakşehir Living Lab Kuluçka Merkezi, Başak Mah. Abdülhamithan Cad. No:5 Başakşehir / İSTANBUL
www.codevist.com - info@codevist.com - [@codeVist](https://twitter.com/codeVist)



Yazarlar ve Emeęi Geçenler

Fatih COŞKUN
Mehmet ÖZ
Türkay DÜZBASAN



codeVist

Başakşehir Living Lab Kuluçka Merkezi, Başak Mah. Abdülhamithan Cad. No:5 Başakşehir / İSTANBUL
www.codevist.com - info@codevist.com - [@codeVist](https://twitter.com/codeVist)



Gizlilik:

Bu doküman gizlilik esaslarına göre hazırlanmıştır. Bu doküman içeriğindeki bilgilerin bütün hakları codeVist firmasına aittir ve ancak referans gösterilerek paylaşılabilir.



codeVist

Başakşehir Living Lab Kuluçka Merkezi, Başak Mah. Abdülhamithan Cad. No:5 Başakşehir / İSTANBUL
www.codevist.com - info@codevist.com - [@codeVist](https://www.instagram.com/codevist)



İÇİNDEKİLER

<u>İSİMLENDİRME VE KODLAMA KURALLARI</u>	3
<u>KODLAMA PRATİKLERİ</u>	7
<u>ARAYÜZ PRATİKLERİ VE STANDARTLARI</u>	16
<u>VERİTABANI KODLAMA STANDARTLARI VE PRATİKLERİ</u>	21
<u>KONFIGURASYON STANDARTLARI</u>	22
<u>PROJE DİZİN AĞACI</u>	23
<u>VERSİYON KONTROL PRATİKLERİ</u>	24
<u>MİMARİ STANDARTLAR VE PRATİKLER</u>	25
<u>KATMANLI MİMARİ STANDARDI</u>	25
<u>MİMARİ PRATİKLER</u>	27



İsmlendirme ve Kodlama Kuralları

1) Tip ve metod değerlerin ismlendirilmesinde Pascal Casing kullanılmalıdır.

```
public class SomeClass
{
    Public SomeMethod () {}
}
```

2) Metod argümanları (**parametreleri**) ve local değişkenler için Camel Notasyon kullanılmalıdır. İlk kelimenin ilk harfi küçük bundan sonra devam eden kelimelerin ilk harfleri büyük olmalı.

```
int number;
void MyMethod (int someNumber) {}
```

3) Interface isimleri "I" ile başlamalıdır.

```
Interface IyInterface () {}
```

4) Sınıf içerisinde yer alan private member değişkenlerin önüne "_" getirilmeli. Geri kalanı da Pascal Casing'e göre tamamlanmalıdır.

```
public class SomeClass
{
    private int _Number;
}
```

5) Özel olarak kodlanan Attribute'lar ismlendirilirken sınıf ismi "**Attribute**" ile bitirilmelidir.

```
public class SomeAttribute: System.Attribute {}
```

6) Özel yazılan Exception'lar ismlendirilirken isim "**Exception**" ile bitirilmelidir.

```
public class BusinessLayerException: System.ApplicationException {}
```

7) Metodlar ismlendirilirken metod isimleri fiil nesne ikilisi kullanılarak ismlendirilir. Metod isimleri fiil ile başlamalı ve nesne ile sonlandırılmalıdır. Böylece fiilin hangi nesne üzerinde hangi işlemi yaptığı anlaşılır kılınmalıdır.

```
public DialogResult ShowDialog () {}
```

```
public DialogResult GosterDiyalogu () {}
```

8) Değer döndüren metodlar dönüş değerini yansıtacak şekilde ismlendirilmelidir.

```
GetObjectState ();
NesneDurumunuAl ();
```



9) Açıklayıcı değişken isimleri kullanılmalıdır.

- Tek karakterlik değişken isimleri kullanılmaktan kaçınılmalıdır. “i” veya “t” gibi bir isim yerine “index” veya “temp” gibi isimler kullanılmalıdır.
- Public ve protected member’lar için Hungarian notasyonundan kaçınılmalıdır.
- İsimler anlamsız yere kısaltılmamalı. Mesela “number” yerine “num” kullanılmaktan kaçınılmalıdır.

10) System namespace’ inin içinde yer alan tipler yerine C#’ ta tanımlı tipler kullanılmalıdır.

`object`

(Doğru)

`Object`

(Yanlış)

`int` (Doğru)

`Int32` (Yanlış)

11) Generic’ leri kullanırken tiplerin yerine büyük harfler kullanılmalı. **“Type”** ile biten Generic isimlerinden kaçınılmalıdır.

//Doğru:

```
public class LinkedList<K,  
T> {}
```

//Sakınılmalı:

```
public class LinkedList<KeyType, DataType>  
{}
```

12) Projelerde anlamlı namespace’ ler hiyerarşik olarak kullanılmalıdır. Her proje tek bir kök namespace’ den başlayarak katman ve modül seviyesinde alt kırılımlara kadar bölünmelidir. Namespace’ ler bir ürün adını veya şirket adını yansıtır olabilir. Projeler için kullanılacak namespace kalıbı şu şekilde olmalıdır.

<Şirket ya da müşteri adı>.<Proje adı>.<Katman adı>.<Modül adı>

13) Tiplerin uzun isimlerini yazmaktansa **“using”** ifadesi kullanılması tercih edilmelidir.

System.Data.SqlClient.SqlCommand yerine :

```
using System.Data.SqlClient;
```

```
..
```

```
..
```

```
..
```

```
SqlCommand command = new SqlCommand ();
```

14) Sabit (**const**) ifadeler tamamen büyük harfler ile yazılmalı ve kelimeler arası **“alt çizgi”** karakteri ile ayrılmalıdır.

```
private const int DEFAULT_SIZE = 100;
```



15) Framework namespace' leri beraber gruplanırken, kendi namespace' lerimiz veya third-party namespace' ler altında gruplanmalı (tepedeki **“using”** satırlarında) ve arasında bir satır boşluk bırakılmalıdır.

```
using System;  
using System.Data;  
using System.Collections;  
using System.Collections.Generic;  
using System.ComponentModel;  
using  
MyCompany;  
using MyControl;
```

16) Delegate' i constructor' ı ile initialize etmek yerine delegate inference kullanılmalıdır.

```
delegate void SomeDelegate ();  
public void SomeMethod ()  
{  
  
    //Doğru  
    SomeDelegate someDelegate = SomeMethod;  
    //Sakınılmalıdır.  
    SomeDelegate someDelegate = new SomeDelegate (SomeMethod);  
}
```

17) Kesin çentikler (**indentation**) kullanılmalı. Standart olmayan tab veya çentikler kullanılmamalıdır. CodeVist içindeki projelerimizde çentik uzunluğu **3 boşluk** olmalıdır.

18) Yazılan Comment' ler dökümente edilen kod ile aynı şekilde girintili yazılmalıdır.

19) Comment' ler yazılırken doğru bir dil yapısı kullanılmalıdır. Aksi halde özentisiz bir geliştirme ortaya çıkacaktır.

20) Bütün member (**üye**) değişkenler class içinde en tepede, property' lerden ve metodlardan bir satır ile ayrılmış bir şekilde tanımlanmış olmalıdır.

```
public class MyClass  
{  
    private int _Number;  
    private string _Name;  
  
    public void SomeMethod1 ()  
    {}  
    public void SomeMethod2 ()  
    {}  
}
```

21) Lokal değişkenler kullanıldıkları yere en yakın satırda tanımlanmalıdır. Metod içi kullanımlarda eğer değişken sayısı çok değil ise metod' un başında tanımlanabilirler.

22) Dosya ismi barındırdığı class'ın adını ve sorumluluk alanını yansıtmalıdır.



23) Partial class' lar kullanılırken class' ların bulundukları dosya isimleri mantıksal isimden sonra oynadıkları rolü üstlenecek şekilde isimlendirilmelidir.

```
//MyClass.cs
public partial class MyClass {}

//MyClass.Designer.cs
public partial class MyClass {}
```

24) Kullanılabileceği yerde yeni bir satıra geçerken her zaman küme parantezi "{ "kullanılmalıdır.

25) Anonymous (**anonim**) metodlar kullanılırken, kod akışını delegate tanımı ile aynı hizada yapılmalıdır.

```
public delegate void SomeDelegate (string name);
```

```
//Doğru
public void InvokeMethod ()
{
    SomeDelegate someDelegate = delegate (string name)
    {
        MessageBox.Show (name);
    };
}
```

```
someDelegate ("CodeVist");
}
```

```
//Sakınılmalı
public void InvokeMethod ()
{
    SomeDelegate someDelegate = delegate (string name) {MessageBox.Show (name) };
    someDelegate ("CodeVist");
}
```

26) Parametresiz anonim metodlarda da parantez kullanılmalıdır.

```
delegate void SomeDelegate ();
```

```
//Doğru
SomeDelegate someDelegate = delegate () {
    MessageBox.Show ();
};
```

```
//Sakınılmalı
SomeDelegate someDelegate = delegate {
    MessageBox.Show ();
};
```




Kodlama Pratikleri

- 1) Bir dosya (.cs dosyası) içine birden fazla class tanımlamaktan kaçınılmalıdır.
- 2) Bir kaynak dosya sadece bir namespace' e ait tipleri içermeli, aynı dosya içinde birden fazla namespace kullanımından kaçınılmalıdır.
- 3) Bir dosya içinde 500' den fazla satır kullanmaktan kaçınılmalıdır. Bu gibi durumlarda **extract class** refactoring tekniği kullanılarak sınıf üzerindenki farklı sorumluluk alanlarına ait metodlar ayrı sınıf dosyaları olarak ayrılabilir. (otomatik üretilen kodlar hariç)
- 4) Metodlar en fazla 40 satır kod içermeli, büyük metodlar **extract method** refactoring metodu kullanılarak küçük metodlara indirgenmelidir. Refactoring için "Resharper" ya da VS 2005 refactoring aracı kullanılmalıdır.
- 5) Bir metoda 5' ten fazla argüman tanımlanacaksa, argüman tanımları bir struct ya da tip güvenli özel obje içinde gönderilmelidir. Bunun için "**convert parameters to objects**" refactoring tekniği kullanılmalıdır. Bu şekilde metod imzası içerisinde yer alan parametrelerin değeri halinde metod imzasında değeri ikiliğe kalmadan bunu çağıran diğer katmanlar değişimden en az şekilde etkilenirler.

```
//Sakınılmalıdır
public void KaydetKullanici (string isim, string soyad, byte yas, string adres) {.....}

//Doğru
public void KaydetKullanici (Kullanici kullanici) {.....}

public class Kullanici
{
    public string Isim;
    public string Soyad;
    public byte Yas;
    public string Adres;
}
```

- 6) Bir satırda en fazla 80 karakter kullanılmalıdır. Uzayan hesaplama ve metin birleştirme gibi işlemler farklı satırlara bölünerek okunabilirlik artırılmalıdır.

```
string message = GetHeader () + GetMenu () + GetMainBody () + SEPERATOR + GetFooter ();
```

```
string message = GetHeader () +
    GetMenu () +
    GetMainBody () +
    SEPERATOR +
    GetFooter ();
```

- 7) Üretilen kod, manuel olarak değiştirilmemelidir. Eğer değiştirilecekse de :
 - a. Kodlama standartlarına uygun bir biçimde değiştirilmelidir.
 - b. Mümkünse kod partial class' lara eklenmelidir.



8) Zaten belli olan bir şeyi ifade eden comment'lerden kaçınılmalıdır. Kodda kullanılan isimlendirme ve teknikler ile kodun kendini açık olarak anlatması yolu tercih edilmelidir. Okunabilir değişken ve metodlardan oluşmuş iyi kod zaten comment gerektirmez. Burada esas almamız gereken kural, parametre ve değişken isimlerinin yapılan işi açık bir şekilde anlatması, yani kodların kendini anlatması gerekir. Bunun yeterli gelmediği algoritmik veya karmaşık operasyonlar için metod içi commentler kullanılmalıdır.

9) Sadece operasyonel varsayımlar ve içindeki algoritmalar dökümanite edilmelidir.

10) Metod seviyesinde dökümantasyondan kaçınılmalıdır.

- API dökümantasyonu için kod harici genişletilebilir bir dökümantasyon oluşturulmalı.
- Metod seviyesinde commentler sadece developerlar için tooltip olmak üzere yapılmalıdır.

11) Sıfır ve bir hariç, bir nümerik değer asla hard-code olarak yazılmamalı; yerine bir sabit (**const**) kullanılmalıdır.

12) "**const**" direktifi sadece doğal sabit değerler için kullanılmalıdır. (Haftanın gün sayısı gibi)

13) Read-only değişkenler yerine "**const**" kullanılmaktan kaçınılmalıdır. Bu durumlar da "readonly" direktifi kullanılmalıdır.

```
public class MyClass
{
    public const int DAYS_IN_WEEK = 7;
    public readonly int Number;
    public MyClass (int someValue)
    {
        Number = someValue;
    }
}
```

14) Gerektiği durumlarda her varsayım doğrulanmalıdır. Bu doğrulamalar Birim testlerin kullanıldığı durumlarda tercih edilmez ya da gözardı edilebilir.

```
using System.Diagnostics;

object GetObject
() {...}

object someObject = GetObject ();
Debug.Assert (someObject != null);
```

15) Yazılan her satır kodun çalışabilirliği test edilmelidir. (**White Box Testing**).

16) Sadece kasıtlı olarak handle edilen exception'lar yakalanmalıdır. Exceptionlar yakalanırken en özel seviyede exceptionların yakalanmasına dikkat edilmelidir. Hataya özel aksiyon almak için en detay seviyede exception hiyerarşisi kullanılmalıdır.



17) “**catch**” içinde “**throw**” ile bir exception atılacaksa, stack yerinin ve orjinal mesajın korunması açısından, daima orjinal exception atılmalıdır (veya orginal exception’ dan üretilen başka bir exception). Exception Handling Block kullanılan durumlarda exception’ın başka bir exception tipi ile değiştirilmesi gerektiğinde bu yapılmalıdır.

```
catch (Exception exception)
{
    MessageBox.Show
    (ex.Message); throw; //Atılan
    exception’ın aynısı
}
```

18) Metod dönüş değeri olarak error kodunun verilmesinden kaçınılmalıdır.

19) Kendi exception class’ larımızı yazarken mutlaka spesifik bir sistem exception sınıfından türetme yolu tercih edilmelidir.

20) Kendi exception’ larımız yazarken :

- a. Exception class’ ından yada spesifik exception’ dan türetmeli
- b. Kendi serialization’ ınımızı sağlamalıyız.

21) Bir assembly içinde birden fazla **Main ()** metodu bulundurmaktan kaçınılmalıdır.

22) Sadece en gerekli tipleri “**public**” olarak işaretlemeliyiz. Diğerleri “internal” olarak işaretlenmelidir. Sınıfların kapsülleme (**encapsulation**) pratiğine uygun olarak dış istemcilere sadece ihtiyaç duyulan member’ ları public olarak sunulmalıdır.

23) Friend assembly kullanımından kaçınılmalıdır. Bu assembly’ ler arası bağımlılığa neden olur.

24) Uygulama assembly’ lerindeki kod minumumda tutulmalıdır. Business logic ayrık class library’ lerde tutulmalıdır.

25) Enum’ lar için varsayılan değer kullanımından kaçınılmalı ve her enum’ a açık değer ataması yapılmalıdır.

```
//Doğru
public enum Color
{
    Red, Green, Blue
}
//Sakınılmalı
public enum
Color
{
    Red = 1 ,Green = 2 ,Blue = 3
}
```



26) Bir enum'a tip atamaktan sakınılmalıdır.

```
//Sakınılmalı
public enum Color: long
{
    Red,Green,Blue
}
```

27) Bir “if” ifadesinde, tek satırlık bir şart bile yazılacak olsa, küme parantezleri kullanılmalıdır.

28) “?.” (*ternary conditional operator*) kullanmaktan kaçınılmalıdır. Bu kodun okunabilirliği zedeler.

```
//Sakınılmalı
string str = (test) ? “doğruysa”: “yanlışsa”;
```

29) Boolean koşul ifadelerinde metod çağırımından kaçınılmalıdır. Bu durumlar da yerel değişkenlere ilgili koşul metodu ile atama yapılarak kullanım gerçekleşmelidir.

```
bool IsEverythingOK ()
{...}

//Sakınılmalı
if (IsEverythingOK ())
{...}

//Doğru
bool ok = IsEverythingOK ();
if (ok)
{....}
```

30) Arrayler (*Dizi*) her zaman indeksi sıfırdan başlayacak şekilde kullanılmalıdır.

31) Referans tiplerini içeren bir array'i her zaman bir loop döngüsü ile initialize edilmelidir. Aksi takdirde dizi içerisinde yer alan objeler kullanım anında null pointer hatası verecektir.

```
public class MyClass {}

const int ARRAY_SIZE = 100;

MyClass [] array = new MyClass [ARRAY_SIZE];
for (int index = 0; index<100; index++)
{
    array [index] = new MyClass ();
}
```

32) “*public*” ve “*protected*” member'lerin yerine kullanıcılara ve türetilen alt sınıflara “*property*”ler sunulabilir.

33) Kalıtım yolu ile metod uyarlamalarında “*new*” qualifier'ını kullanmaktan kaçınılmalı, yerine “*override*” kullanılmalıdır.

34) Sealed olmayan bir sınıfta “*public*” ve “*protected*” metodlar “*virtual*” olarak tanımlanmalıdır.



35) Interop hariç unsafe kod kullanılmamalıdır.

36) Explicit çevrimlerden kaçınılmalıdır. “as” operatörü kullanarak güvenli bir şekilde tip çevrimi yapılabilir.

```
Dog dog =new Dog ();
GermanShepherd shepher = dog as GermenShepherd;

if (shepherd != null)
{.....}
```

37) Bir delegate çağrılmadan önce her zaman null kontrolünden geçirilmelidir.

38) “public” event memberlar’dan kaçınılmalı, onun yerine event accessors kullanılmalıdır.

```
public class MyPublisher
{
    MyDelegate _SomeEvent;
    public event MyDelegate SomeEvent
    {
        add
        {
            _SomeEvent += value;
        }
        remove
        {
            _SomeEvent -= value;
        }
    }
}
```

39) Event handling delegate’ leri yazmaktan kaçınılmalı, yerine EventHandler<T> veya GenericEventHandler kullanılmalıdır.

40) Kasıtlı event atmaktan kaçınılmalıdır. EventHelper ile güvenli bir şekilde eventler atılabilir.

41) SOA mimarisinde olan projelerde orta katman için mutlaka interface’ ler kullanılmalıdır. Interface’ ler unit testlerin yapıldığı durumlarda da kullanılmalı ve böylece mock (**taklit**) testlerin yazımı kolaylaştırılmalıdır. Interface’ ler ile bağımlılığı fazla olan sınıf ve birimlerin testlerinde belli bir interface’ i implement eden mock nesneler yolu ile birim testler yapılmalıdır. Referans için Mock Testing konusunu araştırabilirsiniz.

42) Test edilebilirliğin düşünüldüğü modül veya projelerde tüm Service katmanında yer alan sınıflar tanımlı bir Interface’ i uygulamalıdır.

43) Bir üyeli interface’ lerden kaçınılmalıdır.

44) Bir interface’ in 3-5 arası üye içermesi için gayret gösterilmelidir.

45) Bir interface’ de 20’ den fazla üye olmamalıdır.

46) Bir interface içinde event tanımlamaktan kaçınılmalıdır.

47) Abstract (**Soyut**) bir class kullanırken bir interface de alternatif olarak sunulmalıdır.

codeVist

Başakşehir Living Lab Kuluçka Merkezi, Başak Mah. Abdülhamithan Cad. No:5 Başakşehir / İSTANBUL
www.codevist.com - info@codevist.com - @codeVist



48) Açık interface uygulanması tercih edilmelidir.

49) Bir tipin bir interface' i uyguladığı farz edilmemeli, bu interface' i uygulayıp uygulanmadığı sorgulanmalıdır.

```
SomeType obj1;  
IMyInterface obj2;  
obj2 = obj1 as IMyInterface;  
if (obj2 != null)  
{  
    obj2.Method1 ();  
}  
else {.....}
```

50) Kullanıcılara sunulacak string ifadeler asla hard-code yazılmamalı. Onun yerine resource (**resx**) dosyalarında çekilmelidir. Kullanıcı uyarı ve hata mesajları mutlaka resource dosyalarında tutulmalıdır.

51) Deployment' ta değişecek string ifadeler hard-code yazılmamalıdır (**connection string gibi**). Bunlar için configuration application block' ları kullanılmalı ve string ifadeler ve parametreler XML dosyalarında saklanmalıdır.

52) "" boşluk tanımlı yerine String.Empty kullanılmalı.

```
//Doğru  
string name = String.Empty;  
  
//Sakınılmalı  
string name = "";
```

53) Uzun bir string ifade oluşturulacaksa, string birleştirme işlemleri için performans açısından daha iyi olan **StringBuilder** kullanılmalıdır.

54) Struct yapılarında metod tanımlamaktan kaçınılmalıdır.

55) Statik member değişkenler kullanırsak her zaman statik bir constructor' da tanımlanmalıdır.

56) Early-binding (**Erken Bağlama**) mümkünse asla late-binding (**geç bağlama**) kullanılmamalıdır.

57) Uygulamalarda logging ve exception handling kullanılmalıdır. Bunun için Enterprise Library içindeki **Logging** ve **Exception** blockları konfigüre edilmelidir.

58) "switch" bloğu hariç "goto" asla kullanılmamalıdır.



59) Doğrulama yapan bir **“switch”** bloğunda daima **“default”** case bulunmalıdır.

```
int number = SomeMethod ();
switch (number)
{
    case 1:
        Trace.WriteLine ("Case 1");
        break;
    case 2:
        Trace.WriteLine ("Case 2");
        break;
    default: Debug.Assert
        (false); break;
}
```

60) **“this”** referansı bir constructor’ dan diğerine çağrım yapılmadıkça kullanılmamalıdır.

```
public class MyClass
{
    public MyClass (string message)
    {}
    public class MyClass (): this ("Hello")
    {}
}
```

61) Base class üyelerine ulaşırken **“base”** kullanılmamalıdır (isim çakışması ve çelişki yarattığı durumlar hariç).

```
//Uygun kullanım
public class Dog
{
    public Dog (string name)
    {}
    virtual public void Bark (int howLong)
    {}
}
public class GermanShephard: Dog
{
    public GermanShephard (string name): base (name)
    {}
    override public void Bark (int howLong)
    {
        base.Bark (howLong);
    }
}
```

62) GC.AddMemoryPressure () kullanılmamalıdır. COM ve unmanaged nesnelerin kullanıldığı durumlarda explicit destructor yordamları kullanılmalı ve memory’ de bulunan kullanılmayan kaynaklar boşa çöpe atılmalıdır.

63) HandleCollector’ a (unmanaged kaynaklara ait olan handle’ları takip eden ve belli bir noktada bunlar için garbage collection’ı zorlayan sınıf) güvenilmemelidir.



64) Her zaman default olarak unchecked modda çalışıyor olmalıyız (performans açısından).

```
int CacIPower (int number, int power)
{
    int result = 1;
    for (int count =1; count<= power; count++)
    {
        checked
        {
            result *=number;
        }
    }

    return result;
}
```

65) Conditional compiling'den (**Şartlı Derleme**) (#if....#endif) kaçınılmalı, yerinde **"Conditional"** metodlar kullanılmalıdır.

```
public class MyClass
{
    [Conditional ("MySpecialCondition")]
    public void MyMethod ()
    {}
}
```

66) Generics kullanılan kodda, System.Object'e ve System.Object'ten çevrimlere yer verilmemeli; **kısıtlar** veya **"as"** operatörü kullanılmalıdır.

```
class SomeClass
{}

//Sakınılmalı class
MyClass<T>
{
    void SomeMethod (T t)
    {
        object temp = t;
        SomeClass obj = (SomeClass) temp;
    }
}

//Doğru
class MyClass<T> where T: SomeClass
{
    void SomeMethod (T t)
    {
        SomeClass obj = t;
    }
}
```




67) Generic interface' lerde kısıtlardan kaçınılmalı, yerine Strong-typed tipler kullanılmalıdır.

```
public class Customer {...}

//Sakınılmalı
public interface IList<T> where T: Customer
{.....}

//Doğru
public ICustomerList: IList<Customer>
{....}
```

68) Interface' lerde method spesifik kısıtlar belirtilmelidir.

69) Delegate' lerde kısıt (**where operatörü**) belirtilmemelidir.

70) Eğer bir class' ın hem generic, hem de generic olmayan türleri varsa, daima generic olan kullanılmalıdır.

71) Generic olmayan bir interface' den türeyen generic interfacer için, metodlar üzerinde daima açık interface uygulanmalı ve generic olmayan metodlar daima generic olanlarlar kullanılarak temsil edilmeli.

```
class MyClass: IEnumerable<T>
{
    IEnumerator<T> IEnumerable<T>.GetEnumerator ()
    {.....}
    IEnumerator IEnumerable.GetEnumerator ()
    {
        IEnumerable<T> enumerable = this;
        return enumerable.GetEnumerator ();
    }
}
```



Arayüz Pratikleri ve Standartları

- 1) Arayüz kodlamasında macar notasyon kullanılmalıdır. Kullanılan kontrolleri çağırıştıran prefix (**ön takı**) ile arayüz kontrollü tanımlanmalı ve sonrası Camel olarak devam etmelidir.

Örn: `TextBox txtName;`

Kontrollerde kullanılacak genel prefix tablosu

Prefix	Kontrol
txt	Textbox
grd	Grid
tbl	Table
lb	Listbox
ddl	Dropdown List
tab	Tab
frm	Form
fuc	File Upload Control
btn	Button
chk	Checkbox
rbl	Radio Button List
rtxt	Rich Text Box
lbl	Label
trw	Tree View

- 2) Arayüz tasarımında tüm alan etiketleri sağa hizalı olmalıdır.

Username: codevist
(Use this to log in to the site)

Email address:
(We keep your email address private)

New Password:

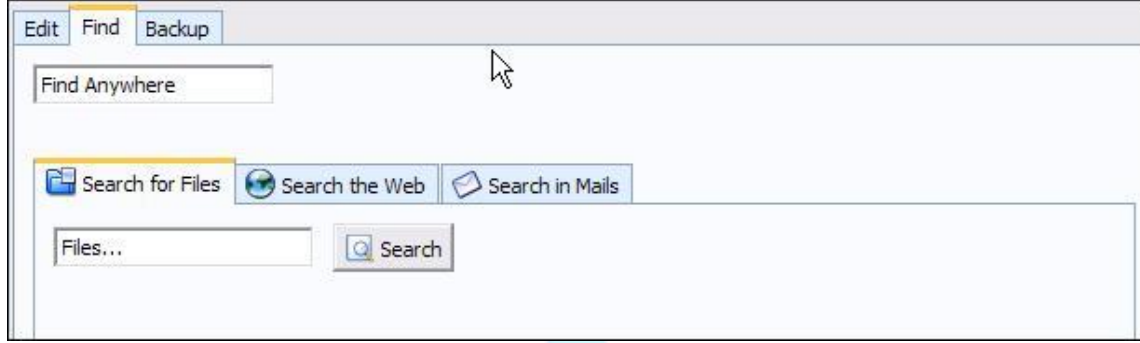
Verify Password:

codeVist

Başakşehir Living Lab Kuluçka Merkezi, Başak Mah. Abdülhamithan Cad. No:5 Başakşehir / İSTANBUL
www.codevist.com - info@codevist.com - [@codeVist](https://twitter.com/codeVist)



- 3) Veri yoğun ve çok fazla parametreden oluşan formlarda kullanılabilirlik açısından mantıksal bölümlere ayrılmış tab sayfaları tercih edilmelidir.



- 4) HTML kodu içerisinde stil ve format kullanımından kaçınılmalıdır. Inline yapılan formatlama kodu HTML dosya boyutunu arttırmakta dolayısıyla performansa negatif etki yapmaktadır. Tüm projelerde global .css ve özel css dosyaları ile css dosya include metodu kullanılmalıdır.

Sakınılmalı

```
<td border="1" style="padding-left:5px">
```

Doğru

```
<td class="label"></td>
```

- 5) Veri formları üzerinden yapılan tüm işlemlerde kullanıcıya başarılı ya da başarısız durum uyarısı standart olarak verilmelidir.
- 6) Kullanıcı tarafından yapılan veri giriş işlemleri sonrasında zorunlu alan veya özel veri doğrulama kontrolleri sonucunda kullanıcıya daima standart uyarı özeti sunulmalı ve bunlar kırmızı font veya sayfa üzerinde DHTML modal dialog windowlar ile sağlanmalıdır. Zorunlu alanlarda mutlaka etiket alanlarında kırmızı * göstergeleri kullanılmalıdır.

Örn: Adınız *:

- 7) Form aksiyon butonları daima kontrollerin altında standart dizilimde olmalıdır.



- 8) Form tipi arayüzlerde giriş kontrolleri birbirine yakın bilgilerin olduğu tasarımsal alt gruplara ayrılarak sunulmalıdır. Örn: Some controls, Some Settings altında 2 bölüme ayrılmıştır.

Some controls

Name

Amount: 0

Type

E-Mail

Comment

Submit

Some settings

☐ Permit others to view my favorites

☒ Use the very high bitrate

- 9) Form sonlarında bulunan aksiyon butonlarında hiç bir zaman İptal, Sil gibi negatif işlemler Kaydet, Güncelle gibi butonlardan önce gelmemeli ve hep en sonda olmalıdır.

Sex: Male

Birthday: Month: Day: Year: 1979

Show my full birthday in my profile.

Hometown:

Country: Turkey

Political Views: Select Political Views:

Religious Views:

Save Changes Cancel

- 10) Formlarda kullanılan kontrollerde mutlaka TAB Index sırasının doğru olmasına özen gösterilmelidir. Kullanıcılar hızlı veri girişleri için TAB tuşundan sıklıkla faydalanmakta ve bu durumda kullanıcı form' un akış yönünde kolaylıkla ilerleyebilmelidir.
- 11) Sayfa yüklenme süresinin uzun olabileceği veya sayfa tam olarak yüklenmeden kullanıcının hatalı işlem yapma olasılığının olduğu durumlarda page postback' leri anında sayfanın render edilmesini önleyip tam olarak yüklenmesi aşamasına kadar sayfa yükleniyor iletisi veya imajını kullanabilirsiniz.



- 12) Site haritasının geniş ve hiyerarşik olarak 3 ve üzerinde seviyeden oluştuğu durumlarda her sayfanın başında ekmek kırıntısı (**Breadcrumbs**) kullanımına özen gösteriniz.



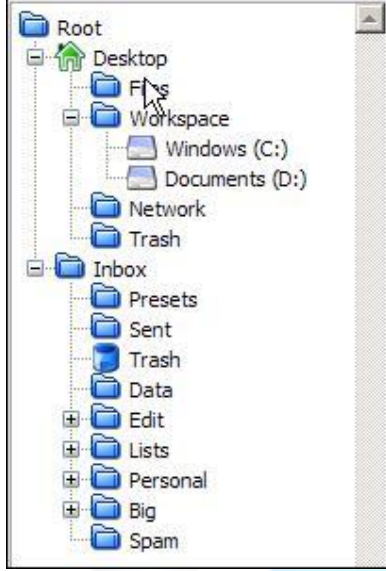
- 13) Veri listelerinin grid ve tablolar ile gösterildiği durumlarda aşağıdaki pratikleri uygulayınız:

- Sıralama: tabloda yer alan başlıklarda çift yönlü sıralama seçeneği
- Sayfalama: 10' UN üzerinde kayıt içeren listelerde projenin standardına göre değişen 10' luk 20' lik kayıtlara bölerek listeyi sunabilirsiniz. Sayfalama çubuğu gridin altında olmalı ve her sayfa rakamla belirtilmelidir.
- Varsayılan Sıralama: tablo mutlaka son kullanıcının kullanım alışkanlığına en uygun kayıt sıralamasına uygun olarak ilk yüklendiğinden gelmelidir.



14) Menü ve navigasyon için koşullara göre 2 alternatif yol tercih edilebilir.

- a. Menü yapısı karmaşık, 3 ve üzerinde derinlik seviyesine sahip ise ağaç menüsü kullanılır



- b. 2 seviyeli navigasyonlarda yatay ya da dikey açılıp kapanabilir menüler tercih edilmelidir. Kullanılan javascriptlerin standart browserlar tarafından desteklenmesie ve performans açısından optimize olmasına özen gösteriniz.



Veritabanı Kodlama Standartları ve Pratikleri

- 1) Stored Procedure isimleri kesinlikle **sp_** ile başlamamalıdır. SQL Server **sp_ prefixi** ile başlayan Spler için önce master SP' leri kontrol eder buda performansa etki eder.
- 2) **TSQL** ifadeleri daima büyük harfle yazılır. SELECT, FROM, SUM, COUNT vb.
- 3) Tablo isimlendirmelerinde **Pascal** notasyon kullanılır. Önr: Uye, UyeRol
- 4) Tablo isimlerinde çoğul kelimeler yerine tekil isimler tercih edilmelidir.

Doğru: Uye
Yanlış: Uyeler

- 5) SP isimlendirmelerinde ana fonksiyonun işletildiği ya da üzerinde işlem yapıldığı tablo ismi başa yazıldıktan sonra yapılan işlem türüne göre işlem türünü gösteren işlem adı ve sonrasında işlemin hangi alanlara ya da aksiyona göre yapıldığını gösteren açıklama bölümü getirilmelidir:

Örn: Tablo adı = Siparis

İşlemler

- i. SELECT : SiparisSelectByMusteriID
 - ii. DELETE : SiparisDeleteByMusteriID
 - iii. UPDATE : SiparisUpdateAsTamamlandi (yapılan aksiyona göre)
 - iv. INSERT : SiparisInsertNew
 - v. BATCH : SiparisBatchTeslimatHazirlik
- 6) SP yazımında **INNER JOIN** içeren **SELECT** ifadelerinde her **JOIN** ayrı satırda ve girintilemeye uyulmuş olarak yazılmalıdır.
 - 7) View isimlendirmeleri **vw prefixi** ile başlar ve sonrasında tuttuğu veriye uygun olarak tablo isimlendirmesine uygun bir isim seçilir. Örn : vwAktifMusteri
 - 8) Tabloda bulunan alanların kısıtlanması ya da ara tablo gibi daha sade veri kümeleri kullanımı için View kullanımı tercih edilebilir.
 - 9) Tablo indexlerinin belirlenmesinde en çok arama yapılan ve filtrelenen alanlar seçilmelidir.
 - 10) Projenin modüllerine uygun olarak veritabanında alt şemalar tanımlanmalıdır. Bu şemalar tablo tanımlarında kullanılarak tabloların gruplandırılması sağlanabilir.

Örn: Proje modülleri ve şemaları Musteri, Siparis, Urun, Sistem Tablo

tanımları: Musteri.Musteri, Musteri.Adres, Siparis.SiparisDetay
 - 11) **Primary key** ve **identity** olan alanlarda kolon isimleri **ID suffix'i** ile bitirilmelidir.
 - 12) Tablo alan (**kolon**) isimlendirmelerinde Pascal notasyon kullanılmalı ve tekil ifadeler yer almalıdır.



Konfigurasyon Standartları

- 1) Canlı sistem kurulumlarında debug modda olmamasına dikkat edilmelidir. Bu mod geliştirme aşamasında kullanılmalıdır.

```
<compilation debug="false">
```

- 2) Tüm web projelerinde özel hata sayfalarının kullanımına ve kullanıcı dostu hata gösteriminin uygulamasına dikkat ediniz.

```
<customErrors mode="RemoteOnly">  
  <error statusCode="404" redirect="NoPage.htm"/>  
  <error statusCode="500" redirect="ServerError.htm"/>  
</customErrors>
```

- 3) Uygulamanın diline uygun culture seçimini genel olarak config dosyasından ayarlayınız.

```
<globalization culture="tr-TR" requestEncoding="UTF-8" responseEncoding="UTF-8"/>
```

- 4) Tüm site genelinde uygulanacak sayfa ayarları için standart olarak aşağıdaki standardı uygulayınız.

```
<pages smartNavigation="true" validateRequest="true"  
maintainScrollPositionOnPostBack="true"></pages>
```

- 5) Web uygulamaların session yönetimi için belirlenmiş gereksinimlere uygun session parametrelerini tanımlayınız.

```
<sessionState timeout="20" mode="InProc"></sessionState>
```

- 6) Config dosyalarında appSettings kullanımı çok artıyor ise harici bir paramtere config dosyası kullanımını tercih ediniz. Bu kullanım için Enterprise Library Configuration Application Block ya da Nini gibi basit knfigurasyon frameworklerini tercih edebilirsiniz.








Proje Dizin Ağacı

Tüm proje dosyaları ve kodları için yazılım uzmanları **C:\Projects** klasöründe çalışırlar. Proje adı veya müşteri adına göre bu klasörde projeye özel alt klasörler açılır. Eğer müşterinin 1'den fazla projesi varsa klasör ismi olarak müşteri adı, sonrasında proje adı için bir alt klasör daha oluşturulur.

Müşteri Projeleri Klasör Yapısı

- C:\Projects\TFF
 - C:\Projects\TFF\FYS
 - C:\Projects\TFF\MilliTakimIstatistik

Proje Klasör Yapısı

 Blueprint <ul style="list-style-type: none"> arch<ul style="list-style-type: none">Release1.0-2008-02-12Release2.0-2008-04-15 doc<ul style="list-style-type: none">analizilerlemekurulummusteridentasarimtesttoplantiyardim lib src<ul style="list-style-type: none">BusinessCommonPersistencePresentationTest	<p>Proje Adı</p> <p>Proje' in geçmi e dönük release arşivleri</p> <p>Sürüm numarası ve tarihe göre klasör</p> <p>“</p> <p>Proje dokümanları</p> <p>Analiz dokümanları (SRS, Prototip) Proje ilerleme raporları</p> <p>Kurulum dokümanları</p> <p>Müşteriden gelen referans dokümanları Teknik tasarım dokümanları (SDS)</p> <p>Test case ve plan dokümanları</p> <p>Müşteri ile gerçekleştirilen toplantı notları Son kullanıcı yardım dokümanları</p> <p>Kullanılan tüm library, dll dosyaları için referans noktası</p> <p>Kaynak kodlar ana klasörü</p> <p>İş katmanı projeleri ana klasörü</p> <p>Proje özelinde ortak diket katman projeleri klasörü</p> <p>Veri eri im projeleri klasörü</p> <p>Arayüz projeleri</p> <p>klasörü Test projeleri</p> <p>klasörü</p>
---	---



Versiyon Kontrol Pratikleri

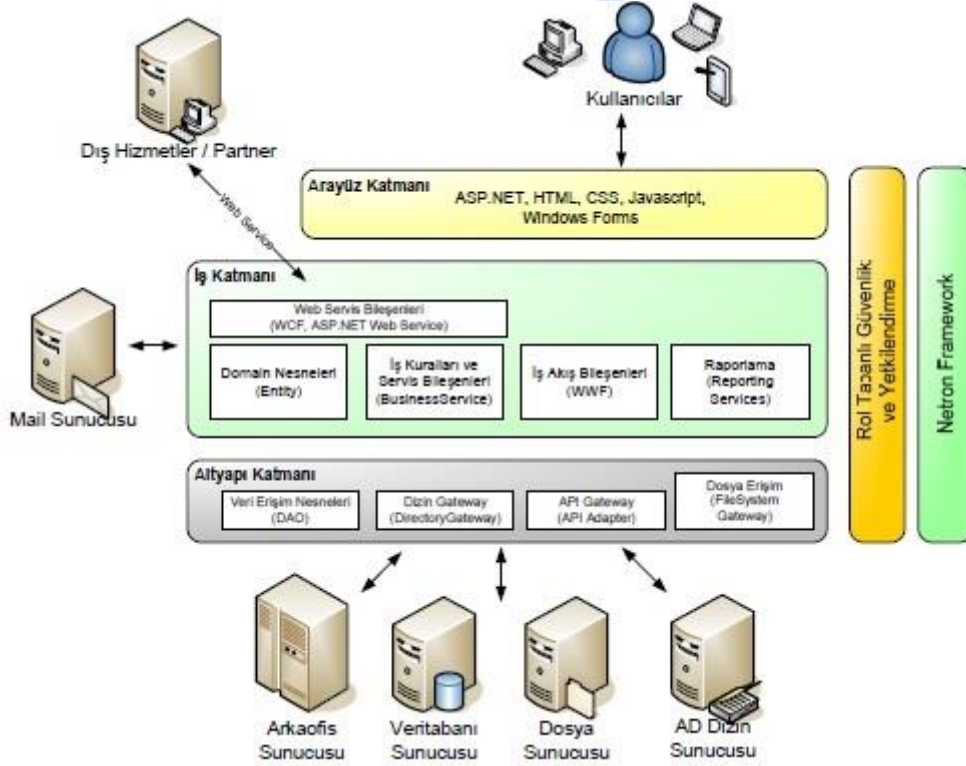
- 1) Her sabah projede kod geliřtirmeye başlamadan VSS (Diğer Versiyon Kontrol Sistemi de olabilir) client' ı üzerinden tüm proje klasörü **Get Latest Version** ile lokale alınır. Bu işlem Visual Studio üzerinde yapılan Get Latest Version işleminden daha performanslı ve güvenlidir. Kod tabanı alındıktan sonra VS editörü üzerinden çalışmaya başlanabilir.
- 2) Her **check-in** işleminde mutlaka tüm projenin başarı ile build edildiğinden emin olunulmalıdır.
- 3) Yapılan check-in işlemlerinde sık sık label ve comment kullanımına dikkat edilmelidir. Bu yorum ve etiketler kodun sürüm geçmişinde önemli bir bilgidir.
- 4) Günlük çalışma içersinde yapılan kodlamalar normal durumlarda mesai bitimine yakın günlük check-in standartlarına uygun olarak kod tabanına entegre edilmelidir.





Mimari Standartlar ve Pratikler

Katmanlı Mimari Standardı



Katmanlı uygulama mimarisi uygulama düzeyindeki mimari yaklaşımı sunmaktadır. Bu mimari kalıptaki ana motivasyonlar ve prensipler şu şekildedir:

- Katmanlar iş sorumluluklarına göre kavramsal modüllere ayrıştırılmıştır. Bu ayrıştırma kodlama açısından sorumluluk alanlarının farklı modüllere bölünerek değişim ve bakım maliyetlerinin azaltılması, performans artışı için yatay ölçeklendirmeye izin vermesi, kurulum aşamalarında birbirinden bağımsız modüler kurulumu desteklemesi ve gerekli lisans maliyetlerinde düşüş sağlaması gibi faydaları sağlamaktadır.
- Sistemin esneklik düzeyi artırılmış ve dış sistemler ile web servisleri aracılığıyla açık standartlara uygun, teknoloji bağımsız olarak entegrasyona izin vermektedir.
- Operasyonel destek anlamında uygulamalarda kullanılan ortak ilgileri Framework (**ortak kütüphane**) içerisine alarak tekrar kullanılabilirlik sağlanmış olmaktadır.
- Global standarda uygun uygulamalar geliştiren ya da destek veren teknik ekip açısından kolay bir ortam oluşturmaktadır.

Bu mimari yapıda olası tüm çerçevenin genel hatları ele alınmış olup, projenin yapısına göre bu yapının bir alt kümesi kullanılmalıdır.



Arayüz Katmanı

Web ve windows arayüz teknolojilerini içeren sadece kullanıcı ile yürütülen iletişim ve akışı yönlendirmekle görevlidir. Herhangi bir iş kuralı içermeksizin girdi doğrulama, kullanıcı uyarıları, Veri giriş, raporlama işlemleri için kullanılmaktadır. Kullanılan teknolojiler ASP.NET, HTML, XML, CSS, Javascript, Windows Forms...

İş Katmanı

İş gereksinimlerinin uygulandığı ve sunulduğu çekirdek katman.

Domain Nesneleri: İş modellerine uygun veri ve davranış içeren birim nesneler. Kayıt bazında bilgi tutarak bunların altyapı ile arayüzü katmanları arasında taşınması ve kayıt bazında bazı ek davranışların eklenmesi aşamalarında kullanılmaktadır.

İş Kuralları ve Servis Bileşenleri: İş kuralları ve mantığının yapılandırıldığı bu alt katman yapılan işlemlerin tek bir transaction içinde sonlandırılması, farklı domain ve veri erişim sınıflarını yönetmekten ve iş kurallarını kontrol etmekten sorumludur.

İş Akış Bileşenleri: İş akışlarını soyut ve bağımsız olarak tasarlayıp süreç yönlendirmeli elektronik akışlar için kullanılmaktadır. WWF ile tasarlanan süreçler çalışma anında dinamik olarak değiştirilebilmekte ve kolay kurulumu ile iş sürekliliğine yardımcı olmaktadır.

Raporlama Bileşenleri: Reporting services veya 3. parti raporlama araçları ile geliştirilmiş ve kullanıma hazır raporlar

Web Servis Bileşenleri: Windows istemcileri ve dış partnerlerin iş servislerini kullanabilmeleri için sunulmuş bir servis arayüz katmanı. Web uygulamalarının web servislerini kullanmadan direk servis bileşenlerini kullanması yeterli iken Windows istemcileri ve dış partnerlerin süreçlere dahil olması için açık standart web service arayüzlerini kullanması sisteme esneklik kazandırmaktadır.

Altyapı Katmanı

Fiziksel sunuculara olan her türlü erişim bu katman tarafından gerçekleştirilir.

Veri Erişim Nesneleri: Veritabanı sunucuları üzerinden her türlü çift yönlü **SQL** işlemleri bu nesneler üzerinden yapılır.

Dizin Gateway: **Active Directory** ve **LDAP** gibi dizin servislerine yapılan sorgulama ve erişimler bu gateway üzerinden yapılır.

API Gateway: Arkaofis yazılımlarına yapılacak entegrasyon arka ofis ürünlerinin sunduğu API' ler üzerinden yapılır. Bu alt katman arka ofisin karmaşık yapısını gizleyerek adaptör görevini yürütmektedir.

Dosya Erişim: Dosya Sunucuları ile olan iletişim ve dosya alışverişi bu bileşenler üzerinden gerçekleştirilir.

Dikey Katmanlar

Dikey katmanlar tüm yatay katmanlar tarafından kullanılabilen ortam etmenleri kapsayan kütüphanelerden oluşmaktadır.

Rol Tabanlı Güvenlik ve Yetkilendirme: tüm kimlik denetimi ve yetkilendirme servislerini barındırır.

CodeVist Framework: Tüm projelerde ortak kullanıma sahip mikro modüller ve destekçi fonksiyonları içeren kütüphane.



Mimari Pratikler

- 1) İş katmanında yer alan tüm servis nesnelerinde metod seviyesinde transaction handling ve exception handling standart olarak yapılmalıdır.

```
public DataTable GetFoo (int foolD)
{
    try
    {
        using (TransactionScope transaction = new TransactionScope ())
        {
            FooDAO fooDao = new FooDAO ();
            fooDao.MakeSomething (foolD);
            fooTable = fooDao.GetFooBylD (foolD);
            transaction.Complete ();
            return fooTable;
        }
    }
    // Exception Yönetimi: Exceptionların yakalanması ve loglanması
    catch (Exception ex)
    {
        HandleException (ex);
    }
}
```

Not: AOP' in uygulandığı projelerde bu enine kesen ilgiler (**cross-cutting concern**) policy Injection Application Block konfigürasyon dosyaları ve postsharp' a yığılır

- 2) Transaction olarak System.Transaction namespace' i kullanılmalı **ADO.NET** transaction ve diğer container bazlı transactionlar' ın bağımlılık ve tasarım kısıtları nedeniyle kullanılmaları sakıncalıdır.
- 3) Arayüz katmanında iş mantığı kesinlikle yazılmamalı ve gerekli mantık servis bile enlerine yüklenmelidir.
- 4) Karmaşık bir formdan servis nesnesine veri taşımada bir çok parametre göndermek yerine tüm parametreleri tek bir veri transfer nesnesinde (**DTO**) birle tirerek göndermek daha uygun olmaktadır. Bu şekilde servis bile inine gönderilen parametrelerin artması ya da değişmesi durumunda Business Service metod imzası değişmez ve katmanlar arası kapsüllenme seviyesi artırılmış olur. Bu yöntem yine Web Service kullanımında tercih edilmeli ve tek çağrı ile işlemler halledilmelidir.

// Önce

```
public void ProcessOrder (OrderEntity order, List<OrderDetailEntity>
orderDetails, DeliveryAdressEntity address, CustomerEntity customerInfo)
```

// Veri Transfer Nesnesine Dönüşüm

```
public class OrderDTO
{
    public OrderEntity Order;
    public List<OrderDetailEntity> OrderDetails;
    public DeliveryAdressEntity Address;
    public CustomerEntity Customer;
}
public void ProcessOrder (OrderDTO orderDto)
```

codeVist

Başakşehir Living Lab Kuluçka Merkezi, Başak Mah. Abdülhamithan Cad. No:5 Başakşehir / İSTANBUL
www.codevist.com - info@codevist.com - @codeVist



- 5) Tüm servis nesneleri mutlaka **CodeVist.Framework.Business** namespace' i altında bulunan **BaseService** sınıfından türetilmeli ve **exception** yönetiminin standart olması sağlanmalıdır.
- 6) Entity' ler üzerinde yapılan kontrol ve yeni property türetimleri partial entity sınıflarında yapılmalıdır.
- 7) Tüm entity sınıfları Web Servislerini destekleyecek şekilde [**Serializable**] attribute' ını uygulamalıdır.
- 8) Projenin gereksinimlerine uygun olarak uygulama denetim logları için aşağıdaki standarda uygun loglama mekanizması uygulanmalıdır.
 - a. Aksiyon Tipi bazında kullanıcının yaptığı işlem (INSERT, SELECT, REPORT, PRINT, DELETE, UPDATE...)
 - b. Log Info :
 - i. User: hangi kullanıcı i lem yaptı
 - ii. ActionDate: hangi tarih ve zamanda
 - iii. ActionParameter: hangi parametreler ile bu işlemi yaptı. Bu bölüme çağrılan servis parametreleri ya da işlem parametreleri gelmelidir.
 - iv. Page: hangi form veya sayfada işlem gerçekleşti

Teşekkürler.

codeVist

Başakşehir Living Lab Kuluçka Merkezi, Başak Mah. Abdülhamithan Cad. No:5 Başakşehir / İSTANBUL
www.codevist.com - info@codevist.com - [@codeVist](https://twitter.com/codeVist)