Dernek Takip Sistemi Form Uygulaması

**Hazırlayan:**

Muharrem GEDİK

**İÇİNDEKİLER**

**1. PROGRAM ÇALIŞMA MANTIĞI…………………………. 3**

**2. ACCESS VERİ TABANI……………………………….…….. 4**

**3. SOLUTION…………………………………………………… 8**

**4. PROGRAM KODLARI……………………………………… 9**

**4.1. Veri Tabanı Bağlantı Metni…………………………... 9**

**4.2. Connect Sınıf Miras İlişkisi………………………….. 10**

**4.3. Sorgu Tasarımları…………………………………….. 11**

**4.4. Data Access Layer…………………………………….. 20**

**4.5. Bussiness Layer……………………………………….. 75**

**4.6. Entity Layer…………………………………………… 97**

**4.7. Presentation Layer……………………………………. 99**

**5.PROGRAM GÖRÜNTÜLERİ………………………………... 262**

1. **PROGRAM ÇALIŞMA MANTIĞI**

Dernek Takip projemizde 3 tür kullanıcı girişi düşünülerek tasarlanmıştır.

1-Admin Yetki = 0

2-Çalışan Yetki = 1

3-Müşteri Yetki = 2

Tüm kullanıcılar aynı “LOGİN PAGE” kısmından daha önceden onlara atanan kullanıcı adı ve şifre bilgileri ile sisteme giriş yapabilmektedirler. Projemizde bir de yetkilendirme işlemi gerçekleştirerek giriş yapılırken kullanıcıların giriş bilgilerini sorgulayarak giriş yapan kullanıcıya yönelik Paneller gösterilmektedir.

**Admin** olarak tanımlanmış bir kullanıcı sisteme giriş yaptığı zaman o kullanıcıya gösterilecek olan işlemler sınırlandırılmamış, kısıtlandırılmamış işlemleri içermektedir.

* Çalışan Ekleme-Silme-Düzenleme-Giriş Bilgileri Düzenleme
* Müşteri Ekleme-Silme-Düzenleme-Giriş Bilgileri Düzenleme
* Aidat Belirleme-Odeme Durum Kontrol
* Listeleme işlemleri
* Grafiksel Gösterimler

**Çalışan** olarak tanımlanmış bir kullanıcı sisteme giriş yaptığı zaman o kullanıcıya gösterilecek olan işlemler sınırlandırılmıştır.

* Kendi Bilgilerini Düzenleyebilme (Bu işlem de Tc numarası, cinsiyet, kan grubu hariç olarak tasarlanmıştır.)
* Müşteri Ekleme-Silme-Düzenleme-Giriş Bilgileri Düzenleme
* Aidat Belirleme-Odeme Durum Kontrol
* Listeleme işlemleri

**Müşteri** olarak tanımlanmış bir kullanıcı sisteme giriş yaptığı zaman o kullanıcıya gösterilecek olan işlemler çok kısıtlanmıştır.

* Kendi Bilgilerini Düzenleyebilme (Bu işlem de Tc numarası, cinsiyet, kan grubu hariç olarak tasarlanmıştır.)
* Aidat Bilgilerini görüntüleme
* Borç Ödeme

1. **ACCESS VERİTABANI TABLO YAPILARI VE İLİŞKİLERİ**

Veri tabanımızda toplamda 6 adet tablomuz bulunmaktadır. Bu tablolarımızın her biri birbiri ile iç içe ve organize bir şekilde çalışmak zorunda bırakılmıştır.

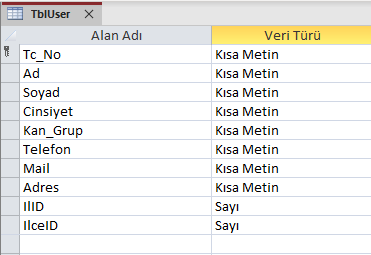
1. **TblUserLogin**



**Şekil 1**

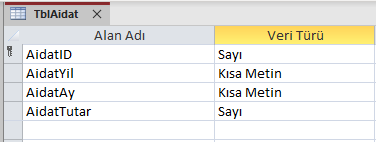
* **kullaniciAd** alanı PRIMARY KEY olarak tanımlanmış olup tüm kullanıcıların kullanıcı adlarının benzersiz olması sağlanmıştır.
* **kullaniciSifre** alanında kullanıcıların giriş şifreleri saklanmaktadır.
* **Auth** alanında kullanıcılara sisteme giriş yaparken kontrol edilebilmesi için yetki verilmesi sağlanmaktadır. Bu alan ile LOGİN kısmı ortak olarak tasarlanmış ve Tüm kullanıcılar LOGIN panelinden kendi yetkilerine göre yönlendirilmiştir.
* **Status** alanında kullanıcıların Aktif/Pasif üye durum bilgileri saklanmakta ve istenirse statü değerleri değiştirilebilmektedir. LOGIN ekranında yetki alanı ile aynı zamanda status alanıda kontrol edilmekte ve Pasif olarak tanımlanmış bir kullanıcının girişine izin verilmemektedir.
* **Tc\_No** alan adı ile TblUser Tablosunda FOREIGN KEY yapısı kullanılarak ilişki kurulmuştur. Bu sayede tablolar birbirine bağlanarak JOIN kullanılarak farklı tablolardan veri çekip ekleme işlemi yapılabilir.

1. **TblUser**



**Şekil 2**

* **Tc\_No** alanı PRIMARY KEY olarak tanımlanmıştır. Hedef kullanıcılarımızın hepsinin devlet tarafından verilmiş benzersiz Tc Kimlik numaraları bulunduğu için tüm sistemimizde Tc\_No alanı ilişkilerde FOREIGN KEY olarak kullanılmıştır.
* **Ad,** **Soyad**, **Cinsiyet**, **Kan\_Grup**, **Telefon**, **Mail**, **Adres** alanlarında kullanıcıların kişisel bilgilerini tutuyoruz.
* **IlID** ve **IlceID** alanlarında Il ve IlIlce tablolarımızdaki PRIMARYKEY değerlerinin FOREIGN KEY birleştirmesini yaparak bu tablomuzu da o tablolar ile birleştirmiş oluyoruz.

1. **TblAidat**

**Şekil 3**

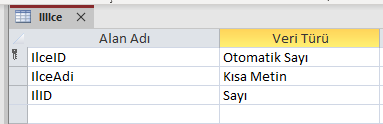
* **AidatID** alanı PRIMARY KEY olarak tanımlanmıştır ve aidatların benzersiz olmasını sağlamaktadır.
* **AidatYil**, **AidatAy** ve **AidatTutar** alanlarında aidatların bilgilerini saklamaktayız.

1. **TblOdemeler**

**Şekil 4**

* **OdemeID** PRIMARY KEY olarak tanımlanmıştır ve Otomatik artan şekilde veri türü belirlenmiştir.
* **Tc\_No** ve **AidatID** değerleri TblUser ve TblAidat tablolarının PRIMARY KEY değerlerinin FOREIGN Key yapısı olarak gelmiştir ve aralarındaki birleşim ilişkisini sağlamaktadır.
* **OdemeDurum** alanı ile kullanıcıların ödeme yapıp yapmadığını TRUE / FALSE olarak belirtebilmekteyiz.

1. **IlIlce**

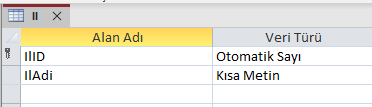


**Şekil 5**

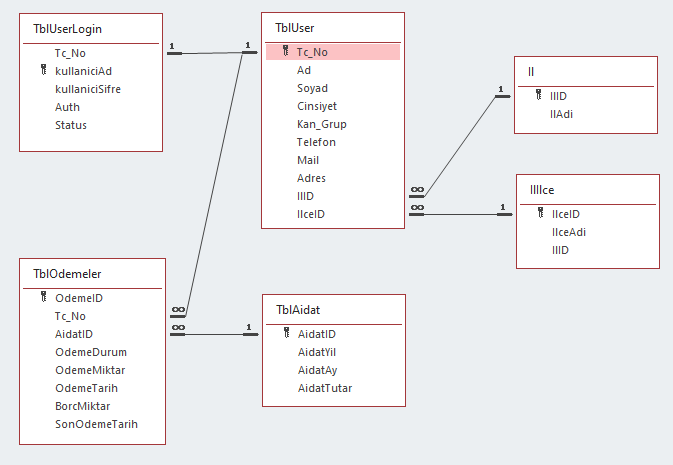
* İlçe bilgilerinin tutulduğu bu tablomuzda **IlceID** alanı PRIMARY KEY olarak tanımlanarak ilçelerin benzersiz olmaması sağlanmış oldu.
* **IlID** alanı ile de Il tablosundan FOREIGN KEY yapısı kullanılarak ilişki kurulması sağlanmıştır.

1. **Il**

**Şekil 6**



* **Il** tablomuzda **IlID** alanı ile PRIMARY KEY olarak illerimizin birbirine herhangi bir şekilde tekrarlama gibi ihtimallerini ortadan kaldırmış olduk ve IlAdi alanı ile de isimlerini saklamaktayız.

**İLİŞKİLER**

TblUser 🡨🡪 TblUserLogin **1-1 ilişkisi**

TblUser 🡨🡪 TblOdemeler **1-n ilişkisi**

TblUser 🡨🡪 Il **n-1 ilişkisi**

TblUser 🡨🡪 IlIlce **n-1 ilişkisi**

TblOdemeler 🡨🡪 TblAidat **n-1 ilişkisi**

1. **SOLUTION**

**metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Şekil 7**

1. **PROGRAM KODLARI**
   1. **DbConnect Class** 🡪 **BAĞLANTI METNİ**

using System.Data.OleDb;

namespace DataAccessLayer.DbConnections

{

public class DbConnect

{

public OleDbConnection baglanti;

public DbConnect()

{

baglanti = new OleDbConnection(@"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=DernekTakipDataBase.accdb;");

}

}

}

Bu C# kodu, OleDbConnection sınıfını kullanarak bir veritabanına bağlanmayı sağlar. Özellikle, Microsoft Access veritabanı için bir bağlantı oluşturan bir sınıf içerir. İşte bu kodun ayrıntılı açıklaması:

1. **using System.Data.OleDb;**: Bu satır, OleDbConnection sınıfını kullanabilmek için gerekli olan OleDb namespace'ini içeri alır.
2. **NamespaceDataAccessLayer.DbConnections**: Bu satır, **DataAccessLayer.DbConnections** adlı bir namespace oluşturur. Bu, kodun başka bir dosyada kullanılması durumunda bu namespace içindeki sınıfların kullanılmasını sağlar.
3. **public class DbConnect**: **DbConnect** adlı bir genel (public) sınıf tanımlanır. Bu sınıf, veritabanı bağlantısı oluşturmak için gerekli olan özellikleri içerir.
4. **public OleDbConnection baglanti;**: **baglanti** adında bir **OleDbConnection** tipinde genel bir değişken (field) tanımlanır. Bu değişken, veritabanına bağlantıyı temsil eder.
5. **public DbConnect()**: Sınıfın kurucu metodu (constructor) tanımlanır. Bu metod, sınıfın bir örneği oluşturulduğunda otomatik olarak çağrılır.
6. **Baglanti = new OleDbConnection(@"Provider=Microsoft.ACE.OLEDB.12.0; Data Source=DernekTakipDataBase.accdb;");**: Bu satırda, OleDbConnection örneği oluşturulur ve **baglanti** değişkenine atanır. Bağlantı dizesi (**Data Source=DernekTakipDataBase.accdb;**) ile Microsoft Access veritabanına bağlanmayı sağlar. Bu dize, veritabanının bulunduğu konumu ve dosya adını içerir.

Bu kod, genel bir veritabanı bağlantısı oluşturmak için kullanılabilir. Ancak, genellikle daha karmaşık uygulamalarda bu tür bağlantıları yönetmek için daha fazla katman ve yöntem eklenir. Örneğin, try-catch blokları ekleyerek hata yönetimi yapılabilir ve bağlantı açma ve kapatma işlemleri daha güvenli bir şekilde gerçekleştirilebilir.

* 1. **DbQuery Class** 🡪 **BAĞLANTI İÇİN GEREKLİ CONSTRUCTOR VE MİRAS**

using System;

using System.Data;

using System.Data.OleDb;

namespace DataAccessLayer.DbConnections

{

public class DbQuery : DbConnect

{

private OleDbDataAdapter dataAdapter;

public DbQuery()

{

dataAdapter = new OleDbDataAdapter();

}

private OleDbConnection openConnection()

{

if (baglanti.State == ConnectionState.Closed || baglanti.State == ConnectionState.Broken)

{

baglanti.Open();

}

return baglanti;

}

Bu C# kodu, OleDbConnection sınıfını ve bağlantı üzerinden veritabanı sorguları yapmak için OleDbDataAdapter sınıfını kullanarak bir sınıf olan **DbQuery**'i içerir. İşte bu kodun ayrıntılı açıklaması:

1. **using System;**: Genel amaçlı .NET kütüphanesini içeri alır.
2. **using System.Data;**: .NET Framework tarafından sağlanan veri işleme sınıflarını içeri alır.
3. **using System.Data.OleDb;**: OleDb bağlantı nesnesi ve diğer ilgili sınıfları içeren OleDb namespace'ini içeri alır.
4. **namespace DataAccessLayer.DbConnections**: Bu satır, **DataAccessLayer.DbConnections** adlı bir namespace oluşturur. Bu namespace içinde **DbQuery** sınıfı yer alır.
5. **public class DbQuery : DbConnect**: **DbQuery** sınıfı, **DbConnect** sınıfından miras alır. Bu, **DbQuery** sınıfının **DbConnect** sınıfındaki **OleDbConnection** nesnesini kullanabilmesini sağlar.
6. **private OleDbDataAdapter dataAdapter;**: **dataAdapter** adında özel bir **OleDbDataAdapter** tipinde alan (field) tanımlanır. Bu nesne, veritabanı ile DataTable arasında veri aktarımını yönetir.
7. **public DbQuery()**: **DbQuery** sınıfının kurucu metodu, **dataAdapter** nesnesini başlatır.
8. **private OleDbConnection openConnection()**: **openConnection** adlı özel bir metod, bağlantının kapalı veya bozuk durumdaysa açar ve bağlantı nesnesini döndürür.
   1. **DbQuery Class 🡪 SORGU İFADELERİ**
      1. **SelectQuery1**

public DataTable SelectQuery1(String \_query, OleDbParameter[] oleDbParameters)

{

OleDbCommand mySelectCommand = new OleDbCommand();

DataTable dataTable = new DataTable();

DataSet ds = new DataSet();

dataTable = null;

try

{

mySelectCommand.Connection = openConnection();

mySelectCommand.CommandText = \_query;

mySelectCommand.Parameters.AddRange(oleDbParameters);

dataAdapter.SelectCommand = mySelectCommand;

dataAdapter.Fill(ds);

dataTable = ds.Tables[0];

}

catch (OleDbException e)

{

Console.WriteLine("Hata Kodu: " + e.ErrorCode);

Console.WriteLine("Hata Mesajı: " + e.Message);

Console.WriteLine("Hata Stack Trace: " + e.StackTrace);

foreach (OleDbError error in e.Errors)

{

Console.WriteLine("Hata: " + error.Message);

}

return null;

}

finally

{

baglanti.Close();

}

return dataTable;

}

Bu C# kodu, bir OleDb veritabanı bağlantısı kullanarak bir SQL sorgusunu çalıştırmak ve sonucunu bir DataTable nesnesi olarak döndürmek için tasarlanmış gibi görünüyor. Kodun genel mantığı şu adımları içerir:

1. **OleDbCommand ve DataTable Oluşturma:**
   * **OleDbCommand** sınıfından bir **mySelectCommand** nesnesi oluşturuluyor.
   * **DataTable** sınıfından bir **dataTable** nesnesi oluşturuluyor.
   * **DataSet** sınıfından bir **ds** nesnesi oluşturuluyor.
2. **Bağlantı Açma:**
   * **openConnection** metodu çağrılarak bir OleDb bağlantısı elde ediliyor.
   * **mySelectCommand.Connection** özelliği, oluşturulan bağlantıya ayarlanıyor.
3. **Sorgu ve Parametre Ayarlama:**
   * **\_query** parametresi, **mySelectCommand.CommandText** özelliği üzerinden sorgu olarak atanıyor.
   * **\_query** ile gelen parametreler (**oleDbParameters**), **mySelectCommand.Parameters.AddRange(oleDbParameters)** yöntemi ile komuta ekleniyor.
4. **DataAdapter ve DataSet Kullanma:**
   * **dataAdapter** nesnesi (bu kod içinde tanımlanmamış gibi görünüyor, muhtemelen başka bir yerde tanımlanmış) kullanılarak **dataAdapter.SelectCommand** özelliği **mySelectCommand** ile ayarlanıyor.
   * **dataAdapter.Fill(ds)** yöntemi ile sorgu sonucu bir **DataSet** elde ediliyor.
5. **DataTable Doldurma:**
   * **dataTable** değişkeni, **ds.Tables[0]** ifadesi ile elde edilen ilk tablo ile dolduruluyor.
6. **Hata Kontrolü:**
   * **OleDbException** türünden hatalar ele alınıyor.
   * Hata mesajları ve detayları **Console.WriteLine** kullanılarak ekrana yazdırılıyor.
   * Hata durumunda **null** döndürülerek işlem sonlandırılıyor.
7. **Bağlantı Kapatma:**
   * **finally** bloğu içinde, bağlantı kapatılıyor (**baglanti.Close()**).
8. **Sonuç Döndürme:**
   * Eğer her şey başarılı ise, doldurulan **dataTable** nesnesi geri döndürülüyor.
     1. **SelectQuery2**

public DataTable SelectQuery2(String \_query)

{

OleDbCommand mySelectCommand = new OleDbCommand();

DataTable dataTable = new DataTable();

DataSet ds = new DataSet();

dataTable = null;

try

{

mySelectCommand.Connection = openConnection();

mySelectCommand.CommandText = \_query;

dataAdapter.SelectCommand = null;

dataAdapter.SelectCommand = mySelectCommand;

dataAdapter.Fill(ds);

dataTable = ds.Tables[0];

}

catch (OleDbException e)

{

Console.WriteLine("Hata Kodu: " + e.ErrorCode);

Console.WriteLine("Hata Mesajı: " + e.Message);

Console.WriteLine("Hata Stack Trace: " + e.StackTrace);

foreach (OleDbError error in e.Errors)

{

Console.WriteLine("Hata: " + error.Message);

}

return null;

}

finally

{

baglanti.Close();

}

return dataTable;

}

Bu C# kodu, bir SQL sorgusunu çalıştırmak için kullanılan **SelectQuery2** adlı bir metodu içerir. Bu metot, veritabanına bağlanmak ve bir SELECT sorgusunu yürütmek için **OleDbConnection**, **OleDbCommand**, **OleDbDataAdapter**, **DataSet** ve **DataTable** sınıflarını kullanır. İşte bu kodun ayrıntılı açıklaması:

1. **public DataTable SelectQuery2(String \_query)**: Veritabanında bir SELECT sorgusunu çalıştırmak için kullanılan genel bir metottur. **\_query** parametresi ile gelen SQL sorgusu kullanılarak işlem yapılır.
2. **OleDbCommand mySelectCommand = new OleDbCommand();**: Bir **OleDbCommand** nesnesi oluşturulur. Bu nesne, SQL sorgusunu ve bağlantı bilgilerini içerir.
3. **DataTable dataTable = new DataTable();**: Bir **DataTable** nesnesi oluşturulur. Bu nesne, sorgu sonuçlarını saklamak için kullanılır.
4. **DataSet ds = new DataSet();**: Bir **DataSet** nesnesi oluşturulur. Bu nesne, verilerin tablo halinde tutulmasını sağlar.
5. **dataTable = null;**: **dataTable** nesnesi null olarak başlatılır.
6. **mySelectCommand.Connection = openConnection();**: Bağlantıyı açan **openConnection** metodunu kullanarak **mySelectCommand** nesnesinin bağlantısını tanımlar.
7. **mySelectCommand.CommandText = \_query;**: SQL sorgusunu belirtilen **\_query** parametresine atar.
8. **dataAdapter.SelectCommand = null;**: **dataAdapter** nesnesinin **SelectCommand** özelliğini null olarak başlatır.
9. **dataAdapter.SelectCommand = mySelectCommand;**: **mySelectCommand** nesnesini **dataAdapter**'ın **SelectCommand** özelliğine atar. Bu, sorgunun **DataSet** üzerine uygulanabilmesi için gerekli adımdır.
10. **dataAdapter.Fill(ds);**: **dataAdapter** nesnesini kullanarak sorguyu çalıştırır ve sonuçları **DataSet**'e doldurur.
11. **dataTable = ds.Tables[0];**: **ds** içindeki ilk tabloyu (**Tables[0]**) **dataTable**'a atar. Bu, sorgu sonuçlarını içeren **DataTable** nesnesini elde eder.
12. **catch (OleDbException e)**: Herhangi bir **OleDbException** hatası durumunda bu blok çalışır. Hata ayrıntıları ekrana yazdırılır ve null değeri döndürülür.
13. **finally**: Bağlantı kapatma işlemi **finally** bloğunda gerçekleşir. Bu, bağlantının ne olursa olsun kapatılmasını sağlar.
14. **return dataTable;**: Metot, sorgu sonuçlarını içeren **dataTable** nesnesini döndürür.

Bu kod, basit bir SELECT sorgusunu çalıştırmak için kullanılabilir. Ancak, gerçek uygulamalarda daha karmaşık senaryolara ve daha güvenli bağlantı yönetimine ihtiyaç duyulabilir.

* + 1. **InsertQuery1**

public bool InsertQuery1(String \_query, OleDbParameter[] oleDbParameters)

{

OleDbCommand myCommand = new OleDbCommand();

try

{

myCommand.Connection = openConnection();

myCommand.CommandText = \_query;

myCommand.Parameters.AddRange(oleDbParameters);

dataAdapter.InsertCommand = myCommand;

myCommand.ExecuteNonQuery();

}

catch (OleDbException e)

{

Console.WriteLine("Hata Kodu: " + e.ErrorCode);

Console.WriteLine("Hata Mesajı: " + e.Message);

Console.WriteLine("Hata Stack Trace: " + e.StackTrace);

foreach (OleDbError error in e.Errors)

{

Console.WriteLine("Hata: " + error.Message);

}

return false;

}

finally

{

baglanti.Close();

}

return true;

}

Bu C# kodu, bir veritabanına INSERT sorgusu göndermek için kullanılan **InsertQuery1** adlı bir metodu içerir. Bu metot, **OleDbCommand** sınıfını ve parametreleri kullanarak bir sorguyu veritabanına gönderir. İşte bu kodun ayrıntılı açıklaması:

1. **public bool InsertQuery1(String \_query, OleDbParameter[] oleDbParameters)**: Veritabanına bir INSERT sorgusunu çalıştırmak için kullanılan genel bir metottur. **\_query** parametresi ile gelen SQL sorgusu ve **oleDbParameters** ile gelen parametreler kullanılarak işlem yapılır.
2. **OleDbCommand myCommand = new OleDbCommand();**: Bir **OleDbCommand** nesnesi oluşturulur. Bu nesne, SQL sorgusunu ve bağlantı bilgilerini içerir.
3. **try**: Sorgunun çalıştırılması için try bloğu başlar.
   * **myCommand.Connection = openConnection();**: Bağlantıyı açan **openConnection** metodunu kullanarak **myCommand** nesnesinin bağlantısını tanımlar.
   * **myCommand.CommandText = \_query;**: SQL sorgusunu belirtilen **\_query** parametresine atar.
   * **myCommand.Parameters.AddRange(oleDbParameters);**: Parametreleri ekler. Bu, özellikle parametrelerin SQL sorgusuna güvenli bir şekilde eklenmesini sağlar.
   * **dataAdapter.InsertCommand = myCommand;**: **dataAdapter** nesnesinin **InsertCommand** özelliğine **myCommand**'ı atar. Bu, sorgunun **DataSet** üzerine uygulanabilmesi için gerekli adımdır.
   * **myCommand.ExecuteNonQuery();**: INSERT sorgusunu veritabanına gönderir ve etkilenen satır sayısını döndürmez. (Bu metodun kullanılmasının sebebi, genellikle INSERT, UPDATE veya DELETE sorgularında kullanılır ve geriye dönüş değeri olarak etkilenen satır sayısını almak mümkündür.)
4. **catch (OleDbException e)**: Herhangi bir **OleDbException** hatası durumunda bu blok çalışır. Hata ayrıntıları ekrana yazdırılır ve false değeri döndürülür.
5. **finally**: Bağlantı kapatma işlemi **finally** bloğunda gerçekleşir. Bu, bağlantının ne olursa olsun kapatılmasını sağlar.
6. **return true;**: INSERT sorgusu başarıyla çalıştırıldıysa true değeri döndürülür.

Bu kod, basit bir INSERT sorgusunu çalıştırmak için kullanılabilir. Ancak, gerçek uygulamalarda daha karmaşık senaryolara ve daha güvenli bağlantı yönetimine ihtiyaç duyulabilir

* + 1. **UpdateQuery1**

public bool UpdateQuery1(String \_query, OleDbParameter[] oleDbParameters)

{

OleDbCommand myCommand = new OleDbCommand();

try

{

myCommand.Connection = openConnection();

myCommand.CommandText = \_query;

myCommand.Parameters.AddRange(oleDbParameters);

dataAdapter.UpdateCommand = myCommand;

myCommand.ExecuteNonQuery();

}

catch (OleDbException e)

{

Console.WriteLine("Hata Kodu: " + e.ErrorCode);

Console.WriteLine("Hata Mesajı: " + e.Message);

Console.WriteLine("Hata Stack Trace: " + e.StackTrace);

foreach (OleDbError error in e.Errors)

{

Console.WriteLine("Hata: " + error.Message);

}

return false;

}

finally

{

baglanti.Close();

}

return true;

}

Bu C# kodu, OleDb veritabanı bağlantısı kullanarak bir güncelleme sorgusunu çalıştırmak ve işlemin başarılı olup olmadığını bir bool değeri olarak döndürmek için tasarlanmış gibi görünüyor. İşte kodun genel mantığı:

1. **OleDbCommand Oluşturma:**
   * **OleDbCommand** sınıfından bir **myCommand** nesnesi oluşturuluyor.
2. **Bağlantı Açma:**
   * **openConnection** fonksiyonu kullanılarak bir OleDb bağlantısı elde ediliyor.
   * **myCommand.Connection** özelliği, oluşturulan bağlantıya ayarlanıyor.
3. **Sorgu ve Parametre Ayarlama:**
   * **\_query** parametresi, **myCommand.CommandText** özelliği üzerinden sorgu olarak atanıyor.
   * **\_query** ile gelen parametreler (**oleDbParameters**), **myCommand.Parameters.AddRange(oleDbParameters)** yöntemi ile komuta ekleniyor.
4. **DataAdapter ve UpdateCommand Kullanma:**
   * **dataAdapter** nesnesi kullanılarak **dataAdapter.UpdateCommand** özelliği **myCommand** ile ayarlanıyor.
5. **Sorgu Çalıştırma:**
   * **myCommand.ExecuteNonQuery()** yöntemi ile güncelleme sorgusu veritabanında çalıştırılıyor.
6. **Hata Kontrolü:**
   * **OleDbException** türünden hatalar ele alınıyor.
   * Hata mesajları ve detayları **Console.WriteLine** kullanılarak ekrana yazdırılıyor.
   * Hata durumunda **false** döndürülerek işlem başarısız olarak işaretleniyor.
7. **Bağlantı Kapatma:**
   * **finally** bloğu içinde, bağlantı kapatılıyor (**baglanti.Close()**).
8. **Sonuç Döndürme:**
   * Eğer her şey başarılı ise, **true** döndürülerek işlem başarılı olarak işaretleniyor.
     1. **DeleteQuery1**

public bool DeleteQuery1(String \_query, OleDbParameter[] oleDbParameters)

{

OleDbCommand myCommand = new OleDbCommand();

try

{

myCommand.Connection = openConnection();

myCommand.CommandText = \_query;

myCommand.Parameters.AddRange(oleDbParameters);

dataAdapter.DeleteCommand = myCommand;

myCommand.ExecuteNonQuery();

}

catch (OleDbException e)

{

Console.WriteLine("Hata Kodu: " + e.ErrorCode);

Console.WriteLine("Hata Mesajı: " + e.Message);

Console.WriteLine("Hata Stack Trace: " + e.StackTrace);

foreach (OleDbError error in e.Errors)

{

Console.WriteLine("Hata: " + error.Message);

}

return false;

}

finally

{

baglanti.Close();

}

return true;

}

Bu C# kodu, OleDb veritabanı bağlantısı kullanarak bir silme (delete) sorgusunu çalıştırmak ve işlemin başarılı olup olmadığını bir bool değeri olarak döndürmek için tasarlanmış gibi görünüyor. İşte kodun genel mantığı:

1. **OleDbCommand Oluşturma:**
   * **OleDbCommand** sınıfından bir **myCommand** nesnesi oluşturuluyor.
2. **Bağlantı Açma:**
   * **openConnection** fonksiyonu kullanılarak bir OleDb bağlantısı elde ediliyor.
   * **myCommand.Connection** özelliği, oluşturulan bağlantıya ayarlanıyor.
3. **Sorgu ve Parametre Ayarlama:**
   * **\_query** parametresi, **myCommand.CommandText** özelliği üzerinden sorgu olarak atanıyor.
   * **\_query** ile gelen parametreler (**oleDbParameters**), **myCommand.Parameters.AddRange(oleDbParameters)** yöntemi ile komuta ekleniyor.
4. **DataAdapter ve DeleteCommand Kullanma:**
   * **dataAdapter** nesnesi kullanılarak **dataAdapter.DeleteCommand** özelliği **myCommand** ile ayarlanıyor.
5. **Sorgu Çalıştırma:**
   * **myCommand.ExecuteNonQuery()** yöntemi ile silme sorgusu veritabanında çalıştırılıyor.
6. **Hata Kontrolü:**
   * **OleDbException** türünden hatalar ele alınıyor.
   * Hata mesajları ve detayları **Console.WriteLine** kullanılarak ekrana yazdırılıyor.
   * Hata durumunda **false** döndürülerek işlem başarısız olarak işaretleniyor.
7. **Bağlantı Kapatma:**
   * **finally** bloğu içinde, bağlantı kapatılıyor (**baglanti.Close()**).
8. **Sonuç Döndürme:**
   * Eğer her şey başarılı ise, **true** döndürülerek işlem başarılı olarak işaretleniyor.
   1. **Data Access Layer**
      1. **DalAidat**

using DataAccessLayer.DbConnections;

using System.Data.OleDb;

using System.Data;

using System;

namespace DataAccessLayer

{

public class DalAidat

{

private DbQuery query;

public DalAidat()

{

query = new DbQuery();

}

* + - 1. **Aidat Tablo , Aidat Ekle , Aidat Sil , Aidat Guncelle Metodları**

public DataTable AidatTablo()

{

string sorgu = "SELECT DISTINCT TblAidat.AidatID, TblAidat.AidatYil, TblAidat.AidatAy, TblAidat.AidatTutar, TblOdemeler.SonOdemeTarih FROM TblAidat LEFT JOIN TblOdemeler ON TblAidat.AidatID = TblOdemeler.AidatID ORDER BY TblAidat.AidatID\r\n";

return query.SelectQuery2(sorgu);

}

public bool AidatEkle(int AidatID, string AidatYil, string AidatAy, int AidatTutar)

{

string sorgu = string.Format("INSERT INTO TblAidat (AidatID,AidatYil,AidatAy,AidatTutar) VALUES (@AidatID,@AidatYil,@AidatAy,@AidatTutar)");

OleDbParameter[] oleParameter = new OleDbParameter[4];

oleParameter[0] = new OleDbParameter("@AidatID", OleDbType.Integer);

oleParameter[1] = new OleDbParameter("@AidatYil", OleDbType.WChar);

oleParameter[2] = new OleDbParameter("@AidatAy", OleDbType.WChar);

oleParameter[3] = new OleDbParameter("@AidatTutar", OleDbType.Integer);

oleParameter[0].Value = Convert.ToString(AidatID);

oleParameter[1].Value = Convert.ToString(AidatYil);

oleParameter[2].Value = Convert.ToString(AidatAy);

oleParameter[3].Value = Convert.ToString(AidatTutar);

return query.InsertQuery1(sorgu, oleParameter);

}

public bool AidatSil(int AidatID)

{

string sorgu = string.Format("DELETE FROM TblAidat WHERE AidatID = @AidatID");

OleDbParameter[] oleParameter = new OleDbParameter[1];

oleParameter[0] = new OleDbParameter("@AidatID", OleDbType.Integer);

oleParameter[0].Value = Convert.ToString(AidatID);

return query.DeleteQuery1(sorgu, oleParameter);

}

public bool AidatGuncelle(int AidatID, string AidatYil, string AidatAy, int AidatTutar)

{

string sorgu = string.Format("UPDATE TblAidat SET AidatID = @AidatID, AidatYil = @AidatYil, AidatAy = @AidatAy, AidatTutar = @AidatTutar WHERE AidatID = @AidatID");

OleDbParameter[] oleParameter = new OleDbParameter[4];

oleParameter[0] = new OleDbParameter("@AidatID", OleDbType.Integer);

oleParameter[1] = new OleDbParameter("@AidatYil", OleDbType.WChar);

oleParameter[2] = new OleDbParameter("@AidatAy", OleDbType.WChar);

oleParameter[3] = new OleDbParameter("@AidatTutar", OleDbType.Integer);

oleParameter[0].Value = Convert.ToString(AidatID);

oleParameter[1].Value = Convert.ToString(AidatYil);

oleParameter[2].Value = Convert.ToString(AidatAy);

oleParameter[3].Value = Convert.ToString(AidatTutar);

return query.UpdateQuery1(sorgu, oleParameter);

}

**Aidat Tablo**

Bu C# kodu, bir veritabanından belirli sorgu koşullarına göre veri çekmek için kullanılır gibi görünmektedir. Kod, bir **DataTable** nesnesi döndüren **AidatTablo** adlı bir metod içermektedir. **DataTable**, veritabanından çekilen tablo benzeri veri kümesini temsil eden bir .NET Framework sınıfıdır.

İşte kodun genel mantığı:

1. **sorgu** adlı bir SQL sorgusu oluşturulmuştur. Bu sorgu, **TblAidat** ve **TblOdemeler** tablolarını birleştirerek belirli alanları seçer. **LEFT JOIN** kullanılarak **TblAidat.AidatID** ve **TblOdemeler.AidatID** alanları arasında birleştirme gerçekleştirilmiştir. **DISTINCT** kullanılarak tekrarlanan satırların önlenmesi sağlanmıştır. Son olarak **ORDER BY TblAidat.AidatID** ifadesi ile sonuçların **AidatID**'ye göre sıralanması istenmiştir.
2. Oluşturulan SQL sorgusu, **query** adlı bir nesnenin **SelectQuery2** metoduna iletilmiştir. Bu metodun geri dönüş değeri, veritabanından çekilen verileri içeren bir **DataTable** nesnesidir.

**Aidat Ekle**

Bu C# kodu, bir veritabanına yeni bir kayıt eklemek için kullanılan bir metodun içeriğini içerir. Aşağıda kodun metinsel açıklaması bulunmaktadır:

1. **AidatEkle** adlı bir metod tanımlanmıştır.
2. Metodun giriş parametreleri şunlardır:
   * **AidatID**: Eklenmek istenen aidatın kimlik numarası (int).
   * **AidatYil**: Eklenmek istenen aidatın yılı (string).
   * **AidatAy**: Eklenmek istenen aidatın ayı (string).
   * **AidatTutar**: Eklenmek istenen aidatın tutarı (int).
3. **string.Format** metodu kullanılarak SQL **INSERT** sorgusu oluşturulmuştur. Bu sorgu, **TblAidat** tablosuna yeni bir kayıt eklemek için kullanılır. Sorguda parametreler (**@AidatID**, **@AidatYil**, **@AidatAy**, **@AidatTutar**) kullanılarak dinamik bir sorgu oluşturulmuştur.
4. **OleDbParameter** tipinde bir dizi (**oleParameter**) oluşturulmuştur. Bu dizi, SQL sorgusundaki parametrelerin türünü ve değerini belirtmek için kullanılır.
5. Oluşturulan parametreler (**oleParameter**) **Convert.ToString** metodunu kullanarak, metodun giriş parametrelerinden alınan değerlere atanmıştır.
6. **query.InsertQuery1** metodu çağrılmıştır. Bu metodun parametreleri, oluşturulan SQL sorgusu ve parametreleri içerir. Bu metodun geri dönüş değeri, SQL sorgusunun başarıyla çalışıp çalışmadığını belirten bir **bool** değeridir.
7. **AidatEkle** metodu sonucunda, yeni aidat kaydının başarıyla eklenip eklenmediği bilgisini döndürür.

**AidatSil**

Bu C# kodu, belirli bir **AidatID** değerine sahip olan kaydı veritabanından silmek için kullanılan bir metodu içermektedir. Aşağıda, bu kodun metinsel açıklaması bulunmaktadır:

1. **AidatSil** adlı bir metod tanımlanmıştır.
2. Metodun giriş parametresi şunlardır:
   * **AidatID**: Silinmek istenen aidatın kimlik numarası (int).
3. **string.Format** metodu kullanılarak SQL **DELETE** sorgusu oluşturulmuştur. Bu sorgu, **TblAidat** tablosundan belirli bir **AidatID**'ye sahip kaydı silmek için kullanılır. Sorguda **@AidatID** parametresi kullanılarak dinamik bir sorgu oluşturulmuştur.
4. **OleDbParameter** tipinde bir dizi (**oleParameter**) oluşturulmuştur. Bu dizi, SQL sorgusundaki parametrenin türünü ve değerini belirtmek için kullanılır.
5. Oluşturulan parametre (**oleParameter**), **Convert.ToString** metodunu kullanarak, metodun giriş parametresinden alınan **AidatID** değerine atanmıştır.
6. **query.DeleteQuery1** metodu çağrılmıştır. Bu metodun parametreleri, oluşturulan SQL sorgusu ve parametreleri içerir. Bu metodun geri dönüş değeri, SQL sorgusunun başarıyla çalışıp çalışmadığını belirten bir **bool** değeridir.
7. **AidatSil** metodu sonucunda, belirtilen **AidatID**'ye sahip kaydın başarıyla silinip silinmediği bilgisini döndürür.

**Aidat Guncelle**

Bu C# kodu, belirli bir **AidatID** değerine sahip olan kaydın verilerini güncellemek için kullanılan bir metodu içermektedir. Aşağıda, bu kodun metinsel açıklaması bulunmaktadır:

1. **AidatGuncelle** adlı bir metod tanımlanmıştır.
2. Metodun giriş parametreleri şunlardır:
   * **AidatID**: Güncellenmek istenen aidatın kimlik numarası (int).
   * **AidatYil**: Güncellenmek istenen aidatın yılı (string).
   * **AidatAy**: Güncellenmek istenen aidatın ayı (string).
   * **AidatTutar**: Güncellenmek istenen aidatın tutarı (int).
3. **string.Format** metodu kullanılarak SQL **UPDATE** sorgusu oluşturulmuştur. Bu sorgu, **TblAidat** tablosundan belirli bir **AidatID**'ye sahip kaydın verilerini güncellemek için kullanılır. Sorguda **@AidatID**, **@AidatYil**, **@AidatAy**, ve **@AidatTutar** parametreleri kullanılarak dinamik bir sorgu oluşturulmuştur.
4. **OleDbParameter** tipinde bir dizi (**oleParameter**) oluşturulmuştur. Bu dizi, SQL sorgusundaki parametrelerin türünü ve değerini belirtmek için kullanılır.
5. Oluşturulan parametreler (**oleParameter**), **Convert.ToString** metodunu kullanarak, metodun giriş parametrelerinden alınan değerlere atanmıştır.
6. **query.UpdateQuery1** metodu çağrılmıştır. Bu metodun parametreleri, oluşturulan SQL sorgusu ve parametreleri içerir. Bu metodun geri dönüş değeri, SQL sorgusunun başarıyla çalışıp çalışmadığını belirten bir **bool** değeridir.
7. **AidatGuncelle** metodu sonucunda, belirtilen **AidatID**'ye sahip kaydın başarıyla güncellenip güncellenmediği bilgisini döndürür.
   * + 1. **AidatOdeme1, AidatOdeme2, AidatOdeme3, AidatOdeme4, AidatOdeme5, AidatOdeme6 Metodları**

public DataTable AidatOdeme1(string Tc\_No)

{

string sorgu = string.Format("SELECT TblUser.Tc\_No, TblUser.Ad, TblUser.Soyad," +

"TblAidat.AidatAy,TblAidat.AidatYil," +

"TblOdemeler.OdemeDurum,TblOdemeler.BorcMiktar,TblOdemeler.SonOdemeTarih,TblOdemeler.OdemeTarih FROM (TblUser INNER JOIN TblOdemeler ON " +

"TblUser.Tc\_No = TblOdemeler.Tc\_No) INNER JOIN TblAidat ON " +

"TblOdemeler.AidatID = TblAidat.AidatID WHERE TblUser.Tc\_No = @Tc\_No");

OleDbParameter[] oleParameter = new OleDbParameter[1];

oleParameter[0] = new OleDbParameter("@Tc\_No", OleDbType.WChar);

oleParameter[0].Value = Convert.ToString(Tc\_No);

return query.SelectQuery1(sorgu, oleParameter);

}

public DataTable AidatOdeme2(string Tc\_No,string yil)

{

string sorgu = string.Format("SELECT TblUser.Tc\_No, TblUser.Ad, TblUser.Soyad," +

"TblAidat.AidatAy,TblAidat.AidatYil," +

"TblOdemeler.OdemeDurum,TblOdemeler.BorcMiktar,TblOdemeler.SonOdemeTarih, TblOdemeler.OdemeTarih FROM (TblUser INNER JOIN TblOdemeler ON " +

"TblUser.Tc\_No = TblOdemeler.Tc\_No) INNER JOIN TblAidat ON " +

"TblOdemeler.AidatID = TblAidat.AidatID WHERE TblUser.Tc\_No = @Tc\_No AND TblAidat.AidatYil =@AidatYil");

OleDbParameter[] oleParameter = new OleDbParameter[2];

oleParameter[0] = new OleDbParameter("@Tc\_No", OleDbType.WChar);

oleParameter[1] = new OleDbParameter("@AidatYil", OleDbType.WChar);

oleParameter[0].Value = Convert.ToString(Tc\_No);

oleParameter[1].Value = Convert.ToString(yil);

return query.SelectQuery1(sorgu, oleParameter);

}

public DataTable AidatOdeme3(string Tc\_No, string yil, string ay)

{

string sorgu = string.Format("SELECT TblUser.Tc\_No, TblUser.Ad, TblUser.Soyad," +

"TblAidat.AidatAy,TblAidat.AidatYil," +

"TblOdemeler.OdemeDurum,TblOdemeler.BorcMiktar,TblOdemeler.SonOdemeTarih, TblOdemeler.OdemeTarih FROM (TblUser INNER JOIN TblOdemeler ON " +

"TblUser.Tc\_No = TblOdemeler.Tc\_No) INNER JOIN TblAidat ON " +

"TblOdemeler.AidatID = TblAidat.AidatID WHERE TblUser.Tc\_No = @Tc\_No AND TblAidat.AidatYil =@AidatYil AND TblAidat.AidatAy=@AidatAy");

OleDbParameter[] oleParameter = new OleDbParameter[3];

oleParameter[0] = new OleDbParameter("@Tc\_No", OleDbType.WChar);

oleParameter[1] = new OleDbParameter("@AidatYil", OleDbType.WChar);

oleParameter[2] = new OleDbParameter("@AidatAy", OleDbType.WChar);

oleParameter[0].Value = Convert.ToString(Tc\_No);

oleParameter[1].Value = Convert.ToString(yil);

oleParameter[2].Value = Convert.ToString(ay);

return query.SelectQuery1(sorgu, oleParameter);

}

public DataTable AidatOdeme4()

{

string sorgu = string.Format("SELECT TblUser.Tc\_No, TblUser.Ad, TblUser.Soyad," +

"TblAidat.AidatAy,TblAidat.AidatYil," +

"TblOdemeler.OdemeDurum,TblOdemeler.BorcMiktar,TblOdemeler.SonOdemeTarih, TblOdemeler.OdemeTarih FROM (TblUser INNER JOIN TblOdemeler ON " +

"TblUser.Tc\_No = TblOdemeler.Tc\_No) INNER JOIN TblAidat ON " +

"TblOdemeler.AidatID = TblAidat.AidatID ");

return query.SelectQuery2(sorgu);

}

public DataTable AidatOdeme5(string yil)

{

string sorgu = string.Format("SELECT TblUser.Tc\_No, TblUser.Ad, TblUser.Soyad," +

"TblAidat.AidatAy,TblAidat.AidatYil," +

"TblOdemeler.OdemeDurum,TblOdemeler.BorcMiktar,TblOdemeler.SonOdemeTarih, TblOdemeler.OdemeTarih FROM (TblUser INNER JOIN TblOdemeler ON " +

"TblUser.Tc\_No = TblOdemeler.Tc\_No) INNER JOIN TblAidat ON " +

"TblOdemeler.AidatID = TblAidat.AidatID WHERE TblAidat.AidatYil =@AidatYil");

OleDbParameter[] oleParameter = new OleDbParameter[1];

oleParameter[0] = new OleDbParameter("@AidatYil", OleDbType.WChar);

oleParameter[0].Value = Convert.ToString(yil);

return query.SelectQuery1(sorgu, oleParameter);

}

public DataTable AidatOdeme6(string yil, string ay)

{

string sorgu = string.Format("SELECT TblUser.Tc\_No, TblUser.Ad, TblUser.Soyad," +

"TblAidat.AidatAy,TblAidat.AidatYil," +

"TblOdemeler.OdemeDurum,TblOdemeler.BorcMiktar,TblOdemeler.SonOdemeTarih, TblOdemeler.OdemeTarih FROM (TblUser INNER JOIN TblOdemeler ON " +

"TblUser.Tc\_No = TblOdemeler.Tc\_No) INNER JOIN TblAidat ON " +

"TblOdemeler.AidatID = TblAidat.AidatID WHERE TblAidat.AidatYil =@AidatYil AND TblAidat.AidatAy=@AidatAy");

OleDbParameter[] oleParameter = new OleDbParameter[2];

oleParameter[0] = new OleDbParameter("@AidatYil", OleDbType.WChar);

oleParameter[1] = new OleDbParameter("@AidatAy", OleDbType.WChar);

oleParameter[0].Value = Convert.ToString(yil);

oleParameter[1].Value = Convert.ToString(ay);

return query.SelectQuery1(sorgu, oleParameter);

}

**AidatOdeme1**

Bu C# kodu, veritabanında belirli bir Tc\_No'ya sahip kullanıcının aidat ödemeleri ile ilgili bilgileri çekmek için kullanılan bir metodu içermektedir. Aşağıda, bu kodun metinsel açıklaması bulunmaktadır:

1. **AidatOdeme1** adlı bir metod tanımlanmıştır.
2. Metodun giriş parametresi şunlardır:
   * **Tc\_No**: Bilgi alınmak istenen kullanıcının TC kimlik numarası (string).
3. **string.Format** metodu kullanılarak karmaşık bir SQL sorgusu oluşturulmuştur. Bu sorgu, **TblUser**, **TblOdemeler** ve **TblAidat** tablolarını içerir. İki INNER JOIN operatörü ile tablolar arasında ilişki kurulmuştur.
4. **OleDbParameter** tipinde bir dizi (**oleParameter**) oluşturulmuştur. Bu dizi, SQL sorgusundaki parametrelerin türünü ve değerini belirtmek için kullanılır.
5. Oluşturulan parametre (**oleParameter**), **Convert.ToString** metodunu kullanarak, metodun giriş parametresinden alınan **Tc\_No** değerine atanmıştır.
6. **query.SelectQuery1** metodu çağrılmıştır. Bu metodun parametreleri, oluşturulan SQL sorgusu ve parametreleri içerir. Bu metodun geri dönüş değeri, sorgunun sonuçlarını içeren bir **DataTable** nesnesidir.
7. **AidatOdeme1** metodu sonucunda, belirtilen **Tc\_No**'ya sahip kullanıcının aidat ödemeleri ile ilgili bilgiler içeren bir **DataTable** döndürülür.

**AidatOdeme2**

Bu C# kodu, belirli bir **Tc\_No**'ya ve belirli bir **yil** değerine sahip olan kullanıcının aidat ödemeleri ile ilgili bilgileri çekmek için kullanılan bir metodu içermektedir. Aşağıda, bu kodun metinsel açıklaması bulunmaktadır:

1. **AidatOdeme2** adlı bir metod tanımlanmıştır.
2. Metodun giriş parametreleri şunlardır:
   * **Tc\_No**: Bilgi alınmak istenen kullanıcının TC kimlik numarası (string).
   * **yil**: Bilgi alınmak istenen aidat yılı (string).
3. **string.Format** metodu kullanılarak karmaşık bir SQL sorgusu oluşturulmuştur. Bu sorgu, **TblUser**, **TblOdemeler** ve **TblAidat** tablolarını içerir. İki INNER JOIN operatörü ile tablolar arasında ilişki kurulmuştur. **WHERE** koşulu, **Tc\_No** ve **AidatYil** parametrelerine göre filtreleme yapar.
4. **OleDbParameter** tipinde bir dizi (**oleParameter**) oluşturulmuştur. Bu dizi, SQL sorgusundaki parametrelerin türünü ve değerini belirtmek için kullanılır.
5. Oluşturulan parametreler (**oleParameter**), **Convert.ToString** metodunu kullanarak, metodun giriş parametrelerinden alınan değerlere atanmıştır.
6. **query.SelectQuery1** metodu çağrılmıştır. Bu metodun parametreleri, oluşturulan SQL sorgusu ve parametreleri içerir. Bu metodun geri dönüş değeri, sorgunun sonuçlarını içeren bir **DataTable** nesnesidir.
7. **AidatOdeme2** metodu sonucunda, belirtilen **Tc\_No** ve **yil**'e sahip kullanıcının aidat ödemeleri ile ilgili bilgiler içeren bir **DataTable** döndürülür.

**AidatOdeme3**

Bu C# kodu, belirli bir **Tc\_No**, **yil** ve **ay** değerlerine sahip olan kullanıcının, belirli bir yıl ve ayda gerçekleştirdiği aidat ödemeleri ile ilgili bilgileri çekmek için kullanılan bir metodu içermektedir. Aşağıda, bu kodun metinsel açıklaması bulunmaktadır:

1. **AidatOdeme3** adlı bir metod tanımlanmıştır.
2. Metodun giriş parametreleri şunlardır:
   * **Tc\_No**: Bilgi alınmak istenen kullanıcının TC kimlik numarası (string).
   * **yil**: Bilgi alınmak istenen aidat yılı (string).
   * **ay**: Bilgi alınmak istenen aidat ayı (string).
3. **string.Format** metodu kullanılarak karmaşık bir SQL sorgusu oluşturulmuştur. Bu sorgu, **TblUser**, **TblOdemeler** ve **TblAidat** tablolarını içerir. İki INNER JOIN operatörü ile tablolar arasında ilişki kurulmuştur. **WHERE** koşulu, **Tc\_No**, **AidatYil** ve **AidatAy** parametrelerine göre filtreleme yapar.
4. **OleDbParameter** tipinde bir dizi (**oleParameter**) oluşturulmuştur. Bu dizi, SQL sorgusundaki parametrelerin türünü ve değerini belirtmek için kullanılır.
5. Oluşturulan parametreler (**oleParameter**), **Convert.ToString** metodunu kullanarak, metodun giriş parametrelerinden alınan değerlere atanmıştır.
6. **query.SelectQuery1** metodu çağrılmıştır. Bu metodun parametreleri, oluşturulan SQL sorgusu ve parametreleri içerir. Bu metodun geri dönüş değeri, sorgunun sonuçlarını içeren bir **DataTable** nesnesidir.
7. **AidatOdeme3** metodu sonucunda, belirtilen **Tc\_No**, **yil** ve **ay**'a sahip kullanıcının aidat ödemeleri ile ilgili bilgiler içeren bir **DataTable** döndürülür.

**AidatOdeme4**

Bu C# kodu, tüm kullanıcıların tüm aidat ödemeleri ile ilgili bilgileri çekmek için kullanılan bir metodu içermektedir. Aşağıda, bu kodun metinsel açıklaması bulunmaktadır:

1. **AidatOdeme4** adlı bir metod tanımlanmıştır.
2. SQL sorgusu içinde, **TblUser**, **TblOdemeler** ve **TblAidat** tabloları arasında INNER JOIN operatörleri kullanılarak birleştirme işlemi yapılmıştır. Bu birleştirme ile tablolar arasındaki ilişkiler kurulmuştur.
3. **SELECT** ifadesi ile belirli alanlar seçilmiştir: kullanıcının TC kimlik numarası (**Tc\_No**), adı (**Ad**), soyadı (**Soyad**), aidatın ayı (**AidatAy**), aidatın yılı (**AidatYil**), ödeme durumu (**OdemeDurum**), borç miktarı (**BorcMiktar**), son ödeme tarihi (**SonOdemeTarih**) ve ödeme tarihi (**OdemeTarih**).
4. **FROM** ifadesi ile hangi tabloların kullanılacağı belirtilmiş ve **WHERE** koşulu olmadan tüm kayıtlar çekilmiştir.
5. **query.SelectQuery2** metodu çağrılmıştır. Bu metodun parametresi, oluşturulan SQL sorgusudur. Bu metodun geri dönüş değeri, sorgunun sonuçlarını içeren bir **DataTable** nesnesidir.
6. **AidatOdeme4** metodu sonucunda, tüm kullanıcıların tüm aidat ödemeleri ile ilgili bilgiler içeren bir **DataTable** döndürülür.

**AidatOdeme5**

Bu C# kodu, belirli bir **yil** değerine sahip olan aidat yılına göre tüm kullanıcıların aidat ödemeleri ile ilgili bilgileri çekmek için kullanılan bir metodu içermektedir. Aşağıda, bu kodun metinsel açıklaması bulunmaktadır:

1. **AidatOdeme5** adlı bir metod tanımlanmıştır.
2. Metodun giriş parametresi şunlardır:
   * **yil**: Bilgi alınmak istenen aidat yılı (string).
3. **string.Format** metodu kullanılarak karmaşık bir SQL sorgusu oluşturulmuştur. Bu sorgu, **TblUser**, **TblOdemeler** ve **TblAidat** tablolarını içerir. İki INNER JOIN operatörü ile tablolar arasında ilişki kurulmuştur. **WHERE** koşulu, **AidatYil** parametresine göre filtreleme yapar.
4. **OleDbParameter** tipinde bir dizi (**oleParameter**) oluşturulmuştur. Bu dizi, SQL sorgusundaki parametrelerin türünü ve değerini belirtmek için kullanılır.
5. Oluşturulan parametre (**oleParameter**), **Convert.ToString** metodunu kullanarak, metodun giriş parametresinden alınan **yil** değerine atanmıştır.
6. **query.SelectQuery1** metodu çağrılmıştır. Bu metodun parametreleri, oluşturulan SQL sorgusu ve parametreleri içerir. Bu metodun geri dönüş değeri, sorgunun sonuçlarını içeren bir **DataTable** nesnesidir.
7. **AidatOdeme5** metodu sonucunda, belirtilen **yil**'e sahip aidat yılında tüm kullanıcıların aidat ödemeleri ile ilgili bilgiler içeren bir **DataTable** döndürülür.

**AidatOdeme6**

Bu C# kodu, belirli bir **yil** ve **ay** değerlerine sahip olan aidat yılı ve ayına göre tüm kullanıcıların aidat ödemeleri ile ilgili bilgileri çekmek için kullanılan bir metodu içermektedir. Aşağıda, bu kodun metinsel açıklaması bulunmaktadır:

1. **AidatOdeme6** adlı bir metod tanımlanmıştır.
2. Metodun giriş parametreleri şunlardır:
   * **yil**: Bilgi alınmak istenen aidat yılı (string).
   * **ay**: Bilgi alınmak istenen aidat ayı (string).
3. **string.Format** metodu kullanılarak karmaşık bir SQL sorgusu oluşturulmuştur. Bu sorgu, **TblUser**, **TblOdemeler** ve **TblAidat** tablolarını içerir. İki INNER JOIN operatörü ile tablolar arasında ilişki kurulmuştur. **WHERE** koşulu, **AidatYil** ve **AidatAy** parametrelerine göre filtreleme yapar.
4. **OleDbParameter** tipinde bir dizi (**oleParameter**) oluşturulmuştur. Bu dizi, SQL sorgusundaki parametrelerin türünü ve değerini belirtmek için kullanılır.
5. Oluşturulan parametreler (**oleParameter**), **Convert.ToString** metodunu kullanarak, metodun giriş parametrelerinden alınan değerlere atanmıştır.
6. **query.SelectQuery1** metodu çağrılmıştır. Bu metodun parametreleri, oluşturulan SQL sorgusu ve parametreleri içerir. Bu metodun geri dönüş değeri, sorgunun sonuçlarını içeren bir **DataTable** nesnesidir.
7. **AidatOdeme6** metodu sonucunda, belirtilen **yil** ve **ay**'a sahip aidat yılı ve ayında tüm kullanıcıların aidat ödemeleri ile ilgili bilgiler içeren bir **DataTable** döndürülür.
   * + 1. **TcQuery , BorcEkleQuery , BorcGuncelleQuery Metodları**

public DataTable TcQuery()

{

string sorgu = "SELECT TblUser.Tc\_No FROM TblUser INNER JOIN TblUserLogin ON TblUser.Tc\_No = TblUserLogin.Tc\_No WHERE TblUserLogin.Auth = '2' AND TblUserLogin.Status = 'A'";

return query.SelectQuery2(sorgu);

}

public bool BorcEkleQuery(string Tc\_No, int AidatID, int OdemeMiktar)

{

string selectAidatQuery = "SELECT AidatAy, AidatYil FROM TblAidat WHERE AidatID = @AidatID";

OleDbParameter[] aidatParameters = new OleDbParameter[]

{

new OleDbParameter("@AidatID", OleDbType.Integer) { Value = AidatID }

};

// Veritabanından AidatAy ve AidatYil bilgilerini al

DataTable aidatTable = query.SelectQuery1(selectAidatQuery, aidatParameters);

if (aidatTable != null && aidatTable.Rows.Count > 0)

{

// Veritabanından alınan bilgileri kullanarak SonOdemeTarih'i oluştur

int aidatAy = Convert.ToInt32(aidatTable.Rows[0]["AidatAy"]);

int aidatYil = Convert.ToInt32(aidatTable.Rows[0]["AidatYil"]);

DateTime sonOdemeTarih = new DateTime(aidatYil, aidatAy, DateTime.DaysInMonth(aidatYil, aidatAy));

// Borç ekleme sorgusu

string \_query = "INSERT INTO TblOdemeler (Tc\_No, AidatID, BorcMiktar, OdemeDurum, SonOdemeTarih) " +

"VALUES (@Tc\_No, @AidatID, @BorcMiktar, @OdemeDurum, @SonOdemeTarih)";

OleDbParameter[] parameters = new OleDbParameter[]

{

new OleDbParameter("@Tc\_No", OleDbType.WChar) { Value = Tc\_No },

new OleDbParameter("@AidatID", OleDbType.Integer) { Value = AidatID },

new OleDbParameter("@BorcMiktar", OleDbType.Integer) { Value = OdemeMiktar },

new OleDbParameter("@OdemeDurum", OleDbType.Boolean) { Value = false },

new OleDbParameter("@SonOdemeTarih", OleDbType.Date) { Value = sonOdemeTarih }

};

// Borç ekleme sorgusunu çalıştır

return query.InsertQuery1(\_query, parameters);

}

// AidatID'ye karşılık gelen bilgileri bulamazsak false döndür

return false;

}

public bool BorcGuncelleQuery(string Tc\_No, int AidatID, int YeniMiktar)

{

string \_query = string.Format("UPDATE TblOdemeler SET BorcMiktar=@BorcMiktar WHERE Tc\_No=@Tc\_No AND AidatID = @AidatID");

OleDbParameter[] oleParameter = new OleDbParameter[3];

oleParameter[0] = new OleDbParameter("@BorcMiktar", OleDbType.Integer);

oleParameter[1] = new OleDbParameter("@Tc\_No", OleDbType.WChar);

oleParameter[2] = new OleDbParameter("@AidatID", OleDbType.Integer);

oleParameter[0].Value = Convert.ToString(YeniMiktar);

oleParameter[1].Value = Convert.ToString(Tc\_No);

oleParameter[2].Value = Convert.ToString(AidatID);

return query.UpdateQuery1(\_query, oleParameter);

}

**TcQuery**

Bu C# kodu, belirli bir koşula uyan kullanıcıların TC kimlik numaralarını çekmek için kullanılan bir metod içermektedir. Aşağıda, bu kodun metinsel açıklaması bulunmaktadır:

1. **TcQuery** adlı bir metod tanımlanmıştır.
2. SQL sorgusu içinde, **TblUser** ve **TblUserLogin** tabloları arasında INNER JOIN operatörü kullanılarak birleştirme işlemi yapılmıştır. Bu birleştirme ile tablolar arasındaki ilişki, **TblUser.Tc\_No** ile **TblUserLogin.Tc\_No** arasındadır.
3. **SELECT** ifadesi ile sadece **TblUser.Tc\_No** alanı seçilmiştir. Bu, sadece kullanıcıların TC kimlik numaralarını içeren bir liste elde etmek amacıyla yapılmıştır.
4. **FROM** ifadesi ile hangi tabloların kullanılacağı belirtilmiş ve **WHERE** koşulu ile sadece belirli koşullara uyan kayıtların seçilmesi sağlanmıştır. Bu koşullar şunlardır:
   * **TblUserLogin.Auth = '2'**: Kullanıcı yetkisi (Auth) '2' olmalıdır.
   * **TblUserLogin.Status = 'A'**: Kullanıcı durumu (Status) 'A' (Aktif) olmalıdır.
5. **query.SelectQuery2** metodu çağrılmıştır. Bu metodun parametresi, oluşturulan SQL sorgusudur. Bu metodun geri dönüş değeri, sorgunun sonuçlarını içeren bir **DataTable** nesnesidir.
6. **TcQuery** metodu sonucunda, belirtilen koşullara uyan kullanıcıların TC kimlik numaralarını içeren bir **DataTable** döndürülür.

**BorcEkleQuery**

Bu C# kodu, bir kullanıcının belirli bir aidat türüne (AidatID) ait olan borcunu eklemek için kullanılan bir metod içermektedir. Aşağıda, bu kodun metinsel açıklaması bulunmaktadır:

1. **BorcEkleQuery** adlı bir metod tanımlanmıştır.
2. Metodun giriş parametreleri şunlardır:
   * **Tc\_No**: Borç eklenen kullanıcının TC kimlik numarası (string).
   * **AidatID**: Borç eklenen aidat türünün ID'si (int).
   * **OdemeMiktar**: Borç miktarı (int).
3. İlk olarak, **SELECT** sorgusu ile veritabanından, belirli bir **AidatID**'ye ait olan aidat türünün ay ve yıl bilgileri (**AidatAy** ve **AidatYil**) çekilir.
4. Eğer bu bilgiler başarıyla alınırsa, **DateTime** sınıfı kullanılarak **SonOdemeTarih** oluşturulur. Bu tarih, aidat türünün son ödeme tarihini temsil eder.
5. **INSERT** sorgusu ile **TblOdemeler** tablosuna, kullanıcının TC kimlik numarası, aidat türü ID'si, borç miktarı, ödeme durumu ve son ödeme tarihi eklenir. Bu sorgu, kullanıcıya ait bir borcu sisteme eklemek için kullanılır.
6. Borç ekleme işlemi başarılıysa, metod **true** değeriyle döner; aksi takdirde, **false** döner.
7. Ayrıca, metodun başında, kullanıcının yetki seviyesinin '2' ve durumunun 'A' (Aktif) olduğu kullanıcıların TC kimlik numaralarını çeken bir **TcQuery** metodunu içerir. Bu, borç eklemek istenen kullanıcıların seçilmesine yardımcı olur.

**BorcGuncelleQuery**

Bu C# kodu, belirli bir kullanıcının belirli bir aidat türüne ait olan borcunu güncellemek için kullanılan bir metod içermektedir. Aşağıda, bu kodun metinsel açıklaması bulunmaktadır:

1. **BorcGuncelleQuery** adlı bir metod tanımlanmıştır.
2. Metodun giriş parametreleri şunlardır:
   * **Tc\_No**: Borcu güncellenen kullanıcının TC kimlik numarası (string).
   * **AidatID**: Borcu güncellenen aidat türünün ID'si (int).
   * **YeniMiktar**: Güncellenen borç miktarı (int).
3. **string.Format** metodu kullanılarak karmaşık bir SQL sorgusu oluşturulmuştur. Bu sorgu, belirli bir **Tc\_No** ve **AidatID**'ye sahip olan borç miktarını günceller.
4. **OleDbParameter** tipinde bir dizi (**oleParameter**) oluşturulmuştur. Bu dizi, SQL sorgusundaki parametrelerin türünü ve değerini belirtmek için kullanılır.
5. Oluşturulan parametreler (**oleParameter**), **Convert.ToString** metodunu kullanarak, metodun giriş parametrelerinden alınan değerlere atanmıştır.
6. **query.UpdateQuery1** metodu çağrılmıştır. Bu metodun parametreleri, oluşturulan SQL sorgusu ve parametreleri içerir. Bu metodun geri dönüş değeri, sorgunun başarıyla güncellenip güncellenmediğini belirten bir **bool** değeridir.
7. **BorcGuncelleQuery** metodu sonucunda, belirtilen kullanıcının belirli aidat türüne ait borcu güncellenmişse **true**, aksi takdirde **false** döndürülür.
   * 1. **DalEmployee**

using DataAccessLayer.DbConnections;

using System;

using System.Data;

using System.Data.OleDb;

namespace DataAccessLayer

{

public class DalEmployee

{

private DbQuery query;

public DalEmployee()

{

query = new DbQuery();

}

* + - 1. **KullaniciTablo , IlCekme , IlceCekme , EmployeeKullaniciAdBelirle**

public DataTable KullaniciTablo(string Auth)

{

string sorgu = string.Format("SELECT TblUser.Tc\_No, TblUser.Ad, TblUser.Soyad, TblUser.Cinsiyet, TblUser.Kan\_Grup, TblUser.Telefon, TblUser.Mail, TblUser.Adres, TblUser.IlID, TblUser.IlceID " +

"FROM TblUser " +

"INNER JOIN TblUserLogin ON TblUser.Tc\_No = TblUserLogin.Tc\_No " +

"WHERE TblUserLogin.Auth = '{0}' " +

"ORDER BY TblUser.Tc\_No", Auth);

return query.SelectQuery2(sorgu);

}

public DataTable IlCekme()

{

string sorgu = string.Format("Select IlID,IlAdi FROM Il ORDER BY IlID");

return query.SelectQuery2(sorgu);

}

public DataTable IlceCekme(int ilID)

{

string sorgu = string.Format("Select IlceID,IlceAdi FROM IlIlce WHERE IlID = @IlID ORDER BY IlceID");

OleDbParameter[] oleParameter = new OleDbParameter[1];

oleParameter[0] = new OleDbParameter("@IlID", OleDbType.Integer);

oleParameter[0].Value = ilID;

return query.SelectQuery1(sorgu, oleParameter);

}

public bool EmployeeKullaniciAdBelirle(string Tc, string kullaniciAd, string parola)

{

string \_query = string.Format("INSERT INTO TblUserLogin (Tc\_No,kullaniciAd,kullaniciSifre,Auth) VALUES (@Tc\_No, @KullaniciAd, @Sifre, @Auth)");

OleDbParameter[] oleParameter = new OleDbParameter[4];

oleParameter[0] = new OleDbParameter("@Tc\_No", OleDbType.WChar);

oleParameter[1] = new OleDbParameter("@kullaniciAd", OleDbType.WChar);

oleParameter[2] = new OleDbParameter("@kullaniciSifre", OleDbType.WChar);

oleParameter[3] = new OleDbParameter("@Auth", OleDbType.WChar);

oleParameter[0].Value = Convert.ToString(Tc);

oleParameter[1].Value = Convert.ToString(kullaniciAd);

oleParameter[2].Value = Convert.ToString(parola);

oleParameter[3].Value = Convert.ToString("1");

return query.InsertQuery1(\_query, oleParameter);

}

**KullaniciTablo**

Bu C# kodu, belirli bir yetki seviyesine (**Auth**) sahip olan kullanıcıların belirli alanlardaki bilgilerini çekmek için kullanılan bir metod içermektedir. Aşağıda, bu kodun metinsel açıklaması bulunmaktadır:

1. **KullaniciTablo** adlı bir metod tanımlanmıştır.
2. Metodun giriş parametresi şunlardır:
   * **Auth**: Kullanıcı yetki seviyesi (string).
3. **string.Format** metodu kullanılarak karmaşık bir SQL sorgusu oluşturulmuştur. Bu sorgu, **TblUser** tablosundan belirli alanları seçer ve **TblUserLogin** tablosu ile **INNER JOIN** operatörü kullanılarak bu iki tablo arasında **Tc\_No** alanı üzerinden birleştirme yapar.
4. **WHERE** koşulu ile sadece belirli bir yetki seviyesine (**Auth**) sahip olan kullanıcılar filtrelenir.
5. **ORDER BY** ifadesi ile sonuçların **Tc\_No**'ya göre sıralanması sağlanır.
6. **query.SelectQuery2** metodu çağrılarak oluşturulan SQL sorgusu çalıştırılır ve sorgunun sonuçları içeren bir **DataTable** nesnesi döndürülür.
7. **KullaniciTablo** metodu sonucunda, belirtilen yetki seviyesine sahip olan kullanıcıların belirli alanlardaki bilgilerini içeren bir **DataTable** döndürülür.

**IlCekme**

Bu C# kodu, bir veritabanındaki "Il" adlı tablodan, il ID'leri (**IlID**) ve il adlarını (**IlAdi**) içeren bilgileri çekmek için kullanılan bir metod içermektedir. Aşağıda, bu kodun metinsel açıklaması bulunmaktadır:

1. **IlCekme** adlı bir metod tanımlanmıştır.
2. Metodun amacı, "Il" tablosundan il ID'leri ve il adlarını çekmektir.
3. **string.Format** metodu kullanılarak karmaşık bir SQL sorgusu oluşturulmuştur. Bu sorgu, "Il" tablosundan il ID'leri ve il adlarını seçer.
4. **ORDER BY** ifadesi ile sonuçların **IlID**'ye göre sıralanması sağlanır.
5. **query.SelectQuery2** metodu çağrılarak oluşturulan SQL sorgusu çalıştırılır ve sorgunun sonuçları içeren bir **DataTable** nesnesi döndürülür.
6. **IlCekme** metodu sonucunda, "Il" tablosundan çekilen il ID'leri ve il adları içeren bir **DataTable** döndürülür.

**IlceCekme**

Bu C# kodu, bir veritabanındaki "IlIlce" adlı tablodan, belirli bir ilin (IlID) ilçe ID'leri (**IlceID**) ve ilçe adlarını (**IlceAdi**) içeren bilgileri çekmek için kullanılan bir metod içermektedir. Aşağıda, bu kodun metinsel açıklaması bulunmaktadır:

1. **IlceCekme** adlı bir metod tanımlanmıştır.
2. Metodun amacı, "IlIlce" tablosundan belirli bir ilin ilçe ID'leri ve ilçe adlarını çekmektir.
3. **string.Format** metodu kullanılarak karmaşık bir SQL sorgusu oluşturulmuştur. Bu sorgu, "IlIlce" tablosundan ilgili ilin (belirli bir **IlID**) ilçe ID'leri ve ilçe adlarını seçer.
4. **OleDbParameter** tipinde bir dizi (**oleParameter**) oluşturulmuştur. Bu dizi, SQL sorgusundaki parametrelerin türünü ve değerini belirtmek için kullanılır. Bu durumda, **@IlID** parametresi ile belirli bir ilin ID'si alınır.
5. Oluşturulan parametre (**oleParameter**), belirtilen ilin ID'sini içerecek şekilde ayarlanır.
6. **query.SelectQuery1** metodu çağrılarak oluşturulan SQL sorgusu ve parametreleri içeren bir **DataTable** nesnesi döndürülür.
7. **IlceCekme** metodu sonucunda, belirtilen ilin ilçe ID'leri ve ilçe adlarını içeren bir **DataTable** döndürülür.

**EmployeeKullaniciAdBelirle**

Bu C# kodu, bir kullanıcıya ait giriş bilgilerini (kullanıcı adı, şifre) belirlemek ve bu kullanıcıyı sisteme eklemek için kullanılan bir metod içermektedir. Aşağıda, bu kodun metinsel açıklaması bulunmaktadır:

1. **EmployeeKullaniciAdBelirle** adlı bir metod tanımlanmıştır.
2. Metodun amacı, belirtilen TC kimlik numarasına (**Tc**), kullanıcı adına (**kullaniciAd**), şifreye (**parola**) ve yetki seviyesine (**Auth**) sahip bir kullanıcıyı sisteme eklemektir.
3. **string.Format** metodu kullanılarak karmaşık bir SQL sorgusu oluşturulmuştur. Bu sorgu, "TblUserLogin" tablosuna belirtilen bilgileri ekler.
4. **OleDbParameter** tipinde bir dizi (**oleParameter**) oluşturulmuştur. Bu dizi, SQL sorgusundaki parametrelerin türünü ve değerini belirtmek için kullanılır.
5. Oluşturulan parametreler (**oleParameter**), metodun giriş parametrelerinden alınan değerlere atanmıştır.
6. **query.InsertQuery1** metodu çağrılarak oluşturulan SQL sorgusu ve parametreleri içeren bir **DataTable** nesnesi döndürülür.
7. **EmployeeKullaniciAdBelirle** metodu sonucunda, belirtilen bilgilerle yeni bir kullanıcı başarıyla eklenmişse **true**, aksi takdirde **false** döndürülür.
   * + 1. **KullaniciEkle , KullaniciSil**

public bool KullaniciEkle(string Tc\_No, string Ad, string Soyad, string Cinsiyet, string Kan\_Grup, string Telefon, string Mail, string Adres, int IlID, int IlceID)

{

string sorgu = string.Format("INSERT INTO TblUser (Tc\_No, Ad, Soyad, Cinsiyet, Kan\_Grup, Telefon, Mail, Adres, IlID, IlceID) VALUES (@Tc\_No, @Ad, @Soyad, @Cinsiyet, @Kan\_Grup, @Telefon, @Mail, @Adres, @IlID, @IlceID)");

OleDbParameter[] oleParameter = new OleDbParameter[10];

oleParameter[0] = new OleDbParameter("@Tc\_No", OleDbType.WChar);

oleParameter[1] = new OleDbParameter("@Ad", OleDbType.WChar);

oleParameter[2] = new OleDbParameter("@Soyad", OleDbType.WChar);

oleParameter[3] = new OleDbParameter("@Cinsiyet", OleDbType.WChar);

oleParameter[4] = new OleDbParameter("@Kan\_Grup", OleDbType.WChar);

oleParameter[5] = new OleDbParameter("@Telefon", OleDbType.WChar);

oleParameter[6] = new OleDbParameter("@Mail", OleDbType.WChar);

oleParameter[7] = new OleDbParameter("@Adres", OleDbType.WChar);

oleParameter[8] = new OleDbParameter("@IlID", OleDbType.Integer);

oleParameter[9] = new OleDbParameter("@IlceID", OleDbType.Integer);

oleParameter[0].Value = Convert.ToString(Tc\_No);

oleParameter[1].Value = Convert.ToString(Ad);

oleParameter[2].Value = Convert.ToString(Soyad);

oleParameter[3].Value = Convert.ToString(Cinsiyet);

oleParameter[4].Value = Convert.ToString(Kan\_Grup);

oleParameter[5].Value = Convert.ToString(Telefon);

oleParameter[6].Value = Convert.ToString(Mail);

oleParameter[7].Value = Convert.ToString(Adres);

oleParameter[8].Value = IlID;

oleParameter[9].Value = IlceID;

return query.InsertQuery1(sorgu, oleParameter);

}

public bool KullaniciSil(string Tc\_No)

{

string sorgu = string.Format("DELETE FROM TblUser WHERE Tc\_No = @Tc\_No");

OleDbParameter[] oleParameter = new OleDbParameter[1];

oleParameter[0] = new OleDbParameter("@Tc\_No", OleDbType.WChar);

oleParameter[0].Value = Convert.ToString(Tc\_No);

return query.DeleteQuery1(sorgu, oleParameter);

}

**KullaniciEkle**

Bu C# kodu, kullanıcı bilgilerini (TC kimlik numarası, ad, soyad, cinsiyet, kan grubu, telefon, e-posta, adres, il ID, ilçe ID) veritabanına eklemek için kullanılan bir metod içermektedir. Aşağıda, bu kodun metinsel açıklaması bulunmaktadır:

1. **KullaniciEkle** adlı bir metod tanımlanmıştır.
2. Metodun amacı, veritabanındaki "TblUser" tablosuna yeni bir kullanıcı eklemektir.
3. **string.Format** metodu kullanılarak karmaşık bir SQL sorgusu oluşturulmuştur. Bu sorgu, "TblUser" tablosuna belirtilen kullanıcı bilgilerini ekler.
4. **OleDbParameter** tipinde bir dizi (**oleParameter**) oluşturulmuştur. Bu dizi, SQL sorgusundaki parametrelerin türünü ve değerini belirtmek için kullanılır.
5. Oluşturulan parametreler (**oleParameter**), metodun giriş parametrelerinden alınan değerlere atanmıştır.
6. **query.InsertQuery1** metodu çağrılarak oluşturulan SQL sorgusu ve parametreleri içeren bir **DataTable** nesnesi döndürülür.
7. **KullaniciEkle** metodu sonucunda, yeni kullanıcı başarıyla eklenmişse **true**, aksi takdirde **false** döndürülür.

**KullaniciSil**

Bu C# kodu, belirli bir kullanıcının TC kimlik numarasına göre veritabanındaki "TblUser" tablosundan silinmesi için kullanılan bir metod içermektedir. İşte bu kodun metinsel açıklaması:

1. **KullaniciSil** adlı bir metod tanımlanmıştır.
2. Metodun amacı, "TblUser" tablosundan belirtilen TC kimlik numarasına sahip bir kullanıcıyı silmektir.
3. **string.Format** metodu kullanılarak karmaşık bir SQL sorgusu oluşturulmuştur. Bu sorgu, "TblUser" tablosundan belirtilen TC kimlik numarasına sahip kullanıcıyı siler.
4. **OleDbParameter** tipinde bir dizi (**oleParameter**) oluşturulmuştur. Bu dizi, SQL sorgusundaki parametrelerin türünü ve değerini belirtmek için kullanılır.
5. Oluşturulan parametre (**oleParameter**), metodun giriş parametresinden alınan TC kimlik numarasına atanmıştır.
6. **query.DeleteQuery1** metodu çağrılarak oluşturulan SQL sorgusu ve parametre içeren bir **DataTable** nesnesi döndürülür.
7. **KullaniciSil** metodu sonucunda, belirtilen TC kimlik numarasına sahip kullanıcı başarıyla silinmişse **true**, aksi takdirde **false** döndürülür.
   * + 1. **KullaniciGuncelle , TcArama , AdSoyadArama**

public bool KullaniciGuncelle(string Tc\_No, string Ad, string SoyAd, string Telefon, string Mail, string Adres, int IlID, int IlceID)

{

string sorgu = string.Format("UPDATE TblUser SET Ad = @Ad, Soyad = @Soyad, Telefon = @Telefon, Mail = @Mail, Adres = @Adres, IlID = @IlID, IlceID = @IlceID WHERE Tc\_No = @Tc\_No");

OleDbParameter[] oleParameters = new OleDbParameter[8];

oleParameters[0] = new OleDbParameter("@Ad", OleDbType.WChar);

oleParameters[1] = new OleDbParameter("@Soyad", OleDbType.WChar);

oleParameters[2] = new OleDbParameter("@Telefon", OleDbType.WChar);

oleParameters[3] = new OleDbParameter("@Mail", OleDbType.WChar);

oleParameters[4] = new OleDbParameter("@Adres", OleDbType.WChar);

oleParameters[5] = new OleDbParameter("@IlID", OleDbType.Integer);

oleParameters[6] = new OleDbParameter("@IlceID", OleDbType.Integer);

oleParameters[7] = new OleDbParameter("@Tc\_No", OleDbType.WChar);

oleParameters[0].Value = Convert.ToString(Ad);

oleParameters[1].Value = Convert.ToString(SoyAd);

oleParameters[2].Value = Convert.ToString(Telefon);

oleParameters[3].Value = Convert.ToString(Mail);

oleParameters[4].Value = Convert.ToString(Adres);

oleParameters[5].Value = Convert.ToString(IlID);

oleParameters[6].Value = Convert.ToString(IlceID);

oleParameters[7].Value = Convert.ToString(Tc\_No);

return query.UpdateQuery1(sorgu, oleParameters);

}

public DataTable TcArama(string Tc\_No)

{

string sorgu = string.Format("SELECT Tc\_No,Ad,Soyad,Cinsiyet,Kan\_Grup,Telefon,Mail,Adres,IlID,IlceID FROM TblUser WHERE Tc\_No=@Tc\_No");

OleDbParameter[] oleParameters = new OleDbParameter[1];

oleParameters[0] = new OleDbParameter("@Tc\_No", OleDbType.WChar);

oleParameters[0].Value = Convert.ToString(Tc\_No);

return query.SelectQuery1(sorgu, oleParameters);

}

public DataTable AdSoyadArama(string Ad, string Soyad)

{

string sorgu = string.Format("SELECT Tc\_No,Ad,Soyad,Cinsiyet,Kan\_Grup,Telefon,Mail,Adres,IlID,IlceID FROM TblUser WHERE Ad=@Ad AND Soyad=@Soyad");

OleDbParameter[] oleParameters = new OleDbParameter[2];

oleParameters[0] = new OleDbParameter("@Ad", OleDbType.WChar);

oleParameters[1] = new OleDbParameter("@Soyad", OleDbType.WChar);

oleParameters[0].Value = Convert.ToString(Ad);

oleParameters[1].Value = Convert.ToString(Soyad);

return query.SelectQuery1(sorgu, oleParameters);

}

**KullaniciGuncelle**

Bu C# kodu, belirli bir kullanıcının TC kimlik numarasına göre veritabanındaki "TblUser" tablosunda güncellenmesi için kullanılan bir metod içermektedir. İşte bu kodun metinsel açıklaması:

1. **KullaniciGuncelle** adlı bir metod tanımlanmıştır.
2. Metodun amacı, "TblUser" tablosundan belirtilen TC kimlik numarasına sahip bir kullanıcının belirli alanlarını güncellemektir.
3. **string.Format** metodu kullanılarak karmaşık bir SQL sorgusu oluşturulmuştur. Bu sorgu, "TblUser" tablosunda belirtilen TC kimlik numarasına sahip kullanıcının belirli alanlarını günceller.
4. **OleDbParameter** tipinde bir dizi (**oleParameters**) oluşturulmuştur. Bu dizi, SQL sorgusundaki parametrelerin türünü ve değerini belirtmek için kullanılır.
5. Oluşturulan parametreler (**oleParameters**), metodun giriş parametrelerinden alınan değerlere atanmıştır.
6. **query.UpdateQuery1** metodu çağrılarak oluşturulan SQL sorgusu ve parametreleri içeren bir **DataTable** nesnesi döndürülür.
7. **KullaniciGuncelle** metodu sonucunda, belirtilen TC kimlik numarasına sahip kullanıcının bilgileri başarıyla güncellenmişse **true**, aksi takdirde **false** döndürülür.

**TcArama**

Bu C# kodu, belirli bir TC kimlik numarasına sahip kullanıcının veritabanındaki "TblUser" tablosunda aranması için kullanılan bir metod içermektedir. İşte bu kodun metinsel açıklaması:

1. **TcArama** adlı bir metod tanımlanmıştır.
2. Metodun amacı, "TblUser" tablosunda belirtilen TC kimlik numarasına sahip bir kullanıcının bilgilerini getirmektir.
3. **string.Format** metodu kullanılarak karmaşık bir SQL sorgusu oluşturulmuştur. Bu sorgu, "TblUser" tablosundan belirtilen TC kimlik numarasına sahip kullanıcının bilgilerini getirir.
4. **OleDbParameter** tipinde bir dizi (**oleParameters**) oluşturulmuştur. Bu dizi, SQL sorgusundaki parametrelerin türünü ve değerini belirtmek için kullanılır.
5. Oluşturulan parametre (**oleParameters**), metodun giriş parametresinden alınan TC kimlik numarasına atanmıştır.
6. **query.SelectQuery1** metodu çağrılarak oluşturulan SQL sorgusu ve parametreyi içeren bir **DataTable** nesnesi döndürülür.
7. **TcArama** metodu sonucunda, belirtilen TC kimlik numarasına sahip kullanıcının bilgilerini içeren bir **DataTable** döndürülür. Eğer kullanıcı bulunamazsa, bu **DataTable** boş olacaktır.

**AdSoyadArama**

Bu C# kodu, "TblUser" tablosunda belirtilen ad ve soyada sahip kullanıcıları aramak için kullanılan bir metod içermektedir. İşte bu kodun metinsel açıklaması:

1. **AdSoyadArama** adlı bir metod tanımlanmıştır.
2. Metodun amacı, "TblUser" tablosunda belirtilen ad ve soyada sahip kullanıcıları getirmektir.
3. **string.Format** metodu kullanılarak karmaşık bir SQL sorgusu oluşturulmuştur. Bu sorgu, "TblUser" tablosundan belirtilen ad ve soyada sahip kullanıcıların bilgilerini getirir.
4. **OleDbParameter** tipinde bir dizi (**oleParameters**) oluşturulmuştur. Bu dizi, SQL sorgusundaki parametrelerin türünü ve değerini belirtmek için kullanılır.
5. Oluşturulan parametreler (**oleParameters**), metodun giriş parametrelerinden alınan ad ve soyada atanmıştır.
6. **query.SelectQuery1** metodu çağrılarak oluşturulan SQL sorgusu ve parametreleri içeren bir **DataTable** nesnesi döndürülür.
7. **AdSoyadArama** metodu sonucunda, belirtilen ad ve soyada sahip kullanıcıların bilgilerini içeren bir **DataTable** döndürülür. Eğer böyle kullanıcılar bulunamazsa, bu **DataTable** boş olacaktır.
   * + 1. **TelArama , MailArama**

public DataTable TelArama(string Tel)

{

string sorgu = string.Format("SELECT Tc\_No,Ad,Soyad,Cinsiyet,Kan\_Grup,Telefon,Mail,Adres,IlID,IlceID FROM TblUser WHERE Telefon=@Telefon");

OleDbParameter[] oleParameters = new OleDbParameter[1];

oleParameters[0] = new OleDbParameter("@Telefon", OleDbType.WChar);

oleParameters[0].Value = Convert.ToString(Tel);

return query.SelectQuery1(sorgu, oleParameters);

}

public DataTable MailArama(string Mail)

{

string sorgu = string.Format("SELECT Tc\_No,Ad,Soyad,Cinsiyet,Kan\_Grup,Telefon,Mail,Adres,IlID,IlceID FROM TblUser WHERE Mail=@Mail");

OleDbParameter[] oleParameters = new OleDbParameter[1];

oleParameters[0] = new OleDbParameter("@Mail", OleDbType.WChar);

oleParameters[0].Value = Convert.ToString(Mail);

return query.SelectQuery1(sorgu, oleParameters);

}

**TelArama**

Bu C# kodu, "TblUser" tablosunda belirtilen telefon numarasına sahip kullanıcıları aramak için kullanılan bir metod içermektedir. İşte bu kodun metinsel açıklaması:

1. **TelArama** adlı bir metod tanımlanmıştır.
2. Metodun amacı, "TblUser" tablosunda belirtilen telefon numarasına sahip kullanıcıları getirmektir.
3. **string.Format** metodu kullanılarak karmaşık bir SQL sorgusu oluşturulmuştur. Bu sorgu, "TblUser" tablosundan belirtilen telefon numarasına sahip kullanıcıların bilgilerini getirir.
4. **OleDbParameter** tipinde bir dizi (**oleParameters**) oluşturulmuştur. Bu dizi, SQL sorgusundaki parametrelerin türünü ve değerini belirtmek için kullanılır.
5. Oluşturulan parametreler (**oleParameters**), metodun giriş parametresinden alınan telefon numarasına atanmıştır.
6. **query.SelectQuery1** metodu çağrılarak oluşturulan SQL sorgusu ve parametreleri içeren bir **DataTable** nesnesi döndürülür.
7. **TelArama** metodu sonucunda, belirtilen telefon numarasına sahip kullanıcıların bilgilerini içeren bir **DataTable** döndürülür. Eğer böyle kullanıcılar bulunamazsa, bu **DataTable** boş olacaktır.

**MailArama**

Bu C# kodu, "TblUser" tablosunda belirtilen e-posta adresine sahip kullanıcıları aramak için kullanılan bir metod içermektedir. İşte bu kodun metinsel açıklaması:

1. **MailArama** adlı bir metod tanımlanmıştır.
2. Metodun amacı, "TblUser" tablosunda belirtilen e-posta adresine sahip kullanıcıları getirmektir.
3. **string.Format** metodu kullanılarak karmaşık bir SQL sorgusu oluşturulmuştur. Bu sorgu, "TblUser" tablosundan belirtilen e-posta adresine sahip kullanıcıların bilgilerini getirir.
4. **OleDbParameter** tipinde bir dizi (**oleParameters**) oluşturulmuştur. Bu dizi, SQL sorgusundaki parametrelerin türünü ve değerini belirtmek için kullanılır.
5. Oluşturulan parametreler (**oleParameters**), metodun giriş parametresinden alınan e-posta adresine atanmıştır.
6. **query.SelectQuery1** metodu çağrılarak oluşturulan SQL sorgusu ve parametreleri içeren bir **DataTable** nesnesi döndürülür.
7. **MailArama** metodu sonucunda, belirtilen e-posta adresine sahip kullanıcıların bilgilerini içeren bir **DataTable** döndürülür. Eğer böyle kullanıcılar bulunamazsa, bu **DataTable** boş olacaktır.
   * 1. **DalGraph**

using DataAccessLayer.DbConnections;

using System;

using System.Data;

using System.Data.OleDb;

using System.Reflection;

using System.Text.RegularExpressions;

namespace DataAccessLayer

{

public class DalGraph

{

private DbQuery \_query;

public DalGraph()

{

\_query = new DbQuery();

}

* + - 1. **JustYil , JustAy , YilandAy , IlCekme**

public DataTable JustYil(string AidatYil)

{ // TblOdemeler.OdemeMiktar,

string sorgu = string.Format("SELECT TblOdemeler.BorcMiktar, TblAidat.AidatTutar, TblAidat.AidatAy FROM TblOdemeler INNER JOIN TblAidat ON TblOdemeler.AidatID = TblAidat.AidatID WHERE TblAidat.AidatYil = @AidatYil");

OleDbParameter[] oleParameters = new OleDbParameter[1];

oleParameters[0] = new OleDbParameter("@AidatYil", OleDbType.WChar);

oleParameters[0].Value = Convert.ToString(AidatYil);

return \_query.SelectQuery1(sorgu, oleParameters);

}

public DataTable JustAy(string AidatAy)

{

string sorgu = string.Format("SELECT TblOdemeler.BorcMiktar, TblAidat.AidatTutar, TblAidat.AidatYil FROM TblOdemeler INNER JOIN TblAidat ON TblOdemeler.AidatID = TblAidat.AidatID WHERE TblAidat.AidatAy = @AidatAy");

OleDbParameter[] oleParameters = new OleDbParameter[1];

oleParameters[0] = new OleDbParameter("@AidatAy", OleDbType.WChar);

oleParameters[0].Value = Convert.ToString(AidatAy);

return \_query.SelectQuery1(sorgu, oleParameters);

}

public DataTable YilandAy(string AidatYil, string AidatAy)

{

string sorgu = @"

SELECT TblOdemeler.BorcMiktar, TblOdemeler.OdemeMiktar, TblAidat.AidatTutar, TblUser.Ad, TblUser.Soyad

FROM (TblUser

INNER JOIN TblOdemeler ON TblUser.Tc\_No = TblOdemeler.Tc\_No)

INNER JOIN TblAidat ON TblOdemeler.AidatID = TblAidat.AidatID

WHERE TblAidat.AidatYil = @AidatYil AND TblAidat.AidatAy = @AidatAy";

OleDbParameter[] oleParameters = new OleDbParameter[2];

oleParameters[0] = new OleDbParameter("@AidatYil", OleDbType.WChar);

oleParameters[1] = new OleDbParameter("@AidatAy", OleDbType.WChar);

oleParameters[0].Value = Convert.ToString(AidatYil);

oleParameters[1].Value = Convert.ToString(AidatAy);

return \_query.SelectQuery1(sorgu, oleParameters);

}

public DataTable IlCekme()

{

string sorgu = string.Format("Select IlID,IlAdi FROM Il ORDER BY IlID");

return \_query.SelectQuery2(sorgu);

}

**JustYil**

Bu C# kodu, "TblOdemeler" ve "TblAidat" tablolarını kullanarak belirli bir yıla ait aidat ödemelerini ve borç miktarlarını getirmek için bir metod içermektedir. İşte bu kodun metinsel açıklaması:

1. **JustYil** adlı bir metod tanımlanmıştır.
2. Metodun amacı, "TblAidat" tablosunda belirtilen yıla ait aidatların bilgileri ile "TblOdemeler" tablosundan alınan borç miktarlarını getirmektir.
3. **string.Format** metodu kullanılarak karmaşık bir SQL sorgusu oluşturulmuştur. Bu sorgu, "TblOdemeler" ve "TblAidat" tablolarını birleştirerek belirli bir yıla ait aidat ödemelerini ve borç miktarlarını getirir.
4. **OleDbParameter** tipinde bir dizi (**oleParameters**) oluşturulmuştur. Bu dizi, SQL sorgusundaki parametrelerin türünü ve değerini belirtmek için kullanılır.
5. Oluşturulan parametre (**oleParameters**), metodun giriş parametresinden alınan yıl bilgisine atanmıştır.
6. **\_query.SelectQuery1** metodu çağrılarak oluşturulan SQL sorgusu ve parametreleri içeren bir **DataTable** nesnesi döndürülür.
7. **JustYil** metodu sonucunda, belirtilen yıla ait aidat ödemelerini ve borç miktarlarını içeren bir **DataTable** döndürülür. Eğer böyle bir veri bulunamazsa, bu **DataTable** boş olacaktır.

**JustAy**

Bu C# kodu, "TblOdemeler" ve "TblAidat" tablolarını kullanarak belirli bir ayda yapılan aidat ödemelerini ve borç miktarlarını getirmek için bir metod içermektedir. İşte bu kodun metinsel açıklaması:

1. **JustAy** adlı bir metod tanımlanmıştır.
2. Metodun amacı, "TblAidat" tablosunda belirtilen ayda yapılan aidat ödemelerinin bilgileri ile "TblOdemeler" tablosundan alınan borç miktarlarını getirmektir.
3. **string.Format** metodu kullanılarak karmaşık bir SQL sorgusu oluşturulmuştur. Bu sorgu, "TblOdemeler" ve "TblAidat" tablolarını birleştirerek belirli bir ayda yapılan aidat ödemelerini ve borç miktarlarını getirir.
4. **OleDbParameter** tipinde bir dizi (**oleParameters**) oluşturulmuştur. Bu dizi, SQL sorgusundaki parametrelerin türünü ve değerini belirtmek için kullanılır.
5. Oluşturulan parametre (**oleParameters**), metodun giriş parametresinden alınan ay bilgisine atanmıştır.
6. **\_query.SelectQuery1** metodu çağrılarak oluşturulan SQL sorgusu ve parametreleri içeren bir **DataTable** nesnesi döndürülür.
7. **JustAy** metodu sonucunda, belirtilen ayda yapılan aidat ödemelerini ve borç miktarlarını içeren bir **DataTable** döndürülür. Eğer böyle bir veri bulunamazsa, bu **DataTable** boş olacaktır.

**YilandAy**

Bu C# kodu, belirli bir yıl ve ayda yapılan aidat ödemelerinin ve borç miktarlarının kullanıcı adı ve soyadıyla birlikte getirilmesi için bir metod içermektedir. İşte bu kodun metinsel açıklaması:

1. **YilandAy** adlı bir metod tanımlanmıştır.
2. Metodun amacı, "TblUser", "TblOdemeler" ve "TblAidat" tablolarını birleştirerek belirli bir yıl ve ayda yapılan aidat ödemelerini ve borç miktarlarını getirmektir.
3. **string.Format** metodu kullanılarak karmaşık bir SQL sorgusu oluşturulmuştur. Bu sorgu, "TblUser" tablosunu "TblOdemeler" ve "TblAidat" tabloları ile birleştirerek belirli bir yıl ve ayda yapılan aidat ödemelerini ve borç miktarlarını getirir.
4. **OleDbParameter** tipinde bir dizi (**oleParameters**) oluşturulmuştur. Bu dizi, SQL sorgusundaki parametrelerin türünü ve değerini belirtmek için kullanılır.
5. Oluşturulan parametreler (**oleParameters**), metodun giriş parametrelerinden alınan yıl ve ay bilgilerine atanmıştır.
6. **\_query.SelectQuery1** metodu çağrılarak oluşturulan SQL sorgusu ve parametreleri içeren bir **DataTable** nesnesi döndürülür.
7. **YilandAy** metodu sonucunda, belirtilen yıl ve ayda yapılan aidat ödemelerini, borç miktarlarını, kullanıcı adını ve soyadını içeren bir **DataTable** döndürülür. Eğer böyle bir veri bulunamazsa, bu **DataTable** boş olacaktır.

**IlCekme**

Bu C# kodu, bir veritabanındaki "Il" tablosundan il ID ve il adı bilgilerini çekmek amacıyla kullanılan bir metod içermektedir. İşte bu kodun metinsel açıklaması:

1. **IlCekme** adlı bir metod tanımlanmıştır.
2. Metodun amacı, "Il" tablosundan il ID ve il adı bilgilerini çekmektir.
3. **string.Format** metodu kullanılarak basit bir SQL sorgusu oluşturulmuştur. Bu sorgu, "Il" tablosundan tüm il ID ve il adı bilgilerini getirir ve bu bilgileri "IlID" sırasına göre sıralar.
4. **\_query.SelectQuery2** metodu çağrılarak oluşturulan SQL sorgusu kullanılarak veritabanından il bilgilerini içeren bir **DataTable** nesnesi getirilir.
5. Bu **DataTable**, ilgili bilgileri içerir ve daha sonra kullanıcı tarafından işlenmek üzere geri döndürülür.

Sonuç olarak, **IlCekme** metodu, veritabanındaki "Il" tablosundan il ID ve il adı bilgilerini çekmek için kullanılır ve bu bilgileri içeren bir **DataTable** nesnesini döndürür.

* + - 1. **IlceCekme , IleGoreKSayi , IlceyeGoreKSayi**

public DataTable IlceCekme(int ilID)

{

string sorgu = string.Format("Select IlceID,IlceAdi FROM IlIlce WHERE IlID = @IlID ORDER BY IlceID");

OleDbParameter[] oleParameter = new OleDbParameter[1];

oleParameter[0] = new OleDbParameter("@IlID", OleDbType.Integer);

oleParameter[0].Value = ilID;

return \_query.SelectQuery1(sorgu, oleParameter);

}

public DataTable IleGoreKSayi(int IlID)

{

string sorgu = @"SELECT Il.IlAdi AS IlAdi,

COUNT(TblUser.Tc\_No) AS KullaniciSayisi

FROM Il INNER JOIN TblUser ON Il.IlID = TblUser.IlID

WHERE Il.IlID = @IlID

GROUP BY Il.IlAdi";

OleDbParameter[] oleParameters = new OleDbParameter[1];

oleParameters[0] = new OleDbParameter("@IlID", OleDbType.Integer);

oleParameters[0].Value = IlID;

return \_query.SelectQuery1(sorgu, oleParameters);

}

public DataTable IlceyeGoreKSayi(int IlceID)

{

string sorgu = @"SELECT IlIlce.IlceAdi AS IlceAdi,

COUNT(TblUser.Tc\_No) AS KullaniciSayisi

FROM IlIlce INNER JOIN TblUser ON IlIlce.IlceID = TblUser.IlceID

WHERE IlIlce.IlceID = @IlceID

GROUP BY IlIlce.IlceAdi";

OleDbParameter[] oleParameters = new OleDbParameter[1];

oleParameters[0] = new OleDbParameter("@IlceID", OleDbType.Integer);

oleParameters[0].Value = IlceID;

return \_query.SelectQuery1(sorgu, oleParameters);

}

**IlceCekme**

Bu C# kodu, bir veritabanındaki "IlIlce" tablosundan belirli bir ilin (ilin ID'sine göre belirlenmiş) ilçe ID ve ilçe adı bilgilerini çekmek amacıyla kullanılan bir metodu içermektedir. İşte bu kodun metinsel açıklaması:

1. **IlceCekme** adlı bir metod tanımlanmıştır.
2. Metodun amacı, "IlIlce" tablosundan belirli bir ilin ilçe ID ve ilçe adı bilgilerini çekmektir.
3. **string.Format** metodu kullanılarak parametreli bir SQL sorgusu oluşturulmuştur. Bu sorgu, "IlIlce" tablosundan belirli bir ilin ilçe ID ve ilçe adı bilgilerini getirir ve bu bilgileri "IlceID" sırasına göre sıralar. WHERE koşulu ile de ilin ID'sine göre filtreleme yapılır.
4. **OleDbParameter** dizisi tanımlanarak, SQL sorgusundaki parametrenin değeri belirlenir. Bu durumda, il ID parametresi belirtilen **ilID** değişkeninin değeriyle doldurulur.
5. **\_query.SelectQuery1** metodu çağrılarak oluşturulan SQL sorgusu ve parametreler kullanılarak veritabanından ilçe bilgilerini içeren bir **DataTable** nesnesi getirilir.
6. Bu **DataTable**, ilgili bilgileri içerir ve daha sonra kullanıcı tarafından işlenmek üzere geri döndürülür.

Sonuç olarak, **IlceCekme** metodu, veritabanındaki "IlIlce" tablosundan belirli bir ilin ilçe ID ve ilçe adı bilgilerini çekmek için kullanılır ve bu bilgileri içeren bir **DataTable** nesnesini döndürür.

**IleGoreKSayi**

Bu C# kodu, belirli bir ilin (ilin ID'sine göre belirlenmiş) kullanıcı sayısını ve il adını çekmek amacıyla kullanılan bir metodu içermektedir. İşte bu kodun metinsel açıklaması:

1. **IleGoreKSayi** adlı bir metod tanımlanmıştır.
2. Metodun amacı, "Il" ve "TblUser" tablolarını birleştirerek belirli bir ilin il adını ve o ile ait kullanıcı sayısını çekmektir.
3. **string.Format** metodu kullanılarak parametreli bir SQL sorgusu oluşturulmuştur. Bu sorgu, "Il" ve "TblUser" tablolarını iç içe birleştirir ve belirli bir ilin ID'sine göre filtreleme yapar. Ayrıca, **COUNT** fonksiyonu kullanılarak ilgili ilin kullanıcı sayısı hesaplanır.
4. **OleDbParameter** dizisi tanımlanarak, SQL sorgusundaki parametrenin değeri belirlenir. Bu durumda, il ID parametresi, belirtilen **IlID** değişkeninin değeriyle doldurulur.
5. **\_query.SelectQuery1** metodu çağrılarak oluşturulan SQL sorgusu ve parametreler kullanılarak veritabanından il adı ve kullanıcı sayısı bilgilerini içeren bir **DataTable** nesnesi getirilir.
6. Bu **DataTable**, ilgili bilgileri içerir ve daha sonra kullanıcı tarafından işlenmek üzere geri döndürülür.

Sonuç olarak, **IleGoreKSayi** metodu, belirli bir ilin kullanıcı sayısını ve il adını çekmek için kullanılır ve bu bilgileri içeren bir **DataTable** nesnesini döndürür.

**IlceyeGoreKSayi**

Bu C# kodu, belirli bir ilçenin (ilçenin ID'sine göre belirlenmiş) kullanıcı sayısını ve ilçe adını çekmek amacıyla kullanılan bir metodu içermektedir. İşte bu kodun metinsel açıklaması:

1. **IlceyeGoreKSayi** adlı bir metod tanımlanmıştır.
2. Metodun amacı, "IlIlce" ve "TblUser" tablolarını birleştirerek belirli bir ilçenin ilçe adını ve o ilçeye ait kullanıcı sayısını çekmektir.
3. **string.Format** metodu kullanılarak parametreli bir SQL sorgusu oluşturulmuştur. Bu sorgu, "IlIlce" ve "TblUser" tablolarını iç içe birleştirir ve belirli bir ilçenin ID'sine göre filtreleme yapar. Ayrıca, **COUNT** fonksiyonu kullanılarak ilgili ilçenin kullanıcı sayısı hesaplanır.
4. **OleDbParameter** dizisi tanımlanarak, SQL sorgusundaki parametrenin değeri belirlenir. Bu durumda, ilçe ID parametresi, belirtilen **IlceID** değişkeninin değeriyle doldurulur.
5. **\_query.SelectQuery1** metodu çağrılarak oluşturulan SQL sorgusu ve parametreler kullanılarak veritabanından ilçe adı ve kullanıcı sayısı bilgilerini içeren bir **DataTable** nesnesi getirilir.
6. Bu **DataTable**, ilgili bilgileri içerir ve daha sonra kullanıcı tarafından işlenmek üzere geri döndürülür.

Sonuç olarak, **IlceyeGoreKSayi** metodu, belirli bir ilçenin kullanıcı sayısını ve ilçe adını çekmek için kullanılır ve bu bilgileri içeren bir **DataTable** nesnesini döndürür.

* + 1. **DalList**

using DataAccessLayer.DbConnections;

using System.Data.OleDb;

using System.Data;

using System.Text.RegularExpressions;

using System.Data.SqlClient;

using System;

using System.Security.Cryptography.X509Certificates;

namespace DataAccessLayer

{

public class DalList

{

private DbQuery \_query;

public DalList()

{

\_query = new DbQuery();

}

public DataTable IlCekme()

{

string sorgu = string.Format("Select IlID,IlAdi FROM Il ORDER BY IlID");

return \_query.SelectQuery2(sorgu);

}

public DataTable IlceCekme(int ilID)

{

string sorgu = string.Format("Select IlceID,IlceAdi FROM IlIlce WHERE IlID = @IlID ORDER BY IlceID");

OleDbParameter[] oleParameter = new OleDbParameter[1];

oleParameter[0] = new OleDbParameter("@IlID", OleDbType.Integer);

oleParameter[0].Value = ilID;

return \_query.SelectQuery1(sorgu, oleParameter);

}

public DataTable KanSorgu1()

{

string sorgu = string.Format("SELECT TblUser.Tc\_No, TblUser.Ad, TblUser.Soyad, TblUser.Cinsiyet," +

" TblUser.Kan\_Grup, TblUser.Telefon, TblUser.Mail, TblUser.Adres " +

"FROM TblUser " +

"INNER JOIN TblUserLogin ON TblUser.Tc\_No = TblUserLogin.Tc\_No " +

"WHERE TblUserLogin.Status = 'A' AND TblUserLogin.Auth = '2'");

return \_query.SelectQuery2(sorgu);

}

public DataTable KanSorgu2(string Grup)

{

string sorgu = string.Format("SELECT TblUser.Tc\_No, TblUser.Ad, TblUser.Soyad, TblUser.Cinsiyet," +

" TblUser.Kan\_Grup, TblUser.Telefon, TblUser.Mail, TblUser.Adres " +

"FROM TblUser " +

"INNER JOIN TblUserLogin ON TblUser.Tc\_No = TblUserLogin.Tc\_No " +

"WHERE TblUserLogin.Status = 'A' AND TblUserLogin.Auth = '2' AND TblUser.Kan\_Grup = @Grup");

OleDbParameter[] oleParameter = new OleDbParameter[1];

oleParameter[0] = new OleDbParameter("@Grup", OleDbType.VarChar);

oleParameter[0].Value = Grup;

return \_query.SelectQuery1(sorgu, oleParameter);

}

Bu metodun amacı, bir veritabanındaki "Il" tablosundan "IlID" ve "IlAdi" sütunlarındaki bilgileri çekmek ve bu bilgileri bir **DataTable** nesnesinde toplamaktır.

1. **string sorgu = string.Format("Select IlID, IlAdi FROM Il ORDER BY IlID");**
   * SQL sorgusu oluşturulur. Bu sorgu, "Il" tablosundan "IlID" ve "IlAdi" sütunlarını seçer ve "IlID" sütununa göre sıralar.
2. **return \_query.SelectQuery2(sorgu);**
   * **\_query** adındaki bir nesnenin **SelectQuery2** metoduna oluşturulan SQL sorgusu iletilir.
   * Bu metodun geri dönüş değeri, veritabanından çekilen verileri içeren bir **DataTable** nesnesidir.
   * Bu **DataTable**, "IlID" ve "IlAdi" sütunlarındaki il bilgilerini içerir.

Sonuç olarak, **IlCekme** metodu, belirtilen SQL sorgusunu kullanarak bir veritabanından il bilgilerini çeker ve bu bilgileri içeren bir **DataTable** nesnesini döndürür.

Bu metod, belirli bir ilin (verilen **ilID** parametresi ile belirlenen ilin) ilçelerini bir veritabanından çekmek amacıyla kullanılır.

1. SQL sorgusu oluşturulur:
   * **IlIlce** tablosundan **IlceID** ve **IlceAdi** sütunları seçilir.
   * **IlID** sütunu, metoda parametre olarak verilen **ilID** değeri ile eşleşen ilçelerin seçilmesi için kullanılır.
   * Sonuçlar **IlceID** sütununa göre sıralanır.
2. Parametre hazırlanır:
   * **OleDbParameter[] oleParameter = new OleDbParameter[1];** ile bir adet parametre oluşturulur.
   * **@IlID** adında bir parametre tanımlanır ve tipi **OleDbType.Integer** olarak belirlenir.
   * Parametreye, metoda geçirilen **ilID** değeri atanır.
3. **\_query.SelectQuery1(sorgu, oleParameter)** çağrılır:
   * **\_query** adlı bir nesnenin **SelectQuery1** metodu çağrılır.
   * Bu metoda oluşturulan SQL sorgusu (**sorgu**) ve parametreler (**oleParameter**) iletilir.
   * Metodun geri dönüş değeri, veritabanından çekilen ilçe bilgilerini içeren bir **DataTable** nesnesidir.

Sonuç olarak, **IlceCekme** metodu, belirli bir ilin ilçelerini çekmek için SQL sorgusu ve parametre kullanır ve bu bilgileri içeren bir **DataTable** nesnesini döndürür.

Bu metod, aktif durumdaki (**Status = 'A'**) ve yetkisi (**Auth**) müşteri (**Auth = '2'**) olan kullanıcıların belirli bilgilerini içeren bir **DataTable** döndürmek üzere kullanılır.

SQL sorgusu şu adımları içerir:

1. **TblUser** ve **TblUserLogin** tabloları arasında bir iç birleştirme (INNER JOIN) yapılır.
2. **TblUser.Tc\_No**, **TblUser.Ad**, **TblUser.Soyad**, **TblUser.Cinsiyet**, **TblUser.Kan\_Grup**, **TblUser.Telefon**, **TblUser.Mail**, **TblUser.Adres** sütunları seçilir.
3. **TblUserLogin.Status** sütunu "A" (aktif) ve **TblUserLogin.Auth** sütunu "2" (müşteri) olan kullanıcıları sınırlar.
4. Sonuçlar, **TblUser.Tc\_No = TblUserLogin.Tc\_No** eşleşen kayıtlar arasında oluşturulan bir iç birleştirme sonucu elde edilir.

Bu sayede, sorgu, aktif müşterilerin belirli bilgilerini içeren bir **DataTable** nesnesini döndürmek üzere **\_query.SelectQuery2** metodunu kullanır.

Bu metod, belirli bir kan grubuna (**Grup** parametresi) sahip, aktif durumdaki (**Status = 'A'**) ve müşteri yetkisine (**Auth = '2'**) sahip kullanıcıların belirli bilgilerini içeren bir **DataTable** döndürmek üzere kullanılır.

SQL sorgusu şu adımları içerir:

1. **TblUser** ve **TblUserLogin** tabloları arasında bir iç birleştirme (INNER JOIN) yapılır.
2. **TblUser.Tc\_No**, **TblUser.Ad**, **TblUser.Soyad**, **TblUser.Cinsiyet**, **TblUser.Kan\_Grup**, **TblUser.Telefon**, **TblUser.Mail**, **TblUser.Adres** sütunları seçilir.
3. **TblUserLogin.Status** sütunu "A" (aktif), **TblUserLogin.Auth** sütunu "2" (müşteri) ve **TblUser.Kan\_Grup** sütunu, metoda geçirilen **Grup** değeriyle eşleşen kullanıcıları sınırlar.
4. **@Grup** parametresi, SQL sorgusunda kullanılmak üzere **OleDbParameter** ile oluşturulur.

Bu sayede, sorgu, belirli bir kan grubuna sahip aktif müşterilerin belirli bilgilerini içeren bir **DataTable** nesnesini döndürmek üzere **\_query.SelectQuery1** metodunu kullanır.

public DataTable StatuSorgu1()

{

string sorgu = string.Format("SELECT TblUser.Tc\_No, TblUser.Ad, TblUser.Soyad, TblUser.Cinsiyet," +

" TblUser.Kan\_Grup, TblUser.Telefon, TblUser.Mail, TblUser.Adres " +

"FROM TblUser " +

"INNER JOIN TblUserLogin ON TblUser.Tc\_No = TblUserLogin.Tc\_No " +

"WHERE TblUserLogin.Auth = '2'");

return \_query.SelectQuery2(sorgu);

}

public DataTable StatuSorgu2(string Statu)

{

string sorgu = string.Format("SELECT TblUser.Tc\_No, TblUser.Ad, TblUser.Soyad, TblUser.Cinsiyet," +

" TblUser.Kan\_Grup, TblUser.Telefon, TblUser.Mail, TblUser.Adres " +

"FROM TblUser " +

"INNER JOIN TblUserLogin ON TblUser.Tc\_No = TblUserLogin.Tc\_No " +

"WHERE TblUserLogin.Status = @Statu AND TblUserLogin.Auth = '2'");

OleDbParameter[] oleParameter = new OleDbParameter[1];

oleParameter[0] = new OleDbParameter("@Statu", OleDbType.VarChar);

oleParameter[0].Value = Statu;

return \_query.SelectQuery1(sorgu, oleParameter);

}

public DataTable BorcSorgu1()

{

string sorgu = string.Format("SELECT TblUser.Tc\_No, TblUser.Ad, TblUser.Soyad, TblUser.Telefon," +

" TblUser.Mail, SUM(TblOdemeler.BorcMiktar) AS ToplamBorc FROM ((TblUser INNER JOIN TblUserLogin" +

" ON TblUser.Tc\_No = TblUserLogin.Tc\_No) INNER JOIN TblOdemeler ON TblUser.Tc\_No = TblOdemeler.Tc\_No)" +

" WHERE TblUserLogin.Auth = '2' GROUP BY TblUser.Tc\_No, TblUser.Ad, TblUser.Soyad, TblUser.Telefon, TblUser.Mail");

return \_query.SelectQuery2(sorgu);

}

public DataTable BorcSorgu2(bool borcBilgi)

{

if (borcBilgi)

{

string sorgu = string.Format("SELECT TblUser.Tc\_No, TblUser.Ad, TblUser.Soyad, TblUser.Telefon, TblUser.Mail," +

" SUM(IIf(IsNull(TblOdemeler.BorcMiktar), 0, TblOdemeler.BorcMiktar)) AS ToplamBorc FROM " +

"(TblUser INNER JOIN TblUserLogin ON TblUser.Tc\_No = TblUserLogin.Tc\_No) LEFT JOIN TblOdemeler ON TblUser.Tc\_No " +

"= TblOdemeler.Tc\_No WHERE TblUserLogin.Auth = '2' GROUP BY TblUser.Tc\_No, TblUser.Ad, TblUser.Soyad, TblUser.Telefon," +

" TblUser.Mail HAVING SUM(IIf(IsNull(TblOdemeler.BorcMiktar), 0, TblOdemeler.BorcMiktar)) > 0");

return \_query.SelectQuery2(sorgu);

}

else

{

string sorgu = string.Format("SELECT TblUser.Tc\_No, TblUser.Ad, TblUser.Soyad, TblUser.Telefon," +

" TblUser.Mail, "+"SUM(IIf(IsNull(TblOdemeler.BorcMiktar), 0, TblOdemeler.BorcMiktar))" +

" AS ToplamBorc FROM" + " (TblUser INNER JOIN TblUserLogin ON TblUser.Tc\_No = TblUserLogin.Tc\_No)" +

" LEFT JOIN TblOdemeler ON TblUser.Tc\_No" +" = TblOdemeler.Tc\_No WHERE TblUserLogin.Auth = '2' GROUP BY TblUser.Tc\_No," +

" TblUser.Ad, TblUser.Soyad, TblUser.Telefon," +

" TblUser.Mail HAVING SUM(IIf(IsNull(TblOdemeler.BorcMiktar), 0, TblOdemeler.BorcMiktar))" +

" = 0 OR SUM(TblOdemeler.BorcMiktar)" +" IS NULL");

OleDbParameter[] oleParameter = new OleDbParameter[1];

oleParameter[0] = new OleDbParameter("@BorcBilgi", OleDbType.Boolean);

oleParameter[0].Value = borcBilgi;

return \_query.SelectQuery1(sorgu, oleParameter);

}

Bu metod, müşteri yetkisine (**Auth = '2'**) sahip olan kullanıcıların belirli bilgilerini içeren bir **DataTable** döndürmek üzere kullanılır.

SQL sorgusu şu adımları içerir:

1. **TblUser** ve **TblUserLogin** tabloları arasında bir iç birleştirme (INNER JOIN) yapılır.
2. **TblUser.Tc\_No**, **TblUser.Ad**, **TblUser.Soyad**, **TblUser.Cinsiyet**, **TblUser.Kan\_Grup**, **TblUser.Telefon**, **TblUser.Mail**, **TblUser.Adres** sütunları seçilir.
3. **TblUserLogin.Auth** sütunu "2" (müşteri) olan kullanıcıları sınırlar.

Bu sayede, sorgu, müşteri yetkisine sahip kullanıcıların belirli bilgilerini içeren bir **DataTable** nesnesini döndürmek üzere **\_query.SelectQuery2** metodunu kullanır.

Bu metod, belirli bir kullanıcı durumuna (**Statu** parametresi) ve müşteri yetkisine (**Auth = '2'**) sahip olan kullanıcıların belirli bilgilerini içeren bir **DataTable** döndürmek üzere kullanılır.

SQL sorgusu şu adımları içerir:

1. **TblUser** ve **TblUserLogin** tabloları arasında bir iç birleştirme (INNER JOIN) yapılır.
2. **TblUser.Tc\_No**, **TblUser.Ad**, **TblUser.Soyad**, **TblUser.Cinsiyet**, **TblUser.Kan\_Grup**, **TblUser.Telefon**, **TblUser.Mail**, **TblUser.Adres** sütunları seçilir.
3. **TblUserLogin.Status** sütunu, metoda geçirilen **Statu** değeri ile eşleşen kullanıcıları sınırlar.
4. **TblUserLogin.Auth** sütunu "2" (müşteri) olan kullanıcıları sınırlar.
5. **@Statu** parametresi, SQL sorgusunda kullanılmak üzere **OleDbParameter** ile oluşturulur.

Bu sayede, sorgu, belirli bir kullanıcı durumuna ve müşteri yetkisine sahip kullanıcıların belirli bilgilerini içeren bir **DataTable** nesnesini döndürmek üzere **\_query.SelectQuery1** metodunu kullanır.

Bu metod, müşteri yetkisine (**Auth = '2'**) sahip olan kullanıcıların adı, soyadı, telefon numarası, e-posta ve toplam borç miktarını içeren bir **DataTable** döndürmek üzere kullanılır.

SQL sorgusu şu adımları içerir:

1. **TblUser**, **TblUserLogin**, ve **TblOdemeler** tabloları arasında iki iç birleştirme (INNER JOIN) yapılır.
2. **TblUser.Tc\_No**, **TblUser.Ad**, **TblUser.Soyad**, **TblUser.Telefon**, **TblUser.Mail**, ve **SUM(TblOdemeler.BorcMiktar) AS ToplamBorc** sütunları seçilir.
3. İlk iç birleştirmede, **TblUser.Tc\_No = TblUserLogin.Tc\_No** eşleşen kayıtlar alınır.
4. İkinci iç birleştirmede, **TblUser.Tc\_No = TblOdemeler.Tc\_No** eşleşen kayıtlar alınır.
5. **TblUserLogin.Auth** sütunu "2" (müşteri) olan kullanıcıları sınırlar.
6. Sonuçlar, müşteriye ait Tc\_No, Ad, Soyad, Telefon, Mail ve bu müşterinin ödemelerinin toplam borç miktarını gruplar.

Bu sayede, sorgu, müşteri yetkisine sahip kullanıcıların adı, soyadı, telefon numarası, e-posta ve toplam borç miktarını içeren bir **DataTable** nesnesini döndürmek üzere **\_query.SelectQuery2** metodunu kullanır.

**BorcSorgu1 Metodu:**

Bu metod, müşteri yetkisine (**Auth = '2'**) sahip olan kullanıcıların adı, soyadı, telefon numarası, e-posta ve toplam borç miktarını içeren bir **DataTable** döndürmek üzere kullanılır. İki iç birleştirme (**INNER JOIN**) ve bir sol birleştirme (**LEFT JOIN**) kullanarak bu bilgileri elde eder. Borcu olan müşterileri gruplayarak toplam borç miktarını hesaplar ve sıfırdan büyük olanları filtreler.

**BorcSorgu2 Metodu:**

Bu metod, iki farklı durumu ele alır:

1. **borcBilgi == true:**
   * Borcu olan müşterileri seçer.
   * **IIf(IsNull(TblOdemeler.BorcMiktar), 0, TblOdemeler.BorcMiktar)** ifadesi, eğer **TblOdemeler.BorcMiktar** null ise 0, değilse **TblOdemeler.BorcMiktar** değerini alır.
   * Borcu olan müşterileri gruplayarak toplam borç miktarını hesaplar ve sıfırdan büyük olanları filtreler.
2. **borcBilgi == false:**
   * Borcu olmayan müşterileri seçer.
   * **IIf(IsNull(TblOdemeler.BorcMiktar), 0, TblOdemeler.BorcMiktar)** ifadesi, eğer **TblOdemeler.BorcMiktar** null ise 0, değilse **TblOdemeler.BorcMiktar** değerini alır.
   * Borcu olmayan müşterileri gruplayarak toplam borç miktarını hesaplar ve bu miktarın sıfır veya null olduğu durumları seçer.

Bu sayede, sorgular, belirli bir durumda olan veya olmayan müşterilerin adı, soyadı, telefon numarası, e-posta ve toplam borç miktarını içeren bir **DataTable** nesnesini döndürmek üzere **\_query.SelectQuery2** ve **\_query.SelectQuery1** metodlarını kullanır.

public DataTable IlIlceTablo(int IlID)

{

string sorgu = string.Format("SELECT TblUser.Tc\_No, TblUser.Ad, TblUser.Soyad, " +

"TblUser.Cinsiyet, TblUser.Kan\_Grup," +

" TblUser.Telefon, TblUser.Mail, TblUser.Adres FROM TblUser" +

" INNER JOIN TblUserLogin ON TblUser.Tc\_No = TblUserLogin.Tc\_No WHERE TblUserLogin.Status " +

"= 'A' AND TblUserLogin.Auth = '2' AND TblUser.IlID = @Il");

OleDbParameter[] oleParameter = new OleDbParameter[1];

oleParameter[0] = new OleDbParameter("@Il", OleDbType.Integer);

oleParameter[0].Value = IlID;

return \_query.SelectQuery1(sorgu, oleParameter);

}

public DataTable TariheGore(DateTime dt1, DateTime dt2)

{

Console.WriteLine(dt1);

Console.WriteLine(dt2);

string sorgu = string.Format("SELECT TblUser.Tc\_No, TblUser.Ad, TblUser.Soyad, TblOdemeler.OdemeTarih FROM " +

"(TblUser INNER JOIN TblUserLogin ON TblUser.Tc\_No = TblUserLogin.Tc\_No) INNER JOIN TblOdemeler ON TblUser.Tc\_No = " +

"TblOdemeler.Tc\_No WHERE TblUserLogin.Status = 'A' AND TblUserLogin.Auth = '2' AND TblOdemeler.OdemeDurum" +

" = true AND TblOdemeler.OdemeTarih " +">= @dt1 AND TblOdemeler.OdemeTarih <= @dt2");

OleDbParameter[] oleParameter = new OleDbParameter[2];

oleParameter[0] = new OleDbParameter("@dt1", OleDbType.Date);

oleParameter[1] = new OleDbParameter("@dt2", OleDbType.Date);

oleParameter[0].Value = dt1;

oleParameter[1].Value = dt2;

return \_query.SelectQuery1(sorgu, oleParameter);

}

Bu metod, belirli bir İlID'ye ait olan, müşteri yetkisine (**Auth = '2'**) ve aktif bir kullanıcı durumuna (**Status = 'A'**) sahip kullanıcıların adı, soyadı, cinsiyet, kan grubu, telefon, e-posta ve adres bilgilerini içeren bir **DataTable** döndürmek üzere kullanılır.

SQL sorgusu şu adımları içerir:

1. **TblUser** ve **TblUserLogin** tabloları arasında bir iç birleştirme (INNER JOIN) yapılır.
2. **TblUser.Tc\_No**, **TblUser.Ad**, **TblUser.Soyad**, **TblUser.Cinsiyet**, **TblUser.Kan\_Grup**, **TblUser.Telefon**, **TblUser.Mail**, **TblUser.Adres** sütunları seçilir.
3. **TblUserLogin.Status** sütunu "A" (aktif) olan kullanıcıları sınırlar.
4. **TblUserLogin.Auth** sütunu "2" (müşteri) olan kullanıcıları sınırlar.
5. **TblUser.IlID** sütunu, metoda geçirilen **IlID** değeri ile eşleşen kullanıcıları sınırlar.
6. **@Il** parametresi, SQL sorgusunda kullanılmak üzere **OleDbParameter** ile oluşturulur.

Bu sayede, sorgu, belirli bir İlID'ye ait olan, müşteri yetkisine ve aktif bir kullanıcı durumuna sahip kullanıcıların belirli bilgilerini içeren bir **DataTable** nesnesini döndürmek üzere **\_query.SelectQuery1** metodunu kullanır.

Bu metod, belirli bir tarih aralığındaki ödemeleri içeren kullanıcı bilgilerini içeren bir **DataTable** döndürmek üzere kullanılır.

1. **TblUser**, **TblUserLogin**, ve **TblOdemeler** tabloları arasında iç içe iki iç birleştirme (INNER JOIN) yapılır.
2. Seçilen sütunlar:
   * **TblUser.Tc\_No**
   * **TblUser.Ad**
   * **TblUser.Soyad**
   * **TblOdemeler.OdemeTarih**
3. Sorgu şu koşulları içerir:
   * **TblUserLogin.Status** sütunu "A" (aktif) olan kullanıcıları sınırlar.
   * **TblUserLogin.Auth** sütunu "2" (müşteri) olan kullanıcıları sınırlar.
   * **TblOdemeler.OdemeDurum** sütunu **true** olan ödemeleri sınırlar.
   * **TblOdemeler.OdemeTarih** sütunu, metoda geçirilen **dt1** ve **dt2** tarihleri arasında olan ödemeleri sınırlar.
4. **@dt1** ve **@dt2** parametreleri, SQL sorgusunda kullanılmak üzere **OleDbParameter** ile oluşturulur.

Bu sayede, sorgu belirli bir tarih aralığındaki, müşteri yetkisine ve aktif bir kullanıcı durumuna sahip kullanıcıların ödeme bilgilerini içeren bir **DataTable** nesnesini döndürmek üzere **\_query.SelectQuery1** metodunu kullanır.

* + 1. **DalUser**

using System;

using System.Data;

using System.Data.OleDb;

using DataAccessLayer.DbConnections;

namespace DataAccessLayer

{

public class DalUser

{

private DbQuery query;

public DalUser()

{

query = new DbQuery();

}

public DataTable IlCekme()

{

string sorgu = string.Format("Select IlID,IlAdi FROM Il ORDER BY IlID");

return query.SelectQuery2(sorgu);

}

public DataTable IlceCekme(int ilID)

{

string sorgu = string.Format("Select IlceID,IlceAdi FROM IlIlce WHERE IlID = @IlID ORDER BY IlceID");

OleDbParameter[] oleParameter = new OleDbParameter[1];

oleParameter[0] = new OleDbParameter("@IlID", OleDbType.Integer);

oleParameter[0].Value = ilID;

return query.SelectQuery1(sorgu, oleParameter);

}

public DataTable KullaniciTablo(string Auth)

{

string sorgu = string.Format("SELECT TblUser.Tc\_No, TblUser.Ad, TblUser.Soyad, TblUser.Cinsiyet, TblUser.Kan\_Grup, TblUser.Telefon, TblUser.Mail, TblUser.Adres, TblUser.IlID, TblUser.IlceID " +

"FROM TblUser " +

"INNER JOIN TblUserLogin ON TblUser.Tc\_No = TblUserLogin.Tc\_No " +

"WHERE TblUserLogin.Auth = '{0}' " +

"ORDER BY TblUser.Tc\_No", Auth);

return query.SelectQuery2(sorgu);

}

Bu metod, bir veritabanındaki "Il" adlı tablodan, "IlID" ve "IlAdi" sütunlarını seçerek sıralı bir şekilde çekme işlemini gerçekleştirir. Bu işlem sonucunda elde edilen verileri içeren bir **DataTable** nesnesini döndürür. Bu **DataTable** verisi, genellikle bir Windows Forms uygulamasında veya başka bir uygulama arayüzünde kullanıcıya belirli bir il seçme veya listeleme imkanı sağlamak amacıyla kullanılır.

Bu metod, veritabanındaki "IlIlce" adlı tablodan, belirli bir "IlID" değerine sahip olan ilçeleri seçerek sıralı bir şekilde çekme işlemini gerçekleştirir. Bu işlem sonucunda elde edilen verileri içeren bir **DataTable** nesnesini döndürür. Bu **DataTable** verisi genellikle bir Windows Forms uygulamasında veya başka bir uygulama arayüzünde, belirli bir ilin ilçelerini kullanıcıya göstermek amacıyla kullanılır.  
  
Bu metod, belirli bir yetki seviyesine sahip kullanıcıları veritabanından seçer ve bu kullanıcıların ilgili bilgilerini içeren bir **DataTable** nesnesini döndürür. SQL sorgusunda, "TblUser" ve "TblUserLogin" tablolarını iç içe birleştirir (INNER JOIN) ve "Auth" parametresine göre filtreleme yapar. Yani, bu metod belirli bir yetki seviyesine sahip kullanıcıları çekmek için kullanılır. Son olarak, kullanıcıları "Tc\_No" sütununa göre sıralar ve elde edilen sonucu **DataTable** olarak geri döndürür.

public bool KullaniciEkle(string Tc\_No, string Ad, string Soyad, string Cinsiyet, string Kan\_Grup, string Telefon, string Mail, string Adres, int IlID, int IlceID)

{

string sorgu = string.Format("INSERT INTO TblUser (Tc\_No, Ad, Soyad, Cinsiyet, Kan\_Grup, Telefon, Mail, Adres, IlID, IlceID) VALUES (@Tc\_No, @Ad, @Soyad, @Cinsiyet, @Kan\_Grup, @Telefon, @Mail, @Adres, @IlID, @IlceID)");

OleDbParameter[] oleParameter = new OleDbParameter[10];

oleParameter[0] = new OleDbParameter("@Tc\_No", OleDbType.WChar);

oleParameter[1] = new OleDbParameter("@Ad", OleDbType.WChar);

oleParameter[2] = new OleDbParameter("@Soyad", OleDbType.WChar);

oleParameter[3] = new OleDbParameter("@Cinsiyet", OleDbType.WChar);

oleParameter[4] = new OleDbParameter("@Kan\_Grup", OleDbType.WChar);

oleParameter[5] = new OleDbParameter("@Telefon", OleDbType.WChar);

oleParameter[6] = new OleDbParameter("@Mail", OleDbType.WChar);

oleParameter[7] = new OleDbParameter("@Adres", OleDbType.WChar);

oleParameter[8] = new OleDbParameter("@IlID", OleDbType.Integer);

oleParameter[9] = new OleDbParameter("@IlceID", OleDbType.Integer);

oleParameter[0].Value = Convert.ToString(Tc\_No);

oleParameter[1].Value = Convert.ToString(Ad);

oleParameter[2].Value = Convert.ToString(Soyad);

oleParameter[3].Value = Convert.ToString(Cinsiyet);

oleParameter[4].Value = Convert.ToString(Kan\_Grup);

oleParameter[5].Value = Convert.ToString(Telefon);

oleParameter[6].Value = Convert.ToString(Mail);

oleParameter[7].Value = Convert.ToString(Adres);

oleParameter[8].Value = IlID;

oleParameter[9].Value = IlceID;

return query.InsertQuery1(sorgu, oleParameter);

}

public bool KullaniciSil(string Tc\_No)

{

string sorgu = string.Format("DELETE FROM TblUser WHERE Tc\_No = @Tc\_No");

OleDbParameter[] oleParameter = new OleDbParameter[1];

oleParameter[0] = new OleDbParameter("@Tc\_No", OleDbType.WChar);

oleParameter[0].Value = Convert.ToString(Tc\_No);

return query.DeleteQuery1(sorgu, oleParameter);

}

Bu metod, yeni bir kullanıcıyı "TblUser" tablosuna eklemek için kullanılır. SQL sorgusunda **INSERT INTO** kullanılarak yeni bir kayıt eklenir. Parametreler, sorguda kullanılmadan önce parametrik olarak belirlenmiştir, bu da SQL enjeksiyonu saldırılarına karşı bir koruma sağlar. Her bir parametre, metodun aldığı parametre değerleriyle uyumlu bir şekilde belirlenir. Son olarak, **query.InsertQuery1** metodu ile SQL sorgusu ve parametreleri kullanılarak veritabanına ekleme işlemi gerçekleştirilir. Ekleme işlemi başarılıysa **true**, aksi takdirde **false** değeri döndürülür.  
  
Bu metod, belirli bir Tc\_No'ya sahip kullanıcıyı "TblUser" tablosundan silmek için kullanılır. SQL sorgusunda **DELETE FROM** kullanılarak belirtilen koşula uyan kayıtlar silinir. Parametreler, sorguda kullanılmadan önce parametrik olarak belirlenmiştir, bu sayede SQL enjeksiyonu saldırılarına karşı bir koruma sağlanmış olur. Silme işlemi, **query.DeleteQuery1** metodu ile SQL sorgusu ve parametreleri kullanılarak gerçekleştirilir. Eğer silme işlemi başarılıysa **true**, aksi takdirde **false** değeri döndürülür.

public bool KullaniciGuncelle(string Tc\_No, string Ad, string SoyAd, string Telefon, string Mail, string Adres, int IlID, int IlceID)

{

string sorgu = string.Format("UPDATE TblUser SET Ad = @Ad, Soyad = @Soyad, Telefon = @Telefon, Mail = @Mail, Adres = @Adres, IlID = @IlID, IlceID = @IlceID WHERE Tc\_No = @Tc\_No");

OleDbParameter[] oleParameters = new OleDbParameter[8];

oleParameters[0] = new OleDbParameter("@Ad", OleDbType.WChar);

oleParameters[1] = new OleDbParameter("@Soyad", OleDbType.WChar);

oleParameters[2] = new OleDbParameter("@Telefon", OleDbType.WChar);

oleParameters[3] = new OleDbParameter("@Mail", OleDbType.WChar);

oleParameters[4] = new OleDbParameter("@Adres", OleDbType.WChar);

oleParameters[5] = new OleDbParameter("@IlID", OleDbType.Integer);

oleParameters[6] = new OleDbParameter("@IlceID", OleDbType.Integer);

oleParameters[7] = new OleDbParameter("@Tc\_No", OleDbType.WChar);

oleParameters[0].Value = Convert.ToString(Ad);

oleParameters[1].Value = Convert.ToString(SoyAd);

oleParameters[2].Value = Convert.ToString(Telefon);

oleParameters[3].Value = Convert.ToString(Mail);

oleParameters[4].Value = Convert.ToString(Adres);

oleParameters[5].Value = Convert.ToString(IlID);

oleParameters[6].Value = Convert.ToString(IlceID);

oleParameters[7].Value = Convert.ToString(Tc\_No);

return query.UpdateQuery1(sorgu, oleParameters);

}

public DataTable TcArama(string Tc\_No)

{

string sorgu = string.Format("SELECT Tc\_No,Ad,Soyad,Cinsiyet,Kan\_Grup,Telefon,Mail,Adres,IlID,IlceID FROM TblUser WHERE Tc\_No=@Tc\_No");

OleDbParameter[] oleParameters = new OleDbParameter[1];

oleParameters[0] = new OleDbParameter("@Tc\_No", OleDbType.WChar);

oleParameters[0].Value = Convert.ToString(Tc\_No);

return query.SelectQuery1(sorgu,oleParameters);

}

public DataTable AdSoyadArama(string Ad, string Soyad)

{

string sorgu = string.Format("SELECT Tc\_No,Ad,Soyad,Cinsiyet,Kan\_Grup,Telefon,Mail,Adres,IlID,IlceID FROM TblUser WHERE Ad=@Ad AND Soyad=@Soyad");

OleDbParameter[] oleParameters = new OleDbParameter[2];

oleParameters[0] = new OleDbParameter("@Ad", OleDbType.WChar);

oleParameters[1] = new OleDbParameter("@Soyad", OleDbType.WChar);

oleParameters[0].Value = Convert.ToString(Ad);

oleParameters[1].Value = Convert.ToString(Soyad);

return query.SelectQuery1(sorgu, oleParameters);

}

public DataTable TelArama(string Tel)

{

string sorgu = string.Format("SELECT Tc\_No,Ad,Soyad,Cinsiyet,Kan\_Grup,Telefon,Mail,Adres,IlID,IlceID FROM TblUser WHERE Telefon=@Telefon");

OleDbParameter[] oleParameters = new OleDbParameter[1];

oleParameters[0] = new OleDbParameter("@Telefon", OleDbType.WChar);

oleParameters[0].Value = Convert.ToString(Tel);

return query.SelectQuery1(sorgu, oleParameters);

}

Bu metod, belirli bir Tc\_No'ya sahip kullanıcının bilgilerini güncellemek için kullanılır. SQL sorgusu **UPDATE** komutu kullanılarak, "TblUser" tablosundaki belirli bir kaydın alanları yeniden değerlendirilir. Parametreler, SQL sorgusundaki ilgili yerlere geçmeden önce parametrik olarak belirlenir. Bu, SQL enjeksiyonu saldırılarına karşı bir güvenlik önlemidir. Güncelleme işlemi, **\_query.UpdateQuery1** metodu ile gerçekleştirilir. Eğer güncelleme işlemi başarılıysa **true**, aksi takdirde **false** değeri döndürülür.

Bu metod, belirli bir Tc\_No'ya sahip kullanıcının bilgilerini getirmek için kullanılır. SQL sorgusu **SELECT** komutu kullanılarak, "TblUser" tablosundan belirli bir Tc\_No'ya sahip olan kaydın bilgileri çekilir. Parametreler, SQL sorgusundaki ilgili yerlere geçmeden önce parametrik olarak belirlenir. Bu, SQL enjeksiyonu saldırılarına karşı bir güvenlik önlemidir. Veritabanında bu kritere uyan kayıtlar, **\_query.SelectQuery1** metodu ile seçilir ve bir **DataTable** olarak geri döndürülür.

Bu metod, belirli bir Ad ve Soyad'a sahip kullanıcının bilgilerini getirmek için kullanılır. SQL sorgusu **SELECT** komutu kullanılarak, "TblUser" tablosundan belirli bir Ad ve Soyad'a sahip olan kaydın bilgileri çekilir. Parametreler, SQL sorgusundaki ilgili yerlere geçmeden önce parametrik olarak belirlenir. Bu, SQL enjeksiyonu saldırılarına karşı bir güvenlik önlemidir. Veritabanında bu kritere uyan kayıtlar, **\_query.SelectQuery1** metodu ile seçilir ve bir **DataTable** olarak geri döndürülür.

Bu metod, belirli bir telefon numarasına sahip kullanıcının bilgilerini getirmek için kullanılır. SQL sorgusu **SELECT** komutu kullanılarak, "TblUser" tablosundan belirli bir telefon numarasına sahip olan kaydın bilgileri çekilir. Parametreler, SQL sorgusundaki ilgili yerlere geçmeden önce parametrik olarak belirlenir. Bu, SQL enjeksiyonu saldırılarına karşı bir güvenlik önlemidir. Veritabanında bu kritere uyan kayıtlar, **\_query.SelectQuery1** metodu ile seçilir ve bir **DataTable** olarak geri döndürülür.

public DataTable MailArama(string Mail)

{

string sorgu = string.Format("SELECT Tc\_No,Ad,Soyad,Cinsiyet,Kan\_Grup,Telefon,Mail,Adres,IlID,IlceID FROM TblUser WHERE Mail=@Mail");

OleDbParameter[] oleParameters = new OleDbParameter[1];

oleParameters[0] = new OleDbParameter("@Mail", OleDbType.WChar);

oleParameters[0].Value = Convert.ToString(Mail);

return query.SelectQuery1(sorgu, oleParameters);

}

public DataTable AdGetir(string Tc\_No)

{

string sorgu = string.Format("SELECT Ad FROM TblUser WHERE Tc\_No=@Tc\_No");

OleDbParameter[] oleParameters = new OleDbParameter[1];

oleParameters[0] = new OleDbParameter("@Tc\_No", OleDbType.WChar);

oleParameters[0].Value = Convert.ToString(Tc\_No);

return query.SelectQuery1(sorgu, oleParameters);

}

public DataTable BorcGetirBos(string Tc\_No)

{

string sorgu = "SELECT A.AidatAy, A.AidatYil, O.BorcMiktar, O.SonOdemeTarih FROM (TblOdemeler O INNER JOIN TblAidat A ON O.AidatID =" +

" A.AidatID) WHERE O.Tc\_No = @tcNo and O.OdemeDurum = @odeme";

OleDbParameter[] oleParameter = new OleDbParameter[2];

oleParameter[0] = new OleDbParameter("@tcNo", OleDbType.WChar);

oleParameter[1] = new OleDbParameter("@odeme", OleDbType.Boolean);

oleParameter[0].Value = Tc\_No;

oleParameter[1].Value = false;

return query.SelectQuery1(sorgu, oleParameter); ;

}

public DataTable BorcGetirYilAy(string Tc\_No, string Yil, string Ay)

{

string sorgu = "SELECT A.AidatAy, A.AidatYil, O.BorcMiktar, O.SonOdemeTarih FROM (TblOdemeler O INNER JOIN TblAidat A ON O.AidatID = A.AidatID)" +

" WHERE O.Tc\_No = @tcNo and O.OdemeDurum = @odeme and A.AidatYil = @yil and A.AidatAy = @ay";

OleDbParameter[] oleParameter = new OleDbParameter[4];

oleParameter[0] = new OleDbParameter("@tcNo", OleDbType.WChar);

oleParameter[1] = new OleDbParameter("@odeme", OleDbType.Boolean);

oleParameter[2] = new OleDbParameter("@yil", OleDbType.WChar);

oleParameter[3] = new OleDbParameter("@ay", OleDbType.WChar);

oleParameter[0].Value = Tc\_No;

oleParameter[1].Value = false;

oleParameter[2].Value = Yil;

oleParameter[3].Value = Ay;

return query.SelectQuery1(sorgu, oleParameter);

}

Bu metod, belirli bir e-posta adresine sahip kullanıcının bilgilerini getirmek için kullanılır. SQL sorgusu **SELECT** komutu kullanılarak, "TblUser" tablosundan belirli bir e-posta adresine sahip olan kaydın bilgileri çekilir. Parametreler, SQL sorgusundaki ilgili yerlere geçmeden önce parametrik olarak belirlenir. Bu, SQL enjeksiyonu saldırılarına karşı bir güvenlik önlemidir. Veritabanında bu kritere uyan kayıtlar, **\_query.SelectQuery1** metodu ile seçilir ve bir **DataTable** olarak geri döndürülür.

Bu metod, belirli bir T.C. Kimlik Numarasına sahip kullanıcının adını getirmek için kullanılır. SQL sorgusu, "TblUser" tablosundan belirli bir T.C. Kimlik Numarasına sahip olan kaydın adını çeker. Parametreler, SQL sorgusundaki ilgili yerlere geçmeden önce parametrik olarak belirlenir. Bu, SQL enjeksiyonu saldırılarına karşı bir güvenlik önlemidir. Veritabanında bu kritere uyan kayıtlar, **\_query.SelectQuery1** metodu ile seçilir ve bir **DataTable** olarak geri döndürülür.  
Bu metod, belirli bir T.C. Kimlik Numarasına sahip olan kullanıcının ödenmemiş aidat borçlarını getirmek için kullanılır. SQL sorgusu, "TblOdemeler" ve "TblAidat" tablolarını birleştirir ve belirli bir T.C. Kimlik Numarasına ve ödenmemiş durumda olan borçları içerir.

Sorgu şu bilgileri getirir:

* **A.AidatAy**: Aidatın ait olduğu ay.
* **A.AidatYil**: Aidatın ait olduğu yıl.
* **O.BorcMiktar**: Borç miktarı.
* **O.SonOdemeTarih**: Son ödeme tarihi.

Parametreler (**@tcNo** ve **@odeme**), SQL sorgusundaki ilgili yerlere geçmeden önce parametrik olarak belirlenir. Bu, SQL enjeksiyonu saldırılarına karşı bir güvenlik önlemidir. Veritabanında bu kritere uyan kayıtlar, **\_query.SelectQuery1** metodu ile seçilir ve bir **DataTable** olarak geri döndürülür.

Bu metod, belirli bir T.C. Kimlik Numarasına, yıl ve ay parametrelerine sahip olan kullanıcının belirli bir yıl ve ay için ödenmemiş aidat borçlarını getirmek için kullanılır. SQL sorgusu, "TblOdemeler" ve "TblAidat" tablolarını birleştirir ve belirli bir T.C. Kimlik Numarasına, belirli bir yıl ve ay, ödenmemiş durumda olan borçları içerir.

Sorgu şu bilgileri getirir:

* **A.AidatAy**: Aidatın ait olduğu ay.
* **A.AidatYil**: Aidatın ait olduğu yıl.
* **O.BorcMiktar**: Borç miktarı.
* **O.SonOdemeTarih**: Son ödeme tarihi.

Parametreler (**@tcNo**, **@odeme**, **@yil** ve **@ay**), SQL sorgusundaki ilgili yerlere geçmeden önce parametrik olarak belirlenir. Bu, SQL enjeksiyonu saldırılarına karşı bir güvenlik önlemidir. Veritabanında bu kritere uyan kayıtlar, **\_query.SelectQuery1** metodu ile seçilir ve bir **DataTable** olarak geri döndürülür.

public int GetAidatID(string Yil, string Ay)

{

string sorgu = "SELECT AidatID FROM TblAidat WHERE AidatYil = @Yil AND AidatAy = @Ay";

OleDbParameter[] oleDbParameters = new OleDbParameter[2];

oleDbParameters[0] = new OleDbParameter("@Yil", OleDbType.WChar);

oleDbParameters[1] = new OleDbParameter("@Ay", OleDbType.WChar);

oleDbParameters[0].Value = Convert.ToString(Yil);

oleDbParameters[1].Value = Convert.ToString(Ay);

DataTable resultTable = query.SelectQuery1(sorgu, oleDbParameters);

if (resultTable != null && resultTable.Rows.Count > 0)

{

int aidatID = Convert.ToInt32(resultTable.Rows[0]["AidatID"]);

return aidatID;

}

else

{

return -1;

}

}

public bool TutarGuncelle(string Tc\_No, string Yil, string Ay, int tutar)

{

int aidatId = GetAidatID(Yil, Ay);

if (aidatId != -1)

{

string sorgu = "UPDATE TblOdemeler SET BorcMiktar = 0, OdemeDurum = true, OdemeMiktar = @tutar, OdemeTarih =@Gunumuz WHERE Tc\_No = @Tc\_No AND AidatID = @AidatID";

OleDbParameter[] oleParameters = new OleDbParameter[4];

oleParameters[0] = new OleDbParameter("@tutar", OleDbType.Integer);

oleParameters[1] = new OleDbParameter("@Gunumuz", OleDbType.Date);

oleParameters[2] = new OleDbParameter("@Tc\_No", OleDbType.WChar);

oleParameters[3] = new OleDbParameter("@AidatID", OleDbType.Integer);

oleParameters[0].Value = tutar;

oleParameters[1].Value = DateTime.Now.Date;

oleParameters[2].Value = Tc\_No;

oleParameters[3].Value = aidatId;

return query.UpdateQuery1(sorgu, oleParameters);

}

else

{

return false;

}

}

Bu C# fonksiyonu, bir veritabanı sorgusu kullanarak belirli bir yıl (Yil) ve ay (Ay) için bir "AidatID" değeri almayı amaçlamaktadır. İşlevin mantığı şu adımları takip eder:

1. **Sorgu Oluşturma:**
   * SQL sorgusu "TblAidat" adlı bir tablodan "AidatYil" ve "AidatAy" sütunlarına göre bir "AidatID" değeri seçmeyi amaçlar.
2. **Parametre Tanımlama:**
   * Sorguda kullanılacak parametreler, OleDbParameter nesneleri aracılığıyla tanımlanır. Bu parametreler, Yil ve Ay değerlerini alır.
3. **Sorgu Çalıştırma:**
   * Tanımlanan sorgu ve parametrelerle birlikte bir sorgu yürütme işlevi çağrılır (muhtemelen "query.SelectQuery1" fonksiyonu). Bu sorgu, veritabanında belirtilen koşullara uyan verileri getirir.
4. **Sonuç Kontrolü:**
   * Sorgunun sonucu bir DataTable nesnesi içinde saklanır.
   * Eğer sonuçta en az bir satır varsa, bu satırdan "AidatID" değeri alınır ve bu değer geri döndürülür.
   * Eğer sonuç boşsa veya nullsa, fonksiyon -1 değerini döndürür.

Bu işlevin temel amacı, belirli bir yıl ve ay için bir "AidatID" değerini veritabanından çekmektir. Eğer çekilen veri yoksa veya hata oluşmuşsa, fonksiyon -1 değerini döndürür.

Bu C# fonksiyonu, ödemeler tablosunda belirli bir kişiye ait belirli bir aidatın ödeme durumunu güncellemek amacıyla tasarlanmıştır. İşlevin mantığı şu adımları takip eder:

1. İlk olarak, belirli bir yıl ve ay için **GetAidatID** fonksiyonu kullanılarak ilgili aidatın ID'si alınır.
2. Eğer aidat ID'si -1 değilse (yani veritabanında böyle bir aidat bulunuyorsa), ödeme durumu güncellemesi için bir SQL sorgusu oluşturulur.
3. SQL sorgusunda kullanılacak parametreler tanımlanır (tutar, günümüz tarihi, TC kimlik numarası ve aidat ID).
4. Parametre değerleri atanır (tutar, günümüz tarihi, TC kimlik numarası ve aidat ID).
5. Oluşturulan sorgu ve parametrelerle birlikte bir güncelleme sorgusu çalıştırılır (muhtemelen "query.UpdateQuery1" fonksiyonu).
6. Güncelleme işlemi başarılıysa fonksiyon **true** değerini döndürür, aksi takdirde **false** değerini döndürür.

Bu fonksiyonun temel amacı, belirli bir TC kimlik numarası, yıl ve ay için ödemeler tablosunda bir ödeme kaydını güncellemektir. Eğer belirli bir aidat ID bulunamazsa (aidat ID -1 olarak dönerse), fonksiyon **false** değerini döndürür ve güncelleme işlemi gerçekleştirilmez.

* + 1. **DalUserLogin**

using System;

using System.Data.OleDb;

using System.Data;

using DataAccessLayer.DbConnections;

using System.Security.Policy;

namespace DataAccessLayer

{

public class DalUserLogin

{

private DbQuery query;

public DalUserLogin()

{

query = new DbQuery();

}

public DataTable SearchKullaniciAdiveSifre(string kullaniciAdi, string sifre)

{

string sorgu = string.Format("Select kullaniciAd, kullaniciSifre, Auth, Status from TblUserLogin where kullaniciAd = @ad and kullaniciSifre = @sifre");

OleDbParameter[] oleParameter = new OleDbParameter[4];

oleParameter[0] = new OleDbParameter("@ad", OleDbType.WChar);

oleParameter[1] = new OleDbParameter("@sifre", OleDbType.WChar);

oleParameter[2] = new OleDbParameter("@Auth", OleDbType.WChar);

oleParameter[3] = new OleDbParameter("@Status", OleDbType.WChar);

oleParameter[0].Value = Convert.ToString(kullaniciAdi);

oleParameter[1].Value = Convert.ToString(sifre);

return query.SelectQuery1(sorgu, oleParameter);

}

public DataTable SearchTcNo(string kullaniciAdi)

{

string sorgu = string.Format("Select Tc\_No from TblUserLogin where kullaniciAd = @ad");

OleDbParameter[] oleParameter = new OleDbParameter[1];

oleParameter[0] = new OleDbParameter("@ad", OleDbType.WChar);

oleParameter[0].Value = Convert.ToString(kullaniciAdi);

return query.SelectQuery1(sorgu, oleParameter);

}

public DataTable SearchKAd(string Tc\_No)

{

string sorgu = "Select kullaniciAd from TblUserLogin where Tc\_No = @Tc\_No";

OleDbParameter[] oleParameter = new OleDbParameter[1];

oleParameter[0] = new OleDbParameter("@Tc\_No", OleDbType.VarChar);

oleParameter[0].Value = Tc\_No;

return query.SelectQuery1(sorgu, oleParameter);

}

public DataTable SearchParola(string Tc)

{

string sorgu = string.Format("Select kullaniciSifre from TblUserLogin where Tc\_No = @Tc");

OleDbParameter[] oleParameter = new OleDbParameter[1];

oleParameter[0] = new OleDbParameter("@Tc", OleDbType.WChar);

oleParameter[0].Value = Convert.ToString(Tc);

return query.SelectQuery1(sorgu, oleParameter);

}

Bu C# kodu, kullanıcı girişi (login) işlemlerini gerçekleştirmek amacıyla tasarlanmış bir sınıftır. İlgili metotlar, kullanıcı adı ve şifre kontrolleri ile kullanıcı bilgilerini sorgular ve bu bilgileri döndürür. İşte sınıftaki metotların açıklamaları:

**SearchKullaniciAdiveSifre Metodu:**

* **public DataTable SearchKullaniciAdiveSifre(string kullaniciAdi, string sifre)**: Veritabanında kullanıcı adı (**kullaniciAd**) ve şifre (**kullaniciSifre**) ile eşleşen kayıtları sorgular ve ilgili bilgileri döndürür.
  + SQL sorgusunu oluşturur ve **kullaniciAdi**, **sifre** parametreleri ile birlikte **\_query.SelectQuery1** metodu aracılığıyla sorguyu çalıştırır.
  + Parametreler: kullaniciAdi, sifre.
  + Dönen sonuçlar **DataTable** şeklinde olup, **kullaniciAd**, **kullaniciSifre**, **Auth** ve **Status** sütunlarını içerir.

**SearchTcNo Metodu:**

* **public DataTable SearchTcNo(string kullaniciAdi)**: Veritabanında kullanıcı adına (**kullaniciAd**) ait Tc\_No'yu sorgular ve ilgili bilgiyi döndürür.
  + SQL sorgusunu oluşturur ve **kullaniciAdi** parametresi ile birlikte **\_query.SelectQuery1** metodu aracılığıyla sorguyu çalıştırır.
  + Parametre: kullaniciAdi.
  + Dönen sonuçlar **DataTable** şeklinde olup, **Tc\_No** sütununu içerir.

**SearchKAd Metodu:**

* **public DataTable SearchKAd(string Tc\_No)**: Veritabanında Tc\_No'ya (**Tc\_No**) ait kullanıcı adını sorgular ve ilgili bilgiyi döndürür.
  + SQL sorgusunu oluşturur ve **Tc\_No** parametresi ile birlikte **\_query.SelectQuery1** metodu aracılığıyla sorguyu çalıştırır.
  + Parametre: Tc\_No.
  + Dönen sonuçlar **DataTable** şeklinde olup, **kullaniciAd** sütununu içerir.

**SearchParola Metodu:**

* **public DataTable SearchParola(string Tc)**: Veritabanında Tc\_No'ya (**Tc\_No**) ait şifreyi sorgular ve ilgili bilgiyi döndürür.
  + SQL sorgusunu oluşturur ve **Tc** parametresi ile birlikte **\_query.SelectQuery1** metodu aracılığıyla sorguyu çalıştırır.
  + Parametre: Tc.
  + Dönen sonuçlar **DataTable** şeklinde olup, **kullaniciSifre** sütununu içerir.

Bu sınıf, kullanıcı girişiyle ilgili temel bilgileri sorgulamak amacıyla kullanılır. **DalUserLogin** sınıfındaki metotlar, kullanıcı adı ve şifre ile ilgili sorguları gerçekleştirir ve sonuçları **DataTable** formatında döndürür.

public bool UserKullaniciAdBelirle(string Tc,string kullaniciAd, string parola)

{

string \_query = string.Format("INSERT INTO TblUserLogin (Tc\_No,kullaniciAd,kullaniciSifre,Auth) VALUES (@Tc\_No, @KullaniciAd, @Sifre, @Auth)");

OleDbParameter[] oleParameter = new OleDbParameter[4];

oleParameter[0] = new OleDbParameter("@Tc\_No", OleDbType.WChar);

oleParameter[1] = new OleDbParameter("@kullaniciAd", OleDbType.WChar);

oleParameter[2] = new OleDbParameter("@kullaniciSifre", OleDbType.WChar);

oleParameter[3] = new OleDbParameter("@Auth", OleDbType.WChar);

oleParameter[0].Value = Convert.ToString(Tc);

oleParameter[1].Value = Convert.ToString(kullaniciAd);

oleParameter[2].Value = Convert.ToString(parola);

oleParameter[3].Value = Convert.ToString("2");

return query.InsertQuery1(\_query, oleParameter);

}

public bool UserKullaniciAdSifreDegistir1(string Tc\_No, string KAd, string Sifre, string Statu)

{

string \_query = "UPDATE TblUserLogin SET kullaniciAd = @KAd, kullaniciSifre = @Sifre, Status=@Statu WHERE Tc\_No = @Tc\_No";

OleDbParameter[] oleParameter = new OleDbParameter[4];

oleParameter[0] = new OleDbParameter("@KAd", OleDbType.VarChar);

oleParameter[1] = new OleDbParameter("@Sifre", OleDbType.VarChar);

oleParameter[2] = new OleDbParameter("@Statu", OleDbType.VarChar);

oleParameter[3] = new OleDbParameter("@Tc\_No", OleDbType.VarChar);

oleParameter[0].Value = KAd;

oleParameter[1].Value = Sifre;

oleParameter[2].Value = Statu;

oleParameter[3].Value = Tc\_No;

return query.UpdateQuery1(\_query, oleParameter);

}

public bool UserKullaniciAdSifreDegistir2(string Tc\_No, string KAd, string Sifre)

{

string \_query = "UPDATE TblUserLogin SET kullaniciAd = @KAd, kullaniciSifre = @Sifre WHERE Tc\_No = @Tc\_No";

OleDbParameter[] oleParameter = new OleDbParameter[3];

oleParameter[0] = new OleDbParameter("@KAd", OleDbType.VarChar);

oleParameter[1] = new OleDbParameter("@Sifre", OleDbType.VarChar);

oleParameter[2] = new OleDbParameter("@Tc\_No", OleDbType.VarChar);

oleParameter[0].Value = KAd;

oleParameter[1].Value = Sifre;

oleParameter[2].Value = Tc\_No;

return query.UpdateQuery1(\_query, oleParameter);

}

Bu C# kodu, kullanıcıyla ilgili işlemleri gerçekleştirmek için tasarlanmış bir sınıf içeriyor. İlgili metotlar, kullanıcı adı ve şifre belirleme, kullanıcı adı ve şifre değiştirme gibi işlemleri gerçekleştirir. İşte bu metotların açıklamaları:

**UserKullaniciAdBelirle Metodu:**

* **public bool UserKullaniciAdBelirle(string Tc, string kullaniciAd, string parola)**: Kullanıcının Tc\_No, kullanıcı adı (**kullaniciAd**), şifre (**parola**) ve yetki (**Auth**) bilgilerini veritabanına ekler.
  + SQL sorgusunu oluşturur ve **Tc**, **kullaniciAd**, **parola** ve **Auth** parametreleri ile birlikte **\_query.InsertQuery1** metodu aracılığıyla sorguyu çalıştırır.
  + Parametreler: Tc, kullaniciAd, parola.
  + İşlem başarılıysa **true**, aksi takdirde **false** döndürür.

**UserKullaniciAdSifreDegistir1 Metodu:**

* **public bool UserKullaniciAdSifreDegistir1(string Tc\_No, string KAd, string Sifre, string Statu)**: Kullanıcının Tc\_No'suna göre kullanıcı adı (**KAd**), şifre (**Sifre**) ve kullanıcı durumu (**Statu**) bilgilerini veritabanında günceller.
  + SQL sorgusunu oluşturur ve **KAd**, **Sifre**, **Statu** ve **Tc\_No** parametreleri ile birlikte **\_query.UpdateQuery1** metodu aracılığıyla sorguyu çalıştırır.
  + Parametreler: Tc\_No, KAd, Sifre, Statu.
  + İşlem başarılıysa **true**, aksi takdirde **false** döndürür.

**UserKullaniciAdSifreDegistir2 Metodu:**

* **public bool UserKullaniciAdSifreDegistir2(string Tc\_No, string KAd, string Sifre)**: Kullanıcının Tc\_No'suna göre kullanıcı adı (**KAd**) ve şifre (**Sifre**) bilgilerini veritabanında günceller.
  + SQL sorgusunu oluşturur ve **KAd**, **Sifre** ve **Tc\_No** parametreleri ile birlikte **\_query.UpdateQuery1** metodu aracılığıyla sorguyu çalıştırır.
  + Parametreler: Tc\_No, KAd, Sifre.
  + İşlem başarılıysa **true**, aksi takdirde **false** döndürür.

Bu sınıf, kullanıcı işlemleri için gerekli veritabanı etkileşimlerini gerçekleştiren metotları içerir. Bu işlemler arasında yeni bir kullanıcı ekleme, kullanıcı bilgilerini güncelleme gibi temel kullanıcı yönetimi işlemleri bulunmaktadır.

public bool UserKullaniciAdDegistir1(string Tc\_No, string KAd, string statu)

{

string \_query = "UPDATE TblUserLogin SET kullaniciAd = @KAd, Status=@Statu WHERE Tc\_No = @Tc\_No";

OleDbParameter[] oleParameter = new OleDbParameter[3];

oleParameter[0] = new OleDbParameter("@KAd", OleDbType.VarChar);

oleParameter[1] = new OleDbParameter("@Statu", OleDbType.VarChar);

oleParameter[2] = new OleDbParameter("@Tc\_No", OleDbType.VarChar);

oleParameter[0].Value = KAd;

oleParameter[1].Value = statu;

oleParameter[2].Value = Tc\_No;

return query.UpdateQuery1(\_query, oleParameter);

}

public bool UserKullaniciAdDegistir2(string Tc\_No, string KAd)

{

string \_query = "UPDATE TblUserLogin SET kullaniciAd = @KAd WHERE Tc\_No = @Tc\_No";

OleDbParameter[] oleParameter = new OleDbParameter[2];

oleParameter[0] = new OleDbParameter("@KAd", OleDbType.VarChar);

oleParameter[1] = new OleDbParameter("@Tc\_No", OleDbType.VarChar);

oleParameter[0].Value = KAd;

oleParameter[1].Value = Tc\_No;

return query.UpdateQuery1(\_query, oleParameter);

}

public bool UserSifreDegistir1(string Tc\_No, string Sifre, string statu)

{

string \_query = "UPDATE TblUserLogin SET kullaniciSifre = @Sifre, Status=@Statu WHERE Tc\_No = @Tc\_No";

OleDbParameter[] oleParameter = new OleDbParameter[3];

oleParameter[0] = new OleDbParameter("@Sifre", OleDbType.VarChar);

oleParameter[1] = new OleDbParameter("@Statu", OleDbType.VarChar);

oleParameter[2] = new OleDbParameter("@Tc\_No", OleDbType.VarChar);

oleParameter[0].Value = Sifre;

oleParameter[1].Value = statu;

oleParameter[2].Value = Tc\_No;

return query.UpdateQuery1(\_query, oleParameter);

}

public bool UserSifreDegistir2(string Tc\_No, string Sifre)

{

string \_query = "UPDATE TblUserLogin SET kullaniciSifre = @Sifre WHERE Tc\_No = @Tc\_No";

OleDbParameter[] oleParameter = new OleDbParameter[2];

oleParameter[0] = new OleDbParameter("@Sifre", OleDbType.VarChar);

oleParameter[1] = new OleDbParameter("@Tc\_No", OleDbType.VarChar);

oleParameter[0].Value = Sifre;

oleParameter[1].Value = Tc\_No;

return query.UpdateQuery1(\_query, oleParameter);

}

Bu kod bloğu, kullanıcı adı ve şifre değişiklik işlemlerini gerçekleştiren metodları içerir. İlgili metotlar aşağıdaki gibidir:

**UserKullaniciAdDegistir1 Metodu:**

* **public bool UserKullaniciAdDegistir1(string Tc\_No, string KAd, string statu)**: Kullanıcının Tc\_No'suna göre kullanıcı adını (**KAd**) ve kullanıcı durumunu (**statu**) günceller.
  + SQL sorgusunu oluşturur ve **KAd**, **statu** ve **Tc\_No** parametreleri ile birlikte **\_query.UpdateQuery1** metodu aracılığıyla sorguyu çalıştırır.
  + Parametreler: Tc\_No, KAd, statu.
  + İşlem başarılıysa **true**, aksi takdirde **false** döndürür.

**UserKullaniciAdDegistir2 Metodu:**

* **public bool UserKullaniciAdDegistir2(string Tc\_No, string KAd)**: Kullanıcının Tc\_No'suna göre kullanıcı adını (**KAd**) günceller.
  + SQL sorgusunu oluşturur ve **KAd** ve **Tc\_No** parametreleri ile birlikte **\_query.UpdateQuery1** metodu aracılığıyla sorguyu çalıştırır.
  + Parametreler: Tc\_No, KAd.
  + İşlem başarılıysa **true**, aksi takdirde **false** döndürür.

**UserSifreDegistir1 Metodu:**

* **public bool UserSifreDegistir1(string Tc\_No, string Sifre, string statu)**: Kullanıcının Tc\_No'suna göre şifresini (**Sifre**) ve kullanıcı durumunu (**statu**) günceller.
  + SQL sorgusunu oluşturur ve **Sifre**, **statu** ve **Tc\_No** parametreleri ile birlikte **\_query.UpdateQuery1** metodu aracılığıyla sorguyu çalıştırır.
  + Parametreler: Tc\_No, Sifre, statu.
  + İşlem başarılıysa **true**, aksi takdirde **false** döndürür.

**UserSifreDegistir2 Metodu:**

* **public bool UserSifreDegistir2(string Tc\_No, string Sifre)**: Kullanıcının Tc\_No'suna göre şifresini (**Sifre**) günceller.
  + SQL sorgusunu oluşturur ve **Sifre** ve **Tc\_No** parametreleri ile birlikte **\_query.UpdateQuery1** metodu aracılığıyla sorguyu çalıştırır.
  + Parametreler: Tc\_No, Sifre.
  + İşlem başarılıysa **true**, aksi takdirde **false** döndürür.

Bu metotlar, kullanıcının giriş bilgilerini güncelleme işlemlerini gerçekleştiren veritabanı sorgularını içermektedir.

public DataTable AidatBos1(string Tc\_No)

{

string sorgu = "SELECT A.AidatAy, A.AidatYil, A.AidatTutar, O.OdemeDurum, O.OdemeTarih, O.SonOdemeTarih FROM (TblOdemeler O INNER JOIN TblAidat A ON O.AidatID = A.AidatID) WHERE O.Tc\_No = @tcNo";

OleDbParameter[] oleParameter = new OleDbParameter[1];

oleParameter[0] = new OleDbParameter("@tcNo", OleDbType.WChar);

oleParameter[0].Value = Tc\_No;

return query.SelectQuery1(sorgu, oleParameter); ;

}

public DataTable AidatAy1(string Tc\_No, string Ay)

{

string sorgu = "SELECT A.AidatAy, A.AidatYil, A.AidatTutar, O.OdemeDurum, O.OdemeTarih, O.SonOdemeTarih FROM (TblOdemeler O INNER JOIN TblAidat A ON O.AidatID = A.AidatID) WHERE O.Tc\_No = @tcNo and A.AidatAy = @ay";

OleDbParameter[] oleParameter = new OleDbParameter[2];

oleParameter[0] = new OleDbParameter("@tcNo", OleDbType.WChar);

oleParameter[1] = new OleDbParameter("@ay", OleDbType.WChar);

oleParameter[0].Value = Tc\_No;

oleParameter[1].Value = Ay;

return query.SelectQuery1(sorgu, oleParameter);

}

public DataTable AidatYil1(string Tc\_No, string Yil)

{

string sorgu = "SELECT A.AidatAy, A.AidatYil, A.AidatTutar, O.OdemeDurum, O.OdemeTarih, O.SonOdemeTarih FROM (TblOdemeler O INNER JOIN TblAidat A ON O.AidatID = A.AidatID) WHERE O.Tc\_No = @tcNo and A.AidatYil = @yil";

OleDbParameter[] oleParameter = new OleDbParameter[2];

oleParameter[0] = new OleDbParameter("@tcNo", OleDbType.WChar);

oleParameter[1] = new OleDbParameter("@yil", OleDbType.WChar);

oleParameter[0].Value = Tc\_No;

oleParameter[1].Value = Yil;

return query.SelectQuery1(sorgu, oleParameter);

}

public DataTable AidatYilAy1(string Tc\_No, string Yil, string Ay)

{

string sorgu = "SELECT A.AidatAy, A.AidatYil, A.AidatTutar, O.OdemeDurum, O.OdemeTarih, O.SonOdemeTarih FROM (TblOdemeler O INNER JOIN TblAidat A ON O.AidatID = A.AidatID) WHERE O.Tc\_No = @tcNo and A.AidatYil = @yil and A.AidatAy = @ay";

OleDbParameter[] oleParameter = new OleDbParameter[3];

oleParameter[0] = new OleDbParameter("@tcNo", OleDbType.WChar);

oleParameter[1] = new OleDbParameter("@yil", OleDbType.WChar);

oleParameter[2] = new OleDbParameter("@ay", OleDbType.WChar);

oleParameter[0].Value = Tc\_No;

oleParameter[1].Value = Yil;

oleParameter[2].Value = Ay;

return query.SelectQuery1(sorgu, oleParameter);

}

public DataTable AidatBos2(string Tc\_No, bool OdemeDurum)

{

string sorgu = "SELECT A.AidatAy, A.AidatYil, A.AidatTutar, O.OdemeDurum, O.OdemeTarih, O.SonOdemeTarih FROM (TblOdemeler O INNER JOIN TblAidat A ON O.AidatID = A.AidatID) WHERE O.Tc\_No = @tcNo and O.OdemeDurum = @odeme";

OleDbParameter[] oleParameter = new OleDbParameter[2];

oleParameter[0] = new OleDbParameter("@tcNo", OleDbType.WChar);

oleParameter[1] = new OleDbParameter("@odeme", OleDbType.Boolean);

oleParameter[0].Value = Tc\_No;

oleParameter[1].Value = OdemeDurum;

return query.SelectQuery1(sorgu, oleParameter);

}

public DataTable AidatAy2(string Tc\_No, string Ay, bool OdemeDurum)

{

string sorgu = "SELECT A.AidatAy, A.AidatYil, A.AidatTutar, O.OdemeDurum, O.OdemeTarih, O.SonOdemeTarih FROM (TblOdemeler O INNER JOIN TblAidat A ON O.AidatID = A.AidatID) WHERE O.Tc\_No = @tcNo and A.AidatAy = @ay and O.OdemeDurum = @odeme";

OleDbParameter[] oleParameter = new OleDbParameter[3];

oleParameter[0] = new OleDbParameter("@tcNo", OleDbType.WChar);

oleParameter[1] = new OleDbParameter("@ay", OleDbType.WChar);

oleParameter[2] = new OleDbParameter("@odeme", OleDbType.Boolean);

oleParameter[0].Value = Tc\_No;

oleParameter[1].Value = Ay;

oleParameter[2].Value = OdemeDurum;

return query.SelectQuery1(sorgu, oleParameter);

}

public DataTable AidatYil2(string Tc\_No, string Yil, bool OdemeDurum)

{

string sorgu = "SELECT A.AidatAy, A.AidatYil, A.AidatTutar, O.OdemeDurum, O.OdemeTarih, O.SonOdemeTarih FROM (TblOdemeler O INNER JOIN TblAidat A ON O.AidatID = A.AidatID) WHERE O.Tc\_No = @tcNo and A.AidatYil = @yil and O.OdemeDurum = @odeme";

OleDbParameter[] oleParameter = new OleDbParameter[3];

oleParameter[0] = new OleDbParameter("@tcNo", OleDbType.WChar);

oleParameter[1] = new OleDbParameter("@yil", OleDbType.WChar);

oleParameter[2] = new OleDbParameter("@odeme", OleDbType.Boolean);

oleParameter[0].Value = Tc\_No;

oleParameter[1].Value = Yil;

oleParameter[2].Value = OdemeDurum;

return query.SelectQuery1(sorgu, oleParameter);

}

public DataTable AidatYilAy2(string Tc\_No, string Yil, string Ay, bool OdemeDurum)

{

string sorgu = "SELECT A.AidatAy, A.AidatYil, A.AidatTutar, O.OdemeDurum, O.OdemeTarih, O.SonOdemeTarih FROM (TblOdemeler O INNER JOIN TblAidat A ON O.AidatID = A.AidatID) WHERE O.Tc\_No = @tcNo and A.AidatYil = @yil and A.AidatAy = @ay and O.OdemeDurum = @odeme";

OleDbParameter[] oleParameter = new OleDbParameter[4];

oleParameter[0] = new OleDbParameter("@tcNo", OleDbType.WChar);

oleParameter[1] = new OleDbParameter("@yil", OleDbType.WChar);

oleParameter[2] = new OleDbParameter("@ay", OleDbType.WChar);

oleParameter[3] = new OleDbParameter("@odeme", OleDbType.Boolean);

oleParameter[0].Value = Tc\_No;

oleParameter[1].Value = Yil;

oleParameter[2].Value = Ay;

oleParameter[3].Value = OdemeDurum;

return query.SelectQuery1(sorgu, oleParameter);

}

Bu kod bloğu, aidat ödemeleri ile ilgili çeşitli sorguları gerçekleştiren metodları içerir. İlgili metotlar aşağıdaki gibidir:

**AidatBos1 Metodu:**

* **public DataTable AidatBos1(string Tc\_No)**: Belirtilen Tc\_No'ya ait tüm aidat bilgilerini getirir.
  + SQL sorgusunu oluşturur ve **Tc\_No** parametresi ile birlikte **\_query.SelectQuery1** metodu aracılığıyla sorguyu çalıştırır.
  + Parametre: Tc\_No.
  + Sonucu **DataTable** olarak döndürür.

**AidatAy1 Metodu:**

* **public DataTable AidatAy1(string Tc\_No, string Ay)**: Belirtilen Tc\_No'ya ve Ay'a ait aidat bilgilerini getirir.
  + SQL sorgusunu oluşturur ve **Tc\_No** ve **Ay** parametreleri ile birlikte **\_query.SelectQuery1** metodu aracılığıyla sorguyu çalıştırır.
  + Parametreler: Tc\_No, Ay.
  + Sonucu **DataTable** olarak döndürür.

**AidatYil1 Metodu:**

* **public DataTable AidatYil1(string Tc\_No, string Yil)**: Belirtilen Tc\_No'ya ve Yıl'a ait aidat bilgilerini getirir.
  + SQL sorgusunu oluşturur ve **Tc\_No** ve **Yil** parametreleri ile birlikte **\_query.SelectQuery1** metodu aracılığıyla sorguyu çalıştırır.
  + Parametreler: Tc\_No, Yil.
  + Sonucu **DataTable** olarak döndürür.

**AidatYilAy1 Metodu:**

* **public DataTable AidatYilAy1(string Tc\_No, string Yil, string Ay)**: Belirtilen Tc\_No'ya, Yıl'a ve Ay'a ait aidat bilgilerini getirir.
  + SQL sorgusunu oluşturur ve **Tc\_No**, **Yil** ve **Ay** parametreleri ile birlikte **\_query.SelectQuery1** metodu aracılığıyla sorguyu çalıştırır.
  + Parametreler: Tc\_No, Yil, Ay.
  + Sonucu **DataTable** olarak döndürür.

**AidatBos2 Metodu:**

* **public DataTable AidatBos2(string Tc\_No, bool OdemeDurum)**: Belirtilen Tc\_No'ya ve ödeme durumuna (**OdemeDurum**) ait aidat bilgilerini getirir.
  + SQL sorgusunu oluşturur ve **Tc\_No** ve **OdemeDurum** parametreleri ile birlikte **\_query.SelectQuery1** metodu aracılığıyla sorguyu çalıştırır.
  + Parametreler: Tc\_No, OdemeDurum.
  + Sonucu **DataTable** olarak döndürür.

**AidatAy2 Metodu:**

* **public DataTable AidatAy2(string Tc\_No, string Ay, bool OdemeDurum)**: Belirtilen Tc\_No'ya, Ay'a ve ödeme durumuna (**OdemeDurum**) ait aidat bilgilerini getirir.
  + SQL sorgusunu oluşturur ve **Tc\_No**, **Ay** ve **OdemeDurum** parametreleri ile birlikte **\_query.SelectQuery1** metodu aracılığıyla sorguyu çalıştırır.
  + Parametreler: Tc\_No, Ay, OdemeDurum.
  + Sonucu **DataTable** olarak döndürür.

**AidatYil2 Metodu:**

* **public DataTable AidatYil2(string Tc\_No, string Yil, bool OdemeDurum)**: Belirtilen Tc\_No'ya, Yıl'a ve ödeme durumuna (**OdemeDurum**) ait aidat bilgilerini getirir.
  + SQL sorgusunu oluşturur ve **Tc\_No**, **Yil** ve **OdemeDurum** parametreleri ile birlikte **\_query.SelectQuery1** metodu aracılığıyla sorguyu çalıştırır.
  + Parametreler: Tc\_No, Yil, OdemeDurum.
  + Sonucu **DataTable** olarak döndürür.

**AidatYilAy2 Metodu:**

* **public DataTable AidatYilAy2(string Tc\_No, string Yil, string Ay, bool OdemeDurum)**: Belirtilen Tc\_No'ya, Yıl'a, Ay'a ve ödeme durumuna (**OdemeDurum**) ait aidat bilgilerini getirir.
  + SQL sorgusunu oluşturur ve **Tc\_No**, **Yil**, **Ay** ve **OdemeDurum** parametreleri ile birlikte **\_query.SelectQuery1** metodu aracılığıyla sorguyu çalıştırır.
  + Parametreler: Tc\_No, Yil, Ay, OdemeDurum.
  + Sonucu **DataTable** olarak döndürür.

Bu metotlar, aidat ödemeleriyle ilgili belirli tarih ve ödeme durumu filtreleri ile bilgi çekmek için kullanılabilir. Filtre kriterlerine bağlı olarak farklı sorgular yapmaktadır.

* 1. **Bussiness Layer**

using DataAccessLayer;

using System.Data;

namespace BusinessLayer

{

public class BlAidat

{

private DalAidat \_dalAidat;

public BlAidat()

{

\_dalAidat = new DalAidat();

}

public DataTable TabloSorgu()

{

DataTable dataTable = \_dalAidat.AidatTablo();

return dataTable;

}

public bool EklemeSorgu(int AidatID, string AidatYil, string AidatAy, int AidatTutar)

{

bool onay = false;

onay = \_dalAidat.AidatEkle(AidatID, AidatYil, AidatAy, AidatTutar);

return onay;

}

public bool SilmeSorgu(int AidatID)

{

bool onay = false;

onay = \_dalAidat.AidatSil(AidatID);

return onay;

}

public bool GuncellemeSorgu(int AidatID, string AidatYil, string AidatAy, int AidatTutar)

{

bool onay = false;

onay = \_dalAidat.AidatGuncelle(AidatID, AidatYil, AidatAy, AidatTutar);

return onay;

}

public DataTable AidatOdeme1(string Tc\_No)

{

DataTable dataTable = new DataTable();

dataTable = \_dalAidat.AidatOdeme1(Tc\_No);

return dataTable;

}

public DataTable AidatOdeme2(string Tc\_No, string yil)

{

DataTable dataTable = new DataTable();

dataTable = \_dalAidat.AidatOdeme2(Tc\_No,yil);

return dataTable;

}

public DataTable AidatOdeme3(string Tc\_No, string yil, string ay)

{

DataTable dataTable = new DataTable();

dataTable = \_dalAidat.AidatOdeme3(Tc\_No,yil,ay);

return dataTable;

}

Bu kod bloğu, iş katmanında aidat işlemlerini yöneten bir sınıfı (**BlAidat**) içermektedir. Aşağıda, bu sınıfın içerdiği metodlar ve işlevleri bulunmaktadır:

**BlAidat Sınıfı:**

* **\_dalAidat Değişkeni**: **DalAidat** sınıfının bir örneğini içerir. Veritabanı işlemlerini bu sınıf aracılığıyla gerçekleştirir.
* **BlAidat Constructor Metodu**: Sınıfın örneği oluşturulduğunda, **\_dalAidat** örneğini başlatır.

**TabloSorgu Metodu:**

* **public DataTable TabloSorgu()**: Bu metod, aidat tablosundaki tüm kayıtları çekmek için kullanılır. **\_dalAidat.AidatTablo()** metodunu çağırır ve sonucu **DataTable** olarak döndürür.

**EklemeSorgu Metodu:**

* **public bool EklemeSorgu(int AidatID, string AidatYil, string AidatAy, int AidatTutar)**: Belirli bir aidat kaydını eklemek için kullanılır. **\_dalAidat.AidatEkle** metodunu çağırır ve işlemin başarılı olup olmadığını bool olarak döndürür.

**SilmeSorgu Metodu:**

* **public bool SilmeSorgu(int AidatID)**: Belirli bir aidat kaydını silmek için kullanılır. **\_dalAidat.AidatSil** metodunu çağırır ve işlemin başarılı olup olmadığını bool olarak döndürür.

**GuncellemeSorgu Metodu:**

* **public bool GuncellemeSorgu(int AidatID, string AidatYil, string AidatAy, int AidatTutar)**: Belirli bir aidat kaydını güncellemek için kullanılır. **\_dalAidat.AidatGuncelle** metodunu çağırır ve işlemin başarılı olup olmadığını bool olarak döndürür.

**AidatOdeme1, AidatOdeme2, AidatOdeme3 Metodları:**

* **public DataTable AidatOdeme1(string Tc\_No)**
* **public DataTable AidatOdeme2(string Tc\_No, string yil)**
* **public DataTable AidatOdeme3(string Tc\_No, string yil, string ay)**

Bu metodlar, belirli kriterlere göre aidat ödemelerini sorgulamak için kullanılır. İlgili **\_dalAidat** metodlarını çağırarak işlemi gerçekleştirir ve sonuçları **DataTable** olarak döndürür.

public DataTable AidatOdeme4()

{

DataTable dataTable = new DataTable();

dataTable = \_dalAidat.AidatOdeme4();

return dataTable;

}

public DataTable AidatOdeme5(string yil)

{

DataTable dataTable = new DataTable();

dataTable = \_dalAidat.AidatOdeme5(yil);

return dataTable;

}

public DataTable AidatOdeme6(string yil, string ay)

{

DataTable dataTable = new DataTable();

dataTable = \_dalAidat.AidatOdeme6(yil, ay);

return dataTable;

}

public DataTable TcSorgu()

{

DataTable dataTable = new DataTable();

dataTable = \_dalAidat.TcQuery();

return dataTable;

}

public bool BorcEkle(string Tc\_No, int AidatID, int OdemeMiktar)

{

bool onay = false;

onay = \_dalAidat.BorcEkleQuery(Tc\_No,AidatID,OdemeMiktar);

return onay;

}

public bool BorcGuncelle(string Tc\_No, int AidatID, int YeniMiktar)

{

bool onay = false;

onay = \_dalAidat.BorcGuncelleQuery(Tc\_No,AidatID,YeniMiktar);

return onay;

}

Bu kod bloğu, **BlAidat** sınıfına ek olarak, aidat ödemeleri ve borç işlemlerini yöneten yeni metodları içermektedir. Aşağıda, bu yeni metodlar ve işlevleri açıklanmaktadır:

**Yeni Metodlar:**

**AidatOdeme4 Metodu:**

* **public DataTable AidatOdeme4()**: Bu metod, tüm aidat ödemelerini sorgulamak için kullanılır. **\_dalAidat.AidatOdeme4** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**AidatOdeme5 Metodu:**

* **public DataTable AidatOdeme5(string yil)**: Belirli bir yılda yapılan aidat ödemelerini sorgulamak için kullanılır. **\_dalAidat.AidatOdeme5** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**AidatOdeme6 Metodu:**

* **public DataTable AidatOdeme6(string yil, string ay)**: Belirli bir yıl ve ayda yapılan aidat ödemelerini sorgulamak için kullanılır. **\_dalAidat.AidatOdeme6** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**TcSorgu Metodu:**

* **public DataTable TcSorgu()**: Tüm kullanıcıların TC kimlik numaralarını sorgulamak için kullanılır. **\_dalAidat.TcQuery** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**BorcEkle Metodu:**

* **public bool BorcEkle(string Tc\_No, int AidatID, int OdemeMiktar)**: Belirli bir kişiye aidat borcu eklemek için kullanılır. **\_dalAidat.BorcEkleQuery** metodunu çağırır ve işlemin başarılı olup olmadığını bool olarak döndürür.

**BorcGuncelle Metodu:**

* **public bool BorcGuncelle(string Tc\_No, int AidatID, int YeniMiktar)**: Belirli bir kişinin aidat borcunu güncellemek için kullanılır. **\_dalAidat.BorcGuncelleQuery** metodunu çağırır ve işlemin başarılı olup olmadığını bool olarak döndürür.

Bu metodlar, **BlAidat** sınıfını daha kapsamlı ve kullanışlı hale getirerek, aidat ve ödeme işlemleriyle ilgili geniş bir işlevsellik sunar.

using DataAccessLayer;

using System;

using System.Data;

namespace BusinessLayer

{

public class BlEmployee

{

private DalEmployee \_dalEmployee;

public BlEmployee()

{

\_dalEmployee = new DalEmployee();

}

public DataTable TabloSorgu(string Auth)

{

DataTable dataTable = \_dalEmployee.KullaniciTablo(Auth);

return dataTable;

}

public DataTable IlSorgu()

{

DataTable dataTable = new DataTable();

dataTable = \_dalEmployee.IlCekme();

return dataTable;

}

public DataTable IlceSorgu(int ilID)

{

DataTable dataTable = new DataTable();

dataTable = \_dalEmployee.IlceCekme(ilID);

return dataTable;

}

public bool KullaniciAdBelirle(string Tc, string Ad)

{

bool onay = false;

Random rnd = new Random(3);

string parolasayi = rnd.Next().ToString();

string kullaniciAd = Ad.Substring(3) + Tc.Substring(3);

string kullaniciSifre = Ad.Substring(2) + Tc.Substring(2) + parolasayi;

onay = \_dalEmployee.EmployeeKullaniciAdBelirle(Tc, kullaniciAd, kullaniciSifre);

return onay;

}

public bool EklemeSorgu(string Tc\_No, string Ad, string Soyad, string Cinsiyet, string Kan\_Grup, string Telefon, string Mail, string Adres, int IlID, int IlceID)

{

bool onay = false;

onay = \_dalEmployee.KullaniciEkle(Tc\_No, Ad, Soyad, Cinsiyet, Kan\_Grup, Telefon, Mail, Adres, IlID, IlceID);

return onay;

}

public bool SilmeSorgu(string Tc\_No)

{

bool onay = false;

onay = \_dalEmployee.KullaniciSil(Tc\_No);

return onay;

}

Bu kod bloğu, **BlEmployee** sınıfına ait metodları içermektedir. Aşağıda, her bir metodun açıklaması bulunmaktadır:

**BlEmployee Sınıfı Metodları:**

**1. TabloSorgu Metodu:**

* **public DataTable TabloSorgu(string Auth)**: Bu metod, kullanıcı tablosundan belirli bir yetkiye sahip kullanıcıları sorgular. **\_dalEmployee.KullaniciTablo(Auth)** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**2. IlSorgu Metodu:**

* **public DataTable IlSorgu()**: Bu metod, illeri sorgular. **\_dalEmployee.IlCekme()** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**3. IlceSorgu Metodu:**

* **public DataTable IlceSorgu(int ilID)**: Belirli bir ilin ilçelerini sorgular. **\_dalEmployee.IlceCekme(ilID)** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**4. KullaniciAdBelirle Metodu:**

* **public bool KullaniciAdBelirle(string Tc, string Ad)**: Kullanıcı adı ve şifresi belirleyerek kullanıcı eklemek için kullanılır. Random bir sayı oluşturur ve adın belirli bir kısmını ve TC'nin belirli bir kısmını içeren bir kullanıcı adı oluşturur. Aynı şekilde, şifreyi de belirli bir kısmını ve random sayıyı içeren bir şekilde oluşturur. Oluşturulan bilgilerle **\_dalEmployee.EmployeeKullaniciAdBelirle** metodunu çağırır ve işlemin başarılı olup olmadığını bool olarak döndürür.

**5. EklemeSorgu Metodu:**

* **public bool EklemeSorgu(string Tc\_No, string Ad, string Soyad, string Cinsiyet, string Kan\_Grup, string Telefon, string Mail, string Adres, int IlID, int IlceID)**: Yeni bir kullanıcı eklemek için kullanılır. **\_dalEmployee.KullaniciEkle** metodunu çağırır ve işlemin başarılı olup olmadığını bool olarak döndürür.

**6. SilmeSorgu Metodu:**

* **public bool SilmeSorgu(string Tc\_No)**: Belirli bir kullanıcıyı silmek için kullanılır. **\_dalEmployee.KullaniciSil** metodunu çağırır ve işlemin başarılı olup olmadığını bool olarak döndürür.

Bu metodlar, **BlEmployee** sınıfını kullanıcı yönetimi ve ilgili sorgular için kullanışlı hale getirir. İlgili işlemleri gerçekleştirmek ve veritabanından bilgileri çekmek için **DalEmployee** sınıfını kullanır.

public bool GuncellemeSorgu(string Tc\_No, string Ad, string Soyad, string Telefon, string Mail, string Adres, int IlID, int IlceID)

{

bool onay = false;

onay = \_dalEmployee.KullaniciGuncelle(Tc\_No, Ad, Soyad, Telefon, Mail, Adres, IlID, IlceID);

return onay;

}

public DataTable TcArama(string Tc\_No)

{

DataTable dataTable = new DataTable();

dataTable = \_dalEmployee.TcArama(Tc\_No);

return dataTable;

}

public DataTable AdSoyadArama(string Ad, string Soyad)

{

DataTable dataTable = new DataTable();

dataTable = \_dalEmployee.AdSoyadArama(Ad, Soyad);

return dataTable;

}

public DataTable TelArama(string Tel)

{

DataTable dataTable = new DataTable();

dataTable = \_dalEmployee.TelArama(Tel);

return dataTable;

}

public DataTable MailArama(string Mail)

{

DataTable dataTable = new DataTable();

dataTable = \_dalEmployee.MailArama(Mail);

return dataTable;

}

}

}

Bu kod bloğu, **BlEmployee** sınıfına ait ekstra metodları içermektedir. Aşağıda, her bir metodun açıklaması bulunmaktadır:

**7. GuncellemeSorgu Metodu:**

* **public bool GuncellemeSorgu(string Tc\_No, string Ad, string Soyad, string Telefon, string Mail, string Adres, int IlID, int IlceID)**: Belirli bir kullanıcının bilgilerini güncellemek için kullanılır. **\_dalEmployee.KullaniciGuncelle** metodunu çağırır ve işlemin başarılı olup olmadığını bool olarak döndürür.

**8. TcArama Metodu:**

* **public DataTable TcArama(string Tc\_No)**: Belirli bir TC numarasına sahip kullanıcıyı aramak için kullanılır. **\_dalEmployee.TcArama** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**9. AdSoyadArama Metodu:**

* **public DataTable AdSoyadArama(string Ad, string Soyad)**: Belirli bir ad ve soyada sahip kullanıcıyı aramak için kullanılır. **\_dalEmployee.AdSoyadArama** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**10. TelArama Metodu:**

* **public DataTable TelArama(string Tel)**: Belirli bir telefon numarasına sahip kullanıcıyı aramak için kullanılır. **\_dalEmployee.TelArama** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**11. MailArama Metodu:**

* **public DataTable MailArama(string Mail)**: Belirli bir e-posta adresine sahip kullanıcıyı aramak için kullanılır. **\_dalEmployee.MailArama** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

Bu metodlar, kullanıcı bilgilerini aramak ve güncellemek için **BlEmployee** sınıfını daha işlevsel hale getirir. İlgili işlemleri gerçekleştirmek ve veritabanından bilgileri çekmek için **DalEmployee** sınıfını kullanır.

Formun Üstü

using DataAccessLayer;

using System.Data;

namespace BusinessLayer

{

public class BlGraph

{

private DalGraph \_dalGraph;

public BlGraph()

{

\_dalGraph = new DalGraph();

}

public DataTable JustYil(string AidatYil)

{

DataTable dataTable = \_dalGraph.JustYil(AidatYil);

return dataTable;

}

public DataTable JustAy(string AidatAy)

{

DataTable dataTable = \_dalGraph.JustAy(AidatAy);

return dataTable;

}

public DataTable YilandAy(string AidatYil, string AidatAy)

{

DataTable dataTable = \_dalGraph.YilandAy(AidatYil, AidatAy);

return dataTable;

}

public DataTable IlSorgu()

{

DataTable dataTable = new DataTable();

dataTable = \_dalGraph.IlCekme();

return dataTable;

}

public DataTable IlceSorgu(int ilID)

{

DataTable dataTable = new DataTable();

dataTable = \_dalGraph.IlceCekme(ilID);

return dataTable;

}

public DataTable IleGoreKSayi(int ilID)

{

DataTable dataTable = new DataTable();

dataTable = \_dalGraph.IleGoreKSayi(ilID);

return dataTable;

}

public DataTable IlceyeGoreKSayi(int ilceID)

{

DataTable dataTable = new DataTable();

dataTable = \_dalGraph.IlceyeGoreKSayi(ilceID);

return dataTable;

}

}

}

Bu kod bloğu, **BlGraph** sınıfına ait metodları içermektedir. Aşağıda, her bir metodun açıklaması bulunmaktadır:

**1. JustYil Metodu:**

* **public DataTable JustYil(string AidatYil)**: Belirli bir yıla ait aidat verilerini çekmek için kullanılır. **\_dalGraph.JustYil** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**2. JustAy Metodu:**

* **public DataTable JustAy(string AidatAy)**: Belirli bir aya ait aidat verilerini çekmek için kullanılır. **\_dalGraph.JustAy** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**3. YilandAy Metodu:**

* **public DataTable YilandAy(string AidatYil, string AidatAy)**: Belirli bir yıl ve ay kombinasyonuna ait aidat verilerini çekmek için kullanılır. **\_dalGraph.YilandAy** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**4. IlSorgu Metodu:**

* **public DataTable IlSorgu()**: Tüm illeri çekmek için kullanılır. **\_dalGraph.IlCekme** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**5. IlceSorgu Metodu:**

* **public DataTable IlceSorgu(int ilID)**: Belirli bir ile ait ilçeleri çekmek için kullanılır. **\_dalGraph.IlceCekme** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**6. IleGoreKSayi Metodu:**

* **public DataTable IleGoreKSayi(int ilID)**: Belirli bir ile ait kullanıcı sayısını çekmek için kullanılır. **\_dalGraph.IleGoreKSayi** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**7. IlceyeGoreKSayi Metodu:**

* **public DataTable IlceyeGoreKSayi(int ilceID)**: Belirli bir ilçeye ait kullanıcı sayısını çekmek için kullanılır. **\_dalGraph.IlceyeGoreKSayi** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

Bu metodlar, grafik oluşturmak ve veri analizi yapmak amacıyla **BlGraph** sınıfını daha işlevsel hale getirir. İlgili işlemleri gerçekleştirmek ve veritabanından bilgileri çekmek için **DalGraph** sınıfını kullanır.

using DataAccessLayer;

using System;

using System.Data;

namespace BusinessLayer

{

public class BlList

{

private DalList \_dalList;

public BlList()

{

\_dalList = new DalList();

}

public DataTable IlSorgu()

{

DataTable dataTable = new DataTable();

dataTable = \_dalList.IlCekme();

return dataTable;

}

public DataTable IlceSorgu(int ilID)

{

DataTable dataTable = new DataTable();

dataTable = \_dalList.IlceCekme(ilID);

return dataTable;

}

public DataTable KanSorgu1()

{

DataTable dataTable = new DataTable();

dataTable = \_dalList.KanSorgu1();

return dataTable;

}

public DataTable KanSorgu2(string Grup)

{

DataTable dataTable = new DataTable();

dataTable = \_dalList.KanSorgu2(Grup);

return dataTable;

}

public DataTable StatuSorgu1()

{

DataTable dataTable = new DataTable();

dataTable = \_dalList.StatuSorgu1();

return dataTable;

}

public DataTable StatuSorgu2(string Statu)

{

DataTable dataTable = new DataTable();

dataTable = \_dalList.StatuSorgu2(Statu);

return dataTable;

}

Bu kod bloğu, **BlList** sınıfına ait metodları içermektedir. Aşağıda, her bir metodun açıklaması bulunmaktadır:

**1. IlSorgu Metodu:**

* **public DataTable IlSorgu()**: Tüm illeri çekmek için kullanılır. **\_dalList.IlCekme** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**2. IlceSorgu Metodu:**

* **public DataTable IlceSorgu(int ilID)**: Belirli bir ile ait ilçeleri çekmek için kullanılır. **\_dalList.IlceCekme** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**3. KanSorgu1 Metodu:**

* **public DataTable KanSorgu1()**: Tüm kan gruplarını çekmek için kullanılır. **\_dalList.KanSorgu1** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**4. KanSorgu2 Metodu:**

* **public DataTable KanSorgu2(string Grup)**: Belirli bir kan grubuna ait kullanıcıları çekmek için kullanılır. **\_dalList.KanSorgu2** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**5. StatuSorgu1 Metodu:**

* **public DataTable StatuSorgu1()**: Tüm kullanıcı statülerini çekmek için kullanılır. **\_dalList.StatuSorgu1** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**6. StatuSorgu2 Metodu:**

* **public DataTable StatuSorgu2(string Statu)**: Belirli bir kullanıcı statüsüne ait kullanıcıları çekmek için kullanılır. **\_dalList.StatuSorgu2** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

Bu metodlar, kullanıcıları ve ilgili bilgileri çekmek için **DalList** sınıfını kullanarak veritabanı işlemlerini gerçekleştirir. İlgili tablolardan gerekli verileri çekerek iş mantığını uygular ve sonuçları **DataTable** formatında döndürür.

public DataTable BorcSorgu1()

{

DataTable dataTable = new DataTable();

dataTable = \_dalList.BorcSorgu1();

return dataTable;

}

public DataTable BorcSorgu2(bool borcBilgi)

{

DataTable dataTable = new DataTable();

dataTable = \_dalList.BorcSorgu2(borcBilgi);

return dataTable;

}

public DataTable IlIlceTablo(int IlId)

{

DataTable dataTable = new DataTable();

dataTable = \_dalList.IlIlceTablo(IlId);

return dataTable;

}

public DataTable TariheGore(DateTime dt1, DateTime dt2)

{

DataTable dataTable = new DataTable();

dataTable = \_dalList.TariheGore(dt1,dt2);

return dataTable;

}

}

}

Bu kod bloğu, **BlList** sınıfına ait ekstra metodları içermektedir. Aşağıda, her bir metodun açıklaması bulunmaktadır:

**7. BorcSorgu1 Metodu:**

* **public DataTable BorcSorgu1()**: Tüm borç bilgilerini çekmek için kullanılır. **\_dalList.BorcSorgu1** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**8. BorcSorgu2 Metodu:**

* **public DataTable BorcSorgu2(bool borcBilgi)**: Belirli bir borç durumuna ait borç bilgilerini çekmek için kullanılır. **\_dalList.BorcSorgu2** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**9. IlIlceTablo Metodu:**

* **public DataTable IlIlceTablo(int IlId, int IlceID)**: Belirli bir ile ve ilçeye ait tabloyu çekmek için kullanılır. **\_dalList.IlIlceTablo** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

Bu metodlar, **\_dalList** nesnesini kullanarak veritabanından ilgili bilgileri çeker ve bu bilgileri **DataTable** formatında döndürür. İlgili veritabanı işlemlerini gerçekleştirir ve sonuçları iş mantığına uygun bir şekilde sunar.

using DataAccessLayer;

using System.Data;

namespace BusinessLayer

{

public class BlUser

{

private DalUser \_dalUser;

public BlUser()

{

\_dalUser = new DalUser();

}

public DataTable IlSorgu()

{

DataTable dataTable = new DataTable();

dataTable = \_dalUser.IlCekme();

return dataTable;

}

public DataTable IlceSorgu(int ilID)

{

DataTable dataTable = new DataTable();

dataTable = \_dalUser.IlceCekme(ilID);

return dataTable;

}

public DataTable TabloSorgu(string Auth)

{

DataTable dataTable = \_dalUser.KullaniciTablo(Auth);

return dataTable;

}

public bool EklemeSorgu(string Tc\_No, string Ad, string Soyad, string Cinsiyet, string Kan\_Grup, string Telefon, string Mail, string Adres, int IlID, int IlceID)

{

bool onay = false;

onay = \_dalUser.KullaniciEkle(Tc\_No, Ad, Soyad, Cinsiyet, Kan\_Grup, Telefon, Mail, Adres, IlID, IlceID);

return onay;

}

public bool SilmeSorgu(string Tc\_No)

{

bool onay = false;

onay = \_dalUser.KullaniciSil(Tc\_No);

return onay;

}

public bool GuncellemeSorgu(string Tc\_No, string Ad, string Soyad, string Telefon, string Mail, string Adres, int IlID, int IlceID)

{

bool onay = false;

onay = \_dalUser.KullaniciGuncelle(Tc\_No, Ad, Soyad, Telefon, Mail, Adres, IlID, IlceID);

return onay;

}

Bu kod bloğu, kullanıcı işlemlerini gerçekleştiren **BlUser** sınıfına ait metodları içermektedir. Aşağıda, her bir metodun açıklaması bulunmaktadır:

**1. IlSorgu Metodu:**

* **public DataTable IlSorgu()**: Tüm illeri çekmek için kullanılır. **\_dalUser.IlCekme** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**2. IlceSorgu Metodu:**

* **public DataTable IlceSorgu(int ilID)**: Belirli bir ile ait ilçeleri çekmek için kullanılır. **\_dalUser.IlceCekme** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**3. TabloSorgu Metodu:**

* **public DataTable TabloSorgu(string Auth)**: Belirli bir yetkiye ait kullanıcıları çekmek için kullanılır. **\_dalUser.KullaniciTablo** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**4. EklemeSorgu Metodu:**

* **public bool EklemeSorgu(string Tc\_No, string Ad, string Soyad, string Cinsiyet, string Kan\_Grup, string Telefon, string Mail, string Adres, int IlID, int IlceID)**: Yeni bir kullanıcı eklemek için kullanılır. **\_dalUser.KullaniciEkle** metodunu çağırır ve işlem başarılıysa **true**, başarısızsa **false** döndürür.

**5. SilmeSorgu Metodu:**

* **public bool SilmeSorgu(string Tc\_No)**: Belirli bir kullanıcıyı silmek için kullanılır. **\_dalUser.KullaniciSil** metodunu çağırır ve işlem başarılıysa **true**, başarısızsa **false** döndürür.

**6. GuncellemeSorgu Metodu:**

* **public bool GuncellemeSorgu(string Tc\_No, string Ad, string Soyad, string Telefon, string Mail, string Adres, int IlID, int IlceID)**: Belirli bir kullanıcının bilgilerini güncellemek için kullanılır. **\_dalUser.KullaniciGuncelle** metodunu çağırır ve işlem başarılıysa **true**, başarısızsa **false** döndürür.

Bu metodlar, **\_dalUser** nesnesini kullanarak veritabanından ilgili bilgileri çeker ve bu bilgileri **DataTable** formatında döndürür. İlgili veritabanı işlemlerini gerçekleştirir ve sonuçları iş mantığına uygun bir şekilde sunar.

public DataTable TcArama(string Tc\_No)

{

DataTable dataTable = new DataTable();

dataTable = \_dalUser.TcArama(Tc\_No);

return dataTable;

}

public DataTable AdSoyadArama(string Ad, string Soyad)

{

DataTable dataTable = new DataTable();

dataTable = \_dalUser.AdSoyadArama(Ad,Soyad);

return dataTable;

}

public DataTable TelArama(string Tel)

{

DataTable dataTable = new DataTable();

dataTable = \_dalUser.TelArama(Tel);

return dataTable;

}

public DataTable MailArama(string Mail)

{

DataTable dataTable = new DataTable();

dataTable = \_dalUser.MailArama(Mail);

return dataTable;

}

public string AdGetir(string Tc\_No)

{

DataTable dataTable = new DataTable();

dataTable = \_dalUser.AdGetir(Tc\_No);

string Ad = null;

if (dataTable != null && dataTable.Rows.Count > 0)

{

Ad = dataTable.Rows[0]["Ad"].ToString();

}

return Ad;

}

public DataTable BorcGetirBos(string Tc\_No)

{

DataTable dataTable = new DataTable();

dataTable = \_dalUser.BorcGetirBos(Tc\_No);

return dataTable;

}

public DataTable BorcGetirYilAy(string Tc\_No, string Yil, string Ay)

{

DataTable dataTable = new DataTable();

dataTable = \_dalUser.BorcGetirYilAy(Tc\_No, Yil, Ay);

return dataTable;

}

public bool TutarGuncelle(string Tc\_No, string Yil, string Ay, int tutar)

{

bool updatesonuc = false;

updatesonuc = \_dalUser.TutarGuncelle(Tc\_No, Yil, Ay, tutar);

return updatesonuc;

}

}

}

Bu kod bloğu, kullanıcı bilgilerini aramak, borç bilgilerini getirmek ve tutarı güncellemek gibi işlemleri gerçekleştiren **BlUser** sınıfına ait ek metodları içermektedir. Aşağıda, her bir metodun açıklaması bulunmaktadır:

**1. TcArama Metodu:**

* **public DataTable TcArama(string Tc\_No)**: Belirli bir TC Kimlik Numarasına ait kullanıcı bilgilerini çekmek için kullanılır. **\_dalUser.TcArama** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**2. AdSoyadArama Metodu:**

* **public DataTable AdSoyadArama(string Ad, string Soyad)**: Belirli bir ad ve soyad kombinasyonuna ait kullanıcı bilgilerini çekmek için kullanılır. **\_dalUser.AdSoyadArama** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**3. TelArama Metodu:**

* **public DataTable TelArama(string Tel)**: Belirli bir telefon numarasına ait kullanıcı bilgilerini çekmek için kullanılır. **\_dalUser.TelArama** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**4. MailArama Metodu:**

* **public DataTable MailArama(string Mail)**: Belirli bir e-posta adresine ait kullanıcı bilgilerini çekmek için kullanılır. **\_dalUser.MailArama** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**5. AdGetir Metodu:**

* **public string AdGetir(string Tc\_No)**: Belirli bir TC Kimlik Numarasına ait kullanıcının adını çekmek için kullanılır. **\_dalUser.AdGetir** metodunu çağırır ve sonucu string olarak döndürür.

**6. BorcGetirBos Metodu:**

* **public DataTable BorcGetirBos(string Tc\_No)**: Belirli bir kullanıcının boş borç bilgilerini çekmek için kullanılır. **\_dalUser.BorcGetirBos** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**7. BorcGetirYilAy Metodu:**

* **public DataTable BorcGetirYilAy(string Tc\_No, string Yil, string Ay)**: Belirli bir kullanıcının belirli bir yıl ve aydaki borç bilgilerini çekmek için kullanılır. **\_dalUser.BorcGetirYilAy** metodunu çağırır ve sonuçları **DataTable** olarak döndürür.

**8. TutarGuncelle Metodu:**

* **public bool TutarGuncelle(string Tc\_No, string Yil, string Ay, int tutar)**: Belirli bir kullanıcının belirli bir yıl ve aydaki borç tutarını güncellemek için kullanılır. **\_dalUser.TutarGuncelle** metodunu çağırır ve işlem başarılıysa **true**, başarısızsa **false** döndürür.

Bu metodlar, **\_dalUser** nesnesini kullanarak veritabanından ilgili bilgileri çeker ve bu bilgileri iş mantığına uygun bir şekilde sunar.

using DataAccessLayer;

using EntityLayer;

using System;

using System.Data;

namespace BusinessLayer

{

public class BlUserLogin

{

private DalUserLogin \_dalUserLogin;

private DalUser \_dalUser;

public BlUserLogin()

{

\_dalUserLogin = new DalUserLogin();

\_dalUser = new DalUser();

}

public EntityTblUserLogin getSorgu(string kullaniciAd, string sifre)

{

EntityTblUserLogin kullaniciGirisi = new EntityTblUserLogin();

DataTable dataTable = new DataTable();

dataTable = \_dalUserLogin.SearchKullaniciAdiveSifre(kullaniciAd, sifre);

foreach (DataRow dr in dataTable.Rows)

{

kullaniciGirisi.kullaniciAd = dr["kullaniciAd"].ToString();

kullaniciGirisi.kullaniciSifre = dr["kullaniciSifre"].ToString();

kullaniciGirisi.Auth = dr["Auth"].ToString();

kullaniciGirisi.Status = dr["Status"].ToString();

}

return kullaniciGirisi;

}

public bool KullaniciAdBelirle(string Tc, string Ad)

{

bool onay = false;

Random rnd = new Random(3);

string parolasayi = rnd.Next().ToString();

string kullaniciAd = Ad.Substring(3) + Tc.Substring(3);

string kullaniciSifre = Ad.Substring(2) + Tc.Substring(2) + parolasayi;

onay = \_dalUserLogin.UserKullaniciAdBelirle(Tc, kullaniciAd, kullaniciSifre);

return onay;

}

public string TcNoAl(string KAd)

{

DataTable dataTable = new DataTable();

dataTable = \_dalUserLogin.SearchTcNo(KAd);

string tcNo = null;

if (dataTable != null && dataTable.Rows.Count > 0)

{

tcNo = dataTable.Rows[0]["Tc\_No"].ToString();

}

return tcNo;

}

Bu kod bloğu, kullanıcı girişi, kullanıcı adı belirleme ve TC No çekme işlemlerini gerçekleştiren **BlUserLogin** sınıfına ait metodları içermektedir. Aşağıda, her bir metodun açıklaması bulunmaktadır:

**1. getSorgu Metodu:**

* **public EntityTblUserLogin getSorgu(string kullaniciAd, string sifre)**: Kullanıcı girişi sorgulama işlemi için kullanılır. **\_dalUserLogin.SearchKullaniciAdiveSifre** metodunu çağırarak, verilen kullanıcı adı ve şifreyle eşleşen kullanıcı bilgilerini bir **EntityTblUserLogin** nesnesi olarak döndürür.

**2. KullaniciAdBelirle Metodu:**

* **public bool KullaniciAdBelirle(string Tc, string Ad)**: Kullanıcı adı belirleme işlemi için kullanılır. Random sayı üreterek, TC kimlik numarası ve isimden türetilmiş bir kullanıcı adı ve şifre oluşturur ve **\_dalUserLogin.UserKullaniciAdBelirle** metodunu çağırarak veritabanına ekler. İşlem başarılıysa **true**, başarısızsa **false** döndürür.

**3. TcNoAl Metodu:**

* **public string TcNoAl(string KAd)**: Belirli bir kullanıcı adına ait TC kimlik numarasını çekmek için kullanılır. **\_dalUserLogin.SearchTcNo** metodunu çağırarak, verilen kullanıcı adıyla eşleşen TC kimlik numarasını döndürür.

Bu metodlar, **\_dalUserLogin** ve **\_dalUser** nesnelerini kullanarak kullanıcı girişi ve kullanıcı bilgilerini çekme işlemlerini gerçekleştirir. Kullanıcı girişi için **\_dalUserLogin**, kullanıcı bilgileri için ise **\_dalUser** sınıfları kullanılmaktadır.

public string KAdGetir(string Tc)

{

DataTable dataTable = new DataTable();

dataTable = \_dalUserLogin.SearchKAd(Tc);

string KAd = null;

if (dataTable != null && dataTable.Rows.Count > 0)

{

KAd = dataTable.Rows[0]["kullaniciAd"].ToString();

}

return KAd;

}

public string ParolaGetir(string Tc)

{

DataTable dataTable = new DataTable();

dataTable = \_dalUserLogin.SearchParola(Tc);

string Parola = null;

if (dataTable != null && dataTable.Rows.Count > 0)

{

Parola = dataTable.Rows[0]["kullaniciSifre"].ToString();

}

return Parola;

}

public bool KullaniciAdSifreDegistir1(string Tc\_No, string KAd,string Sifre, string statu)

{

bool onay = false;

onay = \_dalUserLogin.UserKullaniciAdSifreDegistir1(Tc\_No,KAd,Sifre,statu);

return onay;

}

public bool KullaniciAdSifreDegistir2(string Tc\_No, string KAd, string Sifre)

{

bool onay = false;

onay = \_dalUserLogin.UserKullaniciAdSifreDegistir2(Tc\_No, KAd, Sifre);

return onay;

}

public bool KullaniciAdDegistir1(string Tc\_No, string KAd, string statu)

{

bool onay = false;

onay = \_dalUserLogin.UserKullaniciAdDegistir1(Tc\_No, KAd, statu);

return onay;

}

public bool KullaniciAdDegistir2(string Tc\_No, string KAd)

{

bool onay = false;

onay = \_dalUserLogin.UserKullaniciAdDegistir2(Tc\_No, KAd);

return onay;

}

public bool SifreDegistir1(string Tc\_No, string Sifre, string statu)

{

bool onay = false;

onay = \_dalUserLogin.UserSifreDegistir1(Tc\_No, Sifre, statu);

return onay;

}

Bu metodlar, kullanıcı adı ve şifre işlemlerini gerçekleştiren **BlUserLogin** sınıfına aittir. İlgili metodların açıklamalarını aşağıda bulabilirsiniz:

**1. KAdGetir Metodu:**

* **public string KAdGetir(string Tc)**: Belirli bir TC kimlik numarasına ait kullanıcı adını çekmek için kullanılır. **\_dalUserLogin.SearchKAd** metodunu çağırarak, verilen TC kimlik numarasıyla eşleşen kullanıcı adını döndürür.

**2. ParolaGetir Metodu:**

* **public string ParolaGetir(string Tc)**: Belirli bir TC kimlik numarasına ait şifreyi çekmek için kullanılır. **\_dalUserLogin.SearchParola** metodunu çağırarak, verilen TC kimlik numarasıyla eşleşen şifreyi döndürür.

**3. KullaniciAdSifreDegistir1 Metodu:**

* **public bool KullaniciAdSifreDegistir1(string Tc\_No, string KAd, string Sifre, string statu)**: Belirli bir kullanıcının kullanıcı adı, şifre ve statü bilgilerini değiştirmek için kullanılır. **\_dalUserLogin.UserKullaniciAdSifreDegistir1** metodunu çağırarak, verilen parametrelerle ilgili değişiklikleri gerçekleştirir. İşlem başarılıysa **true**, başarısızsa **false** döndürür.

**4. KullaniciAdSifreDegistir2 Metodu:**

* **public bool KullaniciAdSifreDegistir2(string Tc\_No, string KAd, string Sifre)**: Belirli bir kullanıcının kullanıcı adı ve şifresini değiştirmek için kullanılır. **\_dalUserLogin.UserKullaniciAdSifreDegistir2** metodunu çağırarak, verilen parametrelerle ilgili değişiklikleri gerçekleştirir. İşlem başarılıysa **true**, başarısızsa **false** döndürür.

**5. KullaniciAdDegistir1 Metodu:**

* **public bool KullaniciAdDegistir1(string Tc\_No, string KAd, string statu)**: Belirli bir kullanıcının kullanıcı adını ve statüsünü değiştirmek için kullanılır. **\_dalUserLogin.UserKullaniciAdDegistir1** metodunu çağırarak, verilen parametrelerle ilgili değişiklikleri gerçekleştirir. İşlem başarılıysa **true**, başarısızsa **false** döndürür.

**6. KullaniciAdDegistir2 Metodu:**

* **public bool KullaniciAdDegistir2(string Tc\_No, string KAd)**: Belirli bir kullanıcının kullanıcı adını değiştirmek için kullanılır. **\_dalUserLogin.UserKullaniciAdDegistir2** metodunu çağırarak, verilen parametrelerle ilgili değişiklikleri gerçekleştirir. İşlem başarılıysa **true**, başarısızsa **false** döndürür.

**7. SifreDegistir1 Metodu:**

* **public bool SifreDegistir1(string Tc\_No, string Sifre, string statu)**: Belirli bir kullanıcının şifresini değiştirmek için kullanılır. **\_dalUserLogin.UserSifreDegistir1** metodunu çağırarak, verilen parametrelerle ilgili değişiklikleri gerçekleştirir. İşlem başarılıysa **true**, başarısızsa **false** döndürür.

Formun Üstü

public bool SifreDegistir2(string Tc\_No, string Sifre)

{

bool onay = false;

onay = \_dalUserLogin.UserSifreDegistir2(Tc\_No, Sifre);

return onay;

}

public DataTable TcArama(string Tc\_No)

{

DataTable dataTable = new DataTable();

dataTable = \_dalUser.TcArama(Tc\_No);

return dataTable;

}

public DataTable AidatBosSorgu1(string Tc\_No)

{

DataTable dataTable = new DataTable();

dataTable = \_dalUserLogin.AidatBos1(Tc\_No);

return dataTable;

}

public DataTable AidatAySorgu1(string Tc\_No, string Ay)

{

DataTable dataTable = new DataTable();

dataTable = \_dalUserLogin.AidatAy1(Tc\_No, Ay);

return dataTable;

}

public DataTable AidatYilSorgu1(string Tc\_No, string Yil)

{

DataTable dataTable = new DataTable();

dataTable = \_dalUserLogin.AidatYil1(Tc\_No, Yil);

return dataTable;

}

public DataTable AidatYilAySorgu1(string Tc\_No, string Yil, string Ay)

{

DataTable dataTable = new DataTable();

dataTable = \_dalUserLogin.AidatYilAy1(Tc\_No, Yil, Ay);

return dataTable;

}

public DataTable AidatBosSorgu2(string Tc\_No, bool OdemeDurum)

{

DataTable dataTable = new DataTable();

dataTable = \_dalUserLogin.AidatBos2(Tc\_No,OdemeDurum);

return dataTable;

}

public DataTable AidatAySorgu2(string Tc\_No, string Ay, bool OdemeDurum)

{

DataTable dataTable = new DataTable();

dataTable = \_dalUserLogin.AidatAy2(Tc\_No, Ay, OdemeDurum);

return dataTable;

}

public DataTable AidatYilSorgu2(string Tc\_No, string Yil, bool OdemeDurum)

{

DataTable dataTable = new DataTable();

dataTable = \_dalUserLogin.AidatYil2(Tc\_No, Yil, OdemeDurum);

return dataTable;

}

public DataTable AidatYilAySorgu2(string Tc\_No, string Yil, string Ay, bool OdemeDurum)

{

DataTable dataTable = new DataTable();

dataTable = \_dalUserLogin.AidatYilAy2(Tc\_No, Yil, Ay, OdemeDurum);

return dataTable;

}

}

}

Bu metodlar, kullanıcı bilgileri ve aidat ödeme sorguları ile ilgili işlemleri gerçekleştiren **BlUserLogin** sınıfına aittir. İlgili metodların açıklamalarını aşağıda bulabilirsiniz:

**1. SifreDegistir2 Metodu:**

* **public bool SifreDegistir2(string Tc\_No, string Sifre)**: Belirli bir kullanıcının şifresini değiştirmek için kullanılır. **\_dalUserLogin.UserSifreDegistir2** metodunu çağırarak, verilen parametrelerle ilgili değişiklikleri gerçekleştirir. İşlem başarılıysa **true**, başarısızsa **false** döndürür.

**2. TcArama Metodu:**

* **public DataTable TcArama(string Tc\_No)**: Belirli bir TC kimlik numarasına ait kullanıcıyı aramak için kullanılır. **\_dalUser.TcArama** metodunu çağırarak, verilen TC kimlik numarasıyla eşleşen kullanıcı bilgilerini döndürür.

**3. AidatBosSorgu1 Metodu:**

* **public DataTable AidatBosSorgu1(string Tc\_No)**: Belirli bir kullanıcının ödenmemiş aidatlarını sorgular. **\_dalUserLogin.AidatBos1** metodunu çağırarak, verilen TC kimlik numarasına ait ödenmemiş aidat bilgilerini döndürür.

**4. AidatAySorgu1 Metodu:**

* **public DataTable AidatAySorgu1(string Tc\_No, string Ay)**: Belirli bir kullanıcının belirli bir aydaki ödenmemiş aidatlarını sorgular. **\_dalUserLogin.AidatAy1** metodunu çağırarak, verilen TC kimlik numarasına ve ay bilgisine ait ödenmemiş aidat bilgilerini döndürür.

**5. AidatYilSorgu1 Metodu:**

* **public DataTable AidatYilSorgu1(string Tc\_No, string Yil)**: Belirli bir kullanıcının belirli bir yıldaki ödenmemiş aidatlarını sorgular. **\_dalUserLogin.AidatYil1** metodunu çağırarak, verilen TC kimlik numarasına ve yıl bilgisine ait ödenmemiş aidat bilgilerini döndürür.

**6. AidatYilAySorgu1 Metodu:**

* **public DataTable AidatYilAySorgu1(string Tc\_No, string Yil, string Ay)**: Belirli bir kullanıcının belirli bir yılda ve ayda ödenmemiş aidatlarını sorgular. **\_dalUserLogin.AidatYilAy1** metodunu çağırarak, verilen TC kimlik numarasına, yıl ve ay bilgilerine ait ödenmemiş aidat bilgilerini döndürür.

**7. AidatBosSorgu2 Metodu:**

* **public DataTable AidatBosSorgu2(string Tc\_No, bool OdemeDurum)**: Belirli bir kullanıcının ödenmiş veya ödenmemiş aidatlarını sorgular. **\_dalUserLogin.AidatBos2** metodunu çağırarak, verilen TC kimlik numarasına ve ödeme durumu bilgisine ait aidat bilgilerini döndürür.

**8. AidatAySorgu2 Metodu:**

* **public DataTable AidatAySorgu2(string Tc\_No, string Ay, bool OdemeDurum)**: Belirli bir kullanıcının belirli bir aydaki ödenmiş veya ödenmemiş aidatlarını sorgular. **\_dalUserLogin.AidatAy2** metodunu çağırarak, verilen TC kimlik numarasına, ay ve ödeme durumu bilgilerine ait aidat bilgilerini döndürür.

**9. AidatYilSorgu2 Metodu:**

* **public DataTable AidatYilSorgu2(string Tc\_No, string Yil, bool OdemeDurum)**: Belirli bir kullanıcının belirli bir yıldaki ödenmiş veya ödenmemiş aidatlarını sorgular. **\_dalUserLogin.AidatYil2** metodunu çağırarak, verilen TC kimlik numarasına, yıl ve ödeme durumu bilgilerine ait aidat bilgilerini döndürür.

**10. AidatYilAySorgu2 Metodu:**

* **public DataTable AidatYilAySorgu2(string Tc\_No, string Yil, string Ay, bool OdemeDurum)**: Belirli bir kullanıcının belirli bir yılda ve ayda ödenmiş veya ödenmemiş aidatlarını sorgular. **\_dalUserLogin.AidatYilAy2** metodunu çağırarak, verilen TC kimlik numarasına, yıl, ay ve ödeme durumu bilgilerine ait aidat bilgilerini döndürür.
  1. **Entity Layer**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.NetworkInformation;

using System.Text;

using System.Threading.Tasks;

namespace EntityLayer

{

public class EntityTblUserLogin

{

private string \_kullaniciAd { get; set; }

private string \_kullaniciSifre { get; set; }

private string \_Auth { get; set; }

private string \_Status { get; set; }

public string kullaniciAd

{

get

{

return \_kullaniciAd;

}

set

{

\_kullaniciAd = value;

}

}

public string kullaniciSifre

{

get

{

return \_kullaniciSifre;

}

set

{

\_kullaniciSifre = value;

}

}

public string Auth

{

get

{

return \_Auth;

}

set

{

\_Auth = value;

}

}

public string Status

{

get

{

return \_Status;

}

set

{

\_Status = value;

}

}

}

}

Bu **EntityTblUserLogin** sınıfı, kullanıcı girişiyle ilgili verileri tutan bir sınıftır. İşte sınıfın özellikleri ve açıklamaları:

**EntityTblUserLogin Sınıfı**

Bu sınıf, kullanıcı girişiyle ilgili temel bilgileri içerir.

Özellikler (Properties):

1. **kullaniciAd**: Kullanıcının giriş adını temsil eden özelliktir.
   * **Veri Tipi**: **string**
   * **Açıklama**: Kullanıcı adı, sisteme giriş yaparken kullanıcının kendisini tanımlamasına olanak tanıyan bir değerdir.
2. **kullaniciSifre**: Kullanıcının giriş şifresini temsil eden özelliktir.
   * **Veri Tipi**: **string**
   * **Açıklama**: Kullanıcı şifresi, kullanıcının hesabına güvenli bir şekilde erişmesini sağlayan gizli bir değerdir.
3. **Auth**: Kullanıcının yetki seviyesini temsil eden özelliktir.
   * **Veri Tipi**: **string**
   * **Açıklama**: Kullanıcının sahip olduğu yetki düzeyini belirten bir değerdir. Örneğin, "Admin", "User" gibi.
4. **Status**: Kullanıcının hesap durumunu temsil eden özelliktir.
   * **Veri Tipi**: **string**
   * **Açıklama**: Kullanıcının hesap durumu, genellikle "Active" veya "Inactive" gibi değerler alır.

Notlar:

* Özelliklerin başındaki alt çizgi (**\_**) iki kapsülleme seviyesi arasındaki farkı belirtir. Bu, sadece aynı sınıf içindeki metotların veya özelliklerin bu özelliklere doğrudan erişebileceği anlamına gelir.
* Her özellik, **get** ve **set** accessor'larına sahiptir, bu da dışarıdan bu özelliklere erişim ve bu özelliklere değer atama olanağı sağlar.

Bu sınıfın amacı, kullanıcı girişiyle ilgili temel verileri gruplamak ve bu verilere erişim sağlamaktır.

* 1. **Presentation Layer**

using BusinessLayer;

using DernekUyeTakipSistemi.Admin;

using DernekUyeTakipSistemi.Customer;

using DernekUyeTakipSistemi.Employee;

using EntityLayer;

using System;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi

{

public partial class LoginPage : Form

{

int gvnlkkod;

BlUserLogin \_blUserLogin;

public LoginPage()

{

InitializeComponent();

\_blUserLogin = new BlUserLogin();

txtKAd.MaxLength = 40;

txtParola.MaxLength = 40;

txtKod.MaxLength = 5;

}

private void LoginPage\_Load(object sender, EventArgs e)

{

txtKAd.Focus();

GuvenlikKodYenile();

}

private void GuvenlikKodYenile()

{

Random random = new Random();

gvnlkkod = random.Next(10000, 99999);

lblKod.Text = gvnlkkod.ToString();

}

1. **using System.Data.OleDb;**: Bu satır, **OleDbConnection** sınıfını kullanabilmek için gerekli olan **OleDb** isim alanını içeri alır. Bu, OleDb veritabanı bağlantıları için gerekli olan sınıfları ve özellikleri içerir.
2. **namespace DataAccessLayer.DbConnections**: Bu satır, **DataAccessLayer.DbConnections** adında bir isim alanı (namespace) oluşturur. Bu, bu kodun başka bir dosyada kullanılması durumunda bu namespace içindeki sınıfların kullanılmasını sağlar.
3. **public class DbConnect**: Bu satır, **DbConnect** adında genel bir sınıf tanımlar. Bu sınıf, veritabanı bağlantısı oluşturmak için gerekli olan özellikleri içerir.
4. **public OleDbConnection baglanti;**: **baglanti** adında bir **OleDbConnection** tipinde genel bir değişken (field) tanımlanır. Bu değişken, veritabanına bağlantıyı temsil eder.
5. **public DbConnect()**: Bu satır, sınıfın kurucu metodu (constructor) tanımlar. Bu metod, sınıfın bir örneği oluşturulduğunda otomatik olarak çağrılır.
6. **Baglanti = new OleDbConnection(@"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=DernekTakipDataBase.accdb;");**: Bu satırda, **OleDbConnection** örneği oluşturulur ve **baglanti** değişkenine atanır. Bu bağlantı dizesi, Microsoft Access veritabanına bağlanmayı sağlar. Dize, veritabanının bulunduğu konumu ve dosya adını içerir.

Bu kod, genel bir veritabanı bağlantısı oluşturmak için kullanılır. Ancak, daha karmaşık uygulamalarda genellikle hata yönetimi ve güvenlik önlemleri eklemek için daha fazla kod eklenir. Ayrıca, bağlantı açma ve kapatma işlemleri daha güvenli bir şekilde gerçekleştirilir, örneğin **try-catch** blokları eklenerek hata yönetimi yapılabilir.

private void txtKAd\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Enter)

{

e.SuppressKeyPress = true;

btnOnay.PerformClick();

}

}

private void txtParola\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Enter)

{

e.SuppressKeyPress = true;

btnOnay.PerformClick();

}

}

private void txtKod\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Enter)

{

e.SuppressKeyPress = true;

btnOnay.PerformClick();

}

}

private void txtKod\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsControl(e.KeyChar) && !char.IsDigit(e.KeyChar))

{

e.Handled = true;

}

}

private void yenileToolStripMenuItem\_Click(object sender, EventArgs e)

{

txtKAd.Clear();

txtParola.Clear();

txtKod.Clear();

GuvenlikKodYenile();

txtKAd.Focus();

}

private void yardımToolStripMenuItem\_Click(object sender, EventArgs e)

{

MessageBox.Show("Merhabalar Sayın Kullanıcı, " +

"\nDernek Takip Uygulamamıza HOŞ GELDİNİZ. " +

"\nEkranda görmüş olduğunuz kullanıcı adı ve şifre kısımlarına " +

"\nSize daha önceden verilmiş olan bilgileri girip " +

"\nHemen altında bulunan Yeşil renkli onay kutusu ile GİRİŞ YAPABİLİRSİNİZ." +

"\nEğer programdan çıkmak isterseniz Kırmızı renkli EXIT tuşuna basabilirsiniz. " +

"\nİyi günler dilerim.");

}

private void çıkışToolStripMenuItem\_Click(object sender, EventArgs e)

{

Application.Exit();

}

private void ClearForm()

{

txtKAd.Clear(); txtParola.Clear();txtKod.Clear();

GuvenlikKodYenile();

txtKAd.Focus();

}

Bu C# kodu bir Windows Forms uygulamasındaki olay işleyicilerini içerir. Aşağıda her bir metotun açıklamasını bulabilirsiniz:

1. **txtKAd\_KeyDown Metodu:**
   * Kullanıcı adı (**txtKAd**) TextBox'ına bir tuş basıldığında tetiklenen olay işleyicisidir.
   * Eğer basılan tuş Enter ise, tuşun işlenmesi engellenir (**e.SuppressKeyPress = true;**) ve **btnOnay** adlı butonun tıklama olayı (**PerformClick()**) simüle edilir.
2. **txtParola\_KeyDown Metodu:**
   * Şifre (**txtParola**) TextBox'ına bir tuş basıldığında tetiklenen olay işleyicisidir.
   * Eğer basılan tuş Enter ise, tuşun işlenmesi engellenir ve **btnOnay** butonunun tıklama olayı simüle edilir.
3. **txtKod\_KeyDown Metodu:**
   * Güvenlik kodu (**txtKod**) TextBox'ına bir tuş basıldığında tetiklenen olay işleyicisidir.
   * Eğer basılan tuş Enter ise, tuşun işlenmesi engellenir ve **btnOnay** butonunun tıklama olayı simüle edilir.
4. **txtKod\_KeyPress Metodu:**
   * Güvenlik kodu TextBox'ına bir tuş basıldığında tetiklenen olay işleyicisidir.
   * Eğer basılan tuş kontrol karakteri değilse ve rakam değilse, tuş işlenir (**e.Handled = true;**).
5. **yenileToolStripMenuItem\_Click Metodu:**
   * "Yenile" menü öğesi tıklandığında tetiklenen olay işleyicisidir.
   * Kullanıcı adı, şifre ve güvenlik kodu TextBox'larını temizler, yeni bir güvenlik kodu oluşturur ve kullanıcı adı TextBox'ına odaklanır.
6. **yardımToolStripMenuItem\_Click Metodu:**
   * "Yardım" menü öğesi tıklandığında tetiklenen olay işleyicisidir.
   * Bir bilgi penceresi görüntüler ve uygulama hakkında bilgi içerir.
7. **çıkışToolStripMenuItem\_Click Metodu:**
   * "Çıkış" menü öğesi tıklandığında tetiklenen olay işleyicisidir.
   * Uygulamadan çıkış yapar (**Application.Exit()**).
8. **ClearForm Metodu:**
   * Kullanıcı adı, şifre ve güvenlik kodu TextBox'larını temizler, yeni bir güvenlik kodu oluşturur ve kullanıcı adı TextBox'ına odaklanır.

Bu metotlar, bir giriş formuyla ilgili işlevselliği yönetir ve kullanıcı etkileşimlerine yanıt olarak belirli olayları işler.

private void btnOnay\_Click(object sender, EventArgs e)

{

EntityTblUserLogin \_usrlgn = new EntityTblUserLogin();

\_usrlgn = \_blUserLogin.getSorgu(txtKAd.Text, txtParola.Text);

try

{

if (\_usrlgn.kullaniciAd == txtKAd.Text && \_usrlgn.kullaniciSifre == txtParola.Text && \_usrlgn.Status == "A")

{

if (!string.IsNullOrEmpty(txtKod.Text))

{

if (gvnlkkod == Convert.ToInt32(txtKod.Text))

{

if (\_usrlgn.Auth == (0).ToString())

{

AMenu adminMenu = new AMenu();

adminMenu.Show();

ClearForm();

this.Hide();

}

else if (\_usrlgn.Auth == (1).ToString())

{

EMenu employeeMenu = new EMenu();

employeeMenu.TcNoAlma(\_usrlgn.kullaniciAd);

employeeMenu.Show();

ClearForm();

this.Hide();

}

else if (\_usrlgn.Auth == (2).ToString())

{

CMenu customerMenu = new CMenu();

customerMenu.TcNoAlma(\_usrlgn.kullaniciAd);

customerMenu.Show();

ClearForm();

this.Hide();

}

else

{

MessageBox.Show("Lütfen Hesap Yöneticiniz İle iletişime Geçiniz!\nBir problem Oluştu.");

}

}

else

{

MessageBox.Show("Güvenlik Kodunu YANLIŞ Girdiniz.\nLütfen Tekrar deneyiniz.");

txtKod.Focus();

}

}

else

{

MessageBox.Show("Güvenlik Kodu boş bırakılamaz.");

txtKod.Focus();

}

}

else

{

MessageBox.Show(" Yanlış Kullanıcı Adı veya Şifre.\n\n Tekrar Deneyiniz!");

txtKAd.Clear();

txtParola.Clear();

txtKAd.Focus();

}

}

catch

{

MessageBox.Show("Lütfen \n\nKullanıcı adı ve şifreniz ile \n\ngiriş yapınız.");

txtKAd.Clear();

txtParola.Clear();

txtKAd.Focus();

}

finally

{

GuvenlikKodYenile();

txtKod.Clear();

}

}

Bu C# kod bloğu, bir kullanıcının giriş bilgilerini doğrulayan bir metodun olay işleyicisidir. İşte bu metodun ayrıntılı açıklaması:

1. **private void btnOnay\_Click(object sender, EventArgs e)**
   * "Onay" butonuna tıklandığında tetiklenen olay işleyicisidir.
2. **EntityTblUserLogin \_usrlgn = new EntityTblUserLogin();**
   * **\_usrlgn** adında bir **EntityTblUserLogin** nesnesi oluşturulur. Bu nesne, kullanıcı giriş bilgilerini içeren bir varlık (entity) sınıfından türetilmiştir.
3. **\_usrlgn = \_blUserLogin.getSorgu(txtKAd.Text, txtParola.Text);**
   * **\_blUserLogin** adındaki **BlUserLogin** sınıfının **getSorgu** metodunu kullanarak, kullanıcı adı ve şifreye göre veritabanından gerekli bilgileri çeker.
4. **try { ... }**
   * Ana try-catch bloğu, çeşitli durumları ele almak için kullanılır. Kod içindeki işlemlerde hata olup olmadığını kontrol eder.
5. **if (\_usrlgn.kullaniciAd == txtKAd.Text && \_usrlgn.kullaniciSifre == txtParola.Text && \_usrlgn.Status == "A")**
   * Kullanıcı adı, şifre ve kullanıcının aktif durumu kontrol edilir. Eğer bilgiler doğru ve hesap aktifse:
6. **if (!string.IsNullOrEmpty(txtKod.Text))**
   * Güvenlik kodu TextBox'ı boş olup olmadığı kontrol edilir.
7. **if (gvnlkkod == Convert.ToInt32(txtKod.Text))**
   * Girilen güvenlik kodu doğru ise:
8. **if (\_usrlgn.Auth == (0).ToString())**
   * Kullanıcının yetki seviyesine göre farklı menüler gösterilir.
   * Auth değeri 0 ise admin menüsü, 1 ise çalışan menüsü, 2 ise müşteri menüsü açılır.
9. **else { MessageBox.Show("Lütfen Hesap Yöneticiniz İle iletişime Geçiniz!\nBir problem Oluştu."); }**
   * Auth değeri 0, 1 veya 2 değilse bir hata mesajı gösterilir.
10. **else { MessageBox.Show("Güvenlik Kodunu YANLIŞ Girdiniz.\nLütfen Tekrar deneyiniz."); txtKod.Focus(); }**
    * Girilen güvenlik kodu yanlışsa bir hata mesajı gösterilir ve güvenlik kodu TextBox'a odaklanılır.
11. **else { MessageBox.Show("Güvenlik Kodu boş bırakılamaz."); txtKod.Focus(); }**
    * Güvenlik kodu boşsa bir hata mesajı gösterilir ve güvenlik kodu TextBox'a odaklanılır.
12. **else { MessageBox.Show(" Yanlış Kullanıcı Adı veya Şifre.\n\n Tekrar Deneyiniz!"); txtKAd.Clear(); txtParola.Clear(); txtKAd.Focus(); }**
    * Kullanıcı adı, şifre veya hesap durumu hatalı ise bir hata mesajı gösterilir ve giriş bilgileri temizlenip kullanıcı adı TextBox'a odaklanılır.
13. **catch { MessageBox.Show("Lütfen \n\nKullanıcı adı ve şifreniz ile \n\ngiriş yapınız."); txtKAd.Clear(); txtParola.Clear(); txtKAd.Focus(); }**
    * Try bloğu içindeki herhangi bir hata durumunda genel bir hata mesajı gösterilir.
14. **finally { GuvenlikKodYenile(); txtKod.Clear(); }**
    * Kodun çalışması tamamlandığında, güvenlik kodunu yeniler ve güvenlik kodu TextBox'ını temizler.

Bu metodun amacı, kullanıcının giriş bilgilerini doğrulamak ve yetkilerine göre uygun bir menüyü açmaktır. Hata durumlarına karşı uygun mesajlar gösterilir ve işlemler tamamlandıktan sonra gerekli temizleme işlemleri yapılır.

private void btnExit\_Click(object sender, EventArgs e)

{

DialogResult cıkıs = MessageBox.Show("Çıkış yapmak istediğinize emin misiniz ?", "", MessageBoxButtons.YesNo);

if (cıkıs == DialogResult.Yes)

{

Application.Exit();

}

}

Bu C# kod bloğu, bir Windows Forms uygulamasında çıkış yapma işlemi için bir olay işleyicisi içerir. İşte bu metodun ayrıntılı açıklaması:

1. **private void btnExit\_Click(object sender, EventArgs e)**
   * "Çıkış" butonuna tıklandığında tetiklenen olay işleyicisidir.
2. **DialogResult cıkıs = MessageBox.Show("Çıkış yapmak istediğinize emin misiniz ?", "", MessageBoxButtons.YesNo);**
   * Bir iletişim penceresi (MessageBox) görüntüler ve kullanıcının çıkış yapmak isteyip istemediğini sormak için "Emin misiniz?" sorusu içerir. Kullanıcıya "Evet" ve "Hayır" seçenekleri sunulur.
3. **if (cıkıs == DialogResult.Yes)**
   * Kullanıcı "Evet" seçeneğini seçerse, uygulama kapatılır (**Application.Exit()**).

Bu metodun temel amacı, kullanıcıya çıkış yapmak isteyip istemediğini sormak ve kullanıcının "Evet" seçeneğini seçmesi durumunda uygulamayı kapatmaktır.

using DernekUyeTakipSistemi.Admin.MenuLayer.EmployeeEditLayer.EmployeeEditMenu;

using DernekUyeTakipSistemi.Admin.MenuLayer.GraphLayer.GraphMenu;

using DernekUyeTakipSistemi.Employee.MenuLayer.ListLayer.ListMenu;

using DernekUyeTakipSistemi.Employee.MenuLayer.CustomerEditLayer;

using DernekUyeTakipSistemi.Employee;

using System;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Admin

{

public partial class AMenu : Form

{

EmployeeEditMenu employeeMenu;

CustomerEditMenu customerMenu;

ListMenu listMenu;

DuesMenu duesMenu;

GraphMenu graphMenu;

public AMenu()

{

InitializeComponent();

}

private void btnEmployee\_Click(object sender, EventArgs e)

{

if (employeeMenu == null || employeeMenu.IsDisposed)

{

employeeMenu = new EmployeeEditMenu();

employeeMenu.Show();

this.Hide();

}

else { MessageBox.Show("Çalışan Menüsüne Ulaşırken Bir HATA Oluştu."); }

}

private void btnCustomer\_Click(object sender, EventArgs e)

{

if (customerMenu == null || customerMenu.IsDisposed)

{

customerMenu = new CustomerEditMenu();

customerMenu.Show();

this.Hide();

}

else { MessageBox.Show("Müşteri Menüsüne Ulaşırken Bir HATA Oluştu."); }

}

Bu C# kodu, bir dernek üye takip sisteminde yönetici (admin) kullanıcı arayüzü (UI) için tasarlanmış bir formu temsil eder. Aşağıda kodun metinsel açıklaması bulunmaktadır:

1. **Namespace'ler:**
   * Kod, farklı namespace'leri kullanarak organize edilmiştir. Bu namespace'ler, farklı katmanlardaki sınıfları gruplamak ve organize etmek için kullanılır.
2. **Form Sınıfı ve Nesneler:**
   * **AMenu** sınıfı, admin kullanıcı arayüzünü temsil eden bir formu ifade eder.
   * **EmployeeEditMenu**, **CustomerEditMenu**, **ListMenu**, **DuesMenu**, ve **GraphMenu** sınıfları, farklı menü katmanlarını temsil eden nesnelerdir. Bu sınıflar muhtemelen farklı ekranlarda veya işlevlerde kullanılan kullanıcı arayüzlerini içerir.
3. **Constructor (Yapıcı Metot):**
   * **AMenu** sınıfının yapıcı metodu, formun başlatılması sırasında çağrılır. Bu metotta, başlatma işlemleri ve gerekirse başlangıç değerleri atanır.
4. **Buton Tıklama Olayları:**
   * **btnEmployee\_Click** ve **btnCustomer\_Click** metodları, ilgili butonlara tıklanıldığında gerçekleşen olayları yönetir.
   * Her iki metot da, ilgili menü formunun nesnesini kontrol eder. Eğer ilgili form daha önce oluşturulmamış veya kapatılmışsa, yeni bir nesne oluşturarak ekranda gösterir ve mevcut formu gizler. Aksi halde bir hata mesajı gösterir.

Bu kodun temel amacı, admin kullanıcısının farklı menüleri açmasına olanak tanımak ve bu menüler arasında geçiş yapmasını sağlamaktır.

private void btnAidat\_Click(object sender, EventArgs e)

{

if (duesMenu == null || duesMenu.IsDisposed)

{

duesMenu = new DuesMenu();

duesMenu.Show();

this.Hide();

}

else { MessageBox.Show("Aidat Menüsüne Ulaşırken Bir HATA Oluştu."); }

}

private void btnList\_Click(object sender, EventArgs e)

{

if (listMenu == null || listMenu.IsDisposed)

{

listMenu = new ListMenu();

listMenu.Show();

this.Hide();

}

else { MessageBox.Show("Listeleme Menüsüne Ulaşırken Bir HATA Oluştu."); }

}

private void btnGraph\_Click(object sender, EventArgs e)

{

if (graphMenu == null || graphMenu.IsDisposed)

{

graphMenu = new GraphMenu();

graphMenu.Show();

this.Hide();

}

else { MessageBox.Show("Grafik Menüsüne Ulaşırken Bir HATA Oluştu."); }

}

private void btnExit\_Click(object sender, EventArgs e)

{

DialogResult cıkıs = MessageBox.Show("Çıkış yapmak istediğinize emin misiniz ?", "", MessageBoxButtons.YesNo);

if (cıkıs == DialogResult.Yes)

{

LoginPage loginPage = Application.OpenForms["LoginPage"] as LoginPage;

loginPage.Show();

this.Close();

}

}

Bu C# kodu, bir kullanıcı arayüzünde çeşitli butonlara tıklama olaylarını yöneten bir form sınıfını temsil eder. Aşağıda kodun metinsel açıklaması bulunmaktadır:

1. **Aidat Butonu Tıklama Olayı (btnAidat\_Click):**
   * Eğer **duesMenu** nesnesi daha önce oluşturulmamış veya kapatılmışsa, yeni bir **DuesMenu** nesnesi oluşturularak ekranda gösterilir ve mevcut form gizlenir. Aksi halde bir hata mesajı gösterilir.
2. **Liste Butonu Tıklama Olayı (btnList\_Click):**
   * Eğer **listMenu** nesnesi daha önce oluşturulmamış veya kapatılmışsa, yeni bir **ListMenu** nesnesi oluşturularak ekranda gösterilir ve mevcut form gizlenir. Aksi halde bir hata mesajı gösterilir.
3. **Grafik Butonu Tıklama Olayı (btnGraph\_Click):**
   * Eğer **graphMenu** nesnesi daha önce oluşturulmamış veya kapatılmışsa, yeni bir **GraphMenu** nesnesi oluşturularak ekranda gösterilir ve mevcut form gizlenir. Aksi halde bir hata mesajı gösterilir.
4. **Çıkış Butonu Tıklama Olayı (btnExit\_Click):**
   * Kullanıcıya çıkış yapmak isteyip istemediği sorulur. Eğer kullanıcı "Evet" derse, giriş sayfasının (**LoginPage**) nesnesi alınır ve gösterilir. Mevcut form ise kapatılır.

Bu kodun temel amacı, çeşitli butonlara tıklanıldığında ilgili menüleri veya işlemleri yönetmek ve kullanıcı arayüzü üzerinde geçişler sağlamaktır.

using BusinessLayer;

using DernekUyeTakipSistemi.Admin.MenuLayer.EmployeeEditLayer.MenuLayer;

using DernekUyeTakipSistemi.Employee.MenuLayer.EmployeeInfoLayer.MenuLayer;

using System;

using System.Collections.Generic;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Admin.MenuLayer.EmployeeEditLayer.EmployeeEditMenu

{

public partial class EmployeeEditMenu : Form

{

BlEmployee \_blEmployee;

public EmployeeEditEkle InsertEmployee;

public EmployeeEditDuzenle UpdateEmployee;

public EmployeeEditLogin EmployeeEditLogin;

public EmployeeEditSearch SearchEmployee;

string tcno;

private bool uygulamaKapatildi = false;

public EmployeeEditMenu()

{

InitializeComponent();

\_blEmployee = new BlEmployee();

goster();

}

public void goster()

{

ekran.DataSource = \_blEmployee.TabloSorgu("1");

ekran.AutoResizeColumns();

}

private void dataGridView1\_SelectionChanged(object sender, EventArgs e)

{

if (ekran.SelectedCells.Count > 0)

{

int selectedRowIndex = ekran.SelectedCells[0].RowIndex;

// TC No sütununun index'i

int tcNoColumnIndex = ekran.Columns["Tc\_No"].Index;

// Seçili satırdaki TC No'yu alın

tcno = ekran.Rows[selectedRowIndex].Cells[tcNoColumnIndex].Value.ToString();

if (EmployeeEditLogin != null)

{

EmployeeEditLogin.TcAtama(tcno);

}

if(EmployeeEditLogin == null || EmployeeEditLogin.IsDisposed)

{

EmployeeEditLogin = new EmployeeEditLogin();

EmployeeEditLogin.TcAtama(tcno);

}

}

}

Bu C# kodu, bir çalışanın bilgilerini düzenlemek için kullanılan bir menü formunu temsil eder. Aşağıda kodun metinsel açıklaması bulunmaktadır:

1. **Namespace ve Sınıf Tanımlamaları:**
   * Kod, belirli namespace'ler altında ve **EmployeeEditMenu** adlı bir sınıf içerisinde bulunmaktadır.
2. **Sınıf Değişkenleri ve Nesneler:**
   * **\_blEmployee**: **BlEmployee** tipinde bir nesne oluşturulmuş ve bu nesne, çalışan bilgileriyle ilgili işlemleri gerçekleştiren bir iş katmanı sınıfını temsil eder.
   * **InsertEmployee**, **UpdateEmployee**, **EmployeeEditLogin**, ve **SearchEmployee**: Bu nesneler, farklı işlemleri gerçekleştirmek için kullanılacak olan alt menülerin nesnelerini temsil eder.
   * **tcno**: Seçilen çalışanın TC kimlik numarasını saklamak için kullanılan bir string değişkenidir.
   * **uygulamaKapatildi**: Uygulamanın kapatılıp kapatılmadığını takip etmek için kullanılan bir boolean değişkenidir.
3. **Constructor (Yapıcı Metot):**
   * **EmployeeEditMenu** sınıfının yapıcı metodu, formun başlatılması sırasında çağrılır. Bu metotta, başlatma işlemleri ve gerekirse başlangıç değerleri atanır. Ayrıca **goster** fonksiyonu çağrılarak çalışan bilgileri ekranda gösterilir.
4. **goster Metodu:**
   * **\_blEmployee** nesnesi aracılığıyla çalışan bilgilerini alarak, bu bilgileri bir DataGridView (**ekran**) üzerinde gösterir.
5. **dataGridView1\_SelectionChanged Olayı:**
   * DataGridView'da bir hücre seçildiğinde gerçekleşen olaydır.
   * Seçili satırın indeksini ve TC No sütununun indeksini alır.
   * Seçili satırdaki TC No'yu **tcno** değişkenine atar.
   * Eğer **EmployeeEditLogin** nesnesi varsa ve bu nesne henüz kapatılmamışsa, seçilen TC No'yu bu nesneye atar.
   * Eğer **EmployeeEditLogin** nesnesi null veya kapatılmışsa, yeni bir nesne oluşturarak seçilen TC No'yu bu nesneye atar.

Bu kodun temel amacı, çalışan bilgilerini düzenleme işlemleri için kullanılan bir menü formunu temsil etmek ve bu form üzerinden çeşitli alt menülere geçiş yapabilmektir.

private Dictionary<string, object> GetCellData(int rowIndex, int columnIndex)

{

Dictionary<string, object> cellData = new Dictionary<string, object>();

// Seçilen hücrenin verilerini sözlüğe ekle

foreach (DataGridViewCell cell in ekran.Rows[rowIndex].Cells)

{

cellData[cell.OwningColumn.Name] = cell.Value;

}

return cellData;

}

private void ekran\_CellClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex >= 0 && e.ColumnIndex >= 0)

{

// Seçilen satır ve sütun indekslerini al

int selectedRowIndex = e.RowIndex;

int selectedColumnIndex = e.ColumnIndex;

// Seçilen hücrenin verilerini diğer forma iletmek için olayı tetikle

if (UpdateEmployee != null)

{

UpdateEmployee.OnVeriSecildi(GetCellData(selectedRowIndex, selectedColumnIndex));

}

}

}

public void TcArama(string TcNo)

{

ekran.DataSource = \_blEmployee.TcArama(TcNo);

ekran.AutoResizeColumns();

}

public void AdSoyadArama(string Ad, String Soyad)

{

ekran.DataSource = \_blEmployee.AdSoyadArama(Ad, Soyad);

ekran.AutoResizeColumns();

}

public void TelArama(string Tel)

{

ekran.DataSource = \_blEmployee.TelArama(Tel);

ekran.AutoResizeColumns();

}

public void MailArama(string Mail)

{

ekran.DataSource = \_blEmployee.MailArama(Mail);

ekran.AutoResizeColumns();

}

Bu C# kodu, bir DataGridView üzerinde hücre tıklama olaylarını ele alan ve çalışan bilgilerini aramak için kullanılan bir sınıfı temsil eder. Aşağıda kodun metinsel açıklaması bulunmaktadır:

1. **GetCellData Metodu:**
   * **GetCellData** metodu, belirtilen satır ve sütundaki hücrenin verilerini bir sözlük içinde toplar. Sözlük, hücre değerlerini ilgili sütun adlarıyla eşleştirir.
2. **ekran\_CellClick Olayı:**
   * **ekran\_CellClick** olayı, DataGridView üzerinde bir hücreye tıklandığında gerçekleşir.
   * Seçilen hücrenin satır ve sütun indeksleri alınır.
   * **UpdateEmployee** nesnesi varsa ve null değilse, seçilen hücrenin verilerini **GetCellData** metodu aracılığıyla alır ve **UpdateEmployee** nesnesinin **OnVeriSecildi** metodu iletilir.
3. **TcArama Metodu:**
   * **TcArama** metodu, TC kimlik numarasına göre çalışanları aramak için kullanılır. **\_blEmployee** nesnesi aracılığıyla ilgili iş katmanı sınıfından veri çekilir ve DataGridView'e atanır.
4. **AdSoyadArama Metodu:**
   * **AdSoyadArama** metodu, ad ve soyadına göre çalışanları aramak için kullanılır. **\_blEmployee** nesnesi aracılığıyla ilgili iş katmanı sınıfından veri çekilir ve DataGridView'e atanır.
5. **TelArama Metodu:**
   * **TelArama** metodu, telefon numarasına göre çalışanları aramak için kullanılır. **\_blEmployee** nesnesi aracılığıyla ilgili iş katmanı sınıfından veri çekilir ve DataGridView'e atanır.
6. **MailArama Metodu:**
   * **MailArama** metodu, e-posta adresine göre çalışanları aramak için kullanılır. **\_blEmployee** nesnesi aracılığıyla ilgili iş katmanı sınıfından veri çekilir ve DataGridView'e atanır.

Bu kodun temel amacı, DataGridView üzerindeki hücre tıklama olaylarını ele alarak çalışan bilgilerini görüntülemek ve arama işlemlerini gerçekleştirmektir.

private void btnInsert\_Click(object sender, EventArgs e)

{

if (InsertEmployee == null || InsertEmployee.IsDisposed)

{

InsertEmployee = new EmployeeEditEkle();

InsertEmployee.Show();

}

else

{

InsertEmployee.Show();

}

}

private void btnDelete\_Click(object sender, EventArgs e)

{

bool deleteSonuc = \_blEmployee.SilmeSorgu(tcno);

if (deleteSonuc)

{

MessageBox.Show("Silme işlemi başarılı.");

goster();

}

else

{

MessageBox.Show("Silme işlemi başarısız.");

}

}

private void btnUpdate\_Click(object sender, EventArgs e)

{

if (UpdateEmployee == null || UpdateEmployee.IsDisposed)

{

UpdateEmployee = new EmployeeEditDuzenle();

UpdateEmployee.Show();

}

}

private void btnLogin\_Click(object sender, EventArgs e)

{

if (EmployeeEditLogin == null || EmployeeEditLogin.IsDisposed)

{

EmployeeEditLogin = new EmployeeEditLogin();

EmployeeEditLogin.Show();

}

if(EmployeeEditLogin != null)

{

EmployeeEditLogin.Show();

}

}

Bu C# kodu, bir kullanıcı arayüzünde bulunan butonlara tıklama olaylarını ele alan bir sınıfı temsil eder. Aşağıda kodun metinsel açıklaması bulunmaktadır:

1. **btnInsert\_Click Metodu:**
   * "Ekle" butonuna tıklandığında gerçekleşen olayı ele alır.
   * Eğer **InsertEmployee** nesnesi daha önce oluşturulmamış veya kapatılmışsa, yeni bir **EmployeeEditEkle** nesnesi oluşturularak gösterilir.
   * Aksi halde mevcut nesne gösterilir.
2. **btnDelete\_Click Metodu:**
   * "Sil" butonuna tıklandığında gerçekleşen olayı ele alır.
   * **\_blEmployee.SilmeSorgu** metodu ile seçili çalışanın TC kimlik numarasına göre silme işlemi gerçekleştirilir.
   * Silme işlemi başarılıysa bilgilendirme mesajı gösterilir ve DataGridView güncellenir.
   * Silme işlemi başarısızsa hata mesajı gösterilir.
3. **btnUpdate\_Click Metodu:**
   * "Güncelle" butonuna tıklandığında gerçekleşen olayı ele alır.
   * Eğer **UpdateEmployee** nesnesi daha önce oluşturulmamış veya kapatılmışsa, yeni bir **EmployeeEditDuzenle** nesnesi oluşturularak gösterilir.
4. **btnLogin\_Click Metodu:**
   * "Giriş" butonuna tıklandığında gerçekleşen olayı ele alır.
   * Eğer **EmployeeEditLogin** nesnesi daha önce oluşturulmamış veya kapatılmışsa, yeni bir **EmployeeEditLogin** nesnesi oluşturularak gösterilir.
   * Eğer **EmployeeEditLogin** nesnesi varsa, tekrar gösterilir. Ancak bu kodun mevcut hali biraz gereksiz bir kontrol içermektedir çünkü önce nesne kontrolü yapılmış, ardından aynı nesne yine gösterilmeye çalışılmıştır.

Bu kodun temel amacı, kullanıcı arayüzündeki butonlara tıklandığında ilgili işlemleri gerçekleştirmek ve gerekli ekranları göstermektir.

private void btnSearch\_Click(object sender, EventArgs e)

{

if (SearchEmployee == null || SearchEmployee.IsDisposed)

{

SearchEmployee = new EmployeeEditSearch();

SearchEmployee.Show();

}

else if (SearchEmployee != null)

{

SearchEmployee.Show();

}

}

private void btnRefresh\_Click(object sender, EventArgs e)

{

goster();

}

public void FormlariKapa()

{

if (InsertEmployee != null)

{

InsertEmployee.Close();

}

if (UpdateEmployee != null)

{

UpdateEmployee.Close();

}

if (EmployeeEditLogin != null)

{

EmployeeEditLogin.Close();

}

}

private void btnGeri\_Click(object sender, EventArgs e)

{

FormlariKapa();

uygulamaKapatildi = true;

this.Close();

AMenu aMenu = Application.OpenForms["AMenu"] as AMenu;

aMenu?.Show();

}

Bu C# kodu, bir kullanıcı arayüzünde bulunan butonlara tıklama olaylarını ve bazı işlemleri ele alan bir sınıfı temsil eder. Aşağıda kodun metinsel açıklaması bulunmaktadır:

1. **btnSearch\_Click Metodu:**
   * "Arama" butonuna tıklandığında gerçekleşen olayı ele alır.
   * Eğer **SearchEmployee** nesnesi daha önce oluşturulmamış veya kapatılmışsa, yeni bir **EmployeeEditSearch** nesnesi oluşturularak gösterilir.
   * Eğer **SearchEmployee** nesnesi varsa, tekrar gösterilir. Ancak bu kontrol şu anda gereksiz gibi görünmektedir.
2. **btnRefresh\_Click Metodu:**
   * "Yenile" butonuna tıklandığında gerçekleşen olayı ele alır.
   * **goster** metodunu çağırarak DataGridView'i günceller.
3. **FormlariKapa Metodu:**
   * Açık olan diğer formları kapatmak için kullanılır.
   * **InsertEmployee**, **UpdateEmployee**, ve **EmployeeEditLogin** nesneleri varsa, bunları kapatır.
4. **btnGeri\_Click Metodu:**
   * "Geri" butonuna tıklandığında gerçekleşen olayı ele alır.
   * **FormlariKapa** metodu ile diğer formları kapatır.
   * **uygulamaKapatildi** değişkenini **true** olarak ayarlar.
   * Şu anki formu kapatır.
   * Ana menü formunu (**AMenu**) bulur ve eğer bulunursa gösterir.

Bu kodun temel amacı, kullanıcı arayüzünde bulunan butonlara tıklanması durumunda ilgili işlemleri gerçekleştirmek ve diğer formları kontrol etmek. Özellikle **FormlariKapa** metodu, açık olan diğer formları kapatma işlevini yerine getirir.

private void EmployeeEditMenu\_FormClosing(object sender, FormClosingEventArgs e)

{

if (!uygulamaKapatildi)

{

DialogResult cikis = MessageBox.Show("Çıkış yapmak istediğinize emin misiniz ?", "", MessageBoxButtons.YesNo);

if (cikis == DialogResult.Yes)

{

uygulamaKapatildi = true;

Application.Exit();

}

else

{

e.Cancel = true;

}

}

}

Bu C# kodu, bir form kapatılmak istendiğinde gerçekleşen olayı ele alır. Aşağıda kodun metinsel açıklaması bulunmaktadır:

* **Form Kapatılma Olayı (EmployeeEditMenu\_FormClosing):**
  + Bu olay, **EmployeeEditMenu** formu kapatılmaya çalışıldığında gerçekleşir.
  + **FormClosingEventArgs** tipindeki **e** parametresi aracılığıyla form kapatma işlemi kontrol edilir.
  + **uygulamaKapatildi** değişkeni, uygulamanın zaten kapatılıp kapatılmadığını takip eder.
  + **Kontroller:**
    - Eğer **uygulamaKapatildi** false ise (uygulama kapatılmamışsa):
      * Kullanıcıya "Çıkış yapmak istediğinize emin misiniz?" diye soran bir ileti penceresi gösterilir (**MessageBox**).
      * Kullanıcı "Evet" derse:
        + **uygulamaKapatildi** değişkeni true olarak ayarlanır.
        + **Application.Exit()** metodu çağrılarak uygulama kapatılır.
      * Kullanıcı "Hayır" derse:
        + **e.Cancel** özelliği true olarak ayarlanır, bu da formun kapatma işleminin iptal edileceğini belirtir.

Bu kodun temel amacı, kullanıcı uygulamadan çıkış yapmak istediğinde bir onay ileti penceresi göstermek ve kullanıcının seçimine göre işlemi yönlendirmektir.

using BusinessLayer;

using System;

using System.Collections.Generic;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Admin.MenuLayer.EmployeeEditLayer.MenuLayer

{

public partial class EmployeeEditDuzenle : Form

{

BlEmployee \_blEmployee;

public EmployeeEditDuzenle()

{

InitializeComponent();

\_blEmployee = new BlEmployee();

txtTc.MaxLength = 11;

txtAd.MaxLength = 15;

txtSoyad.MaxLength = 15;

txtTel.MaxLength = 15;

txtmail.MaxLength = 60;

txtAdres.MaxLength = 60;

this.AcceptButton = btnGuncelle;

FillIl();

}

private void txtTc\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsDigit(e.KeyChar) && e.KeyChar != (char)Keys.Back)

{

e.Handled = true;

}

}

private void txtAd\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsLetter(e.KeyChar) && e.KeyChar != (char)Keys.Back && e.KeyChar != (char)Keys.Space)

{

e.Handled = true;

}

}

private void txtSoyad\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsLetter(e.KeyChar) && e.KeyChar != (char)Keys.Back && e.KeyChar != (char)Keys.Space)

{

e.Handled = true;

}

}

Bu C# kodu, bir kullanıcı arayüzü formunu temsil eder. Aşağıda kodun metinsel açıklaması bulunmaktadır:

1. **Sınıf ve Değişken Tanımlamaları:**
   * **EmployeeEditDuzenle** sınıfı, bir formu temsil eder.
   * **\_blEmployee**: **BlEmployee** tipinde bir nesne oluşturularak, çalışan işlemlerini gerçekleştiren iş katmanı sınıfını temsil eder.
2. **Yapıcı Metot (EmployeeEditDuzenle):**
   * Form başlatıldığında çalışan metotlardır.
   * İlgili metotlar çağrılarak formun başlangıç durumu ayarlanır.
   * Textbox'ların maksimum karakter uzunlukları belirlenir.
   * **AcceptButton** özelliği, Enter tuşuna basıldığında **btnGuncelle** butonuna odaklanmasını sağlar.
   * **FillIl** metodu, muhtemelen bir ComboBox'ı il ilişkilendirmek için çağrılır (ancak kod içinde gözükmemektedir).
3. **txtTc\_KeyPress Metodu:**
   * "Tc" TextBox'ına girilen karakterlerin kontrolü yapılır.
   * Eğer karakter bir rakam değilse ve geri tuşu da değilse (**Keys.Back**), karakter kabul edilmez.
4. **txtAd\_KeyPress ve txtSoyad\_KeyPress Metotları:**
   * "Ad" ve "Soyad" TextBox'larına girilen karakterlerin kontrolü yapılır.
   * Eğer karakter bir harf değilse, geri tuşu değiilse ve boşluk karakteri de değilse, karakter kabul edilmez.

Bu kodun temel amacı, bir kullanıcı arayüzü formunda çalışan işlemlerini gerçekleştirmek ve belirli TextBox'ların kullanıcı girişlerini sınırlamak için gerekli kontrolleri sağlamaktır.

private void txtTel\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsDigit(e.KeyChar) && e.KeyChar != (char)Keys.Back)

{

e.Handled = true;

}

}

private void imlec(int position)

{

txtTel.SelectionStart = position;

txtTel.Focus();

}

private int eskiuzunluk = 0;

private void txtTel\_TextChanged(object sender, EventArgs e)

{

string text = txtTel.Text.Replace("(", "").Replace(")", "").Replace(" ", "");

StringBuilder formattedNumber = new StringBuilder();

int yeniuzunluk = text.Length;

if (yeniuzunluk > eskiuzunluk)

{

if (text.Length > 0)

{

formattedNumber.Append("(" + text.Substring(0, Math.Min(3, text.Length)) + ")");

yeniuzunluk = text.Length;

}

if (text.Length > 3)

{

formattedNumber.Append(" " + text.Substring(3, Math.Min(3, text.Length - 3)));

yeniuzunluk = text.Length;

}

if (text.Length > 6)

{

formattedNumber.Append(" " + text.Substring(6, Math.Min(2, text.Length - 6)));

yeniuzunluk = text.Length;

}

if (text.Length > 8)

{

formattedNumber.Append(" " + text.Substring(8, Math.Max(0, text.Length - 8)));

yeniuzunluk = text.Length;

}

txtTel.Text = formattedNumber.ToString();

txtTel.SelectionStart = txtTel.Text.Length;

}

if (yeniuzunluk < eskiuzunluk)

{

if (text.Length == 3)

{

imlec(4);

}

if (text.Length == 0)

{

txtTel.Text = string.Empty;

}

}

eskiuzunluk = yeniuzunluk;

}

Bu C# kodu, bir telefon numarası TextBox'ının kullanıcı girişini kontrol etmek ve formatlamak için kullanılmaktadır. Aşağıda kodun metinsel açıklaması bulunmaktadır:

1. **txtTel\_KeyPress Metodu:**
   * "Telefon Numarası" TextBox'ına girilen karakterlerin kontrolü yapılır.
   * Eğer karakter bir rakam değilse ve geri tuşu değiilse (**Keys.Back**), karakter kabul edilmez.
2. **imlec Metodu:**
   * TextBox içinde belirli bir konuma imleci (cursor) yerleştirmek için kullanılır.
   * **position** parametresi, imlecin yerleştirileceği konumu belirtir.
   * **SelectionStart** özelliği ile imleç konumu belirlenir.
3. **txtTel\_TextChanged Metodu:**
   * "Telefon Numarası" TextBox'ında metin değiştiğinde gerçekleşen olayı ele alır.
   * Girilen metni formatlamak ve belirli bir telefon numarası formatına dönüştürmek amacıyla kullanılır.
   * **text** değişkeni, metinden parantez, boşluk ve diğer karakterleri temizleyerek bir sayı dizisini temsil eder.
   * **formattedNumber** adında bir **StringBuilder** nesnesi oluşturulur ve metin belirli bir formata göre düzenlenir.
   * **eskiuzunluk** değişkeni, önceki metnin uzunluğunu tutar.
   * Eğer yeni eklenen karakterler varsa, belirli aralıklarda boşluk ve parantez ekleyerek telefon numarasını formatlar.
   * **eskiuzunluk** ve **yeniuzunluk** değişkenleri, metnin uzunluğunu takip eder.
   * Eğer karakter silinirse, önceki imleç konumu kontrol edilir ve gerekirse yeni bir imleç konumu belirlenir.

Bu kodun temel amacı, kullanıcının telefon numarası girişini kontrol etmek ve belirli bir formata göre düzenlemektir.

private void txtmail\_TextChanged(object sender, EventArgs e)

{

KontrolEt();

}

private int KontrolEt()

{

string email = txtmail.Text;

if (IsValidEmail(email))

{

txtmail.ForeColor = Color.Black;

return 1;

}

else

{

txtmail.ForeColor = Color.Red;

return 0;

}

}

private bool IsValidEmail(string email)

{

string emailPattern = @"^[a-zA-Z0-9.\_-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$";

return System.Text.RegularExpressions.Regex.IsMatch(email, emailPattern);

}

private void FillIl()

{

cmil.DataSource = \_blEmployee.IlSorgu();

cmil.DisplayMember = "IlAdi";

cmil.ValueMember = "IlID";

cmil.SelectedIndex = -1;

}

private void cmil\_SelectedIndexChanged(object sender, EventArgs e)

{

if (cmil.SelectedValue != null)

{

DataRowView selectedRow = cmil.SelectedItem as DataRowView;

if (selectedRow != null)

{

int selectedIlID = Convert.ToInt32(selectedRow["IlID"]);

FillIlce(selectedIlID);

}

}

}

private void FillIlce(int ilId)

{

cmilce.DataSource = \_blEmployee.IlceSorgu(ilId);

cmilce.DisplayMember = "IlceAdi";

cmilce.ValueMember = "IlceID";

cmilce.SelectedIndex = -1;

}

Bu C# kodu, bir kullanıcı arayüzü formunu temsil eder. Aşağıda kodun metinsel açıklaması bulunmaktadır:

1. **txtmail\_TextChanged Metodu:**
   * "Eposta" TextBox'ındaki metin değiştikçe gerçekleşen olayı ele alır.
   * **KontrolEt** metodu çağrılarak eposta doğruluğu kontrol edilir.
2. **KontrolEt Metodu:**
   * Eposta adresinin geçerli olup olmadığını kontrol eden bir metodtur.
   * **IsValidEmail** metodunu kullanarak eposta doğruluğunu kontrol eder.
   * Eğer eposta geçerli ise metin rengini siyah olarak ayarlar, aksi takdirde kırmızı yapar.
   * Eposta geçerli ise 1, geçerli değilse 0 döndürür.
3. **IsValidEmail Metodu:**
   * Verilen eposta adresinin belirli bir desene uygun olup olmadığını kontrol eder.
   * Desen, standart bir eposta adresi formatını kontrol eden bir regular expression (regex) içerir.
4. **FillIl Metodu:**
   * Bir ComboBox (**cmil**) içine il verilerini dolduran bir metottur.
   * **\_blEmployee.IlSorgu()** metodu kullanılarak il verileri alınır.
   * ComboBox'ın **DisplayMember** ve **ValueMember** özellikleri ayarlanır.
   * ComboBox'ın başlangıçta seçili bir öğe olmaması için **SelectedIndex** -1 olarak ayarlanır.
5. **cmil\_SelectedIndexChanged Metodu:**
   * İl ComboBox'ındaki seçili öğe değiştikçe gerçekleşen olayı ele alır.
   * Seçilen ilin **IlID** değeri alınarak **FillIlce** metodu çağrılır.
6. **FillIlce Metodu:**
   * İlçe ComboBox'ına ilçe verilerini dolduran bir metottur.
   * **\_blEmployee.IlceSorgu(ilId)** metodu kullanılarak ilçe verileri alınır.
   * ComboBox'ın **DisplayMember** ve **ValueMember** özellikleri ayarlanır.
   * ComboBox'ın başlangıçta seçili bir öğe olmaması için **SelectedIndex** -1 olarak ayarlanır.

Bu kodun temel amacı, bir kullanıcı arayüzü formunda eposta doğruluğunu kontrol etmek, il ve ilçe verilerini ComboBox'lara doldurmak ve bu veriler üzerinde etkileşim sağlamaktır.

private void FillIlce(int ilId)

{

cmilce.DataSource = \_blEmployee.IlceSorgu(ilId);

cmilce.DisplayMember = "IlceAdi";

cmilce.ValueMember = "IlceID";

cmilce.SelectedIndex = -1;

}

private void ClearForm()

{

// Formdaki tüm kontrol değerlerini temizleme işlemi

foreach (Control control in Controls)

{

if (control is System.Windows.Forms.TextBox textBox)

{

textBox.Clear();

}

if (control is System.Windows.Forms.ComboBox comboBox)

{

comboBox.SelectedIndex = -1;

}

}

}

public void OnVeriSecildi(Dictionary<string, object> rowData)

{

txtTc.Text = rowData["Tc\_No"].ToString();

txtAd.Text = rowData["Ad"].ToString();

txtSoyad.Text = rowData["Soyad"].ToString();

cmCins.SelectedItem = rowData["Cinsiyet"].ToString();

cmKan.SelectedItem = rowData["Kan\_Grup"].ToString();

txtTel.Text = rowData["Telefon"].ToString();

txtmail.Text = rowData["Mail"].ToString();

txtAdres.Text = rowData["Adres"].ToString();

cmil.SelectedValue = rowData["IlID"];

cmilce.SelectedValue = rowData["IlceID"];

}

private bool KontrolleriGecerliMi()

{

if (txtAd.Text.Length <= 2 || txtSoyad.Text.Length <= 2)

{

MessageBox.Show("Lütfen Gerçek Adınızı ve Soyadınızı Giriniz.");

return false;

}

if (txtTel.Text.Length != 15)

{

MessageBox.Show("Lütfen Telefon numaranızı uygun formatta giriniz.");

return false;

}

if (KontrolEt() != 1)

{

MessageBox.Show("Lütfen MAİL adresinizi uygun formatta giriniz.");

return false;

}

if (cmil.SelectedIndex == -1 || cmilce.SelectedIndex == -1)

{

MessageBox.Show("Lütfen İl ve İlçe Seçiniz.");

return false;

}

if (txtAdres.Text.Length <= 7)

{

MessageBox.Show("Lütfen Açık Adres Bilginizi Giriniz.");

return false;

}

return true;

}

1. **FillIlce Metodu:**
   * İl ComboBox'ındaki seçilen ilin ilçe verilerini dolduran bir metottur.
   * **\_blEmployee.IlceSorgu(ilId)** metodu kullanılarak ilçe verileri alınır.
   * ComboBox'ın **DisplayMember** ve **ValueMember** özellikleri ayarlanır.
   * ComboBox'ın başlangıçta seçili bir öğe olmaması için **SelectedIndex** -1 olarak ayarlanır.
2. **ClearForm Metodu:**
   * Formdaki tüm TextBox ve ComboBox kontrol değerlerini temizleme işlemi yapar.
   * **foreach** döngüsü ile formdaki kontrol elemanlarına erişilir.
   * TextBox ise **Clear** metodu ile temizlenir, ComboBox ise **SelectedIndex** -1 olarak ayarlanır.
3. **OnVeriSecildi Metodu:**
   * DataGridView'dan seçilen bir satırın verilerini alarak, bu verilerle formdaki alanları dolduran bir metodur.
   * **rowData** parametresi, bir satırdaki verileri içerir.
   * TextBox ve ComboBox'lar bu verilere göre doldurulur.
4. **KontrolleriGecerliMi Metodu:**
   * Formdaki giriş alanlarının geçerliliğini kontrol eden bir metottur.
   * Gerçek ad ve soyadın en az 2 karakter uzunluğunda olup olmadığı kontrol edilir.
   * Telefon numarasının 15 karakter uzunluğunda olup olmadığı kontrol edilir.
   * Eposta adresinin geçerli olup olmadığı **KontrolEt** metodu kullanılarak kontrol edilir.
   * İl ve ilçe seçimlerinin yapılmış olup olmadığı kontrol edilir.
   * Açık adres bilgisinin en az 7 karakter uzunluğunda olup olmadığı kontrol edilir.
   * Eğer tüm kontroller geçerliyse **true** döndürülür, aksi takdirde **false** döndürülür.

Bu kod bloğu, bir formun bazı işlevlerini gerçekleştirmektedir. **OnVeriSecildi** metodu genellikle DataGridView'dan seçilen bir satırın verilerini form elemanlarına aktarmak için kullanılır. **KontrolleriGecerliMi** metodu, kullanıcının formdaki alanları doğru bir şekilde doldurup doldurmadığını kontrol eder. Eğer kontrol edilen koşullar sağlanmazsa, kullanıcıya uygun hata mesajları gösterilir.

private void btnGuncelle\_Click(object sender, EventArgs e)

{

if (!KontrolleriGecerliMi())

{

return;

}

int ilID = cmil.SelectedIndex >= 0 ? Convert.ToInt32(cmil.SelectedValue) : -1;

int ilceID = cmilce.SelectedIndex >= 0 ? Convert.ToInt32(cmilce.SelectedValue) : -1;

bool insertSonuc = \_blEmployee.GuncellemeSorgu(txtTc.Text, txtAd.Text, txtSoyad.Text, txtTel.Text, txtmail.Text, txtAdres.Text, ilID, ilceID);

if (insertSonuc)

{

MessageBox.Show("Güncelleme Başarılı!");

ClearForm();

EmployeeEditMenu.EmployeeEditMenu usrmenu = Application.OpenForms["EmployeeEditMenu"] as EmployeeEditMenu.EmployeeEditMenu;

usrmenu?.goster();

txtAd.Focus();

}

else

{

MessageBox.Show("Kayıt gerçekleşirken bir hata oluştu.");

}

}

Bu kod, bir form üzerinde bulunan "Güncelle" butonuna tıklanıldığında gerçekleşen olayı (event) temsil eder. İşte kodun metinsel açıklaması:

1. **KontrolleriGecerliMi Metodu:**
   * Form üzerindeki giriş alanlarının geçerliliğini kontrol eden bir metottur.
   * Eğer kontroller geçerli değilse, işlemi sonlandırır ve geri döner.
2. **ilID ve ilceID Değişkenleri:**
   * İl ve ilçe ComboBox'larında seçili öğelerin ID'lerini tutan değişkenler.
   * Eğer seçili bir il veya ilçe yoksa, -1 değerine atanır.
3. **\_blEmployee.GuncellemeSorgu Metodu:**
   * Business Layer'da bulunan **\_blEmployee** nesnesinin **GuncellemeSorgu** metodunu çağırarak veritabanında güncelleme işlemi gerçekleştirilir.
   * Metoda, güncellenecek kaydın detayları ve il/ilçe ID'leri parametre olarak verilir.
   * Metodun dönüş değeri, güncelleme işleminin başarılı olup olmadığını belirten bir boolean değeridir.
4. **Güncelleme Sonuçlarına Göre Mesaj Kutuları:**
   * Eğer güncelleme başarılıysa, kullanıcıya "Güncelleme Başarılı!" mesajını gösterir.
   * Formdaki tüm giriş alanlarını temizler (**ClearForm** metodu).
   * Kullanıcının görebileceği bir başka form olan **EmployeeEditMenu**'yu günceller (**goster** metodunu çağırır).
   * İlk giriş alanına odaklanır (focus).
   * Eğer güncelleme başarısızsa, "Kayıt gerçekleşirken bir hata oluştu." mesajını gösterir.

Bu kod bloğu, bir form üzerindeki giriş alanlarındaki verileri kullanarak bir güncelleme işlemi gerçekleştirir. Eğer güncelleme başarılıysa, kullanıcıya bilgi mesajları gösterilir ve form temizlenir. Eğer güncelleme başarısızsa, kullanıcıya bir hata mesajı gösterilir.

using BusinessLayer;

using DernekUyeTakipSistemi.Admin.MenuLayer.EmployeeEditLayer.EmployeeEditMenu;

using System;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Admin.MenuLayer.EmployeeEditLayer.MenuLayer

{

public partial class EmployeeEditEkle : Form

{

BlEmployee \_blEmployee;

public EmployeeEditEkle()

{

InitializeComponent();

\_blEmployee = new BlEmployee();

txtTc.MaxLength = 11;

txtAd.MaxLength = 15;

txtSoyad.MaxLength = 15;

txtTel.MaxLength = 15;

txtmail.MaxLength = 60;

txtAdres.MaxLength = 60;

this.AcceptButton = btnEkle;

FillIl();

}

private void txtTc\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsDigit(e.KeyChar) && e.KeyChar != (char)Keys.Back)

{

e.Handled = true;

}

}

private void txtAd\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsLetter(e.KeyChar) && e.KeyChar != (char)Keys.Back && e.KeyChar != (char)Keys.Space)

{

e.Handled = true;

}

}

private void txtSoyad\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsLetter(e.KeyChar) && e.KeyChar != (char)Keys.Back && e.KeyChar != (char)Keys.Space)

{

e.Handled = true;

}

}

private void txtTel\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsDigit(e.KeyChar) && e.KeyChar != (char)Keys.Back)

{

e.Handled = true;

}

}

private void imlec(int position)

{

txtTel.SelectionStart = position;

txtTel.Focus();

}

Bu kod bloğu, bir kullanıcının personel eklemek için kullandığı bir formu temsil eder. İşte kodun metinsel açıklaması:

1. **BlEmployee Nesnesi Oluşturma:**
   * **\_blEmployee** adında bir **BlEmployee** nesnesi oluşturulur. Bu nesne, iş katmanındaki işlemleri gerçekleştirmek için kullanılır.
2. **Giriş Kontrolleri:**
   * Form üzerinde bulunan metin kutularına (TextBox) sadece belirli karakter türlerinin girilmesini sağlayan **KeyPress** olayları tanımlanmıştır.
   * **txtTc**, **txtAd**, ve **txtSoyad** kutularına sadece rakam ve harf girişi kabul ederken, **txtTel** kutusuna sadece rakam girişi kabul eder.
   * Bu kontroller, kullanıcının geçersiz karakter girmesini önler.
3. **Telefon Numarası İçin Formatlama Fonksiyonu:**
   * **imlec** fonksiyonu, telefon numarası kutusuna girilen rakamları belirli bir formatta göstermek için kullanılır.
   * Girilen rakamların sayısına bağlı olarak telefon numarasını parantez içinde gruplar ve boşluklar ekler.
   * Bu fonksiyon, **txtTel\_TextChanged** olayında çağrılır ve her karakter girişinde telefon numarasını formatlar.
4. **Kontrol ve Sınırlamalar:**
   * Her bir metin kutusu için **MaxLength** özellikleri belirlenmiştir. Bu, kullanıcının belirli bir karakter sınırını aşmasını önler.
   * **AcceptButton** özelliği, form üzerinde Enter tuşuna basıldığında çağrılacak olan butonu belirler. Bu durumda **btnEkle** butonu seçilmiştir.
5. **İl ve İlçe Bilgilerini Doldurma:**
   * **FillIl** metodu, ComboBox'lara il ve ilçe bilgilerini doldurmak için kullanılır.
   * İl bilgileri **\_blEmployee.IlSorgu** metodundan alınır ve il ilçe ComboBox'larına atanır.
   * İl ComboBox'u seçildiğinde, ilgili ilin ilçeleri **\_blEmployee.IlceSorgu** metodu ile alınarak ilçe ComboBox'ına atanır.

Bu kod bloğu, kullanıcıya personel eklemek için bir form sağlar. Giriş kontrolleri ve formatlama işlemleri, kullanıcının geçerli ve uygun verileri girmesini sağlamak amacıyla kullanılır.

private int eskiuzunluk = 0;

private void txtTel\_TextChanged(object sender, EventArgs e)

{

string text = txtTel.Text.Replace("(", "").Replace(")", "").Replace(" ", "");

StringBuilder formattedNumber = new StringBuilder();

int yeniuzunluk = text.Length;

if (yeniuzunluk > eskiuzunluk)

{

if (text.Length > 0)

{

formattedNumber.Append("(" + text.Substring(0, Math.Min(3, text.Length)) + ")");

yeniuzunluk = text.Length;

}

if (text.Length > 3)

{

formattedNumber.Append(" " + text.Substring(3, Math.Min(3, text.Length - 3)));

yeniuzunluk = text.Length;

}

if (text.Length > 6)

{

formattedNumber.Append(" " + text.Substring(6, Math.Min(2, text.Length - 6)));

yeniuzunluk = text.Length;

}

if (text.Length > 8)

{

formattedNumber.Append(" " + text.Substring(8, Math.Max(0, text.Length - 8)));

yeniuzunluk = text.Length;

}

txtTel.Text = formattedNumber.ToString();

txtTel.SelectionStart = txtTel.Text.Length;

}

if (yeniuzunluk < eskiuzunluk)

{

if (text.Length == 3)

{

imlec(4);

}

if (text.Length == 0)

{

txtTel.Text = string.Empty;

}

}

eskiuzunluk = yeniuzunluk;

}

private void txtmail\_TextChanged(object sender, EventArgs e)

{

KontrolEt();

}

**Telefon Numarası Formatlama Mantığı**

1. **Değişkenlerin Tanımlanması:**
   * **eskiuzunluk**: Önceki karakter uzunluğunu tutan bir değişken.
   * **text**: Metin kutusundan alınan telefon numarasını temsil eden bir dize.
   * **formattedNumber**: Yeni formata dönüştürülen telefon numarasını tutan **StringBuilder**.
2. **Telefon Numarasının Formatlanması:**
   * **txtTel\_TextChanged** olayı tetiklendiğinde, önceki ve yeni karakter uzunlukları karşılaştırılır.
   * Eğer yeni uzunluk öncekiden büyükse, kullanıcının girişini takip ederek telefon numarasını belirli bir formata dönüştürür.
   * Her bir durumda, belirli bir sayıda karakter eklenir ve metin kutusu güncellenir.
   * **imlec** fonksiyonu, telefon numarasının belirli bir konumuna imleci yerleştirir.
   * Son olarak, önceki uzunluk güncellenir.
3. **Telefon Numarası Silme Kontrolü:**
   * Eğer yeni uzunluk öncekiden küçükse, kullanıcının karakter sildiği anlamına gelir.
   * Eğer sildiği karakter sayısı 3 ise, imleç konumunu 4. karaktere getirir.
   * Eğer telefon numarası tamamen silinmişse, metin kutusunu temizler.

**Email Geçerliliği Kontrolü Mantığı**

1. **txtmail\_TextChanged Olayı:**
   * Email metin kutusu (**txtmail**) üzerinde bir değişiklik olduğunda tetiklenir.
   * **KontrolEt** fonksiyonu çağrılır.
2. **Email Geçerliliği Kontrolü:**
   * **KontrolEt** fonksiyonu, metin kutusundaki email adresinin geçerli olup olmadığını kontrol eder.
   * Geçerli bir email adresi ise metin rengini siyah yapar, değilse kırmızı yapar.
   * Fonksiyon, geçerli olup olmadığına dair bir sayı döndürür (1: geçerli, 0: geçersiz).

private int KontrolEt()

{

string email = txtmail.Text;

if (IsValidEmail(email))

{

txtmail.ForeColor = Color.Black;

return 1;

}

else

{

txtmail.ForeColor = Color.Red;

return 0;

}

}

private bool IsValidEmail(string email)

{

string emailPattern = @"^[a-zA-Z0-9.\_-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$";

return System.Text.RegularExpressions.Regex.IsMatch(email, emailPattern);

}

private void FillIl()

{

cmil.DataSource = \_blEmployee.IlSorgu();

cmil.DisplayMember = "IlAdi";

cmil.ValueMember = "IlID";

cmil.SelectedIndex = -1;

}

private void cmil\_SelectedIndexChanged(object sender, EventArgs e)

{

if (cmil.SelectedValue != null)

{

DataRowView selectedRow = cmil.SelectedItem as DataRowView;

if (selectedRow != null)

{

int selectedIlID = Convert.ToInt32(selectedRow["IlID"]);

FillIlce(selectedIlID);

}

}

}

private void FillIlce(int ilId)

{

cmilce.DataSource = \_blEmployee.IlceSorgu(ilId);

cmilce.DisplayMember = "IlceAdi";

cmilce.ValueMember = "IlceID";

cmilce.SelectedIndex = -1;

}

Bu kod bloğu, bir Windows Forms uygulamasında kullanılan bir dizi fonksiyonun mantığını içerir. İşte bu kodun metinsel açıklaması:

**Email Geçerliliği Kontrolü (KontrolEt ve IsValidEmail Fonksiyonları)**

1. **KontrolEt Fonksiyonu:**
   * **txtmail** adlı TextBox'tan alınan email adresini kontrol eder.
   * Eğer email geçerliyse, metin rengini siyah yapar ve 1 değerini döndürür.
   * Geçerli değilse, metin rengini kırmızı yapar ve 0 değerini döndürür.
2. **IsValidEmail Fonksiyonu:**
   * Parametre olarak gelen email adresini, belirli bir formata uygun olup olmadığını kontrol eder.
   * **^[a-zA-Z0-9.\_-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$** şeklindeki regex deseni ile kontrol yapar.
   * Eğer email adresi belirtilen formata uygunsa, **true** döndürür. Aksi takdirde, **false** döndürür.

**İl ve İlçe Listesi Doldurma (FillIl, cmil\_SelectedIndexChanged, ve FillIlce Fonksiyonları)**

1. **FillIl Fonksiyonu:**
   * İl isimlerini ve ID'lerini içeren bir DataTable'ı **cmil** adlı ComboBox'a doldurur.
   * Gösterilecek metin (DisplayMember) "IlAdi", değer (ValueMember) "IlID" olarak belirlenir.
   * ComboBox'ı başlangıçta seçili bir öğe olmayacak şekilde ayarlar.
2. **cmil\_SelectedIndexChanged Olayı:**
   * **cmil** ComboBox'ında seçim değiştiğinde tetiklenir.
   * Seçilen öğenin DataRowView olarak alınması ve bu öğenin "IlID" değerinin alınarak **FillIlce** fonksiyonunun çağrılması sağlanır.
3. **FillIlce Fonksiyonu:**
   * Parametre olarak alınan il ID'sine bağlı olarak ilçe isimlerini ve ID'lerini içeren bir DataTable'ı **cmilce** adlı ComboBox'a doldurur.
   * Gösterilecek metin (DisplayMember) "IlceAdi", değer (ValueMember) "IlceID" olarak belirlenir.
   * ComboBox'ı başlangıçta seçili bir öğe olmayacak şekilde ayarlar.

private bool KontrolleriGecerliMi()

{

if (txtTc.Text.Length != 11)

{

MessageBox.Show("Lütfen T.C Kimlik Numaranızı Kontrol Ediniz.");

return false;

}

if (txtAd.Text.Length <= 2 || txtSoyad.Text.Length <= 2)

{

MessageBox.Show("Lütfen Gerçek Adınızı ve Soyadınızı Giriniz.");

return false;

}

if (cmCins.SelectedIndex == -1 || cmKan.SelectedIndex == -1)

{

MessageBox.Show("Lütfen Cinsiyetinizi ve Kan Grubunuzu Seçtiğinizden Emin olunuz.");

return false;

}

if (txtTel.Text.Length != 15)

{

MessageBox.Show("Lütfen Telefon numaranızı uygun formatta giriniz.");

return false;

}

if (KontrolEt() != 1)

{

MessageBox.Show("Lütfen MAİL adresinizi uygun formatta giriniz.");

return false;

}

if (cmil.SelectedIndex == -1 || cmilce.SelectedIndex == -1)

{

MessageBox.Show("Lütfen İl ve İlçe Seçiniz.");

return false;

}

if (txtAdres.Text.Length <= 7)

{

MessageBox.Show("Lütfen Açık Adres Bilginizi Giriniz.");

return false;

}

return true;

}

Bu kod bloğu, bir Windows Forms uygulamasında kullanılan iki fonksiyonu içermektedir. Aşağıda her iki fonksiyonun da mantığını metinsel olarak açıklıyorum:

**KontrolleriGecerliMi Fonksiyonu:**

* **Amaç:** Bu fonksiyon, formdaki giriş değerlerinin geçerliliğini kontrol eder.
* **Mantık:**
  + TC Kimlik Numarası, Ad, Soyad, Cinsiyet, Kan Grubu, Telefon, Email, İl, İlçe ve Adres gibi form elemanlarındaki değerlerin geçerliliğini kontrol eder.
  + Eğer bir kontrol başarısız olursa, kullanıcıya bir MessageBox ile hata mesajı gösterilir ve fonksiyon **false** döner.
  + Tüm kontroller başarılı ise, fonksiyon **true** döner.

Geçerlilik Kontrolleri:

1. **TC Kimlik Numarası Kontrolü:**
   * TC Kimlik Numarasının 11 haneli olup olmadığı kontrol edilir.
2. **Ad ve Soyad Uzunluk Kontrolü:**
   * Gerçek ad ve soyadın minimum uzunlukları kontrol edilir.
3. **Cinsiyet ve Kan Grubu Seçimi Kontrolü:**
   * Cinsiyet ve kan grubu ComboBox'larının seçili olup olmadığı kontrol edilir.
4. **Telefon Numarası Uzunluk Kontrolü:**
   * Telefon numarasının 15 karakter uzunluğunda olup olmadığı kontrol edilir.
5. **Email Geçerliliği Kontrolü:**
   * **KontrolEt** fonksiyonu ile emailin geçerli olup olmadığı kontrol edilir.
6. **İl ve İlçe Seçimi Kontrolü:**
   * İl ve ilçe ComboBox'larının seçili olup olmadığı kontrol edilir.
7. **Adres Uzunluk Kontrolü:**
   * Açık adres bilgisinin minimum uzunluğu kontrol edilir.

private void btnEkle\_Click(object sender, EventArgs e)

{

if (!KontrolleriGecerliMi())

{

return;

}

string cinsiyet = cmCins.SelectedItem?.ToString() ?? string.Empty;

string kanGrubu = cmKan.SelectedItem?.ToString() ?? string.Empty;

int ilID = cmil.SelectedIndex >= 0 ? Convert.ToInt32(cmil.SelectedValue) : -1;

int ilceID = cmilce.SelectedIndex >= 0 ? Convert.ToInt32(cmilce.SelectedValue) : -1;

bool insertSonuc = \_blEmployee.EklemeSorgu(txtTc.Text, txtAd.Text, txtSoyad.Text, cinsiyet, kanGrubu, txtTel.Text, txtmail.Text, txtAdres.Text, ilID, ilceID);

if (insertSonuc)

{

\_blEmployee.KullaniciAdBelirle(txtTc.Text, txtAd.Text);

MessageBox.Show("Kayıt Başarılı!");

EmployeeEditMenu.EmployeeEditMenu empMenu = Application.OpenForms["EmployeeEditMenu"] as EmployeeEditMenu.EmployeeEditMenu;

empMenu.goster();

}

else

{

MessageBox.Show("Kayıt gerçekleşirken bir hata oluştu.");

}

}  
  
Bu kod bloğu, bir Windows Forms uygulamasında kullanılan bir "Ekle" butonuna tıklandığında çalışacak olan bir fonksiyonu içermektedir. Aşağıda fonksiyonun mantığını metinsel olarak açıklıyorum:

**btnEkle\_Click Fonksiyonu:**

* **Amaç:** Bu fonksiyon, form üzerindeki giriş değerlerinin geçerliliğini kontrol eder ve ardından bu değerleri kullanarak bir yeni kayıt ekler.
* **Mantık:**
  + **KontrolleriGecerliMi** fonksiyonu çağrılarak formdaki giriş değerlerinin geçerliliği kontrol edilir.
  + Geçerlilik kontrolü başarısız olursa, fonksiyon işlemleri sonlandırır.
  + Cinsiyet ve kan grubu değerleri ComboBox'lardan alınır.
  + İl ve ilçe değerleri seçilmişse ilgili değerler alınır, seçilmemişse -1 atanır.
  + **\_blEmployee.EklemeSorgu** fonksiyonu çağrılarak yeni kayıt eklenir.
  + Kullanıcı adı belirleme işlemi için **\_blEmployee.KullaniciAdBelirle** fonksiyonu çağrılır.
  + Eğer kayıt ekleme işlemi başarılıysa, kullanıcıya bir bilgi mesajı gösterilir ve **EmployeeEditMenu** formu güncellenir.
  + Eğer kayıt ekleme işlemi başarısız olursa, kullanıcıya bir hata mesajı gösterilir.

Notlar:

* **SelectedItem** kullanılarak ComboBox'tan seçilen değer alınır. Eğer bir değer seçilmemişse, **null** değeri kullanılır ve **??** operatörü ile bir boş dize (**string.Empty**) atanır.
* **\_blEmployee.EklemeSorgu** fonksiyonunun geri dönüş değeri başarı durumunu belirtir.

using BusinessLayer;

using System;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Admin.MenuLayer.EmployeeEditLayer.MenuLayer

{

public partial class EmployeeEditLogin : Form

{

int gvnlkkod;

public string Tc\_no;

BlUserLogin \_blUserLogin;

public EmployeeEditLogin()

{

InitializeComponent();

txtKAd1.MaxLength = 9;

txtKAd2.MaxLength = 9;

txtSifre1.MaxLength = 9;

txtSifre2.MaxLength = 9;

txtGuvenlik.MaxLength = 6;

\_blUserLogin = new BlUserLogin();

this.AcceptButton = btnGuncelle;

}

private void LoginEmployee\_Load(object sender, System.EventArgs e)

{

this.Size = new System.Drawing.Size(310, 105);

GuvenlikKodYenile();

mvctKAd.Visible = false;

mvctParola.Visible = false;

gbKAd.Visible = false;

gbParola.Visible = false;

GuvenlikKod.Visible = false;

txtGuvenlik.Visible = false;

lb5.Visible = false;

btnGuncelle.Visible = false;

}

private void txtGuvenlik\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsDigit(e.KeyChar) && e.KeyChar != (char)Keys.Back)

{

e.Handled = true;

}

}

public void TcAtama(string tc)

{

Tc\_no = tc;

mvctKAd.Text = \_blUserLogin.KAdGetir(Tc\_no);

mvctParola.Text = \_blUserLogin.ParolaGetir(Tc\_no);

}

public void MevcutKAd()

{

mvctKAd.Visible = true;

mvctKAd.Text = \_blUserLogin.KAdGetir(Tc\_no);

}

public void MevcutParola()

{

mvctParola.Visible = true;

mvctParola.Text = \_blUserLogin.ParolaGetir(Tc\_no);

}

Bu kod bloğu, bir Windows Forms uygulamasında kullanılan bir "EmployeeEditLogin" formunun kodlarını içermektedir. Aşağıda kodun metinsel açıklamasını bulabilirsiniz:

**EmployeeEditLogin Formu:**

* **Amaç:** Bu form, bir çalışanın kullanıcı adı ve şifresini düzenlemek için kullanılır.
* **Mantık:**
  + **txtKAd1**, **txtKAd2**, **txtSifre1**, **txtSifre2** ve **txtGuvenlik** adlı TextBox kontrollerinin uzunluk sınırlamaları belirlenir.
  + **\_blUserLogin** adlı **BlUserLogin** sınıfından bir örnek oluşturulur.
  + Form yüklenirken (**LoginEmployee\_Load** metodu), bazı kontrol ve öğeler gizlenir veya boyutlandırılır.
  + Kullanıcıya ait güvenlik kodu (**txtGuvenlik**) sadece sayı girişine izin veren bir kontrola sahiptir (**txtGuvenlik\_KeyPress**).
  + **TcAtama** fonksiyonu, çalışanın TC kimlik numarasını alarak ilgili işlemleri gerçekleştirir. Bu işlemler arasında kullanıcı adını ve şifreyi getirme (**\_blUserLogin.KAdGetir** ve **\_blUserLogin.ParolaGetir**) işlemleri bulunur.
  + **MevcutKAd** ve **MevcutParola** fonksiyonları, mevcut kullanıcı adını ve parolayı görüntülemek üzere ilgili kontrol ve öğeleri görünür kılar.

Notlar:

* **MaxLength** özelliği ile TextBox'ların maksimum uzunluğu sınırlanır.
* **\_blUserLogin** sınıfı, muhtemelen iş katmanında kullanıcıyla ilgili işlemleri gerçekleştiren bir sınıftır.
* **LoginEmployee\_Load** metodu, form yüklendiğinde yapılacak başlangıç ayarlarını içerir.
* **txtGuvenlik\_KeyPress** metodu, güvenlik kodu TextBox'ına sadece sayı girişine izin.

private void cbKAd\_CheckedChanged(object sender, EventArgs e)

{

HangisiSecili();

}

private void cbParola\_CheckedChanged(object sender, EventArgs e)

{

HangisiSecili();

}

Bu kod parçası, iki CheckBox kontrolü olan **cbKAd** ve **cbParola**nın durum değişikliklerini takip eden iki olayı içerir. Aşağıda kodun mantığına dair metinsel bir açıklama bulunmaktadır.

* **cbKAd\_CheckedChanged Metodu:**
  + Bu metodun amacı, "cbKAd" adlı CheckBox kontrolünün durumu değiştiğinde tetiklenir.
  + **HangisiSecili** fonksiyonunu çağırarak, hangi CheckBox'ın seçili olduğunu kontrol eder ve ilgili işlemleri gerçekleştirir.
* **cbParola\_CheckedChanged Metodu:**
  + Bu metodun amacı, "cbParola" adlı CheckBox kontrolünün durumu değiştiğinde tetiklenir.
  + **HangisiSecili** fonksiyonunu çağırarak, hangi CheckBox'ın seçili olduğunu kontrol eder ve ilgili işlemleri gerçekleştirir.
* **HangisiSecili Fonksiyonu:**
  + Bu fonksiyonun amacı, hangi CheckBox'ın seçili olduğunu kontrol eder.
  + Eğer "cbKAd" seçiliyse, muhtemelen bir kullanıcı adı ile ilgili işlemler gerçekleştirilecek.
  + Eğer "cbParola" seçiliyse, muhtemelen bir parola ile ilgili işlemler gerçekleştirilecek.

Bu kod parçası, CheckBox'ların durum değişikliklerine tepki vererek kullanıcı arayüzünde hangi işlemlerin gerçekleştirileceğini belirlemek üzere tasarlanmıştır. Ancak, işlevselliğin tam anlaşılması ve detaylı bir açıklama yapılabilmesi için **HangisiSecili** fonksiyonunun içeriği önemlidir. Bu fonksiyonun içeriğine bağlı olarak, seçilen CheckBox'a göre belirli işlemlerin tetikleneceği düşünülebilir.

private void HangisiSecili()

{

if (cbKAd.Checked && !cbParola.Checked)

{

//Sadece kullanıcı adı seçili.

MevcutKAd();

mvctParola.Visible = false;

this.Size = new System.Drawing.Size(310, 300);

gbKAd.Visible = true;

gbParola.Visible = false;

GuvenlikKod.Visible = true;

txtGuvenlik.Visible = true;

lb5.Visible = true;

btnGuncelle.Visible = true;

radioButton1.Visible = true;

radioButton2.Visible = true;

gbKAd.Location = new System.Drawing.Point(25, 70);

GuvenlikKod.Location = new System.Drawing.Point(159, 154);

lb5.Location = new System.Drawing.Point(65, 183);

txtGuvenlik.Location = new System.Drawing.Point(148, 183);

btnGuncelle.Location = new System.Drawing.Point(207, 209);

radioButton1.Location = new System.Drawing.Point(15, 150);

radioButton2.Location = new System.Drawing.Point(15, 172);

}

if (!cbKAd.Checked && cbParola.Checked)

{

//Sadece parola seçili.

MevcutParola();

mvctKAd.Visible = false;

this.Size = new System.Drawing.Size(310, 300);

gbParola.Visible = true;

gbKAd.Visible = false;

GuvenlikKod.Visible = true;

txtGuvenlik.Visible = true;

lb5.Visible = true;

btnGuncelle.Visible = true;

radioButton1.Visible = true;

radioButton2.Visible = true;

gbParola.Location = new System.Drawing.Point(25, 70);

GuvenlikKod.Location = new System.Drawing.Point(159, 154);

lb5.Location = new System.Drawing.Point(65, 183);

txtGuvenlik.Location = new System.Drawing.Point(148, 183);

btnGuncelle.Location = new System.Drawing.Point(207, 209);

radioButton1.Location = new System.Drawing.Point(15, 150);

radioButton2.Location = new System.Drawing.Point(15, 172);

}

if (cbKAd.Checked && cbParola.Checked)

{

//İkiside seçiliyse.

MevcutKAd();

MevcutParola();

this.Size = new System.Drawing.Size(310, 385);

gbKAd.Visible = true;

gbParola.Visible = true;

GuvenlikKod.Visible = true;

txtGuvenlik.Visible = true;

lb5.Visible = true;

btnGuncelle.Visible = true;

radioButton1.Visible = true;

radioButton2.Visible = true;

gbKAd.Location = new System.Drawing.Point(25, 70);

gbParola.Location = new System.Drawing.Point(25, 150);

GuvenlikKod.Location = new System.Drawing.Point(159, 231);

lb5.Location = new System.Drawing.Point(65, 263);

txtGuvenlik.Location = new System.Drawing.Point(148, 263);

btnGuncelle.Location = new System.Drawing.Point(208, 295);

radioButton1.Location = new System.Drawing.Point(26, 295);

radioButton2.Location = new System.Drawing.Point(26, 317);

}

if (!cbKAd.Checked && !cbParola.Checked)

{

mvctKAd.Visible = false;

mvctParola.Visible = false;

this.Size = new System.Drawing.Size(310, 100);

gbKAd.Visible = false;

gbParola.Visible = false;

GuvenlikKod.Visible = false;

txtGuvenlik.Visible = false;

lb5.Visible = false;

btnGuncelle.Visible = false;

radioButton1.Visible = false;

radioButton2.Visible = false;

}

}

Bu kod bloğu, **cbKAd** ve **cbParola** adlı iki CheckBox kontrolünün durumlarına göre kullanıcı arayüzünü düzenleyen bir fonksiyon içermektedir. İşte bu fonksiyonun metinsel açıklaması:

* **HangisiSecili Fonksiyonu:**
  + Bu fonksiyon, **cbKAd** ve **cbParola** CheckBox kontrollerinin durumlarına göre farklı kullanıcı arayüzü düzenlemelerini yapar.
  + İlk olarak, eğer sadece **cbKAd** seçiliyse:
    - Kullanıcı adı ile ilgili işlemleri gerçekleştiren **MevcutKAd** fonksiyonunu çağırır.
    - Kullanıcı arayüzünü sadece kullanıcı adı ile ilgili alanları içerecek şekilde günceller.
  + Ardından, eğer sadece **cbParola** seçiliyse:
    - Parola ile ilgili işlemleri gerçekleştiren **MevcutParola** fonksiyonunu çağırır.
    - Kullanıcı arayüzünü sadece parola ile ilgili alanları içerecek şekilde günceller.
  + Daha sonra, hem **cbKAd** hem de **cbParola** seçiliyse:
    - Hem kullanıcı adı hem de parola ile ilgili işlemleri gerçekleştiren **MevcutKAd** ve **MevcutParola** fonksiyonlarını çağırır.
    - Kullanıcı arayüzünü hem kullanıcı adı hem de parola ile ilgili alanları içerecek şekilde günceller.
  + Son olarak, her iki CheckBox de seçili değilse:
    - Tüm ilgili alanları ve kontrolleri gizler, yani kullanıcı arayüzünü temizler.
    - Boyutu daha küçük bir pencereye ayarlar.
  + Bu fonksiyon, CheckBox'ların durumlarına göre hangi işlemlerin gerçekleştirileceğini belirleyerek kullanıcı arayüzünü dinamik olarak uyarlar.

private void GuvenlikKodYenile()

{

Random random = new Random();

gvnlkkod = random.Next(10000, 99999);

GuvenlikKod.Text = gvnlkkod.ToString();

}

private string Statu()

{

if (radioButton1.Checked)

{

return "A";

}

if (radioButton2.Checked)

{

return "D";

}

return "A";

}

private bool KontrollerGecerliMi()

{

if (cbKAd.Checked && cbParola.Checked)

{

if (txtKAd1.Text == txtKAd2.Text)

{

if (txtSifre1.Text == txtSifre2.Text)

{

if (txtGuvenlik.Text == GuvenlikKod.Text)

{

string statu = Statu();

bool onay = \_blUserLogin.KullaniciAdSifreDegistir1(Tc\_no, txtKAd1.Text, txtSifre1.Text, statu);

return onay;

}

else

{

MessageBox.Show("Güvenlik kodunu yanlış girdiniz! \nLütfen Güvenlik kodunu kontrol ediniz.");

txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

else

{

MessageBox.Show("Şifreler Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtSifre1.Text = null; txtSifre2.Text = null;

txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

else

{

MessageBox.Show("Kullanıcı Adları Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtKAd1.Text = null; txtKAd2.Text = null; txtSifre1.Text = null;

txtSifre2.Text = null; txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

if (!cbKAd.Checked && cbParola.Checked)

{

if (txtSifre1.Text == txtSifre2.Text)

{

if (txtGuvenlik.Text == GuvenlikKod.Text)

{

string statu = Statu();

bool onay = \_blUserLogin.SifreDegistir1(Tc\_no, txtSifre1.Text, statu);

return onay;

}

else

{

MessageBox.Show("Güvenlik kodunu yanlış girdiniz! \nLütfen Güvenlik kodunu kontrol ediniz.");

txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

else

{

MessageBox.Show("Şifreler Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtSifre1.Text = null; txtSifre2.Text = null;

txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

if (cbKAd.Checked && !cbParola.Checked)

{

if (txtKAd1.Text == txtKAd2.Text)

{

if (txtGuvenlik.Text == GuvenlikKod.Text)

{

string statu = Statu();

bool onay = \_blUserLogin.KullaniciAdDegistir1(Tc\_no, txtKAd1.Text, statu);

return onay;

}

else

{

MessageBox.Show("Güvenlik kodunu yanlış girdiniz! \nLütfen Güvenlik kodunu kontrol ediniz.");

txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

else

{

MessageBox.Show("Kullanıcı Adları Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtKAd1.Text = null; txtKAd2.Text = null; txtSifre1.Text = null;

txtSifre2.Text = null; txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

if (!cbKAd.Checked && !cbParola.Checked)

{

return false;

}

else { return false; }

}Formun Üstü

Bu kod bloğu, kullanıcının değiştirmek istediği kullanıcı adı ve şifre bilgilerini güncellemek için bir dizi kontrolü gerçekleştiren bir dizi fonksiyon içermektedir. İşte bu fonksiyonların metinsel açıklamaları:

* **GuvenlikKodYenile Fonksiyonu:**
  + Bu fonksiyon, güvenlik kodunu rastgele bir değerle doldurarak ve kullanıcı arayüzünde gösterir.
* **Statu Fonksiyonu:**
  + Bu fonksiyon, RadioButton'ların durumlarına göre bir kullanıcı statüsü belirler. Eğer radioButton1 seçiliyse "A", radioButton2 seçiliyse "D", herhangi biri seçili değilse varsayılan olarak "A" döndürür.
* **KontrollerGecerliMi Fonksiyonu:**
  + Bu fonksiyon, CheckBox'ların durumlarına ve kullanıcının girmiş olduğu değerlere göre değişik işlemler gerçekleştirir.
  + Eğer hem **cbKAd** hem de **cbParola** seçiliyse:
    - Kullanıcı adı ve şifre kontrol edilir. Eğer her ikisi de geçerliyse, değişiklik yapılır.
  + Eğer sadece **cbParola** seçiliyse:
    - Sadece şifre kontrol edilir. Eğer geçerliyse, şifre değişikliği yapılır.
  + Eğer sadece **cbKAd** seçiliyse:
    - Sadece kullanıcı adı kontrol edilir. Eğer geçerliyse, kullanıcı adı değişikliği yapılır.
  + Eğer hiçbiri seçili değilse, herhangi bir değişiklik yapılmaz.
  + Kullanıcıya uygun mesajlar gösterilerek gerekli kontroller yapılır ve işlemin onaylanıp onaylanmadığına dair bir değer döndürülür.

private void btnGuncelle\_Click(object sender, EventArgs e)

{

if (KontrollerGecerliMi())

{

//Veri tabanına veri yollanacak ve update edilecek.

MessageBox.Show("Başarılı işlem.");

this.Size = new System.Drawing.Size(310, 105);

GuvenlikKodYenile();

gbKAd.Visible = false; txtKAd1.Clear(); txtKAd2.Clear();

gbParola.Visible = false; txtSifre1.Clear(); txtSifre2.Clear();

GuvenlikKod.Visible = false;

txtGuvenlik.Visible = false; txtGuvenlik.Clear();

lb5.Visible = false;

btnGuncelle.Visible = false;

cbKAd.Checked = false;

cbParola.Checked = false;

}

}

Bu kod bloğu, kullanıcının "Güncelle" butonuna tıklaması durumunda gerçekleşen işlemleri içerir. İşte bu kodun metinsel açıklamaları:

* **btnGuncelle\_Click Fonksiyonu:**
  + Bu fonksiyon, güncelleme butonuna tıklandığında çalışır.
  + **KontrollerGecerliMi** fonksiyonu ile kullanıcının girdiği bilgilerin geçerli olup olmadığı kontrol edilir.
  + Eğer kontroller geçerliyse, başarılı bir işlem mesajı gösterilir ve aşağıdaki adımlar gerçekleşir:
    - Formun boyutu küçültülür.
    - Güvenlik kodu yenilenir.
    - Kullanıcı adı ve şifre giriş alanları temizlenir.
    - İlgili kontroller (GroupBox, TextBox'lar, vb.) gizlenir.
    - CheckBox'lar seçili değil hale getirilir.

using BusinessLayer;

using System;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Admin.MenuLayer.EmployeeEditLayer.MenuLayer

{

public partial class EmployeeEditSearch : Form

{

BlEmployee \_blEmployee;

public EmployeeEditSearch()

{

InitializeComponent();

\_blEmployee = new BlEmployee();

txt1.KeyPress += new KeyPressEventHandler(txt1\_KeyPress);

txt2.KeyPress += new KeyPressEventHandler(txt2\_KeyPress);

this.AcceptButton = btnSearch;

}

private void rbTc\_CheckedChanged(object sender, EventArgs e)

{

if (rbTc.Checked)

{

txt1.Text = null;

txt2.Text = null;

lb1.Visible = true;

lb2.Visible = false;

txt1.Visible = true;

txt2.Visible = false;

lb1.Text = "Tc No";

lb2.Text = null;

txt1.MaxLength = 11;

}

}

private void rbAdSoyad\_CheckedChanged(object sender, EventArgs e)

{

if (rbAdSoyad.Checked)

{

txt1.Text = null;

txt2.Text = null;

lb1.Visible = true;

lb2.Visible = true;

txt1.Visible = true;

txt2.Visible = true;

lb1.Text = "Ad";

lb2.Text = "Soyad";

txt1.MaxLength = 15;

txt2.MaxLength = 15;

}

}

private void rbTel\_CheckedChanged(object sender, EventArgs e)

{

if (rbTel.Checked)

{

txt1.Text = null;

txt2.Text = null;

lb1.Visible = true;

lb2.Visible = false;

txt1.Visible = true;

txt2.Visible = false;

lb1.Text = "Tel No";

txt1.MaxLength = 15;

}

}

private void rbMail\_CheckedChanged(object sender, EventArgs e)

{

if (rbMail.Checked)

{

txt1.Text = null;

txt2.Text = null;

lb1.Visible = true;

lb2.Visible = false;

txt1.Visible = true;

txt2.Visible = false;

lb1.Text = "Mail";

txt1.MaxLength = 28;

}

}

Bu kod bloğu, "EmployeeEditSearch" adlı bir form içerisinde kullanılan metinsel arama işlemlerini gerçekleştirmek için tasarlanmıştır. İşte bu kodun metinsel açıklamaları:

* **EmployeeEditSearch Sınıfı:**
  + Bu sınıf, çalışan arama işlemleri için bir formu temsil eder.
  + **BlEmployee** tipinde bir **\_blEmployee** nesnesi oluşturularak iş mantığı işlemleri bu sınıf üzerinden gerçekleştirilir.
  + **txt1** ve **txt2** isimli TextBox kontrol elemanları, sayısal ve metinsel giriş sınırlamalarına uygun hale getirilmiştir. Ayrıca, Enter tuşuna basıldığında arama işlemini gerçekleştirecek şekilde ayarlanmıştır.
* **rbTc\_CheckedChanged Metodu:**
  + T.C. Kimlik Numarasına göre arama seçildiğinde çalışan bu metod içerisindeki kontrolleri günceller.
  + İlgili TextBox'lar ve Label'lar görünür veya görünmez hale getirilir, ilgili metinler ve uzunluk sınırlamaları ayarlanır.
* **rbAdSoyad\_CheckedChanged Metodu:**
  + Ad ve soyada göre arama seçildiğinde çalışan bu metod içerisindeki kontrolleri günceller.
  + İlgili TextBox'lar ve Label'lar görünür veya görünmez hale getirilir, ilgili metinler ve uzunluk sınırlamaları ayarlanır.
* **rbTel\_CheckedChanged Metodu:**
  + Telefon numarasına göre arama seçildiğinde çalışan bu metod içerisindeki kontrolleri günceller.
  + İlgili TextBox'lar ve Label'lar görünür veya görünmez hale getirilir, ilgili metinler ve uzunluk sınırlamaları ayarlanır.
* **rbMail\_CheckedChanged Metodu:**
  + E-posta adresine göre arama seçildiğinde çalışan bu metod içerisindeki kontrolleri günceller.
  + İlgili TextBox'lar ve Label'lar görünür veya görünmez hale getirilir, ilgili metinler ve uzunluk sınırlamaları ayarlanır.

private void txt1\_KeyPress(object sender, KeyPressEventArgs e)

{

if (rbTc.Checked || rbTel.Checked)

{

if (!char.IsDigit(e.KeyChar) && e.KeyChar != (char)Keys.Back)

{

e.Handled = true;

}

}

if (rbAdSoyad.Checked)

{

if (!char.IsLetter(e.KeyChar) && e.KeyChar != (char)Keys.Back && e.KeyChar != (char)Keys.Space)

{

e.Handled = true;

}

}

}

private void txt2\_KeyPress(object sender, KeyPressEventArgs e)

{

if (rbAdSoyad.Checked)

{

if (!char.IsLetter(e.KeyChar) && e.KeyChar != (char)Keys.Back && e.KeyChar != (char)Keys.Space)

{

e.Handled = true;

}

}

}

private void imlec(int position)

{

txt1.SelectionStart = position;

txt1.Focus();

}

private int eskiuzunluk = 0;

private int KontrolEt()

{

string email = txt1.Text;

if (IsValidEmail(email))

{

txt1.ForeColor = Color.Black;

return 1;

}

else

{

txt1.ForeColor = Color.Red;

return 0;

}

}

**txt1\_KeyPress Metodu:**

* Bu metod, **txt1** isimli TextBox üzerine basılan tuşları dinler.
* Eğer "T.C. Kimlik Numarası" veya "Telefon Numarası" seçiliyse, sadece sayısal girişe izin verir. Geri silme tuşuna basıldığında veya sayısal olmayan bir karakter girildiğinde işlemi engeller.
* Eğer "Ad ve Soyad" seçiliyse, sadece harf girişine izin verir. Geri silme tuşuna basıldığında veya harf olmayan bir karakter girildiğinde işlemi engeller.

**txt2\_KeyPress Metodu:**

* Bu metod, **txt2** isimli TextBox üzerine basılan tuşları dinler.
* Eğer "Ad ve Soyad" seçiliyse, sadece harf girişine izin verir. Geri silme tuşuna basıldığında veya harf olmayan bir karakter girildiğinde işlemi engeller.

**imlec Metodu:**

* Bu metod, **txt1** TextBox'ındaki imleci belirtilen konuma taşır.

**eskiuzunluk Değişkeni:**

* Bu değişken, **txt1\_TextChanged** metodu içindeki kontrol amaçları için kullanılır.

**KontrolEt Metodu:**

* Bu metod, **txt1** TextBox'ındaki metni kontrol eder ve geçerli bir e-posta adresi olup olmadığını kontrol eder.
* Eğer geçerli bir e-posta adresi ise metni siyah renkte gösterir ve 1 değerini döndürür.
* Geçerli bir e-posta adresi değilse metni kırmızı renkte gösterir ve 0 değerini döndürür.

private bool IsValidEmail(string email)

{

string emailPattern = @"^[a-zA-Z0-9.\_-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$";

return System.Text.RegularExpressions.Regex.IsMatch(email, emailPattern);

}

private void txt1\_TextChanged(object sender, EventArgs e)

{

if (rbTel.Checked)

{

string text = txt1.Text.Replace("(", "").Replace(")", "").Replace(" ", "");

StringBuilder formattedNumber = new StringBuilder();

int yeniuzunluk = text.Length;

if (yeniuzunluk > eskiuzunluk)

{

if (text.Length > 0)

{

formattedNumber.Append("(" + text.Substring(0, Math.Min(3, text.Length)) + ")");

yeniuzunluk = text.Length;

}

if (text.Length > 3)

{

formattedNumber.Append(" " + text.Substring(3, Math.Min(3, text.Length - 3)));

yeniuzunluk = text.Length;

}

if (text.Length > 6)

{

formattedNumber.Append(" " + text.Substring(6, Math.Min(2, text.Length - 6)));

yeniuzunluk = text.Length;

}

if (text.Length > 8)

{

formattedNumber.Append(" " + text.Substring(8, Math.Max(0, text.Length - 8)));

yeniuzunluk = text.Length;

}

txt1.Text = formattedNumber.ToString();

txt1.SelectionStart = txt1.Text.Length;

}

if (yeniuzunluk < eskiuzunluk)

{

if (text.Length == 3)

{

imlec(4);

}

if (text.Length == 0)

{

txt1.Text = string.Empty;

}

}

eskiuzunluk = yeniuzunluk;

}

if (rbMail.Checked)

{

KontrolEt();

}

}

**IsValidEmail Metodu:**

* Bu metod, bir e-posta adresinin geçerli olup olmadığını kontrol eder.
* **emailPattern** değişkeni, bir e-posta adresinin geçerli olabilmesi için kabul edilebilir bir deseni temsil eder.
* **Regex.IsMatch** metodu, verilen e-posta adresinin belirtilen desene uyup uymadığını kontrol eder ve sonucu boolean olarak döndürür.

**txt1\_TextChanged Metodu:**

* Bu metod, **txt1** TextBox'ındaki metni izler ve değişiklikleri kontrol eder.
* Eğer "Telefon Numarası" seçiliyse:
  + Girilen metni telefon numarası formatına dönüştürür (örneğin, (123) 456 78 90).
  + Geri silme tuşuna basıldığında uygun konuma imleci taşır.
* Eğer "E-posta" seçiliyse:
  + **KontrolEt** metodu ile e-posta adresinin geçerliliğini kontrol eder.
* **eskiuzunluk** değişkeni, önceki metin uzunluğunu saklar.

**Not:** Kodlar arasında bir mantık hatası veya eksik bir parça bulunmadığından emin olunuz. Eğer belirli bir kısımda daha fazla açıklama isterseniz, lütfen belirtiniz.

private void btnSearch\_Click(object sender, EventArgs e)

{

if (rbTc.Checked)

{

EmployeeEditMenu.EmployeeEditMenu empMenu = Application.OpenForms["EmployeeEditMenu"] as EmployeeEditMenu.EmployeeEditMenu;

empMenu.TcArama(txt1.Text);

this.Close();

}

if (rbAdSoyad.Checked)

{

EmployeeEditMenu.EmployeeEditMenu empMenu = Application.OpenForms["EmployeeEditMenu"] as EmployeeEditMenu.EmployeeEditMenu;

empMenu.AdSoyadArama(txt1.Text, txt2.Text);

this.Close();

}

if (rbTel.Checked)

{

EmployeeEditMenu.EmployeeEditMenu empMenu = Application.OpenForms["EmployeeEditMenu"] as EmployeeEditMenu.EmployeeEditMenu;

empMenu.TelArama(txt1.Text);

this.Close();

}

if (rbMail.Checked)

{

EmployeeEditMenu.EmployeeEditMenu empMenu = Application.OpenForms["EmployeeEditMenu"] as EmployeeEditMenu.EmployeeEditMenu;

empMenu.MailArama(txt1.Text);

this.Close();

}

}

**btnSearch\_Click Metodu:**

* Bu metod, "Ara" butonuna tıklandığında gerçekleşen olayları içerir.
* Her bir durumda, seçili olan arama tipine göre uygun işlemi gerçekleştirir:
  1. Eğer "Tc No" seçiliyse:
     + **EmployeeEditMenu** formunu açar ve **TcArama** metodunu çağırarak Tc No'ya göre arama yapılmasını sağlar.
     + Formu kapatır.
  2. Eğer "Ad Soyad" seçiliyse:
     + **EmployeeEditMenu** formunu açar ve **AdSoyadArama** metodunu çağırarak Ad ve Soyad'a göre arama yapılmasını sağlar.
     + Formu kapatır.
  3. Eğer "Telefon Numarası" seçiliyse:
     + **EmployeeEditMenu** formunu açar ve **TelArama** metodunu çağırarak Telefon Numarası'na göre arama yapılmasını sağlar.
     + Formu kapatır.
  4. Eğer "E-posta" seçiliyse:
     + **EmployeeEditMenu** formunu açar ve **MailArama** metodunu çağırarak E-posta'ya göre arama yapılmasını sağlar.
     + Formu kapatır.

using BusinessLayer;

using System;

using System.Collections.Generic;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using ZedGraph;

namespace DernekUyeTakipSistemi.Admin.MenuLayer.GraphLayer.GraphMenu

{

public partial class GraphMenu : Form

{

BlGraph \_blGraph;

string secilenYil;

string secilenAy;

int secilenIl;

int secilenIlce;

DataTable \_dataTable;

private bool uygulamaKapatildi = false;

public GraphMenu()

{

InitializeComponent();

\_blGraph = new BlGraph();

\_dataTable = new DataTable();

cb1.Visible = false;

cb2.Visible = false;

lb1.Visible = false;

lb2.Visible = false;

cbSec.SelectedIndex = 0;

}

private void ClearGraph()

{

GraphPane myPane = zedGraphControl1.GraphPane;

// Grafik üzerindeki çizgileri ve çubukları temizle

myPane.CurveList.Clear();

myPane.GraphObjList.Clear();

// Eksenleri sıfırla

myPane.XAxis.Title.Text = string.Empty;

myPane.YAxis.Title.Text = string.Empty;

myPane.Title.Text = string.Empty;

// Eksen etiketlerini temizle

myPane.XAxis.Scale.TextLabels = new string[] { };

myPane.XAxis.Type = AxisType.Text;

// Yeniden çizim ve geçersiz kılma

zedGraphControl1.AxisChange();

zedGraphControl1.Invalidate();

}

Formun Üstü

**GraphMenu Sınıfı ve ClearGraph Metodu:**

* **GraphMenu** sınıfı, grafiklerin çizildiği bir pencereyi temsil eder.
* **\_blGraph** değişkeni, iş katmanındaki **BlGraph** sınıfının bir örneğini oluşturur.
* **secilenYil**, **secilenAy**, **secilenIl**, ve **secilenIlce** değişkenleri, kullanıcının seçim yaptığı yılı, ayı, il ve ilçeyi saklar.
* **\_dataTable** değişkeni, grafikte kullanılacak olan verileri saklamak için bir **DataTable** oluşturur.
* **cb1**, **cb2**, **lb1**, ve **lb2** kontrol elemanları, kullanıcının seçtiği kriterlere göre görünürlük ayarı yapmak için kullanılır.
* **cbSec** ComboBox'ı, grafik tipi seçimini sağlar ve başlangıçta 0. indekste bulunan değeri seçili kılar.
* **ClearGraph** metodunun amacı, grafik üzerindeki tüm çizgileri temizlemek ve eksenleri sıfırlamaktır.

**ClearGraph Metodu:**

* + **zedGraphControl1** adlı ZedGraph kontrolündeki grafik öğelerini temizler.
  + Grafik üzerindeki çizgileri ve çubukları siler.
  + Eksenleri sıfırlar.
  + Eksen etiketlerini temizler.
  + Yeniden çizim ve geçersiz kılma işlemlerini gerçekleştirir.

private void cbSec\_SelectedIndexChanged(object sender, EventArgs e)

{

ClearGraph();

if (cbSec.SelectedIndex == 1)

{

// Yıla göre gelir dağılımı

cb1.Visible = true;

cb2.Visible = false;

lb1.Visible = true;

lb2.Visible = false;

lb1.Text = "Yil";

YillariComboBoxaEkle();

}

if (cbSec.SelectedIndex == 2)

{

// Aya göre gelir dağılımı

cb1.Visible = false;

cb2.Visible = true;

lb1.Visible = false;

lb2.Visible = true;

lb2.Text = "Ay";

AylariComboBoxaEkle();

}

if (cbSec.SelectedIndex == 3)

{

cb1.Visible = true;

cb2.Visible = true;

lb1.Visible = true;

lb2.Visible = true;

lb1.Text = "Yil";

lb2.Text = "Ay";

YillariComboBoxaEkle();

AylariComboBoxaEkle();

}

if (cbSec.SelectedIndex == 4)

{

cb1.Visible = true;

cb2.Visible = false;

lb1.Visible = true;

lb2.Visible = false;

lb1.Text = "İl";

IlleriComboBoxaEkle();

}

if (cbSec.SelectedIndex == 5)

{

cb1.Visible = true;

cb2.Visible = true;

lb1.Visible = true;

lb2.Visible = true;

lb1.Text = "İl";

lb2.Text = "İlçe";

IlleriComboBoxaEkle();

}

}

private void YillariComboBoxaEkle()

{

// Combobox'ı temizle

cb1.DataSource = null;

cb1.Items.Clear();

// Yılları ekleyin

for (int yil = 2005; yil <= 2030; yil++)

{cb1.Items.Add(yil.ToString()); }}

**cbSec\_SelectedIndexChanged ve YillariComboBoxaEkle Metodları:**

* **cbSec\_SelectedIndexChanged** metodu, ComboBox'ın seçili öğesinin değiştiğinde tetiklenir. Bu metod, seçilen grafik türüne bağlı olarak diğer kontrol elemanlarının görünürlüğünü ve içeriğini ayarlar.
* **YillariComboBoxaEkle** metodu, yılları ComboBox'a ekler. Bu metot, ComboBox'ın veri kaynağını temizler, ardından belirli bir aralıktaki yılları ekler.

**cbSec\_SelectedIndexChanged Metodu:**

* + **ClearGraph** metodunu çağırarak grafik penceresini temizler.
  + **cbSec** ComboBox'ındaki seçili öğeye göre farklı durumları kontrol eder ve ilgili kontrol elemanlarının görünürlüğünü ayarlar.
  + Eğer seçili grafik türü "Yıla göre gelir dağılımı" ise, yıl ComboBox'ını görünür kılar ve yılları ekler.
  + Eğer seçili grafik türü "Aya göre gelir dağılımı" ise, ay ComboBox'ını görünür kılar ve ayları ekler.
  + Eğer seçili grafik türü "Yıla ve Aya göre gelir dağılımı" ise, hem yıl hem de ay ComboBox'larını görünür kılar ve yılları, ayları ekler.
  + Eğer seçili grafik türü "İle göre gelir dağılımı" ise, il ComboBox'ını görünür kılar ve illeri ekler.
  + Eğer seçili grafik türü "İl ve İlçe göre gelir dağılımı" ise, hem il hem de ilçe ComboBox'larını görünür kılar ve illeri, ilçeleri ekler.

**YillariComboBoxaEkle Metodu:**

* + ComboBox'ın veri kaynağını temizler ve öğeleri temizler.
  + For döngüsü ile belirli bir yıl aralığındaki yılları ComboBox'a ekler.

Bu kod bloğu, ComboBox'ların içeriğini ve görünürlüğünü değiştirerek kullanıcının belirli grafik türlerini seçmesini sağlar.

private void AylariComboBoxaEkle()

{

// Combobox'ı temizle

cb2.DataSource = null;

cb2.Items.Clear();

// Ayları ekleyin

string[] aylar = new string[]

{

"Ocak", "Şubat", "Mart", "Nisan", "Mayıs", "Haziran",

"Temmuz", "Ağustos", "Eylül", "Ekim", "Kasım", "Aralık"

};

cb2.Items.AddRange(aylar);

}

private void IlleriComboBoxaEkle()

{

cb1.DataSource = null;

cb1.DisplayMember = null;

cb1.ValueMember = null;

cb1.SelectedIndex = -1;

cb1.DataSource = \_blGraph.IlSorgu();

cb1.DisplayMember = "IlAdi";

cb1.ValueMember = "IlID";

cb1.SelectedIndex = -1;

}

private void IlceleriComboBoxaEkle(int IlID)

{

cb2.DataSource = null;

cb2.DisplayMember = null;

cb2.ValueMember = null;

cb2.SelectedIndex = -1;

cb2.DataSource = \_blGraph.IlceSorgu(IlID);

cb2.DisplayMember = "IlceAdi";

cb2.ValueMember = "IlceID";

cb2.SelectedIndex = -1;

}

private string GetFormattedMonthIndex(int selectedIndex)

{

// Seçilen index değerini bir arttır ve string olarak al

int realIndex = selectedIndex + 1;

string formattedIndex = realIndex.ToString();

// Eğer değer 10'dan küçükse başına 0 ekle

if (realIndex < 10)

{

formattedIndex = "0" + formattedIndex;

}

return formattedIndex;

}

**AylariComboBoxaEkle Metodu:**

* Bu metod, ay isimlerini içeren bir dizi oluşturur ve bu ay isimlerini ComboBox'a ekler.

**IlleriComboBoxaEkle Metodu:**

* Bu metod, **\_blGraph.IlSorgu()** metodunu kullanarak illeri sorgular ve elde edilen veriyi **cb1** (il ComboBox'ı) içine ekler. DisplayMember ve ValueMember özellikleri belirlenerek görünen değer ve değer değeri belirlenir.

**IlceleriComboBoxaEkle Metodu:**

* Bu metod, **\_blGraph.IlceSorgu(IlID)** metodunu kullanarak ilçeleri sorgular ve elde edilen veriyi **cb2** (ilçe ComboBox'ı) içine ekler. DisplayMember ve ValueMember özellikleri belirlenerek görünen değer ve değer değeri belirlenir.

**GetFormattedMonthIndex Metodu:**

* Bu metod, ComboBox'ta seçilen ayın sırasını alır ve bu değeri bir artırarak string olarak döndürür. Eğer değer 10'dan küçükse, başına 0 eklenir.

Bu metodlar, ComboBox'ların içeriğini belirli bir şekilde doldurarak, kullanıcının ilgili seçenekleri seçmesine olanak tanır.

private void cb1\_SelectedIndexChanged(object sender, EventArgs e)

{

if (cbSec.SelectedIndex == 1)

{

secilenYil = cb1.SelectedItem?.ToString();

if (!string.IsNullOrEmpty(secilenYil))

{

// Tüm kullanıcıların aynı yıla ait borç bilgilerini çek

\_dataTable = \_blGraph.JustYil(secilenYil);

YilveyaAy("Yıl'a Göre Borç Bilgisi", "Ay", "Borç ve Aidat Miktarı", "AidatAy");

}

}

if (cbSec.SelectedIndex == 3 && cb2.SelectedIndex != -1)

{

secilenYil = cb1.SelectedItem?.ToString();

secilenAy = cb2.SelectedItem?.ToString();

if (!string.IsNullOrEmpty(secilenYil) && !string.IsNullOrEmpty(secilenAy))

{

string formattedAy = GetFormattedMonthIndex(cb2.SelectedIndex);

\_dataTable = \_blGraph.YilandAy(secilenYil, formattedAy);

YilveAy("Yıl ve Ay'a Göre Borç Bilgisi", "Kullanıcılar", "Borç ve Aidat Miktarı");

}

}

if (cbSec.SelectedIndex == 4)

{

// İl seçili ise

secilenIl = cb1.SelectedIndex;

secilenIl++;

\_dataTable = \_blGraph.IleGoreKSayi(secilenIl);

IleGore("İL'e göre Kullanıcı Dağılımı", "Il", "Kullanıcı Sayısı", "IlAdi");

}

if (cbSec.SelectedIndex == 5)

{

DataRowView selectedRow = cb1.SelectedItem as DataRowView;

if (selectedRow != null)

{

int selectedIlID = Convert.ToInt32(selectedRow["IlID"]);

IlceleriComboBoxaEkle(selectedIlID);

}

}

}

private void cb2\_SelectedIndexChanged(object sender, EventArgs e)

{

if (cbSec.SelectedIndex == 2)

{

secilenAy = cb2.SelectedItem?.ToString();

if (!string.IsNullOrEmpty(secilenAy))

{

// Tüm kullanıcıların aynı yıla ait borç bilgilerini çek

string formattedAy = GetFormattedMonthIndex(cb2.SelectedIndex);

\_dataTable = \_blGraph.JustAy(formattedAy);

YilveyaAy("Ay'a Göre Borç Bilgisi", "Yıl", "Borç ve Aidat Miktarı", "AidatYil");

}

}

if (cbSec.SelectedIndex == 3 && cb1.SelectedIndex != -1)

{

secilenYil = cb1.SelectedItem?.ToString();

secilenAy = cb2.SelectedItem?.ToString();

if (!string.IsNullOrEmpty(secilenYil) && !string.IsNullOrEmpty(secilenAy))

{

string formattedAy = GetFormattedMonthIndex(cb2.SelectedIndex);

\_dataTable = \_blGraph.YilandAy(secilenYil, formattedAy);

YilveAy("Yıl ve Ay'a Göre Borç Bilgisi", "Kullanıcılar", "Borç ve Aidat Miktarı");

}

}

if (cbSec.SelectedIndex == 5 && cb1.SelectedIndex != -1)

{

secilenIlce = cb2.SelectedIndex > 0 ? Convert.ToInt32(cb2.SelectedValue) : -1;

\_dataTable = \_blGraph.IlceyeGoreKSayi(secilenIlce);

IlceyeGore("İL ve İLÇEYE'e göre Kullanıcı Dağılımı", "Ilçe", "Kullanıcı Sayısı", "IlceAdi");

}

}

private void YilveyaAy(string title, string xAxisTitle, string yAxisTitle, string groupByField)

{

GraphPane myPane = zedGraphControl1.GraphPane;

myPane.Title.Text = title;

myPane.XAxis.Title.Text = xAxisTitle;

myPane.YAxis.Title.Text = yAxisTitle;

myPane.CurveList.Clear();

myPane.GraphObjList.Clear();

List<string> categories = new List<string>();

List<double> borcBilgisi = new List<double>();

List<double> aidatBilgisi = new List<double>();

var groupedData = \_dataTable.AsEnumerable()

.GroupBy(row => row.Field<string>(groupByField))

.Select(group => new

{

Category = group.Key,

ToplamBorc = group.Sum(row => row.Field<int>("BorcMiktar")),

ToplamAidat = group.Sum(row => row.Field<int>("AidatTutar"))

})

.OrderBy(group => int.Parse(group.Category));

// Gruplanmış veriyi al ve grafiğe ekle

foreach (var group in groupedData)

{

categories.Add(group.Category);

borcBilgisi.Add(group.ToplamBorc);

aidatBilgisi.Add(group.ToplamAidat);

}

// Çizimi gerçekleştirin

BarItem borcBarItem = myPane.AddBar("Borç Miktarı", null, borcBilgisi.ToArray(), Color.Blue);

borcBarItem.Bar.Fill.Type = FillType.Solid;

BarItem aidatBarItem = myPane.AddBar("Aidat Miktarı", null, aidatBilgisi.ToArray(), Color.Red);

aidatBarItem.Bar.Fill.Type = FillType.Solid;

// X ekseni etiketlerini ayarlama

myPane.XAxis.Scale.TextLabels = categories.ToArray();

myPane.XAxis.Type = AxisType.Text;

// ZedGraphControl'u güncelleyin

zedGraphControl1.AxisChange();

zedGraphControl1.Invalidate();

}

**cb1\_SelectedIndexChanged ve cb2\_SelectedIndexChanged Metotları:**

* Bu metotlar, ComboBox'ların seçilen öğesi değiştikçe hangi işlemlerin yapılacağını kontrol eder.
* Hangi ComboBox'ın seçili olduğuna (**cbSec.SelectedIndex**) göre farklı işlemler gerçekleştirir.

**YilveyaAy Metodu:**

* Bu metod, yıla veya aya göre borç bilgisi grafiği çizimi yapar.
* Grafik üzerindeki çizgileri temizler (**ClearGraph** metodu).
* Veri tablosunu belirli bir alana göre gruplar ve bu gruplardan kategori ve toplam borç/aidat miktarlarını çıkarır.
* ZedGraphControl kullanarak bar grafiği oluşturur.
* Kategori isimleri, borç miktarları ve aidat miktarları listelerini oluşturarak grafiği çizer.
* X ekseni etiketlerini ve diğer grafiği özelliklerini ayarlar.
* Çizimi günceller ve ZedGraphControl'u yeniden çizer.

Bu metot, belirli bir alana göre gruplandırılmış veriler üzerinde çalışarak grafik çizimi gerçekleştirir. Bu sayede seçilen yıl veya ay gibi kriterlere göre borç ve aidat miktarlarını içeren bir çubuk grafiği oluşturulmuş olur.

private void YilveAy(string title, string xAxisTitle, string yAxisTitle)

{

GraphPane myPane = zedGraphControl1.GraphPane;

myPane.Title.Text = title;

myPane.XAxis.Title.Text = xAxisTitle;

myPane.YAxis.Title.Text = yAxisTitle;

myPane.CurveList.Clear();

myPane.GraphObjList.Clear();

List<string> categories = new List<string>();

List<double> borcBilgisi = new List<double>();

List<double> odemeBilgisi = new List<double>();

List<double> aidatBilgisi = new List<double>();

var individualData = \_dataTable.AsEnumerable()

.Select(row => new

{

Category = $"{row.Field<string>("Ad")} {row.Field<string>("Soyad")}",

Borc = row.Field<int>("BorcMiktar"),

Odeme = row.Field<int?>("OdemeMiktar") ?? 0,

Aidat = row.Field<int>("AidatTutar"),

})

.OrderBy(entry => entry.Category);

// Gruplanmış veriyi al ve grafiğe ekle

foreach (var entry in individualData)

{

categories.Add($"{entry.Category}");

borcBilgisi.Add(entry.Borc);

aidatBilgisi.Add(entry.Aidat);

odemeBilgisi.Add(entry.Odeme);

}

double barWidth = 2; // Çubuk genişliğini ayarla

double barSpace = 10; // Çubuklar arasındaki boşluğu ayarla

// Çubukları temsil eden bir PointPairList oluştur

PointPairList borcPoints = new PointPairList();

PointPairList odemePoints = new PointPairList();

PointPairList aidatPoints = new PointPairList();

for (int i = 0; i < categories.Count; i++)

{

double xValue = i + 1;

// Çubukların değerlerini PointPairList'e ekleyin

borcPoints.Add(xValue, borcBilgisi[i]);

odemePoints.Add(xValue + barWidth + barSpace, odemeBilgisi[i]);

aidatPoints.Add(xValue + 2 \* (barWidth + barSpace), aidatBilgisi[i]);

}

// Çubukları grafik pane'ine ekleyin

BarItem borcBarItem = myPane.AddBar("Borç Miktarı", borcPoints, Color.Blue);

borcBarItem.Bar.Fill.Type = FillType.Solid;

BarItem aidatBarItem = myPane.AddBar("Aidat Miktarı", aidatPoints, Color.Red);

aidatBarItem.Bar.Fill.Type = FillType.Solid;

BarItem odemeBarItem = myPane.AddBar("Ödeme Miktarı", odemePoints, Color.Green);

odemeBarItem.Bar.Fill.Type = FillType.Solid;

// X ekseni etiketlerini ayarlama

myPane.XAxis.Scale.TextLabels = categories.ToArray();

myPane.XAxis.Type = AxisType.Text;

// ZedGraphControl'u güncelleyin

zedGraphControl1.AxisChange();

zedGraphControl1.Invalidate();

}

Bu metodun mantığı şu adımları içerir:

1. **Grafik Pane Ayarları:**
   * **GraphPane** öğesini alır ve başlıkları temizler.
   * Grafik üzerindeki çizgileri ve çubukları temizler.
   * Kullanılacak listeleri (kategoriler ve miktarlar) temizler.
2. **Veri Hazırlama:**
   * **individualData** adlı bir sorgu ile her bir kullanıcının adı, soyadı, borç miktarı, ödeme miktarı (varsa), aidat miktarı gibi bilgileri içeren veri kümesini oluşturur.
   * Bu veriyi isme göre sıralar.
3. **Listeleri Doldurma:**
   * **categories**, **borcBilgisi**, **odemeBilgisi** ve **aidatBilgisi** adlı listeleri doldurur.
   * Kullanıcının adı ve soyadını birleştirerek bir kategori oluşturur.
   * Borç, ödeme ve aidat miktarlarını ilgili listelere ekler.
4. **Çubuk Oluşturma:**
   * **barWidth** ve **barSpace** değişkenleri ile çubuk genişliği ve çubuklar arasındaki boşluk ayarlanır.
   * **PointPairList** oluşturarak her bir çubuğun değerini belirler.
   * For döngüsü ile her kullanıcı için bir çubuk oluşturulur ve X ekseni üzerindeki konumu belirlenir.
5. **Çubukları Grafiğe Ekleme:**
   * Grafik pane'ine borç, aidat ve ödeme miktarlarını temsil eden çubukları ekler.
   * Çubukların renklerini ayarlar.
6. **Eksen Ayarları:**
   * X ekseni etiketlerini kategori adları olarak belirler.
   * X ekseni tipini metin olarak ayarlar.
7. **Güncelleme ve Yeniden Çizim:**
   * ZedGraphControl'u günceller ve yeniden çizer.

Bu metodun temel amacı, her bir kullanıcının adı, soyadı, borç miktarı, ödeme miktarı (varsa), aidat miktarı gibi bilgilerini içeren veri setini kullanarak, kullanıcı bazında bir çubuk grafik çizmektir. Bu grafik, her bir kullanıcının borç, ödeme ve aidat miktarlarını yan yana gösterir.

private void IleGore(string title, string xAxisTitle, string yAxisTitle, string groupByField)

{

GraphPane myPane = zedGraphControl1.GraphPane;

myPane.Title.Text = title;

myPane.XAxis.Title.Text = xAxisTitle;

myPane.YAxis.Title.Text = yAxisTitle;

myPane.CurveList.Clear();

myPane.GraphObjList.Clear();

List<string> categories = new List<string>();

List<double> kullaniciSayisi = new List<double>();

// Gruplanmış veriyi al ve grafiğe ekle

foreach (DataRow row in \_dataTable.Rows)

{

string category = row.Field<string>(groupByField);

double userCount = Convert.ToDouble(row["KullaniciSayisi"]);

categories.Add(category);

kullaniciSayisi.Add(userCount);

}

// Çizimi gerçekleştirin

BarItem kullaniciBarItem = myPane.AddBar("Kullanıcı Sayısı", null, kullaniciSayisi.ToArray(), Color.Green);

kullaniciBarItem.Bar.Fill.Type = FillType.Solid;

// X ekseni etiketlerini ayarlama

myPane.XAxis.Scale.TextLabels = categories.ToArray();

myPane.XAxis.Type = AxisType.Text;

// ZedGraphControl'u güncelleyin

zedGraphControl1.AxisChange();

zedGraphControl1.Invalidate();

}

private void IlceyeGore(string title, string xAxisTitle, string yAxisTitle, string groupByField)

{

GraphPane myPane = zedGraphControl1.GraphPane;

myPane.Title.Text = title;

myPane.XAxis.Title.Text = xAxisTitle;

myPane.YAxis.Title.Text = yAxisTitle;

myPane.CurveList.Clear();

myPane.GraphObjList.Clear();

List<string> categories = new List<string>();

List<double> kullaniciSayisi = new List<double>();

// Gruplanmış veriyi al ve grafiğe ekle

foreach (DataRow row in \_dataTable.Rows)

{

string category = row.Field<string>(groupByField);

double userCount = Convert.ToDouble(row["KullaniciSayisi"]);

categories.Add(category);

kullaniciSayisi.Add(userCount);

}

// Çizimi gerçekleştirin

BarItem kullaniciBarItem = myPane.AddBar("Kullanıcı Sayısı", null, kullaniciSayisi.ToArray(), Color.Green);

kullaniciBarItem.Bar.Fill.Type = FillType.Solid;

// X ekseni etiketlerini ayarlama

myPane.XAxis.Scale.TextLabels = categories.ToArray();

myPane.XAxis.Type = AxisType.Text;

// ZedGraphControl'u güncelleyin

zedGraphControl1.AxisChange();

zedGraphControl1.Invalidate();

}

private void btnGeri\_Click(object sender, EventArgs e)

{

uygulamaKapatildi = true;

this.Close();

AMenu aMenu = Application.OpenForms["AMenu"] as AMenu;

aMenu?.Show();

}

private void GraphMenu\_FormClosing(object sender, FormClosingEventArgs e)

{

if (!uygulamaKapatildi)

{

DialogResult cikis = MessageBox.Show("Çıkış yapmak istediğinize emin misiniz ?", "", MessageBoxButtons.YesNo);

if (cikis == DialogResult.Yes)

{

uygulamaKapatildi = true;

Application.Exit();

}

else

{

e.Cancel = true;

}

}

}

1. **IleGore Metodu:**
   * Grafik üzerinde kullanıcı sayısını, belirli bir kategoriye göre gösterir.
   * **groupByField** parametresi, hangi alanın gruplandırma için kullanılacağını belirler.
   * **categories** ve **kullaniciSayisi** adlı listeleri kullanarak her bir kategoriyi ve bu kategoriye ait kullanıcı sayısını alır.
   * Grafik pane'ine çubuk ekleyerek kullanıcı sayısını görselleştirir.
2. **IlceyeGore Metodu:**
   * Grafik üzerinde ilçeye göre kullanıcı sayısını gösterir.
   * **groupByField** parametresi, hangi alanın gruplandırma için kullanılacağını belirler.
   * **categories** ve **kullaniciSayisi** adlı listeleri kullanarak her bir ilçeyi ve bu ilçeye ait kullanıcı sayısını alır.
   * Grafik pane'ine çubuk ekleyerek kullanıcı sayısını görselleştirir.
3. **btnGeri\_Click Metodu:**
   * Formu kapatır ve AMenu formunu açar.
   * Uygulamanın kapatılıp kapatılmadığını kontrol etmek için **uygulamaKapatildi** değişkenini kullanır.
4. **GraphMenu\_FormClosing Metodu:**
   * Form kapatılmak istendiğinde çalışır.
   * **uygulamaKapatildi** değişkeni kontrol edilir.
   * Eğer uygulama kapatılmak istenmiyorsa, bir çıkış onayı ister. Onay gelirse uygulamayı kapatır, gelmezse formun kapatılmasını iptal eder.

Bu kodlar, grafik üzerinde farklı kategorilere göre kullanıcı sayısını gösteren metotları içerir. Ayrıca, formun kapatılması durumunda çıkış onayı istenir ve uygulamanın tamamen kapatılması sağlanır.

Formun Üstü

using BusinessLayer;

using System;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Customer.MenuLayer.CustomerInfoLayer.MenuLayer

{

public partial class CustomerInfoLogin : Form

{

int gvnlkkod;

public string Tc\_no;

BlUserLogin \_blUserLogin;

public CustomerInfoLogin()

{

InitializeComponent();

txtKAd1.MaxLength = 13;

txtKAd2.MaxLength = 13;

txtSifre1.MaxLength = 13;

txtSifre2.MaxLength = 13;

txtGuvenlik.MaxLength = 6;

\_blUserLogin = new BlUserLogin();

this.AcceptButton = btnGuncelle;

}

private void UserName\_Load(object sender, System.EventArgs e)

{

this.Size = new System.Drawing.Size(310, 105);

GuvenlikKodYenile();

mvctKAd.Visible = false;

mvctParola.Visible = false;

gbKAd.Visible = false;

gbParola.Visible = false;

GuvenlikKod.Visible = false;

txtGuvenlik.Visible = false;

lb5.Visible = false;

btnGuncelle.Visible = false;

}

private void txtGuvenlik\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsDigit(e.KeyChar) && e.KeyChar != (char)Keys.Back)

{

e.Handled = true;

}

}

private void GuvenlikKodYenile()

{

Random random = new Random();

gvnlkkod = random.Next(10000, 99999);

GuvenlikKod.Text = gvnlkkod.ToString();

}

public void TcAtama(string tc)

{

Tc\_no = tc;

}

public void MevcutKAd()

{

mvctKAd.Visible = true;

mvctKAd.Text = \_blUserLogin.KAdGetir(Tc\_no);

}

public void MevcutParola()

{

mvctParola.Visible = true;

mvctParola.Text = \_blUserLogin.ParolaGetir(Tc\_no);

}

private void HangisiSecili()

{

if (cbKAd.Checked && !cbParola.Checked)

{

//Sadece kullanıcı adı seçili.

MevcutKAd();

mvctParola.Visible = false;

this.Size = new System.Drawing.Size(310, 300);

gbKAd.Visible = true;

gbParola.Visible = false;

GuvenlikKod.Visible = true;

txtGuvenlik.Visible = true;

lb5.Visible = true;

btnGuncelle.Visible = true;

gbKAd.Location = new System.Drawing.Point(25, 70);

GuvenlikKod.Location = new System.Drawing.Point(159, 154);

lb5.Location = new System.Drawing.Point(65, 183);

txtGuvenlik.Location = new System.Drawing.Point(148, 183);

btnGuncelle.Location = new System.Drawing.Point(207, 209);

}

if (!cbKAd.Checked && cbParola.Checked)

{

//Sadece parola seçili.

MevcutParola();

mvctKAd.Visible = false;

this.Size = new System.Drawing.Size(310, 300);

gbParola.Visible = true;

gbKAd.Visible = false;

GuvenlikKod.Visible = true;

txtGuvenlik.Visible = true;

lb5.Visible = true;

btnGuncelle.Visible = true;

gbParola.Location = new System.Drawing.Point(25, 70);

GuvenlikKod.Location = new System.Drawing.Point(159, 154);

lb5.Location = new System.Drawing.Point(65, 183);

txtGuvenlik.Location = new System.Drawing.Point(148, 183);

btnGuncelle.Location = new System.Drawing.Point(207, 209);

}

if (cbKAd.Checked && cbParola.Checked)

{

//İkiside seçiliyse.

MevcutKAd();

MevcutParola();

this.Size = new System.Drawing.Size(310, 385);

gbKAd.Visible = true;

gbParola.Visible = true;

GuvenlikKod.Visible = true;

txtGuvenlik.Visible = true;

lb5.Visible = true;

btnGuncelle.Visible = true;

gbKAd.Location = new System.Drawing.Point(25, 70);

gbParola.Location = new System.Drawing.Point(25, 150);

GuvenlikKod.Location = new System.Drawing.Point(159, 231);

lb5.Location = new System.Drawing.Point(65, 263);

txtGuvenlik.Location = new System.Drawing.Point(148, 263);

btnGuncelle.Location = new System.Drawing.Point(208, 295);

}

if (!cbKAd.Checked && !cbParola.Checked)

{

mvctKAd.Visible = false;

mvctParola.Visible = false;

this.Size = new System.Drawing.Size(310, 100);

gbKAd.Visible = false;

gbParola.Visible = false;

GuvenlikKod.Visible = false;

txtGuvenlik.Visible = false;

lb5.Visible = false;

btnGuncelle.Visible = false;

}

}  
 private void cbKAd\_CheckedChanged(object sender, EventArgs e)

{

HangisiSecili();

}

private void cbParola\_CheckedChanged(object sender, EventArgs e)

{

HangisiSecili();

}

private bool KontrollerGecerliMi()

{

if (cbKAd.Checked && cbParola.Checked)

{

if (txtKAd1.Text == txtKAd2.Text)

{

if (txtSifre1.Text == txtSifre2.Text)

{

if (txtGuvenlik.Text == GuvenlikKod.Text)

{

bool onay = \_blUserLogin.KullaniciAdSifreDegistir2(Tc\_no, txtKAd1.Text, txtSifre1.Text);

return onay;

}

else

{

MessageBox.Show("Güvenlik kodunu yanlış girdiniz! \nLütfen Güvenlik kodunu kontrol ediniz.");

txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

else if (txtSifre1.Text != txtSifre2.Text && txtGuvenlik.Text != GuvenlikKod.Text)

{

MessageBox.Show("Güvenlik Kodu YANLIŞ! \nŞifreler Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtKAd1.Text = null; txtKAd2.Text = null; txtSifre1.Text = null;

txtSifre2.Text = null; txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

else

{

MessageBox.Show("Şifreler Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtSifre1.Text = null; txtSifre2.Text = null;

txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

else if (txtKAd1.Text != txtKAd2.Text && txtGuvenlik.Text != GuvenlikKod.Text)

{

MessageBox.Show("Güvenlik Kodu YANLIŞ! \nKullanıcı Adları Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtKAd1.Text = null; txtKAd2.Text = null; txtSifre1.Text = null;

txtSifre2.Text = null; txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

else

{

MessageBox.Show("Kullanıcı Adları Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtKAd1.Text = null; txtKAd2.Text = null; txtSifre1.Text = null;

txtSifre2.Text = null; txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

if (!cbKAd.Checked && cbParola.Checked)

{

if (txtSifre1.Text == txtSifre2.Text)

{

if (txtGuvenlik.Text == GuvenlikKod.Text)

{

bool onay = \_blUserLogin.SifreDegistir2(Tc\_no, txtSifre1.Text);

return onay;

}

else

{

MessageBox.Show("Güvenlik kodunu yanlış girdiniz! \nLütfen Güvenlik kodunu kontrol ediniz.");

txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

else if (txtSifre1.Text != txtSifre2.Text && txtGuvenlik.Text != GuvenlikKod.Text)

{

MessageBox.Show("Güvenlik Kodu YANLIŞ! \nŞifreler Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtSifre1.Text = null; txtSifre2.Text = null; txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

else

{

MessageBox.Show("Şifreler Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtSifre1.Text = null; txtSifre2.Text = null; txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

if (cbKAd.Checked && !cbParola.Checked)

{

if (txtKAd1.Text == txtKAd2.Text)

{

if (txtGuvenlik.Text == GuvenlikKod.Text)

{

bool onay = \_blUserLogin.KullaniciAdDegistir2(Tc\_no, txtKAd1.Text);

return onay;

}

else

{

MessageBox.Show("Güvenlik kodunu yanlış girdiniz! \nLütfen Güvenlik kodunu kontrol ediniz.");

txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

else if (txtKAd1.Text != txtKAd2.Text && txtGuvenlik.Text != GuvenlikKod.Text)

{

MessageBox.Show("Güvenlik Kodu YANLIŞ! \nKullanıcı Adları Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtKAd1.Text = null; txtKAd2.Text = null; txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

else

{

MessageBox.Show("Kullanıcı Adları Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtKAd1.Text = null; txtKAd2.Text = null; txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

if (!cbKAd.Checked && !cbParola.Checked)

{

return false;

}

else { return false; }

}

private void btnGuncelle\_Click(object sender, EventArgs e)

{

if (KontrollerGecerliMi())

{

//Veri tabanına veri yollanacak ve update edilecek.

MessageBox.Show("Başarılı işlem.");

this.Size = new System.Drawing.Size(310, 105);

GuvenlikKodYenile();

gbKAd.Visible = false; txtKAd1.Clear(); txtKAd2.Clear();

gbParola.Visible = false; txtSifre1.Clear(); txtSifre2.Clear();

GuvenlikKod.Visible = false;

txtGuvenlik.Visible = false; txtGuvenlik.Clear();

lb5.Visible = false;

btnGuncelle.Visible = false;

cbKAd.Checked = false;

cbParola.Checked = false;

}

}

}

}

Bu kod, bir kullanıcının kullanıcı adını ve/veya şifresini güncelleyebileceği bir Windows Forms uygulaması içerir. İşlevselliği şu adımlarla açıklayabiliriz:

1. **cbKAd\_CheckedChanged ve cbParola\_CheckedChanged Metotları:**
   * Kullanıcı adı veya şifre değiştirme seçenekleri işaretlendiğinde çağrılır.
   * **HangisiSecili** metodu çağrılarak, kullanıcının hangi bilgileri güncellemek istediğine karar verilir.
2. **KontrollerGecerliMi Metodu:**
   * Kullanıcının girdiği bilgilerin geçerliliğini kontrol eder.
   * İki durumu ele alır: Kullanıcı adı değiştirme ve/veya şifre değiştirme.
   * Kullanıcı adı ve şifre değiştirme durumlarında, gerekli kontroller yapılır.
   * Güvenlik kodu doğruluğu kontrol edilir.
   * Tüm kontroller geçerliyse, **\_blUserLogin** sınıfının uygun metodları çağrılarak kullanıcının bilgileri güncellenir.
3. **btnGuncelle\_Click Metodu:**
   * Kullanıcının güncelleme butonuna tıklaması durumunda çağrılır.
   * **KontrollerGecerliMi** metodu ile giriş kontrolleri yapılır.
   * Kontroller geçerliyse, bilgiler veritabanında güncellenir ve kullanıcıya başarılı işlem mesajı gösterilir. Formun görünürlüğü ve boyutu eski haline getirilir.

Bu kodun temel mantığı, kullanıcının istediği bilgileri seçmesine ve bu bilgileri güncellemesine izin vermek, ayrıca giriş kontrolleri yaparak güvenliği sağlamaktır.

using BusinessLayer;

using System;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Customer.MenuLayer.CustomerInfoLayer.MenuLayer

{

public partial class CustomerInfoPersonal : Form

{

public string Tc\_no;

BlUser \_blUser;

BlUserLogin \_blUserLogin;

DataTable dt;

public CustomerInfoPersonal()

{

InitializeComponent();

\_blUser = new BlUser();

\_blUserLogin = new BlUserLogin();

dt = new DataTable();

txtTc.MaxLength = 11;

txtAd.MaxLength = 15;

txtSoyad.MaxLength = 15;

txtTel.MaxLength = 15;

txtMail.MaxLength = 60;

txtAdres.MaxLength = 60;

FillIl();

}

public void TcGetir(string tc)

{

Tc\_no = tc;

FormGuncelle();

}

private void ClearForm()

{

// Formdaki tüm kontrol değerlerini temizleme işlemi

foreach (Control control in Controls)

{

if (control is System.Windows.Forms.TextBox textBox)

{

textBox.Clear();

}

else if (control is System.Windows.Forms.ComboBox comboBox)

{

comboBox.SelectedIndex = -1;

}

}

}

private void FormGuncelle()

{

dt = \_blUserLogin.TcArama(Tc\_no);

if (dt.Rows.Count > 0)

{

txtTc.Text = dt.Rows[0]["Tc\_No"].ToString();

txtAd.Text = dt.Rows[0]["Ad"].ToString();

txtSoyad.Text = dt.Rows[0]["Soyad"].ToString();

cmCins.Text = dt.Rows[0]["Cinsiyet"].ToString();

cmKan.Text = dt.Rows[0]["Kan\_Grup"].ToString();

txtTel.Text = dt.Rows[0]["Telefon"].ToString();

txtMail.Text = dt.Rows[0]["Mail"].ToString();

txtAdres.Text = dt.Rows[0]["Adres"].ToString();

cmil.SelectedValue = dt.Rows[0]["IlID"].ToString();

cmilce.SelectedValue = dt.Rows[0]["IlceID"].ToString();

}

else

{

ClearForm();

}

}

private void txtAd\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsLetter(e.KeyChar) && e.KeyChar != (char)Keys.Back && e.KeyChar != (char)Keys.Space)

{

e.Handled = true;

}

}

private void txtSoyad\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsLetter(e.KeyChar) && e.KeyChar != (char)Keys.Back && e.KeyChar != (char)Keys.Space)

{

e.Handled = true;

}

}

private void imlec(int position)

{

txtTel.SelectionStart = position;

txtTel.Focus();

}

private int eskiuzunluk = 0;

private void txtTel\_TextChanged(object sender, EventArgs e)

{

string text = txtTel.Text.Replace("(", "").Replace(")", "").Replace(" ", "");

StringBuilder formattedNumber = new StringBuilder();

int yeniuzunluk = text.Length;

if (yeniuzunluk > eskiuzunluk)

{

if (text.Length > 0)

{

formattedNumber.Append("(" + text.Substring(0, Math.Min(3, text.Length)) + ")");

yeniuzunluk = text.Length;

}

if (text.Length > 3)

{

formattedNumber.Append(" " + text.Substring(3, Math.Min(3, text.Length - 3)));

yeniuzunluk = text.Length;

}

if (text.Length > 6)

{

formattedNumber.Append(" " + text.Substring(6, Math.Min(2, text.Length - 6)));

yeniuzunluk = text.Length;

}

if (text.Length > 8)

{

formattedNumber.Append(" " + text.Substring(8, Math.Max(0, text.Length - 8)));

yeniuzunluk = text.Length;

}

txtTel.Text = formattedNumber.ToString();

txtTel.SelectionStart = txtTel.Text.Length;

}

if (yeniuzunluk < eskiuzunluk)

{

if (text.Length == 3)

{

imlec(4);

}

if (text.Length == 0)

{

txtTel.Text = string.Empty;

}

}

eskiuzunluk = yeniuzunluk;

}  
 private void txtTel\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsDigit(e.KeyChar) && e.KeyChar != (char)Keys.Back)

{

e.Handled = true;

}

}

private void txtMail\_TextChanged(object sender, EventArgs e)

{

KontrolEt();

}

private int KontrolEt()

{

string email = txtMail.Text;

if (IsValidEmail(email))

{

txtMail.ForeColor = Color.Black;

return 1;

}

else

{

txtMail.ForeColor = Color.Red;

return 0;

}

}

private bool IsValidEmail(string email)

{

string emailPattern = @"^[a-zA-Z0-9.\_-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$";

return System.Text.RegularExpressions.Regex.IsMatch(email, emailPattern);

}

private void FillIl()

{

cmil.DataSource = \_blUser.IlSorgu();

cmil.DisplayMember = "IlAdi";

cmil.ValueMember = "IlID";

cmil.SelectedItem = -1;

}

private void cmil\_SelectedIndexChanged(object sender, EventArgs e)

{

if (cmil.SelectedValue != null)

{

DataRowView selectedRow = cmil.SelectedItem as DataRowView;

if (selectedRow != null)

{

int selectedIlID = Convert.ToInt32(selectedRow["IlID"]);

FillIlce(selectedIlID);

}

}

}

private void FillIlce(int ilId)

{

cmilce.DataSource = \_blUser.IlceSorgu(ilId);

cmilce.DisplayMember = "IlceAdi";

cmilce.ValueMember = "IlceID";

cmilce.SelectedItem = -1;

}

private bool KontrolleriGecerliMi()

{

if (txtAd.Text.Length <= 2 || txtSoyad.Text.Length <= 2)

{

MessageBox.Show("Lütfen Gerçek Adınızı ve Soyadınızı Giriniz.");

return false;

}

if (txtTel.Text.Length != 15)

{

MessageBox.Show("Lütfen Telefon numaranızı uygun formatta giriniz.");

return false;

}

if (KontrolEt() != 1)

{

MessageBox.Show("Lütfen MAİL adresinizi uygun formatta giriniz.");

return false;

}

if (cmil.SelectedIndex == -1 || cmilce.SelectedIndex == -1)

{

MessageBox.Show("Lütfen İl ve İlçe Seçiniz.");

return false;

}

if (txtAdres.Text.Length <= 7)

{

MessageBox.Show("Lütfen Açık Adres Bilginizi Giriniz.");

return false;

}

return true;

}

private void btnGuncelle\_Click(object sender, EventArgs e)

{

if (!KontrolleriGecerliMi())

{

return;

}

int ilID = cmil.SelectedIndex >= 0 ? Convert.ToInt32(cmil.SelectedValue) : -1;

int ilceID = cmilce.SelectedIndex >= 0 ? Convert.ToInt32(cmilce.SelectedValue) : -1;

bool insertSonuc = \_blUser.GuncellemeSorgu(txtTc.Text, txtAd.Text, txtSoyad.Text, txtTel.Text, txtMail.Text, txtAdres.Text, ilID, ilceID);

if (insertSonuc)

{

MessageBox.Show("Güncelleme Başarılı!");

ClearForm();

FormGuncelle();

txtAd.Focus();

CMenu cMenu = Application.OpenForms["CMenu"] as CMenu;

cMenu.SetLabelText();

}

else

{

MessageBox.Show("Kayıt gerçekleşirken bir hata oluştu.");

}

}

}

}

Bu kod, kullanıcının bilgilerini güncelleyebileceği bir formu içerir. İşlevselliği şu adımlarla açıklayabiliriz:

1. **txtTel\_KeyPress Metodu:**
   * Telefon numarası alanına sadece rakam girişi yapılmasını sağlar.
2. **txtMail\_TextChanged ve IsValidEmail Metotları:**
   * Email alanına yapılan her değişiklikte, **IsValidEmail** metodu çağrılarak e-posta adresinin doğruluğu kontrol edilir.
   * **IsValidEmail** metodu, e-posta adresinin belirli bir desene uygun olup olmadığını kontrol eder.
3. **FillIl ve FillIlce Metotları:**
   * İl ve ilçe combobox'larını doldurmak için kullanılır.
   * **\_blUser.IlSorgu()** ve **\_blUser.IlceSorgu(ilId)** metotları ile ilgili veriler alınıp combobox'lara atanır.
4. **cmil\_SelectedIndexChanged Metodu:**
   * İl combobox'ında bir seçim yapıldığında çağrılır.
   * Seçilen ilin ID'si alınarak, bu ilin ilçeleri **FillIlce** metodu ile ilgili combobox'a eklenir.
5. **KontrolleriGecerliMi Metodu:**
   * Girilen bilgilerin geçerliliğini kontrol eder.
   * Gerçek ad ve soyadın uzunluğunu, telefon numarası formatını, e-posta adresini, seçilen il ve ilçeyi, adres bilgisini kontrol eder.
   * Eğer bir hata varsa, kullanıcıya uygun bir hata mesajı gösterir ve **false** döner.
6. **btnGuncelle\_Click Metodu:**
   * Güncelleme butonuna tıklandığında çağrılır.
   * **KontrolleriGecerliMi** metodu ile giriş kontrolleri yapılır.
   * Kontroller geçerliyse, il ve ilçe ID'leri alınır.
   * **\_blUser.GuncellemeSorgu** metodu ile veritabanında güncelleme işlemi yapılır.
   * İşlem başarılıysa, kullanıcıya başarılı bir mesaj gösterilir, form temizlenir ve gerekli güncellemeler yapılır. Ayrıca, ana menüdeki bir etiket güncellenir.
   * İşlem başarısızsa, kullanıcıya bir hata mesajı gösterilir.

Bu kod, kullanıcı bilgilerini güncelleyen bir formun temel mantığını içermektedir.

using DernekUyeTakipSistemi.Customer.MenuLayer.CustomerInfoLayer.MenuLayer;

using System;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Customer.MenuLayer

{

public partial class CustomerInfoMenu : Form

{

private CustomerInfoLogin UserName;

private CustomerInfoPersonal UserUpdate;

private bool uygulamaKapatildi = false;

public static string Tc\_no { get; set; }

public CustomerInfoMenu()

{

InitializeComponent();

}

private void btnUserName\_Click(object sender, EventArgs e)

{

if (UserName == null || UserName.IsDisposed)

{

UserName = new CustomerInfoLogin();

UserName.FormClosed += (s, args) => this.Focus();

UserName.Show();

UserName.TcAtama(Tc\_no);

}

}

private void btnUpdate\_Click(object sender, EventArgs e)

{

if (UserUpdate == null || UserUpdate.IsDisposed)

{

UserUpdate = new CustomerInfoPersonal();

UserUpdate.FormClosed += (s, args) => this.Focus();

UserUpdate.Show();

UserUpdate.TcGetir(Tc\_no);

}

}

private void btnGeri\_Click(object sender, EventArgs e)

{

FormlariKapa();

uygulamaKapatildi = true;

this.Close();

CMenu cMenu = Application.OpenForms["CMenu"] as CMenu;

cMenu?.Show();

}

public void FormlariKapa()

{

if (UserName != null)

{

UserName.Close();

}

if (UserUpdate != null)

{

UserUpdate.Close();

}

}

private void CustomerInfoMenu\_FormClosing(object sender, FormClosingEventArgs e)

{

if (!uygulamaKapatildi)

{

DialogResult cikis = MessageBox.Show("Çıkış yapmak istediğinize emin misiniz ?", "", MessageBoxButtons.YesNo);

if (cikis == DialogResult.Yes)

{

uygulamaKapatildi = true;

Application.Exit();

}

else

{

e.Cancel = true;

}

}

}

}

}

Bu kod, bir müşterinin bilgilerini görüntüleyebileceği ve güncelleyebileceği bir menüyü içermektedir. İşlevselliği şu adımlarla açıklanabilir:

1. **Form Tanımlamaları:**
   * **CustomerInfoMenu** sınıfı, **Form** sınıfından türetilir.
   * **CustomerInfoLogin** ve **CustomerInfoPersonal** formlarını özel sınıf değişkenleri olarak içerir.
   * **Tc\_no** adında bir static property içerir, bu property üzerinden TC kimlik numarası diğer formlara aktarılır.
2. **Constructor (CustomerInfoMenu Metodu):**
   * Formun başlatılması sırasında çağrılır.
   * **InitializeComponent** metodu ile form öğeleri başlatılır.
3. **btnUserName\_Click ve btnUpdate\_Click Metotları:**
   * "Kullanıcı Adı Değiştir" ve "Kişisel Bilgileri Güncelle" butonlarına tıklama olaylarıdır.
   * İlgili formlar (**CustomerInfoLogin** ve **CustomerInfoPersonal**) daha önce oluşturulmuşsa, tekrar oluşturmadan gösterilir. Her formun kendi özel metotları çağrılarak TC kimlik numarası aktarılır.
4. **btnGeri\_Click Metodu:**
   * "Geri" butonuna tıklama olayıdır.
   * **FormlariKapa** metodu çağrılarak açık olan formlar kapatılır.
   * Ana menü formu açılarak bu form gizlenir.
5. **FormlariKapa Metodu:**
   * Açık olan **CustomerInfoLogin** ve **CustomerInfoPersonal** formlarını kapatır.
6. **CustomerInfoMenu\_FormClosing Metodu:**
   * Form kapatılmaya çalışıldığında çağrılır.
   * Eğer uygulama daha önce kapatılmamışsa, bir çıkış uyarısı gösterir.
   * Kullanıcı evet derse uygulamayı kapatır, hayır derse formu kapatmayı iptal eder.

Bu kod, müşteri bilgilerini görüntüleyen ve güncelleyen bir menüyü kontrol etmek için kullanılır.

using BusinessLayer;

using System;

using System.Diagnostics;

using System.Drawing;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Customer.MenuLayer

{

public partial class CustomerDebt : Form

{

BlUser \_blUser;

public static string Tc\_No { get; set; }

private bool uygulamaKapatildi = false;

public CustomerDebt()

{

InitializeComponent();

\_blUser = new BlUser();

goster();

}

private void ekran\_CellClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex >= 0)

{

DataGridViewRow selectedRow = dataGridView1.Rows[e.RowIndex];

cbYil.SelectedItem = selectedRow.Cells["AidatYil"].Value.ToString();

int ay = Convert.ToInt32(selectedRow.Cells["AidatAy"].Value);

if (ay < 10) { ay = ay - 1; string ayNo = '0' + ay.ToString();cbAy.SelectedIndex = Convert.ToInt32(ayNo); }

if (ay >= 10) { ay = ay - 1; cbAy.SelectedIndex = ay; }

}

}

private void ekran\_CellFormatting(object sender, DataGridViewCellFormattingEventArgs e)

{

if (e.RowIndex >= 0)

{

int sonOdemeTarihIndex = dataGridView1.Columns["SonOdemeTarih"].Index;

if (e.ColumnIndex == sonOdemeTarihIndex)

{

DateTime sonOdemeTarih = Convert.ToDateTime(dataGridView1.Rows[e.RowIndex].Cells[sonOdemeTarihIndex].Value);

DateTime simdikiTarih = DateTime.Now.Date;

if (sonOdemeTarih < simdikiTarih)

{

dataGridView1.Rows[e.RowIndex].DefaultCellStyle.BackColor = Color.Red;

}

else if ((sonOdemeTarih - simdikiTarih).Days <= 5)

{

dataGridView1.Rows[e.RowIndex].DefaultCellStyle.BackColor = Color.Yellow;

}

}

}

}

private void btnGeri\_Click(object sender, EventArgs e)

{

uygulamaKapatildi = true;

this.Close();

CMenu cMenu = Application.OpenForms["CMenu"] as CMenu;

cMenu?.Show();

}

public void goster()

{

dataGridView1.DataSource = \_blUser.BorcGetirBos(Tc\_No);

dataGridView1.Refresh();

ToplamBorcGuncelle();

}

private void BorcMenu\_Load(object sender, EventArgs e)

{

goster();

}

private void Hangisi()

{

if (cbYil.SelectedIndex == -1 && cbAy.SelectedIndex == -1)

{

goster();

}

if (cbYil.SelectedIndex != -1 && cbAy.SelectedIndex == -1)

{

ToplamBorcGuncelle();

}

if (cbYil.SelectedIndex == -1 && cbAy.SelectedIndex != -1)

{

ToplamBorcGuncelle();

}

if (cbYil.SelectedIndex != -1 && cbAy.SelectedIndex != -1)

{

string yil = cbYil.Text;

string Ay = cbAy.SelectedItem.ToString();

int AyNumarasi = cbAy.Items.IndexOf(Ay) + 1;

string ay = AyNumarasi < 10 ? "0" + AyNumarasi.ToString() : AyNumarasi.ToString();

dataGridView1.DataSource = \_blUser.BorcGetirYilAy(Tc\_No, yil, ay);

ToplamBorcGuncelle();

string borc = txtBorc.Text;

txtTutar.Text = borc;

}

}

private void cbYil\_SelectedIndexChanged(object sender, EventArgs e)

{

Hangisi();

}

private void cbAy\_SelectedIndexChanged(object sender, EventArgs e)

{

Hangisi();

}

private void btnOdeme\_Click(object sender, EventArgs e)

{

if (cbYil.SelectedIndex == -1 && cbAy.SelectedIndex == -1)

{

goster();

}

if (cbYil.SelectedIndex != -1 && cbAy.SelectedIndex == -1)

{

goster();

cbYil.SelectedIndex = -1;

}

if (cbYil.SelectedIndex == -1 && cbAy.SelectedIndex != -1)

{

goster();

cbAy.SelectedIndex = -1;

}

if (cbAy.SelectedIndex != -1 && cbAy.SelectedIndex != -1)

{

int tutar = Convert.ToInt32(txtTutar.Text);

if (tutar >= 1)

{

string yil = cbYil.Text;

string Ay = cbAy.SelectedItem.ToString();

int AyNumarasi = cbAy.Items.IndexOf(Ay) + 1;

string ay = AyNumarasi < 10 ? "0" + AyNumarasi.ToString() : AyNumarasi.ToString();

bool updateSonuc = false;

updateSonuc = \_blUser.TutarGuncelle(Tc\_No, yil, ay, tutar);

if (updateSonuc)

{

MessageBox.Show("Borç Ödeme İşlemi Başarılı.");

goster();

cbYil.SelectedIndex = -1;

cbAy.SelectedIndex = -1;

txtTutar.Clear();

}

else

{

MessageBox.Show("Borç Ödeme İşlemi Başarısız.");

goster();

cbYil.SelectedIndex = -1;

cbAy.SelectedIndex = -1;

}

}

if (tutar == 0)

{

goster();

cbYil.SelectedIndex = -1;

cbAy.SelectedIndex = -1;

}

}

}

private void ToplamBorcGuncelle()

{

int toplamBorc = 0;

foreach (DataGridViewRow row in dataGridView1.Rows)

{

int borcMiktar;

if (row.Cells["BorcMiktar"].Value != null && int.TryParse(row.Cells["BorcMiktar"].Value.ToString(), out borcMiktar))

{

toplamBorc += borcMiktar;

}

else

{

MessageBox.Show("BorcMiktar sütunu geçersiz bir değer içeriyor.");

}

}

txtBorc.Text = toplamBorc.ToString();

}

private void btnTemizle\_Click(object sender, EventArgs e)

{

cbYil.SelectedIndex = -1;

cbAy.SelectedIndex = -1;

txtTutar.Text = null;

}

private void CustomerDebt\_FormClosing(object sender, FormClosingEventArgs e)

{

if (!uygulamaKapatildi)

{

DialogResult cikis = MessageBox.Show("Çıkış yapmak istediğinize emin misiniz ?", "", MessageBoxButtons.YesNo);

if (cikis == DialogResult.Yes)

{

uygulamaKapatildi = true;

Application.Exit();

}

else

{

e.Cancel = true;

}

}

}

}

}

Bu kod, bir müşterinin borç bilgilerini görüntüleyebileceği, ödeme yapabileceği ve bu ödemeleri takip edebileceği bir menüyü içermektedir. İşlevselliği şu adımlarla açıklanabilir:

1. **Form Tanımlamaları:**
   * **CustomerDebt** sınıfı, **Form** sınıfından türetilir.
   * **BlUser** tipinde bir özel sınıf değişkeni içerir.
   * **Tc\_No** adında bir static property içerir, bu property üzerinden TC kimlik numarası diğer sınıflara aktarılır.
2. **Constructor (CustomerDebt Metodu):**
   * Formun başlatılması sırasında çağrılır.
   * **InitializeComponent** metodu ile form öğeleri başlatılır.
   * **BlUser** sınıfından bir örnek oluşturulur ve **\_blUser** değişkenine atanır.
   * **goster** metodu çağrılarak müşterinin borç bilgileri DataGridView üzerinde gösterilir.
3. **ekran\_CellClick Metodu:**
   * DataGridView hücrelerine tıklama olayıdır.
   * Seçili satırın bilgileri ComboBox'lara aktarılır.
4. **ekran\_CellFormatting Metodu:**
   * DataGridView hücre biçimlendirme olayıdır.
   * **SonOdemeTarih** sütunundaki tarih değerlerine göre satır renkleri belirlenir.
5. **btnGeri\_Click Metodu:**
   * "Geri" butonuna tıklama olayıdır.
   * Ana menü formu açılarak bu form gizlenir.
6. **goster Metodu:**
   * **\_blUser.BorcGetirBos** metodu ile müşterinin borç bilgileri çekilir ve DataGridView üzerinde gösterilir.
7. **BorcMenu\_Load Metodu:**
   * Form yüklenirken çağrılır.
   * **goster** metodu çağrılarak başlangıçta borç bilgileri gösterilir.
8. **Hangisi Metodu:**
   * ComboBox'larda seçili yıl ve ay değerlerine göre borç bilgilerini güncelleyen metodur.
9. **cbYil\_SelectedIndexChanged ve cbAy\_SelectedIndexChanged Metotları:**
   * ComboBox'lardan birinde seçim yapıldığında çağrılır.
   * **Hangisi** metodu çağrılarak borç bilgileri güncellenir.
10. **btnOdeme\_Click Metodu:**
    * "Ödeme Yap" butonuna tıklama olayıdır.
    * Seçili yıl ve ay değerlerine göre belirlenen tutarda ödeme yapılır.
    * Ödeme sonucuna göre bilgilendirme mesajları gösterilir.
11. **ToplamBorcGuncelle Metodu:**
    * DataGridView'daki borç miktarları toplanarak toplam borç bilgisini günceller.
12. **btnTemizle\_Click Metodu:**
    * "Temizle" butonuna tıklama olayıdır.
    * ComboBox'ları ve tutar TextBox'ını temizler.
13. **CustomerDebt\_FormClosing Metodu:**
    * Form kapatılmaya çalışıldığında çağrılır.
    * Eğer uygulama daha önce kapatılmamışsa, bir çıkış uyarısı gösterir.
    * Kullanıcı evet derse uygulamayı kapatır, hayır derse formu kapatmayı iptal eder.

Bu kod, müşterinin borç bilgilerini yönetmesine olanak tanıyan bir formu kontrol etmek için kullanılır.

using BusinessLayer;

using System;

using System.Drawing;

using System.Windows.Forms;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace DernekUyeTakipSistemi.Customer.MenuLayer

{

public partial class CustomerDues : Form

{

BlUserLogin \_blUserLogin;

public static string Tc\_no { get; set; }

private bool uygulamaKapatildi = false;

public CustomerDues()

{

InitializeComponent();

cbOdeme.SelectedItem = "Tümü";

\_blUserLogin = new BlUserLogin();

}

private void ekran\_CellClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex >= 0)

{

DataGridViewRow selectedRow = ekran.Rows[e.RowIndex];

cbYil.SelectedItem = selectedRow.Cells["AidatYil"].Value.ToString();

int ayNumarasi = Convert.ToInt32(selectedRow.Cells["AidatAy"].Value);

// Ay isimlerini içeren bir dizi oluştur

string[] ayIsimleri = { "Ocak", "Şubat", "Mart", "Nisan", "Mayıs", "Haziran", "Temmuz", "Ağustos", "Eylül", "Ekim", "Kasım", "Aralık" };

// Ay numarasına karşılık gelen ay ismini bul

string seciliAy = ayIsimleri[ayNumarasi - 1];

// Combobox'ta göster

cbAy.SelectedItem = seciliAy;

}

}

private void ekran\_CellFormatting(object sender, DataGridViewCellFormattingEventArgs e)

{

if (e.RowIndex >= 0)

{

// Örneğin, SonOdemeTarih sütunu 5. sütun, OdemeDurum sütunu ise 3. sütun ise

int sonOdemeTarihIndex = ekran.Columns["SonOdemeTarih"].Index;

int odemeDurumIndex = ekran.Columns["OdemeDurum"].Index;

if (e.ColumnIndex == sonOdemeTarihIndex || e.ColumnIndex == odemeDurumIndex)

{

// Son ödeme tarihi değerini al

DateTime sonOdemeTarih = Convert.ToDateTime(ekran.Rows[e.RowIndex].Cells[sonOdemeTarihIndex].Value);

// Odeme durumu değerini al

bool odemeDurum = Convert.ToBoolean(ekran.Rows[e.RowIndex].Cells[odemeDurumIndex].Value);

// Günümüz tarihini al

DateTime simdikiTarih = DateTime.Now;

// Eğer son ödeme tarihi geçmiş ve ödeme durumu false ise, satırın arka plan rengini kırmızı yap

if (sonOdemeTarih < simdikiTarih && !odemeDurum)

{

ekran.Rows[e.RowIndex].DefaultCellStyle.BackColor = Color.Red;

}

// Eğer ödeme durumu true ise, satırın arka plan rengini yeşil yap

else if (odemeDurum)

{

ekran.Rows[e.RowIndex].DefaultCellStyle.BackColor = Color.Green;

}

// Eğer son ödeme tarihine 5 gün kala ise, satırın arka plan rengini sarı yap

else if ((sonOdemeTarih - simdikiTarih).Days <= 5)

{

ekran.Rows[e.RowIndex].DefaultCellStyle.BackColor = Color.Yellow;

}

}

}

}

private void btnTemizle\_Click(object sender, EventArgs e)

{

cbYil.SelectedItem = null;

cbAy.SelectedItem = null;

cbOdeme.SelectedItem = "Tümü";

}

private void btnListele\_Click(object sender, EventArgs e)

{

string itemSelect = (string)cbOdeme.SelectedItem;

if (cbYil.SelectedItem != null && cbAy.SelectedItem != null && itemSelect == "Tümü")

{

string Yil = cbYil.SelectedItem.ToString();

string ay = cbAy.SelectedItem.ToString();

int AyNumarasi = cbAy.Items.IndexOf(ay) + 1;

string Ay = AyNumarasi < 10 ? "0" + AyNumarasi.ToString() : AyNumarasi.ToString();

ekran.DataSource = \_blUserLogin.AidatYilAySorgu1(Tc\_no, Yil, Ay);

}

if (cbYil.SelectedItem != null && cbAy.SelectedItem != null && itemSelect == "Ödenmiş")

{

string Yil = cbYil.SelectedItem.ToString();

string ay = cbAy.SelectedItem.ToString();

int AyNumarasi = cbAy.Items.IndexOf(ay) + 1;

string Ay = AyNumarasi < 10 ? "0" + AyNumarasi.ToString() : AyNumarasi.ToString();

bool OdemeDurum = true;

ekran.DataSource = \_blUserLogin.AidatYilAySorgu2(Tc\_no, Yil, Ay, OdemeDurum);

}

if (cbYil.SelectedItem != null && cbAy.SelectedItem != null && itemSelect == "Ödenmemiş")

{

string Yil = cbYil.SelectedItem.ToString();

string ay = cbAy.SelectedItem.ToString();

int AyNumarasi = cbAy.Items.IndexOf(ay) + 1;

string Ay = AyNumarasi < 10 ? "0" + AyNumarasi.ToString() : AyNumarasi.ToString();

bool OdemeDurum = false;

ekran.DataSource = \_blUserLogin.AidatYilAySorgu2(Tc\_no, Yil, Ay, OdemeDurum);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\\

if (cbYil.SelectedItem != null && cbAy.SelectedItem == null && itemSelect == "Tümü")

{

string Yil = cbYil.SelectedItem.ToString();

ekran.DataSource = \_blUserLogin.AidatYilSorgu1(Tc\_no, Yil);

}

if (cbYil.SelectedItem != null && cbAy.SelectedItem == null && itemSelect == "Ödenmiş")

{

string Yil = cbYil.SelectedItem.ToString();

bool OdemeDurum = true;

ekran.DataSource = \_blUserLogin.AidatYilSorgu2(Tc\_no, Yil, OdemeDurum);

}

if (cbYil.SelectedItem != null && cbAy.SelectedItem == null && itemSelect == "Ödenmemiş")

{

string Yil = cbYil.SelectedItem.ToString();

bool OdemeDurum = false;

ekran.DataSource = \_blUserLogin.AidatYilSorgu2(Tc\_no, Yil, OdemeDurum);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\\

if (cbYil.SelectedItem == null && cbAy.SelectedItem != null && itemSelect == "Tümü")

{

string ay = cbAy.SelectedItem.ToString();

int AyNumarasi = cbAy.Items.IndexOf(ay) + 1;

string Ay = AyNumarasi < 10 ? "0" + AyNumarasi.ToString() : AyNumarasi.ToString();

ekran.DataSource = \_blUserLogin.AidatAySorgu1(Tc\_no, Ay);

}

if (cbYil.SelectedItem == null && cbAy.SelectedItem != null && itemSelect == "Ödenmiş")

{

string ay = cbAy.SelectedItem.ToString();

int AyNumarasi = cbAy.Items.IndexOf(ay) + 1;

string Ay = AyNumarasi < 10 ? "0" + AyNumarasi.ToString() : AyNumarasi.ToString();

bool OdemeDurum = true;

ekran.DataSource = \_blUserLogin.AidatAySorgu2(Tc\_no, Ay, OdemeDurum);

}

if (cbYil.SelectedItem == null && cbAy.SelectedItem != null && itemSelect == "Ödenmemiş")

{

string ay = cbAy.SelectedItem.ToString();

int AyNumarasi = cbAy.Items.IndexOf(ay) + 1;

string Ay = AyNumarasi < 10 ? "0" + AyNumarasi.ToString() : AyNumarasi.ToString();

bool OdemeDurum = false;

ekran.DataSource = \_blUserLogin.AidatAySorgu2(Tc\_no, Ay, OdemeDurum);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\\

if (cbYil.SelectedItem == null && cbAy.SelectedItem == null && itemSelect == "Tümü")

{

ekran.DataSource = \_blUserLogin.AidatBosSorgu1(Tc\_no);

}

if (cbYil.SelectedItem == null && cbAy.SelectedItem == null && itemSelect == "Ödenmiş")

{

bool OdemeDurum = true;

ekran.DataSource = \_blUserLogin.AidatBosSorgu2(Tc\_no, OdemeDurum);

}

if (cbYil.SelectedItem == null && cbAy.SelectedItem == null && itemSelect == "Ödenmemiş")

{

bool OdemeDurum = false;

ekran.DataSource = \_blUserLogin.AidatBosSorgu2(Tc\_no, OdemeDurum);

}

}

private void btnGeri\_Click(object sender, EventArgs e)

{

uygulamaKapatildi = true;

this.Close();

CMenu cMenu = Application.OpenForms["CMenu"] as CMenu;

cMenu?.Show();

}

private void CustomerDues\_FormClosing(object sender, FormClosingEventArgs e)

{

if (!uygulamaKapatildi)

{

DialogResult cikis = MessageBox.Show("Çıkış yapmak istediğinize emin misiniz ?", "", MessageBoxButtons.YesNo);

if (cikis == DialogResult.Yes)

{

uygulamaKapatildi = true;

Application.Exit();

}

else

{

e.Cancel = true;

}

}

}

}

}

Bu kod, bir müşterinin ödeme durumlarına göre aidat bilgilerini filtreleyebileceği ve bu bilgileri görebileceği bir formu içermektedir. İşlevselliği şu adımlarla açıklanabilir:

1. **Form Tanımlamaları:**
   * **CustomerDues** sınıfı, **Form** sınıfından türetilir.
   * **BlUserLogin** tipinde bir özel sınıf değişkeni içerir.
   * **Tc\_no** adında bir static property içerir, bu property üzerinden TC kimlik numarası diğer sınıflara aktarılır.
2. **Constructor (CustomerDues Metodu):**
   * Formun başlatılması sırasında çağrılır.
   * **InitializeComponent** metodu ile form öğeleri başlatılır.
   * ComboBox'ın varsayılan seçeneği "Tümü" olarak atanır.
   * **BlUserLogin** sınıfından bir örnek oluşturulur ve **\_blUserLogin** değişkenine atanır.
3. **ekran\_CellClick Metodu:**
   * DataGridView hücrelerine tıklama olayıdır.
   * Seçili satırın bilgileri ComboBox'lara aktarılır.
4. **ekran\_CellFormatting Metodu:**
   * DataGridView hücre biçimlendirme olayıdır.
   * **SonOdemeTarih** ve **OdemeDurum** sütunlarına göre satır renkleri belirlenir.
5. **btnTemizle\_Click Metodu:**
   * "Temizle" butonuna tıklama olayıdır.
   * ComboBox'ları temizler.
6. **btnListele\_Click Metodu:**
   * "Listele" butonuna tıklama olayıdır.
   * ComboBox'lardan seçilen değerlere göre aidat bilgilerini filtreleyerek DataGridView'e yükler.
   * Çeşitli durumları kontrol ederek, uygun sorgu metodunu kullanır.
7. **btnGeri\_Click Metodu:**
   * "Geri" butonuna tıklama olayıdır.
   * Ana menü formu açılarak bu form gizlenir.
8. **CustomerDues\_FormClosing Metodu:**
   * Form kapatılmaya çalışıldığında çağrılır.
   * Eğer uygulama daha önce kapatılmamışsa, bir çıkış uyarısı gösterir.
   * Kullanıcı evet derse uygulamayı kapatır, hayır derse formu kapatmayı iptal eder.

Bu kod, müşterinin ödeme durumlarına göre aidat bilgilerini filtreleyebileceği bir formu kontrol etmek için kullanılır.

using BusinessLayer;

using DernekUyeTakipSistemi.Customer.MenuLayer;

using System;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Customer

{

public partial class CMenu : Form

{

private CustomerInfoMenu UserMenu;

private CustomerDues AidatMenu;

private CustomerDebt BorcMenu;

public string kullaniciAdi;

BlUserLogin \_blUserLogin;

BlUser \_blUser;

private string tc\_no;

public CMenu()

{

InitializeComponent();

\_blUserLogin = new BlUserLogin();

\_blUser = new BlUser();

}

public void TcNoAlma(string a)

{

tc\_no = \_blUserLogin.TcNoAl(a);

SetLabelText();

}

public void SetLabelText()

{

string isim = \_blUser.AdGetir(tc\_no);

label1.Text = "Merhaba Sayın " + isim.ToUpper();

}

private void btnUser\_Click(object sender, EventArgs e)

{

if (UserMenu == null || UserMenu.IsDisposed)

{

UserMenu = new CustomerInfoMenu();

UserMenu.Show();

CustomerInfoMenu.Tc\_no = tc\_no;

this.Hide();

}

}

private void btnAidat\_Click(object sender, EventArgs e)

{

if (AidatMenu == null || AidatMenu.IsDisposed)

{

AidatMenu = new CustomerDues();

AidatMenu.Show();

CustomerDues.Tc\_no = tc\_no;

this.Hide();

}

}

private void btnBorc\_Click(object sender, EventArgs e)

{

if (BorcMenu == null || BorcMenu.IsDisposed)

{

BorcMenu = new CustomerDebt();

CustomerDebt.Tc\_No = tc\_no;

BorcMenu.Show();

this.Hide();

}

}

private void btnExit\_Click(object sender, EventArgs e)

{

DialogResult cıkıs = MessageBox.Show("Çıkış yapmak istediğinize emin misiniz ?", "", MessageBoxButtons.YesNo);

if (cıkıs == DialogResult.Yes)

{

LoginPage loginPage = Application.OpenForms["LoginPage"] as LoginPage;

loginPage.Show();

this.Close();

}

}

}

}

Bu kod, bir dernek üye takip sisteminin müşteri menüsünü kontrol etmek için kullanılır. İşlevselliği aşağıdaki adımlarla açıklanabilir:

1. **Form Tanımlamaları:**
   * **CMenu** sınıfı, **Form** sınıfından türetilir.
   * **CustomerInfoMenu**, **CustomerDues**, ve **CustomerDebt** tiplerinde form değişkenleri içerir.
   * **BlUserLogin** ve **BlUser** tiplerinde iş mantığı sınıflarının örneklerini içerir.
   * **kullaniciAdi** ve **tc\_no** adında string değişkenleri içerir.
2. **Constructor (CMenu Metodu):**
   * Formun başlatılması sırasında çağrılır.
   * **InitializeComponent** metodu ile form öğeleri başlatılır.
   * **\_blUserLogin** ve **\_blUser** değişkenleri oluşturulur.
3. **TcNoAlma Metodu:**
   * Dışarıdan bir TC numarası alır ve **\_blUserLogin** sınıfının **TcNoAl** metodunu kullanarak bu TC numarasını **tc\_no** değişkenine atar.
   * Ardından **SetLabelText** metodunu çağırarak kullanıcı adını belirler ve etikete yazar.
4. **SetLabelText Metodu:**
   * **\_blUser** sınıfının **AdGetir** metodunu kullanarak **tc\_no**'ya karşılık gelen kullanıcının adını alır.
   * Bu adı, bir etiketin metin özelliğine ekleyerek kullanıcıya hitap eden bir mesaj oluşturur.
5. **btnUser\_Click, btnAidat\_Click, btnBorc\_Click Metodları:**
   * Sırasıyla "Kullanıcı Bilgileri", "Aidat Bilgileri" ve "Borç Bilgileri" butonlarına tıklama olaylarıdır.
   * İlgili formların örnekleri oluşturulur, gerekli TC numarası bu formlara atanır, mevcut form gizlenir ve yeni form gösterilir.
   * Her form için sadece bir örnek oluşturulur ve daha önce oluşturulmuşsa veya kapatılmışsa, yeni bir örnek oluşturulur ve gösterilir.
6. **btnExit\_Click Metodu:**
   * "Çıkış" butonuna tıklama olayıdır.
   * Kullanıcıya çıkış yapmak isteyip istemediğini soran bir iletişim kutusu gösterir.
   * Eğer kullanıcı çıkış yapmak isterse, giriş sayfasını açar ve mevcut formu kapatır.

Bu kod, bir müşterinin kullanıcı bilgilerini, aidat bilgilerini ve borç bilgilerini görebileceği bir menüyü kontrol etmek için kullanılır.

Formun Üstü

using BusinessLayer;

using DernekUyeTakipSistemi.Employee.MenuLayer.ListLayer.ListMenu;

using DernekUyeTakipSistemi.Employee.MenuLayer.EmployeeInfoLayer;

using DernekUyeTakipSistemi.Employee.MenuLayer.CustomerEditLayer.MenuLayer;

using DernekUyeTakipSistemi.Employee.MenuLayer.DuesLayer.MenuLayer;

using DernekUyeTakipSistemi.Employee.MenuLayer.CustomerEditLayer;

using System;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Employee

{

public partial class EMenu : Form

{

BlUserLogin \_blUserLogin;

BlUser \_blUser;

EmployeeInfoMenu BilgilerimMenu;

CustomerEditMenu CustomerEditMenu;

DuesMenu Aidat;

ListMenu Listeleme;

public string kullaniciAdi;

public string tc\_no;

public EMenu()

{

InitializeComponent();

\_blUserLogin = new BlUserLogin();

\_blUser = new BlUser();

}

public void TcNoAlma(string a)

{

tc\_no = \_blUserLogin.TcNoAl(a);

SetLabelText();

}

public void SetLabelText()

{

string isim = \_blUser.AdGetir(tc\_no);

label1.Text = "Merhaba Sayın " + isim.ToUpper();

}

private void btnBilgilerim\_Click(object sender, EventArgs e)

{

if (BilgilerimMenu == null || BilgilerimMenu.IsDisposed)

{

BilgilerimMenu = new EmployeeInfoMenu();

BilgilerimMenu.Show();

EmployeeInfoMenu.Tc\_no = tc\_no;

this.Hide();

}

}

private void btnKullanici\_Click(object sender, EventArgs e)

{

if (CustomerEditMenu == null || CustomerEditMenu.IsDisposed)

{

CustomerEditMenu = new CustomerEditMenu();

CustomerEditMenu.Show();

this.Hide();

}

}

private void btnAidat\_Click(object sender, EventArgs e)

{

if (Aidat == null || Aidat.IsDisposed)

{

Aidat = new DuesMenu();

Aidat.Show();

this.Hide();

}

}

private void btnListele\_Click(object sender, EventArgs e)

{

if (Listeleme == null || Listeleme.IsDisposed)

{

Listeleme = new ListMenu();

Listeleme.Show();

this.Hide();

}

}

private void btnExit\_Click(object sender, EventArgs e)

{

DialogResult cıkıs = MessageBox.Show("Çıkış yapmak istediğinize emin misiniz ?", "", MessageBoxButtons.YesNo);

if (cıkıs == DialogResult.Yes)

{

LoginPage loginPage = Application.OpenForms["LoginPage"] as LoginPage;

loginPage.Show();

this.Close();

}

}

}

}

Bu kod, bir çalışanın (employee) kullanıcı arayüzünü kontrol etmek için kullanılır. İşlevselliği aşağıdaki adımlarla açıklanabilir:

1. **Form Tanımlamaları:**
   * **EMenu** sınıfı, **Form** sınıfından türetilir.
   * **\_blUserLogin** ve **\_blUser** tiplerinde iş mantığı sınıflarının örneklerini içerir.
   * **BilgilerimMenu**, **CustomerEditMenu**, **Aidat**, ve **Listeleme** tiplerinde form değişkenleri içerir.
   * **kullaniciAdi** ve **tc\_no** adında string değişkenleri içerir.
2. **Constructor (EMenu Metodu):**
   * Formun başlatılması sırasında çağrılır.
   * **InitializeComponent** metodu ile form öğeleri başlatılır.
   * **\_blUserLogin** ve **\_blUser** değişkenleri oluşturulur.
3. **TcNoAlma Metodu:**
   * Dışarıdan bir TC numarası alır ve **\_blUserLogin** sınıfının **TcNoAl** metodunu kullanarak bu TC numarasını **tc\_no** değişkenine atar.
   * Ardından **SetLabelText** metodunu çağırarak kullanıcı adını belirler ve etikete yazar.
4. **SetLabelText Metodu:**
   * **\_blUser** sınıfının **AdGetir** metodunu kullanarak **tc\_no**'ya karşılık gelen kullanıcının adını alır.
   * Bu adı, bir etiketin metin özelliğine ekleyerek kullanıcıya hitap eden bir mesaj oluşturur.
5. **btnBilgilerim\_Click, btnKullanici\_Click, btnAidat\_Click, ve btnListele\_Click Metodları:**
   * Sırasıyla "Bilgilerim", "Kullanıcı Düzenle", "Aidat", ve "Listeleme" butonlarına tıklama olaylarıdır.
   * İlgili formların örnekleri oluşturulur, gerekli TC numarası bu formlara atanır, mevcut form gizlenir ve yeni form gösterilir.
   * Her form için sadece bir örnek oluşturulur ve daha önce oluşturulmuşsa veya kapatılmışsa, yeni bir örnek oluşturulur ve gösterilir.
6. **btnExit\_Click Metodu:**
   * "Çıkış" butonuna tıklama olayıdır.
   * Kullanıcıya çıkış yapmak isteyip istemediğini soran bir iletişim kutusu gösterir.
   * Eğer kullanıcı çıkış yapmak isterse, giriş sayfasını açar ve mevcut formu kapatır.

Bu kod, bir çalışanın bilgilerini, müşteri düzenleme işlemlerini, aidat takibini ve müşteri listeleme işlemlerini gerçekleştirmek için kullanılan bir menüyü kontrol etmektedir.

using BusinessLayer;

using DataAccessLayer;

using DernekUyeTakipSistemi.Admin;

using DernekUyeTakipSistemi.Employee.MenuLayer.CustomerEditLayer.MenuLayer;

using System;

using System.Collections.Generic;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Employee.MenuLayer.CustomerEditLayer

{

public partial class CustomerEditMenu : Form

{

BlUser \_blUser;

public CustomerEditEkle CustomerEditEkle;

public CustemerEditDuzenle CustemerEditDuzenle;

public CustomerEditSearch CustomerEditSearch;

public CustomerEditLogin CustomerEditLogin;

public string Tc\_no;

private bool uygulamaKapatildi = false;

public CustomerEditMenu()

{

InitializeComponent();

\_blUser = new BlUser();

goster();

}

public void goster()

{

ekran.DataSource = \_blUser.TabloSorgu("2");

ekran.AutoResizeColumns();

}

private void btnEkle\_Click(object sender, EventArgs e)

{

if (CustomerEditEkle == null || CustomerEditEkle.IsDisposed)

{

CustomerEditEkle = new CustomerEditEkle();

CustomerEditEkle.Show();

}

}

private void btnSil\_Click(object sender, EventArgs e)

{

bool deleteSonuc = \_blUser.SilmeSorgu(Tc\_no);

if (deleteSonuc)

{

MessageBox.Show("Silme işlemi başarılı.");

goster();

}

else

{

MessageBox.Show("Silme işlemi başarısız.");

}

}

private void btnGuncelle\_Click(object sender, EventArgs e)

{

if (CustemerEditDuzenle == null || CustemerEditDuzenle.IsDisposed)

{

CustemerEditDuzenle = new CustemerEditDuzenle();

CustemerEditDuzenle.Show();

}

}

private void btnLogin\_Click(object sender, EventArgs e)

{

if (CustomerEditLogin == null || CustomerEditLogin.IsDisposed)

{

CustomerEditLogin = new CustomerEditLogin();

CustomerEditLogin.Show();

}

if(CustomerEditLogin != null)

{

CustomerEditLogin.Show();

}

}

private void btnGeri\_Click(object sender, EventArgs e)

{

FormlariKapa();

EMenu eMenu = Application.OpenForms["EMenu"] as EMenu;

AMenu aMenu = Application.OpenForms["AMenu"] as AMenu;

if (aMenu != null)

{

uygulamaKapatildi = true;

this.Close();

aMenu.Show();

}

if (eMenu != null)

{

uygulamaKapatildi = true;

this.Close();

eMenu.Show();

}

}

public void FormlariKapa()

{

if (CustomerEditEkle != null)

{

CustomerEditEkle.Close();

}

if (CustemerEditDuzenle != null)

{

CustemerEditDuzenle.Close();

}

}

private void dataGridView1\_SelectionChanged(object sender, EventArgs e)

{

if (ekran.SelectedCells.Count > 0)

{

int selectedRowIndex = ekran.SelectedCells[0].RowIndex;

// TC No sütununun index'i

int tcNoColumnIndex = ekran.Columns["Tc\_No"].Index;

// Seçili satırdaki TC No'yu alın

string tcNo = ekran.Rows[selectedRowIndex].Cells[tcNoColumnIndex].Value.ToString();

Tc\_no = tcNo;

if (CustomerEditLogin != null)

{

CustomerEditLogin.TcAtama(tcNo);

}

if(CustomerEditLogin == null || CustomerEditLogin.IsDisposed)

{

CustomerEditLogin = new CustomerEditLogin();

CustomerEditLogin.TcAtama(tcNo);

}

}

}

private Dictionary<string, object> GetCellData(int rowIndex, int columnIndex)

{

Dictionary<string, object> cellData = new Dictionary<string, object>();

// Seçilen hücrenin verilerini sözlüğe ekle

foreach (DataGridViewCell cell in ekran.Rows[rowIndex].Cells)

{

cellData[cell.OwningColumn.Name] = cell.Value;

}

return cellData;

}

private void ekran\_CellClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex >= 0 && e.ColumnIndex >= 0)

{

// Seçilen satır ve sütun indekslerini al

int selectedRowIndex = e.RowIndex;

int selectedColumnIndex = e.ColumnIndex;

// Seçilen hücrenin verilerini diğer forma iletmek için olayı tetikle

if (CustemerEditDuzenle != null)

{

CustemerEditDuzenle.OnVeriSecildi(GetCellData(selectedRowIndex, selectedColumnIndex));

}

}

}

private void btnSearch\_Click(object sender, EventArgs e)

{

if (CustomerEditSearch == null || CustomerEditSearch.IsDisposed)

{

CustomerEditSearch = new CustomerEditSearch();

CustomerEditSearch.Show();

}

else

{

CustomerEditSearch.Show();

}

}

public void TcArama(string TcNo)

{

ekran.DataSource = \_blUser.TcArama(TcNo);

ekran.AutoResizeColumns();

}

public void AdSoyadArama(string Ad, String Soyad)

{

ekran.DataSource = \_blUser.AdSoyadArama(Ad, Soyad);

ekran.AutoResizeColumns();

}

public void TelArama(string Tel)

{

ekran.DataSource = \_blUser.TelArama(Tel);

ekran.AutoResizeColumns();

}

public void MailArama(string Mail)

{

ekran.DataSource = \_blUser.MailArama(Mail);

ekran.AutoResizeColumns();

}

private void btnRefresh\_Click(object sender, EventArgs e)

{

goster();

}

private void CustomerEditMenu\_FormClosing(object sender, FormClosingEventArgs e)

{

if (!uygulamaKapatildi)

{

DialogResult cikis = MessageBox.Show("Çıkış yapmak istediğinize emin misiniz ?", "", MessageBoxButtons.YesNo);

if (cikis == DialogResult.Yes)

{

uygulamaKapatildi = true;

Application.Exit();

}

else

{

e.Cancel = true;

}

}

}

}

}

Bu kod, bir çalışanın müşteri düzenleme işlemlerini gerçekleştirebileceği bir menüyü kontrol eder. İşlevselliği şu adımlarla açıklanabilir:

1. **Form Tanımlamaları:**
   * **CustomerEditMenu** sınıfı, **Form** sınıfından türetilir.
   * **\_blUser** tipinde bir iş mantığı sınıfının örneğini içerir.
   * **CustomerEditEkle**, **CustemerEditDuzenle**, **CustomerEditSearch**, ve **CustomerEditLogin** tiplerinde form değişkenleri içerir.
   * **Tc\_no** adında bir string değişkeni bulunur.
   * **uygulamaKapatildi** adında bir boolean değişkeni bulunur.
2. **Constructor (CustomerEditMenu Metodu):**
   * Formun başlatılması sırasında çağrılır.
   * **InitializeComponent** metodu ile form öğeleri başlatılır.
   * **\_blUser** değişkeni oluşturulur ve **goster** metodu çağrılır.
3. **goster Metodu:**
   * **\_blUser** sınıfının **TabloSorgu** metodunu kullanarak müşteri bilgilerini içeren bir DataGridView kontrolüne veri yükler.
   * DataGridView otomatik olarak sütunları boyutlandırır.
4. **"Ekle", "Sil", "Güncelle", "Login", "Geri" Butonlarına Tıklama Olayları:**
   * İlgili butonlara tıklanınca gerçekleşen olaylardır.
   * Her bir buton için ilgili formun örneği oluşturulur ve eğer daha önce oluşturulmamış veya kapatılmamışsa, yeni form gösterilir.
   * "Sil" butonuna tıklanınca **\_blUser** sınıfının **SilmeSorgu** metodunu kullanarak seçili müşteriyi siler ve sonucuna göre bir mesaj kutusu gösterir.
5. **dataGridView1\_SelectionChanged Metodu:**
   * DataGridView'da bir satır seçildiğinde tetiklenen olaydır.
   * Seçilen satırdaki TC No'yu alır ve **Tc\_no** değişkenine atar.
   * Eğer **CustomerEditLogin** formu varsa, bu forma seçilen TC No'yu ileterek ilgili işlemleri gerçekleştirir.
   * Eğer **CustomerEditLogin** formu yoksa veya kapatılmışsa, yeni bir örnek oluşturur ve seçilen TC No'yu ileterek gösterir.
6. **ekran\_CellClick Metodu:**
   * DataGridView hücresine tıklanınca gerçekleşen olaydır.
   * Seçilen hücrenin verilerini **CustemerEditDuzenle** formuna iletmek için bu formu tetikler.
7. **"Ara", "Yenile" Butonlarına Tıklama Olayları:**
   * "Ara" butonları için **CustomerEditSearch** formunu gösterir.
   * "Yenile" butonu için **goster** metodunu çağırarak DataGridView'i günceller.
8. **TcArama, AdSoyadArama, TelArama, MailArama Metodları:**
   * İlgili arama kriterlerine göre **\_blUser** sınıfının ilgili metotlarını çağırarak DataGridView'i günceller.
9. **CustomerEditMenu\_FormClosing Metodu:**
   * Form kapatılmak istendiğinde gerçekleşen olaydır.
   * Kullanıcıya çıkış yapmak isteyip istemediğini soran bir iletişim kutusu gösterir.
   * Eğer kullanıcı çıkış yapmak isterse, uygulama kapatıldı işaretini true yapar ve uygulamayı kapatır.

Bu kod, müşteri düzenleme menüsünün temel işlevselliğini içerir ve müşteri ekleme, silme, güncelleme, arama gibi işlemleri gerçekleştirebilecek bir arayüz sunar.

using BusinessLayer;

using System;

using System.Collections.Generic;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Employee.MenuLayer.CustomerEditLayer.MenuLayer

{

public partial class CustemerEditDuzenle : Form

{

BlUser \_blUser;

public CustemerEditDuzenle()

{

InitializeComponent();

\_blUser = new BlUser();

txtTc.MaxLength = 11;

txtAd.MaxLength = 15;

txtSoyad.MaxLength = 15;

txtTel.MaxLength = 15;

txtMail.MaxLength = 60;

txtAdres.MaxLength = 60;

this.AcceptButton = btnGuncelle;

FillIl();

}

private void txtAd\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsLetter(e.KeyChar) && e.KeyChar != (char)Keys.Back && e.KeyChar != (char)Keys.Space)

{

e.Handled = true;

}

}

private void txtSoyad\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsLetter(e.KeyChar) && e.KeyChar != (char)Keys.Back && e.KeyChar != (char)Keys.Space)

{

e.Handled = true;

}

}

private void txtTel\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsDigit(e.KeyChar) && e.KeyChar != (char)Keys.Back)

{

e.Handled = true;

}

}

private void imlec(int position)

{

txtTel.SelectionStart = position;

txtTel.Focus();

}

private int eskiuzunluk = 0;

private void txtTel\_TextChanged(object sender, EventArgs e)

{

string text = txtTel.Text.Replace("(", "").Replace(")", "").Replace(" ", "");

StringBuilder formattedNumber = new StringBuilder();

int yeniuzunluk = text.Length;

if (yeniuzunluk > eskiuzunluk)

{

if (text.Length > 0)

{

formattedNumber.Append("(" + text.Substring(0, Math.Min(3, text.Length)) + ")");

yeniuzunluk = text.Length;

}

if (text.Length > 3)

{

formattedNumber.Append(" " + text.Substring(3, Math.Min(3, text.Length - 3)));

yeniuzunluk = text.Length;

}

if (text.Length > 6)

{

formattedNumber.Append(" " + text.Substring(6, Math.Min(2, text.Length - 6)));

yeniuzunluk = text.Length;

}

if (text.Length > 8)

{

formattedNumber.Append(" " + text.Substring(8, Math.Max(0, text.Length - 8)));

yeniuzunluk = text.Length;

}

txtTel.Text = formattedNumber.ToString();

txtTel.SelectionStart = txtTel.Text.Length;

}

if (yeniuzunluk < eskiuzunluk)

{

if (text.Length == 3)

{

imlec(4);

}

if (text.Length == 0)

{

txtTel.Text = string.Empty;

}

}

eskiuzunluk = yeniuzunluk;

}

private void txtmail\_TextChanged(object sender, EventArgs e)

{

KontrolEt();

}

private int KontrolEt()

{

string email = txtMail.Text;

if (IsValidEmail(email))

{

txtMail.ForeColor = Color.Black;

return 1;

}

else

{

txtMail.ForeColor = Color.Red;

return 0;

}

}

private bool IsValidEmail(string email)

{

string emailPattern = @"^[a-zA-Z0-9.\_-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$";

return System.Text.RegularExpressions.Regex.IsMatch(email, emailPattern);

}

private void FillIl()

{

cmil.DataSource = \_blUser.IlSorgu();

cmil.DisplayMember = "IlAdi";

cmil.ValueMember = "IlID";

cmil.SelectedIndex = -1;

}

private void cmil\_SelectedIndexChanged(object sender, EventArgs e)

{

if (cmil.SelectedValue != null)

{

DataRowView selectedRow = cmil.SelectedItem as DataRowView;

if (selectedRow != null)

{

int selectedIlID = Convert.ToInt32(selectedRow["IlID"]);

FillIlce(selectedIlID);

}

}

}

private void FillIlce(int ilId)

{

cmilce.DataSource = \_blUser.IlceSorgu(ilId);

cmilce.DisplayMember = "IlceAdi";

cmilce.ValueMember = "IlceID";

cmilce.SelectedIndex = -1;

}

public void OnVeriSecildi(Dictionary<string, object> rowData)

{

txtTc.Text = rowData["Tc\_No"].ToString();

txtAd.Text = rowData["Ad"].ToString();

txtSoyad.Text = rowData["Soyad"].ToString();

cmCins.SelectedItem = rowData["Cinsiyet"].ToString();

cmKan.SelectedItem = rowData["Kan\_Grup"].ToString();

txtTel.Text = rowData["Telefon"].ToString();

txtMail.Text = rowData["Mail"].ToString();

txtAdres.Text = rowData["Adres"].ToString();

cmil.SelectedValue = rowData["IlID"];

cmilce.SelectedValue = rowData["IlceID"];

}

private bool KontrolleriGecerliMi()

{

if (txtAd.Text.Length <= 2 || txtSoyad.Text.Length <= 2)

{

MessageBox.Show("Lütfen Gerçek Adınızı ve Soyadınızı Giriniz.");

return false;

}

if (txtTel.Text.Length != 15)

{

MessageBox.Show("Lütfen Telefon numaranızı uygun formatta giriniz.");

return false;

}

if (KontrolEt() != 1)

{

MessageBox.Show("Lütfen MAİL adresinizi uygun formatta giriniz.");

return false;

}

if (cmil.SelectedIndex == -1 || cmilce.SelectedIndex == -1)

{

MessageBox.Show("Lütfen İl ve İlçe Seçiniz.");

return false;

}

if (txtAdres.Text.Length <= 7)

{

MessageBox.Show("Lütfen Açık Adres Bilginizi Giriniz.");

return false;

}

return true;

}

private void ClearForm()

{

// Formdaki tüm kontrol değerlerini temizleme işlemi

foreach (Control control in Controls)

{

if (control is System.Windows.Forms.TextBox textBox)

{

textBox.Clear();

}

else if (control is System.Windows.Forms.ComboBox comboBox)

{

comboBox.SelectedIndex = -1;

}

}

}

private void btnGuncelle\_Click(object sender, EventArgs e)

{

if (!KontrolleriGecerliMi())

{

return;

}

int ilID = cmil.SelectedIndex >= 0 ? Convert.ToInt32(cmil.SelectedValue) : -1;

int ilceID = cmilce.SelectedIndex >= 0 ? Convert.ToInt32(cmilce.SelectedValue) : -1;

bool insertSonuc = \_blUser.GuncellemeSorgu(txtTc.Text, txtAd.Text, txtSoyad.Text, txtTel.Text, txtMail.Text, txtAdres.Text, ilID, ilceID);

if (insertSonuc)

{

MessageBox.Show("Güncelleme Başarılı!");

ClearForm();

CustomerEditMenu usrmenu = Application.OpenForms["CustomerEditMenu"] as CustomerEditMenu;

usrmenu?.goster();

txtAd.Focus();

}

else

{

MessageBox.Show("Kayıt gerçekleşirken bir hata oluştu.");

}

}

}

}

Bu kod, müşteri bilgilerini düzenlemek amacıyla kullanılan bir formu kontrol eder. Aşağıda, kodun temel mantığını metinsel olarak açıklamalarla birlikte bulabilirsiniz:

1. **Form Tanımlamaları:**
   * **CustemerEditDuzenle** sınıfı, **Form** sınıfından türetilir.
   * **\_blUser** tipinde bir iş mantığı sınıfının örneğini içerir.
   * Kullanıcı girişi, sadece harf, sayı ve belirli özel karakterlere izin verir.
   * Telefon numarası alanı, girilen değeri otomatik olarak biçimlendirir.
2. **Constructor (CustemerEditDuzenle Metodu):**
   * Formun başlatılması sırasında çağrılır.
   * **\_blUser** değişkeni oluşturulur.
   * Metodun içinde, çeşitli kontrol alanlarının maksimum uzunlukları ve kabul edilebilir karakter türleri ayarlanır.
   * "AcceptButton" özelliği, Enter tuşuna basıldığında çağrılacak olan butonu belirler.
   * İl ve ilçe bilgilerini dolduran **FillIl** metodu çağrılır.
3. **TextBox KeyPress Olayları:**
   * **txtAd**, **txtSoyad** ve **txtTel** TextBox'ları için sadece belirli türde karakter girişine izin verir.
4. **TextBox TextChanged Olayları:**
   * **txtTel** TextBox'ı için, girilen telefon numarasını otomatik olarak biçimlendirir.
5. **KontrolEt ve IsValidEmail Metodları:**
   * **txtMail** TextBox'ındaki e-posta adresinin geçerliliğini kontrol eder.
6. **FillIl ve FillIlce Metodları:**
   * İl ve ilçe ComboBox'larını dolduran metotlardır.
   * **\_blUser** sınıfının **IlSorgu** ve **IlceSorgu** metotları kullanılarak veritabanından il ve ilçe bilgileri alınır.
7. **cmil\_SelectedIndexChanged Metodu:**
   * İl ComboBox'ında bir seçim yapıldığında çağrılır.
   * Seçilen ilin ilçelerini dolduran **FillIlce** metodu çağrılır.
8. **OnVeriSecildi Metodu:**
   * **CustomerEditMenu** formundan seçilen müşteri verilerini bu forma aktaran metodur.
9. **KontrolleriGecerliMi Metodu:**
   * Girilen bilgilerin geçerliliğini kontrol eder.
   * Örneğin, ad ve soyadın minimum uzunlukta olup olmadığını, telefon numarasının uygun formatta olup olmadığını kontrol eder.
10. **ClearForm Metodu:**
    * Formdaki tüm kontrol değerlerini temizler.
11. **btnGuncelle\_Click Metodu:**
    * Güncelleme işlemi için butona tıklandığında çağrılır.
    * **KontrolleriGecerliMi** metodunu kullanarak girilen bilgilerin geçerliliğini kontrol eder.
    * **\_blUser** sınıfının **GuncellemeSorgu** metodunu kullanarak müşteri bilgilerini günceller.
    * Güncelleme işlemi başarılıysa kullanıcıya bir bilgi iletilir, form temizlenir ve **CustomerEditMenu** formunda bulunan DataGridView güncellenir.

using BusinessLayer;

using System;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Employee.MenuLayer.CustomerEditLayer.MenuLayer

{

public partial class CustomerEditEkle : Form

{

BlUser \_blUser;

BlUserLogin \_blUserLogin;

public CustomerEditEkle()

{

InitializeComponent();

\_blUser = new BlUser();

\_blUserLogin = new BlUserLogin();

txtTc.MaxLength = 11;

txtAd.MaxLength = 15;

txtSoyad.MaxLength = 15;

txtTel.MaxLength = 15;

txtmail.MaxLength = 27;

txtAdres.MaxLength = 40;

this.AcceptButton = btnEkle;

FillIl();

}

private void btnEkle\_Click(object sender, EventArgs e)

{

if (!KontrolleriGecerliMi())

{

return;

}

string cinsiyet = cmCins.SelectedItem?.ToString() ?? string.Empty;

string kanGrubu = cmKan.SelectedItem?.ToString() ?? string.Empty;

int ilID = cmil.SelectedIndex >= 0 ? Convert.ToInt32(cmil.SelectedValue) : -1;

int ilceID = cmilce.SelectedIndex >= 0 ? Convert.ToInt32(cmilce.SelectedValue) : -1;

bool insertSonuc = \_blUser.EklemeSorgu(txtTc.Text, txtAd.Text, txtSoyad.Text, cinsiyet, kanGrubu, txtTel.Text, txtmail.Text, txtAdres.Text, ilID, ilceID);

if (insertSonuc)

{

\_blUserLogin.KullaniciAdBelirle(txtTc.Text, txtAd.Text);

MessageBox.Show("Kayıt Başarılı!");

ClearForm();

CustomerEditMenu usrmenu = Application.OpenForms["CustomerEditMenu"] as CustomerEditMenu;

usrmenu.goster();

txtTc.Focus();

}

else

{

MessageBox.Show("Kayıt gerçekleşirken bir hata oluştu.");

}

}

private bool KontrolleriGecerliMi()

{

if (txtTc.Text.Length != 11)

{

MessageBox.Show("Lütfen T.C Kimlik Numaranızı Kontrol Ediniz.");

return false;

}

if (txtAd.Text.Length <= 3 || txtSoyad.Text.Length <= 3)

{

MessageBox.Show("Lütfen Gerçek Adınızı ve Soyadınızı Giriniz.");

return false;

}

if (cmCins.SelectedIndex == -1 || cmKan.SelectedIndex == -1)

{

MessageBox.Show("Lütfen Cinsiyetinizi ve Kan Grubunuzu Seçtiğinizden Emin olunuz.");

return false;

}

if (txtTel.Text.Length != 15)

{

MessageBox.Show("Lütfen Telefon numaranızı uygun formatta giriniz.");

return false;

}

if (KontrolEt() != 1)

{

MessageBox.Show("Lütfen MAİL adresinizi uygun formatta giriniz.");

return false;

}

if (cmil.SelectedIndex == -1 || cmilce.SelectedIndex == -1)

{

MessageBox.Show("Lütfen İl ve İlçe Seçiniz.");

return false;

}

if (txtAdres.Text.Length <= 7)

{

MessageBox.Show("Lütfen Açık Adres Bilginizi Giriniz.");

return false;

}

return true;

}

private void ClearForm()

{

// Formdaki tüm kontrol değerlerini temizleme işlemi

foreach (Control control in Controls)

{

if (control is System.Windows.Forms.TextBox textBox)

{

textBox.Clear();

}

else if (control is System.Windows.Forms.ComboBox comboBox)

{

comboBox.SelectedIndex = -1;

}

}

}

private void txtTc\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsDigit(e.KeyChar) && e.KeyChar != (char)Keys.Back)

{

e.Handled = true;

}

}

private void txtAd\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsLetter(e.KeyChar) && e.KeyChar != (char)Keys.Back && e.KeyChar != (char)Keys.Space)

{

e.Handled = true;

}

}

private void txtSoyad\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsLetter(e.KeyChar) && e.KeyChar != (char)Keys.Back && e.KeyChar != (char)Keys.Space)

{

e.Handled = true;

}

}

private void txtTel\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsDigit(e.KeyChar) && e.KeyChar != (char)Keys.Back)

{

e.Handled = true;

}

}

private void imlec(int position)

{

txtTel.SelectionStart = position;

txtTel.Focus();

}

private int eskiuzunluk = 0;

private void txtTel\_TextChanged(object sender, EventArgs e)

{

string text = txtTel.Text.Replace("(", "").Replace(")", "").Replace(" ", "");

StringBuilder formattedNumber = new StringBuilder();

int yeniuzunluk = text.Length;

if (yeniuzunluk > eskiuzunluk)

{

if (text.Length > 0)

{

formattedNumber.Append("(" + text.Substring(0, Math.Min(3, text.Length)) + ")");

yeniuzunluk = text.Length;

}

if (text.Length > 3)

{

formattedNumber.Append(" " + text.Substring(3, Math.Min(3, text.Length - 3)));

yeniuzunluk = text.Length;

}

if (text.Length > 6)

{

formattedNumber.Append(" " + text.Substring(6, Math.Min(2, text.Length - 6)));

yeniuzunluk = text.Length;

}

if (text.Length > 8)

{

formattedNumber.Append(" " + text.Substring(8, Math.Max(0, text.Length - 8)));

yeniuzunluk = text.Length;

}

txtTel.Text = formattedNumber.ToString();

txtTel.SelectionStart = txtTel.Text.Length;

}

if (yeniuzunluk < eskiuzunluk)

{

if (text.Length == 3)

{

imlec(4);

}

if (text.Length == 0)

{

txtTel.Text = string.Empty;

}

}

eskiuzunluk = yeniuzunluk;

}

private void txtmail\_TextChanged(object sender, EventArgs e)

{

KontrolEt();

}

private int KontrolEt()

{

string email = txtmail.Text;

if (IsValidEmail(email))

{

txtmail.ForeColor = Color.Black;

return 1;

}

else

{

txtmail.ForeColor = Color.Red;

return 0;

}

}

private bool IsValidEmail(string email)

{

string emailPattern = @"^[a-zA-Z0-9.\_-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$";

return System.Text.RegularExpressions.Regex.IsMatch(email, emailPattern);

}

private void FillIl()

{

cmil.DataSource = \_blUser.IlSorgu();

cmil.DisplayMember = "IlAdi";

cmil.ValueMember = "IlID";

cmil.SelectedIndex = -1;

}

private void cmil\_SelectedIndexChanged(object sender, EventArgs e)

{

if (cmil.SelectedValue != null)

{

DataRowView selectedRow = cmil.SelectedItem as DataRowView;

if (selectedRow != null)

{

int selectedIlID = Convert.ToInt32(selectedRow["IlID"]);

FillIlce(selectedIlID);

}

}

}

private void FillIlce(int ilId)

{

cmilce.DataSource = \_blUser.IlceSorgu(ilId);

cmilce.DisplayMember = "IlceAdi";

cmilce.ValueMember = "IlceID";

cmilce.SelectedIndex = -1;

}

}

}

Bu kod, müşteri eklemek amacıyla kullanılan bir formu kontrol eder. Aşağıda, kodun temel mantığını metinsel olarak açıklamalarla birlikte bulabilirsiniz:

1. **Form Tanımlamaları:**
   * **CustomerEditEkle** sınıfı, **Form** sınıfından türetilir.
   * **\_blUser** ve **\_blUserLogin** tipinde iki iş mantığı sınıfının örneğini içerir.
   * Kullanıcı girişi, sadece belirli türde karakterlere izin verir.
   * Telefon numarası alanı, girilen değeri otomatik olarak biçimlendirir.
2. **Constructor (CustomerEditEkle Metodu):**
   * Formun başlatılması sırasında çağrılır.
   * **\_blUser** ve **\_blUserLogin** değişkenleri oluşturulur.
   * Metodun içinde, çeşitli kontrol alanlarının maksimum uzunlukları ve kabul edilebilir karakter türleri ayarlanır.
   * "AcceptButton" özelliği, Enter tuşuna basıldığında çağrılacak olan butonu belirler.
   * İl ve ilçe bilgilerini dolduran **FillIl** metodu çağrılır.
3. **TextBox KeyPress Olayları:**
   * **txtTc**, **txtAd**, **txtSoyad** ve **txtTel** TextBox'ları için sadece belirli türde karakter girişine izin verir.
4. **TextBox TextChanged Olayları:**
   * **txtTel** ve **txtmail** TextBox'ları için, girilen telefon numarasını ve e-posta adresini otomatik olarak biçimlendirir.
5. **KontrolleriGecerliMi Metodu:**
   * Girilen bilgilerin geçerliliğini kontrol eder.
   * Örneğin, T.C. kimlik numarasının uzunluğunu, ad ve soyadın minimum uzunlukta olup olmadığını, cinsiyet ve kan grubu seçilip seçilmediğini kontrol eder.
6. **ClearForm Metodu:**
   * Formdaki tüm kontrol değerlerini temizler.
7. **btnEkle\_Click Metodu:**
   * Ekleme işlemi için butona tıklandığında çağrılır.
   * **KontrolleriGecerliMi** metodunu kullanarak girilen bilgilerin geçerliliğini kontrol eder.
   * **\_blUser** sınıfının **EklemeSorgu** ve **\_blUserLogin** sınıfının **KullaniciAdBelirle** metotlarını kullanarak müşteri bilgilerini ekler ve kullanıcı adını belirler.
   * Ekleme işlemi başarılıysa kullanıcıya bir bilgi iletilir, form temizlenir ve **CustomerEditMenu** formunda bulunan DataGridView güncellenir.
8. **txtTc\_KeyPress Metodu:**
   * T.C. kimlik numarası TextBox'ında sadece sayı girişine izin verir.
9. **FillIl ve FillIlce Metodları:**
   * İl ve ilçe ComboBox'larını dolduran metotlardır.
   * **\_blUser** sınıfının **IlSorgu** ve **IlceSorgu** metotları kullanılarak veritabanından il ve ilçe bilgileri alınır.

using BusinessLayer;

using System;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Employee.MenuLayer.CustomerEditLayer.MenuLayer

{

public partial class CustomerEditLogin : Form

{

int gvnlkkod;

public string Tc\_no;

BlUserLogin \_blUserLogin;

public CustomerEditLogin()

{

InitializeComponent();

txtKAd1.MaxLength = 13;

txtKAd2.MaxLength = 13;

txtSifre1.MaxLength = 13;

txtSifre2.MaxLength = 13;

txtGuvenlik.MaxLength = 6;

\_blUserLogin = new BlUserLogin();

this.AcceptButton = btnGuncelle;

}

private void UserName\_Load(object sender, System.EventArgs e)

{

this.Size = new System.Drawing.Size(310, 105);

GuvenlikKodYenile();

mvctKAd.Visible = false;

mvctParola.Visible = false;

gbKAd.Visible = false;

gbParola.Visible = false;

GuvenlikKod.Visible = false;

txtGuvenlik.Visible = false;

lb5.Visible = false;

btnGuncelle.Visible = false;

}

private void txtGuvenlik\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsDigit(e.KeyChar) && e.KeyChar != (char)Keys.Back)

{

e.Handled = true;

}

}

public void TcAtama(string tc)

{

Tc\_no = tc;

mvctKAd.Text = \_blUserLogin.KAdGetir(Tc\_no);

mvctParola.Text = \_blUserLogin.ParolaGetir(Tc\_no);

}

public void MevcutKAd()

{

mvctKAd.Visible = true;

mvctKAd.Text = \_blUserLogin.KAdGetir(Tc\_no);

}

public void MevcutParola()

{

mvctParola.Visible = true;

mvctParola.Text = \_blUserLogin.ParolaGetir(Tc\_no);

}

private void cbKAd\_CheckedChanged(object sender, EventArgs e)

{

HangisiSecili();

}

private void cbParola\_CheckedChanged(object sender, EventArgs e)

{

HangisiSecili();

}

private void HangisiSecili()

{

if (cbKAd.Checked && !cbParola.Checked)

{

//Sadece kullanıcı adı seçili.

MevcutKAd();

mvctParola.Visible = false;

this.Size = new System.Drawing.Size(310, 300);

gbKAd.Visible = true;

gbParola.Visible = false;

GuvenlikKod.Visible = true;

txtGuvenlik.Visible = true;

lb5.Visible = true;

btnGuncelle.Visible = true;

radioButton1.Visible = true;

radioButton2.Visible = true;

gbKAd.Location = new System.Drawing.Point(25, 70);

GuvenlikKod.Location = new System.Drawing.Point(159, 154);

lb5.Location = new System.Drawing.Point(65, 183);

txtGuvenlik.Location = new System.Drawing.Point(148, 183);

btnGuncelle.Location = new System.Drawing.Point(207, 209);

radioButton1.Location = new System.Drawing.Point(15, 150);

radioButton2.Location = new System.Drawing.Point(15, 172);

}

if (!cbKAd.Checked && cbParola.Checked)

{

//Sadece parola seçili.

MevcutParola();

mvctKAd.Visible = false;

this.Size = new System.Drawing.Size(310, 300);

gbParola.Visible = true;

gbKAd.Visible = false;

GuvenlikKod.Visible = true;

txtGuvenlik.Visible = true;

lb5.Visible = true;

btnGuncelle.Visible = true;

radioButton1.Visible = true;

radioButton2.Visible = true;

gbParola.Location = new System.Drawing.Point(25, 70);

GuvenlikKod.Location = new System.Drawing.Point(159, 154);

lb5.Location = new System.Drawing.Point(65, 183);

txtGuvenlik.Location = new System.Drawing.Point(148, 183);

btnGuncelle.Location = new System.Drawing.Point(207, 209);

radioButton1.Location = new System.Drawing.Point(15, 150);

radioButton2.Location = new System.Drawing.Point(15, 172);

}

if (cbKAd.Checked && cbParola.Checked)

{

//İkiside seçiliyse.

MevcutKAd();

MevcutParola();

this.Size = new System.Drawing.Size(310, 385);

gbKAd.Visible = true;

gbParola.Visible = true;

GuvenlikKod.Visible = true;

txtGuvenlik.Visible = true;

lb5.Visible = true;

btnGuncelle.Visible = true;

radioButton1.Visible = true;

radioButton2.Visible = true;

gbKAd.Location = new System.Drawing.Point(25, 70);

gbParola.Location = new System.Drawing.Point(25, 150);

GuvenlikKod.Location = new System.Drawing.Point(159, 231);

lb5.Location = new System.Drawing.Point(65, 263);

txtGuvenlik.Location = new System.Drawing.Point(148, 263);

btnGuncelle.Location = new System.Drawing.Point(208, 295);

radioButton1.Location = new System.Drawing.Point(26, 295);

radioButton2.Location = new System.Drawing.Point(26, 317);

}

if (!cbKAd.Checked && !cbParola.Checked)

{

mvctKAd.Visible = false;

mvctParola.Visible = false;

this.Size = new System.Drawing.Size(310, 100);

gbKAd.Visible = false;

gbParola.Visible = false;

GuvenlikKod.Visible = false;

txtGuvenlik.Visible = false;

lb5.Visible = false;

btnGuncelle.Visible = false;

radioButton1.Visible = false;

radioButton2.Visible = false;

}

}

private void GuvenlikKodYenile()

{

Random random = new Random();

gvnlkkod = random.Next(10000, 99999);

GuvenlikKod.Text = gvnlkkod.ToString();

}

private string Statu()

{

if (radioButton1.Checked)

{

return "A";

}

if (radioButton2.Checked)

{

return "D";

}

return "A";

}

private bool KontrollerGecerliMi()

{

if (cbKAd.Checked && cbParola.Checked)

{

if (txtKAd1.Text == txtKAd2.Text)

{

if (txtSifre1.Text == txtSifre2.Text)

{

if (txtGuvenlik.Text == GuvenlikKod.Text)

{

string statu = Statu();

bool onay = \_blUserLogin.KullaniciAdSifreDegistir1(Tc\_no, txtKAd1.Text, txtSifre1.Text, statu);

return onay;

}

else

{

MessageBox.Show("Güvenlik kodunu yanlış girdiniz! \nLütfen Güvenlik kodunu kontrol ediniz.");

txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

else if (txtSifre1.Text != txtSifre2.Text && txtGuvenlik.Text != GuvenlikKod.Text)

{

MessageBox.Show("Güvenlik Kodu YANLIŞ! \nŞifreler Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtKAd1.Text = null; txtKAd2.Text = null; txtSifre1.Text = null;

txtSifre2.Text = null; txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

else

{

MessageBox.Show("Şifreler Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtSifre1.Text = null; txtSifre2.Text = null;

txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

else if (txtKAd1.Text != txtKAd2.Text && txtGuvenlik.Text != GuvenlikKod.Text)

{

MessageBox.Show("Güvenlik Kodu YANLIŞ! \nKullanıcı Adları Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtKAd1.Text = null; txtKAd2.Text = null; txtSifre1.Text = null;

txtSifre2.Text = null; txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

else

{

MessageBox.Show("Kullanıcı Adları Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtKAd1.Text = null; txtKAd2.Text = null; txtSifre1.Text = null;

txtSifre2.Text = null; txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

if (!cbKAd.Checked && cbParola.Checked)

{

if (txtSifre1.Text == txtSifre2.Text)

{

if (txtGuvenlik.Text == GuvenlikKod.Text)

{

string statu = Statu();

bool onay = \_blUserLogin.SifreDegistir1(Tc\_no, txtSifre1.Text, statu);

return onay;

}

else

{

MessageBox.Show("Güvenlik kodunu yanlış girdiniz! \nLütfen Güvenlik kodunu kontrol ediniz.");

txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

else if (txtSifre1.Text != txtSifre2.Text && txtGuvenlik.Text != GuvenlikKod.Text)

{

MessageBox.Show("Güvenlik Kodu YANLIŞ! \nŞifreler Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtSifre1.Text = null; txtSifre2.Text = null; txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

else

{

MessageBox.Show("Şifreler Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtSifre1.Text = null; txtSifre2.Text = null;

txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

if (cbKAd.Checked && !cbParola.Checked)

{

if (txtKAd1.Text == txtKAd2.Text)

{

if (txtGuvenlik.Text == GuvenlikKod.Text)

{

string statu = Statu();

bool onay = \_blUserLogin.KullaniciAdDegistir1(Tc\_no, txtKAd1.Text, statu);

return onay;

}

else

{

MessageBox.Show("Güvenlik kodunu yanlış girdiniz! \nLütfen Güvenlik kodunu kontrol ediniz.");

txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

else if (txtKAd1.Text != txtKAd2.Text && txtGuvenlik.Text != GuvenlikKod.Text)

{

MessageBox.Show("Güvenlik Kodu YANLIŞ! \nKullanıcı Adları Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtKAd1.Text = null; txtKAd2.Text = null; txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

else

{

MessageBox.Show("Kullanıcı Adları Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtKAd1.Text = null; txtKAd2.Text = null; txtSifre1.Text = null;

txtSifre2.Text = null; txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

if (!cbKAd.Checked && !cbParola.Checked)

{

return false;

}

else { return false; }

}

private void btnGuncelle\_Click(object sender, EventArgs e)

{

if (KontrollerGecerliMi())

{

//Veri tabanına veri yollanacak ve update edilecek.

MessageBox.Show("Başarılı işlem.");

this.Size = new System.Drawing.Size(310, 105);

GuvenlikKodYenile();

gbKAd.Visible = false; txtKAd1.Clear(); txtKAd2.Clear();

gbParola.Visible = false; txtSifre1.Clear(); txtSifre2.Clear();

GuvenlikKod.Visible = false;

txtGuvenlik.Visible = false; txtGuvenlik.Clear();

lb5.Visible = false;

btnGuncelle.Visible = false;

cbKAd.Checked = false;

cbParola.Checked = false;

}

}

}

}

Bu kod, müşteri veya kullanıcı bilgilerinin güncellendiği bir formu kontrol eder. Aşağıda, kodun temel mantığını metinsel olarak açıklamalarla birlikte bulabilirsiniz:

1. **Form Tanımlamaları:**
   * **CustomerEditLogin** sınıfı, **Form** sınıfından türetilir.
   * **\_blUserLogin** tipinde bir iş mantığı sınıfının örneği içerir.
   * Kullanıcı girişi, sadece belirli türde karakterlere izin verir.
2. **Constructor (CustomerEditLogin Metodu):**
   * Formun başlatılması sırasında çağrılır.
   * Metodun içinde, çeşitli kontrol alanlarının maksimum uzunlukları ayarlanır.
   * "AcceptButton" özelliği, Enter tuşuna basıldığında çağrılacak olan butonu belirler.
3. **Form Yüklendiğinde (UserName\_Load Metodu):**
   * Formun başlatılması ve yüklenmesi sırasında gerçekleşen olayları kontrol eder.
   * Bazı bileşenlerin görünürlüğü ve boyutları başlangıçta ayarlanır.
   * Güvenlik kodu oluşturulur ve bazı kontroller gizlenir.
4. **TextBox KeyPress Olayı (txtGuvenlik\_KeyPress Metodu):**
   * Güvenlik kodu giriş alanında sadece sayı girişine izin verir.
5. **Public Metotlar (TcAtama, MevcutKAd, MevcutParola):**
   * **TcAtama**: Forma bir T.C. kimlik numarası atanır ve bu kimlik numarasına göre mevcut kullanıcı adı ve parola bilgileri getirilir.
   * **MevcutKAd**: Mevcut kullanıcı adını gösteren bir etiket görünür hale getirilir.
   * **MevcutParola**: Mevcut parolayı gösteren bir etiket görünür hale getirilir.
6. **CheckBox Olayları (cbKAd\_CheckedChanged, cbParola\_CheckedChanged):**
   * Kullanıcı, kullanıcı adını veya parolayı güncellemek istediğinde gerçekleşen olayları kontrol eder.
   * Hangi seçeneklerin işaretlendiğine göre bazı kontrollerin görünürlüğünü ayarlar.
7. **HangisiSecili Metodu:**
   * Kullanıcının hangi seçenekleri işaretlediğini kontrol eder ve buna göre formun görünürlüğünü ayarlar.
8. **GuvenlikKodYenile Metodu:**
   * Güvenlik kodunu rastgele bir sayıyla yeniler.
9. **Statu Metodu:**
   * Kullanıcının seçtiği durumu (aktif veya pasif) belirler.
10. **KontrollerGecerliMi Metodu:**
    * Kullanıcının girdiği bilgilerin geçerliliğini kontrol eder.
    * Eğer kullanıcı adı veya parola güncelleme seçenekleri işaretlenmişse, gerekli kontroller yapılır.
    * Geçerliyse, işlemler veritabanına iletilmeye hazır hale getirilir.
11. **btnGuncelle\_Click Metodu:**
    * Güncelleme butonuna tıklandığında çağrılır.
    * **KontrollerGecerliMi** metodu kullanılarak girilen bilgilerin geçerliliği kontrol edilir.
    * Geçerliyse, işlem başarılı mesajı gösterilir ve formun bazı alanları temizlenip boyutu küçültülür.

Formun Üstü

using BusinessLayer;

using System;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using System.Linq;

namespace DernekUyeTakipSistemi.Employee.MenuLayer.CustomerEditLayer.MenuLayer

{

public partial class CustomerEditSearch : Form

{

public CustomerEditSearch()

{

InitializeComponent();

txt1.KeyPress += new KeyPressEventHandler(txt1\_KeyPress);

txt2.KeyPress += new KeyPressEventHandler(txt2\_KeyPress);

}

private void rbTc\_CheckedChanged(object sender, EventArgs e)

{

if (rbTc.Checked)

{

txt1.Text = null;

txt2.Text = null;

lb1.Visible = true;

lb2.Visible = false;

txt1.Visible = true;

txt2.Visible = false;

lb1.Text = "Tc No";

lb2.Text = null;

txt1.MaxLength = 11;

}

}

private void rbAdSoyad\_CheckedChanged(object sender, EventArgs e)

{

if (rbAdSoyad.Checked)

{

txt1.Text = null;

txt2.Text = null;

lb1.Visible = true;

lb2.Visible = true;

txt1.Visible = true;

txt2.Visible = true;

lb1.Text = "Ad";

lb2.Text = "Soyad";

txt1.MaxLength = 15;

txt2.MaxLength = 15;

}

}

private void rbTel\_CheckedChanged(object sender, EventArgs e)

{

if (rbTel.Checked)

{

txt1.Text = null;

txt2.Text = null;

lb1.Visible = true;

lb2.Visible = false;

txt1.Visible = true;

txt2.Visible = false;

lb1.Text = "Tel No";

txt1.MaxLength = 15;

}

}

private void rbMail\_CheckedChanged(object sender, EventArgs e)

{

if (rbMail.Checked)

{

txt1.Text = null;

txt2.Text = null;

lb1.Visible = true;

lb2.Visible = false;

txt1.Visible = true;

txt2.Visible = false;

lb1.Text = "Mail";

txt1.MaxLength = 28;

}

}

private void txt1\_KeyPress(object sender, KeyPressEventArgs e)

{

if (rbTc.Checked || rbTel.Checked)

{

if (!char.IsDigit(e.KeyChar) && e.KeyChar != (char)Keys.Back)

{

e.Handled = true;

}

}

if (rbAdSoyad.Checked)

{

if (!char.IsLetter(e.KeyChar) && e.KeyChar != (char)Keys.Back && e.KeyChar != (char)Keys.Space)

{

e.Handled = true;

}

}

}

private void txt2\_KeyPress(object sender, KeyPressEventArgs e)

{

if (rbAdSoyad.Checked)

{

if (!char.IsLetter(e.KeyChar) && e.KeyChar != (char)Keys.Back && e.KeyChar != (char)Keys.Space)

{

e.Handled = true;

}

}

}

private void imlec(int position)

{

txt1.SelectionStart = position;

txt1.Focus();

}

private int eskiuzunluk = 0;

private int KontrolEt()

{

string email = txt1.Text;

if (IsValidEmail(email))

{

txt1.ForeColor = Color.Black;

return 1;

}

else

{

txt1.ForeColor = Color.Red;

return 0;

}

}

private bool IsValidEmail(string email)

{

string emailPattern = @"^[a-zA-Z0-9.\_-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$";

return System.Text.RegularExpressions.Regex.IsMatch(email, emailPattern);

}

private void txt1\_TextChanged(object sender, EventArgs e)

{

if (rbTel.Checked)

{

string text = txt1.Text.Replace("(", "").Replace(")", "").Replace(" ", "");

StringBuilder formattedNumber = new StringBuilder();

int yeniuzunluk = text.Length;

if (yeniuzunluk > eskiuzunluk)

{

if (text.Length > 0)

{

formattedNumber.Append("(" + text.Substring(0, Math.Min(3, text.Length)) + ")");

yeniuzunluk = text.Length;

}

if (text.Length > 3)

{

formattedNumber.Append(" " + text.Substring(3, Math.Min(3, text.Length - 3)));

yeniuzunluk = text.Length;

}

if (text.Length > 6)

{

formattedNumber.Append(" " + text.Substring(6, Math.Min(2, text.Length - 6)));

yeniuzunluk = text.Length;

}

if (text.Length > 8)

{

formattedNumber.Append(" " + text.Substring(8, Math.Max(0, text.Length - 8)));

yeniuzunluk = text.Length;

}

txt1.Text = formattedNumber.ToString();

txt1.SelectionStart = txt1.Text.Length;

}

if (yeniuzunluk < eskiuzunluk)

{

if (text.Length == 3)

{

imlec(4);

}

if (text.Length == 0)

{

txt1.Text = string.Empty;

}

}

eskiuzunluk = yeniuzunluk;

}

if (rbMail.Checked)

{

KontrolEt();

}

}

private void btnSearch\_Click(object sender, EventArgs e)

{

if (rbTc.Checked)

{

CustomerEditMenu CustomerEditMenu = Application.OpenForms["CustomerEditMenu"] as CustomerEditMenu;

CustomerEditMenu.TcArama(txt1.Text);

this.Close();

}

if (rbAdSoyad.Checked)

{

CustomerEditMenu CustomerEditMenu = Application.OpenForms["CustomerEditMenu"] as CustomerEditMenu;

CustomerEditMenu.AdSoyadArama(txt1.Text, txt2.Text);

this.Close();

}

if (rbTel.Checked)

{

CustomerEditMenu CustomerEditMenu = Application.OpenForms["CustomerEditMenu"] as CustomerEditMenu;

CustomerEditMenu.TelArama(txt1.Text);

this.Close();

}

if (rbMail.Checked)

{

CustomerEditMenu CustomerEditMenu = Application.OpenForms["CustomerEditMenu"] as CustomerEditMenu;

CustomerEditMenu.MailArama(txt1.Text);

this.Close();

}

}

}

}

Bu kod, müşteri arama işlemlerini gerçekleştirmek için kullanılan bir formu kontrol eder. Aşağıda, kodun temel mantığını metinsel olarak açıklamalarla birlikte bulabilirsiniz:

1. **Form Tanımlamaları:**
   * **CustomerEditSearch** sınıfı, **Form** sınıfından türetilir.
   * TextBox girişlerinin düzenlenmesi için olaylar ve metotlar içerir.
2. **Constructor (CustomerEditSearch Metodu):**
   * Formun başlatılması sırasında çağrılır.
   * TextBox'lara özel KeyPress olayları atanır.
3. **RadioButton Olayları (rbTc\_CheckedChanged, rbAdSoyad\_CheckedChanged, rbTel\_CheckedChanged, rbMail\_CheckedChanged):**
   * Hangi türde arama yapılacağını belirleyen RadioButton'lar için olaylar.
   * Seçilen seçeneğe göre TextBox'ların görünürlüğü ve diğer bazı özellikler ayarlanır.
4. **KeyPress Olayları (txt1\_KeyPress, txt2\_KeyPress):**
   * TextBox'lara sadece belirli karakterlerin girilmesine izin veren olaylar.
   * **txt1** için sayı veya harf, **txt2** için sadece harf ve boşluk.
5. **Metotlar (imlec, KontrolEt, IsValidEmail):**
   * **imlec**: TextBox içindeki imleci belirtilen konuma taşıyan metot.
   * **KontrolEt**: Girilen metnin e-posta formatına uygun olup olmadığını kontrol eden metot.
   * **IsValidEmail**: E-posta geçerliliğini kontrol eden yardımcı metot.
6. **TextChanged Olayı (txt1\_TextChanged):**
   * Telefon numarası girildiğinde otomatik olarak biçimlendiren olay.
   * Girilen e-posta adresini kontrol etmek için **KontrolEt** metodu çağrılır.
7. **Button Click Olayı (btnSearch\_Click):**
   * "Ara" butonuna tıklandığında çağrılır.
   * Seçilen arama türüne göre ilgili metotlar çağrılarak arama işlemi gerçekleştirilir.
   * **CustomerEditMenu** formundaki ilgili metotlar çağrılır ve arama sonuçları işlenir.
   * Form kapatılır.

Formun Üstü

using BusinessLayer;

using System;

using System.Data;

using System.Drawing;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Employee.MenuLayer.DuesLayer.MenuLayer

{

public partial class PaymentDues : Form

{

BlAidat \_blaidat;

public int \_AidatID;

private DataView NoView;

private DataView OnView;

private DataView TumView;

private DataTable sorgu1;

private DataTable sorgu2;

private DataTable sorgu3;

private DataTable sorgu4;

private DataTable sorgu5;

private DataTable sorgu6;

public PaymentDues()

{

InitializeComponent();

\_blaidat = new BlAidat();

txtTc.MaxLength = 11;

}

private void AidatOdeme\_Load(object sender, EventArgs e)

{

rbTum.Visible = false;

rbOk.Visible = false;

rbNo.Visible = false;

rbTum.Checked = false;

dataGridView1.AutoResizeColumns();

}

private void Secildi()

{

if (!cbTum.Checked && !cbBireysel.Checked)

{

lbTc.Visible = false; txtTc.Visible = false; gbOdeme.Visible = false;

}

}

private void cbTum\_CheckedChanged(object sender, EventArgs e)

{

if (cbTum.Checked)

{

rbTum.Visible = true;

rbOk.Visible = true;

rbNo.Visible = true;

rbTum.Checked = true;

cbBireysel.Checked = false;

lbTc.Visible = false; txtTc.Visible = false;

gbOdeme.Visible = true;

}

if (!cbTum.Checked)

{

rbTum.Visible = false;

rbOk.Visible = false;

rbNo.Visible = false;

rbTum.Checked = false;

cmOdemeYil.Text = null;

cmOdemeAy.Text = null;

}

Secildi();

}

private void cbBireysel\_CheckedChanged(object sender, EventArgs e)

{

if (cbBireysel.Checked)

{

cbTum.Checked = false;

lbTc.Visible = true; txtTc.Visible = true;

if (txtTc.Text.Length == 11)

{

gbOdeme.Visible = true;

}

else

{

cmOdemeYil.Text = null;

cmOdemeAy.Text = null;

gbOdeme.Visible = false;

}

}

if (!cbTum.Checked)

{

txtTc.Clear();

cmOdemeYil.Text = null;

cmOdemeAy.Text = null;

}

Secildi();

}

private void txtTc\_TextChanged(object sender, EventArgs e)

{

if (txtTc.Text.Length == 11)

{

rbTum.Visible = true;

rbOk.Visible = true;

rbNo.Visible = true;

rbTum.Checked = true;

gbOdeme.Visible = true;

}

else

{

rbTum.Visible = false;

rbOk.Visible = false;

rbNo.Visible = false;

rbTum.Checked = false;

gbOdeme.Visible = false;

}

}

private void txtTc\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsDigit(e.KeyChar) && e.KeyChar != (char)Keys.Back)

{

e.Handled = true;

}

}

private void btnListele\_Click(object sender, EventArgs e)

{

if (cbBireysel.Checked && cmOdemeYil.SelectedItem == null && cmOdemeAy.SelectedItem == null)

{

string Tc\_No = txtTc.Text;

sorgu1 = \_blaidat.AidatOdeme1(Tc\_No);

dataGridView1.DataSource = sorgu1;

}

if (cbBireysel.Checked && cmOdemeYil.SelectedItem != null && cmOdemeAy.SelectedItem == null)

{

// Kişinin tc no ve yıla göre tüm aidatları.

string Tc\_No = txtTc.Text;

string Yil = cmOdemeYil.Text;

sorgu2 = \_blaidat.AidatOdeme2(Tc\_No, Yil);

dataGridView1.DataSource = sorgu2;

}

if (cbBireysel.Checked && cmOdemeYil.SelectedItem != null && cmOdemeAy.SelectedItem != null)

{

// Kişinin Tc no, yil ve aya göre tüm aidatları

string Tc\_No = txtTc.Text;

string Yil = cmOdemeYil.Text;

string secilenAy = cmOdemeAy.SelectedItem.ToString();

int ayNumarasi = cmOdemeAy.Items.IndexOf(secilenAy) + 1;

string Ay = ayNumarasi < 10 ? "0" + ayNumarasi.ToString() : ayNumarasi.ToString();

sorgu3 = \_blaidat.AidatOdeme3(Tc\_No, Yil, Ay);

dataGridView1.DataSource = sorgu3;

}

if (cbTum.Checked && cmOdemeYil.SelectedItem == null && cmOdemeAy.SelectedItem == null)

{

// Herkesin tüm aidatları.

sorgu4 = \_blaidat.AidatOdeme4();

dataGridView1.DataSource = sorgu4;

}

if (cbTum.Checked && cmOdemeYil.SelectedItem != null && cmOdemeAy.SelectedItem == null)

{

// Herkesin yila göre tüm aidatları.

string Yil = cmOdemeYil.Text;

sorgu5 = \_blaidat.AidatOdeme5(Yil);

dataGridView1.DataSource = sorgu5;

}

if (cbTum.Checked && cmOdemeYil.SelectedItem != null && cmOdemeAy.SelectedItem != null)

{

// Herkesin yila ve aya göre tüm aidatları.

string Yil = cmOdemeYil.Text;

string secilenAy = cmOdemeAy.SelectedItem.ToString();

int ayNumarasi = cmOdemeAy.Items.IndexOf(secilenAy) + 1;

string Ay = ayNumarasi < 10 ? "0" + ayNumarasi.ToString() : ayNumarasi.ToString();

sorgu6 = \_blaidat.AidatOdeme6(Yil, Ay);

dataGridView1.DataSource = sorgu6;

}

}

Bu kod, aidat ödeme işlemlerini yöneten bir formu kontrol eder. Aşağıda, kodun temel mantığını metinsel olarak açıklamalarla birlikte bulabilirsiniz:

1. **Form Tanımlamaları:**
   * **PaymentDues** sınıfı, **Form** sınıfından türetilir.
   * **\_blaidat** adında bir **BlAidat** nesnesi tanımlanır. Bu nesne, iş mantığı işlemlerini gerçekleştiren sınıfa aittir.
   * Çeşitli DataView ve DataTable nesneleri tanımlanır.
2. **Constructor (PaymentDues Metodu):**
   * Formun başlatılması sırasında çağrılır.
   * **\_blaidat** nesnesi oluşturulur ve **txtTc** TextBox'unun maksimum uzunluğu 11 olarak ayarlanır.
3. **Form Load Olayı (AidatOdeme\_Load):**
   * Formun yüklenmesi sırasında çağrılır.
   * Bazı kontrollerin görünürlüğü ayarlanır.
4. **RadioButton Olayları (cbTum\_CheckedChanged, cbBireysel\_CheckedChanged):**
   * Kullanıcı, "Tüm" ya da "Bireysel" ödeme seçeneklerinden birini seçer.
   * Seçilen seçeneklere göre bazı kontrollerin görünürlüğü ayarlanır.
5. **TextBox Olayları (txtTc\_TextChanged, txtTc\_KeyPress):**
   * **txtTc** TextBox'unun metin değişikliği ve tuş basma olayları.
   * Girilen Tc kimlik numarasının uzunluğuna göre bazı kontrollerin görünürlüğü ayarlanır ve sınırlamalar eklenir.
6. **Button Click Olayı (btnListele\_Click):**
   * "Listele" butonuna tıklandığında çağrılır.
   * Seçilen seçeneklere göre aidat ödeme sorguları yapılır ve sonuçlar **dataGridView1** datagridview kontrolüne yüklenir.
7. **Aidat Ödeme Sorguları:**
   * **\_blaidat** sınıfındaki çeşitli metotlar kullanılarak aidat ödeme sorguları gerçekleştirilir.
   * Sorgu sonuçları, DataGridView kontrolüne atanır ve kullanıcıya gösterilir.

private void btnTemizle\_Click(object sender, EventArgs e)

{

cmOdemeAy.SelectedItem = null;

cmOdemeYil.SelectedItem = null;

cmOdemeAy.SelectedIndex = -1;

cmOdemeYil.SelectedIndex = -1;

sorgu1 = null;

sorgu2 = null;

sorgu3 = null;

sorgu4 = null;

sorgu5 = null;

sorgu6 = null;

}

private void NoFilter(DataTable sorgu)

{

NoView = new DataView(sorgu);

NoView.RowFilter = "OdemeDurum=false";

dataGridView1.DataSource = NoView;

dataGridView1.AutoResizeColumns();

}

private void OnFilter(DataTable sorgu)

{

OnView = new DataView(sorgu);

OnView.RowFilter = "OdemeDurum=true";

dataGridView1.DataSource = OnView;

dataGridView1.AutoResizeColumns();

}

private void TumFilter(DataTable sorgu)

{

TumView = new DataView(sorgu);

dataGridView1.DataSource = TumView;

dataGridView1.AutoResizeColumns();

}

private void rbOk\_CheckedChanged(object sender, EventArgs e)

{

if (sorgu1 != null) OnFilter(sorgu1);

if (sorgu2 != null) OnFilter(sorgu2);

if (sorgu3 != null) OnFilter(sorgu3);

if (sorgu4 != null) OnFilter(sorgu4);

if (sorgu5 != null) OnFilter(sorgu5);

if (sorgu6 != null) OnFilter(sorgu6);

}

private void rbNo\_CheckedChanged(object sender, EventArgs e)

{

if (sorgu1 != null) NoFilter(sorgu1);

if (sorgu2 != null) NoFilter(sorgu2);

if (sorgu3 != null) NoFilter(sorgu3);

if (sorgu4 != null) NoFilter(sorgu4);

if (sorgu5 != null) NoFilter(sorgu5);

if (sorgu6 != null) NoFilter(sorgu6);

}

private void rbTum\_CheckedChanged(object sender, EventArgs e)

{

if (sorgu1 != null) TumFilter(sorgu1);

if (sorgu2 != null) TumFilter(sorgu2);

if (sorgu3 != null) TumFilter(sorgu3);

if (sorgu4 != null) TumFilter(sorgu4);

if (sorgu5 != null) TumFilter(sorgu5);

if (sorgu6 != null) TumFilter(sorgu6);

}

private void dataGridView1\_CellFormatting(object sender, DataGridViewCellFormattingEventArgs e)

{

if (e.RowIndex >= 0)

{

// Örneğin, SonOdemeTarih sütunu 5. sütun, OdemeDurum sütunu ise 3. sütun ise

int sonOdemeTarihIndex = dataGridView1.Columns["SonOdemeTarih"].Index;

int odemeDurumIndex = dataGridView1.Columns["OdemeDurum"].Index;

if (e.ColumnIndex == sonOdemeTarihIndex || e.ColumnIndex == odemeDurumIndex)

{

// Son ödeme tarihi değerini al

DateTime sonOdemeTarih = Convert.ToDateTime(dataGridView1.Rows[e.RowIndex].Cells[sonOdemeTarihIndex].Value);

// Odeme durumu değerini al

bool odemeDurum = Convert.ToBoolean(dataGridView1.Rows[e.RowIndex].Cells[odemeDurumIndex].Value);

// Günümüz tarihini al

DateTime simdikiTarih = DateTime.Now;

// Eğer son ödeme tarihi geçmiş ve ödeme durumu false ise, satırın arka plan rengini kırmızı yap

if (sonOdemeTarih < simdikiTarih && !odemeDurum)

{

dataGridView1.Rows[e.RowIndex].DefaultCellStyle.BackColor = Color.Red;

}

// Eğer ödeme durumu true ise, satırın arka plan rengini yeşil yap

if (odemeDurum)

{

dataGridView1.Rows[e.RowIndex].DefaultCellStyle.BackColor = Color.Green;

}

// Eğer son ödeme tarihine 5 gün kala ise, satırın arka plan rengini sarı yap

if(sonOdemeTarih >= simdikiTarih && !odemeDurum)

{

dataGridView1.Rows[e.RowIndex].DefaultCellStyle.BackColor = Color.White;

}

}

}

}

}

}

Bu kod, aidat ödeme işlemlerini içeren bir Windows Forms uygulaması formunun bazı ek işlevselliğini açıklar. İşte kodun temel mantığı:

1. **Temizle Butonu (btnTemizle\_Click):**
   * Kullanıcı, "Temizle" butonuna tıkladığında çağrılır.
   * ComboBox'ların seçili öğeleri ve DataTable'ları temizler.
2. **Veri Filtreleme Metodları (NoFilter, OnFilter, TumFilter):**
   * Veri görünümünü filtrelemek için kullanılır.
   * **NoFilter**: Ödeme durumu false olanları filtreler.
   * **OnFilter**: Ödeme durumu true olanları filtreler.
   * **TumFilter**: Tüm veriyi gösterir.
3. **RadioButton Olayları (rbOk\_CheckedChanged, rbNo\_CheckedChanged, rbTum\_CheckedChanged):**
   * Kullanıcı, ödeme durumuna göre filtreleme yapmak için bu radio butonları kullanır.
   * Seçilen duruma göre uygun filtreleme metodu çağrılır.
4. **DataGridView Hücre Biçimlendirme Olayı (dataGridView1\_CellFormatting):**
   * DataGridView'deki hücreleri biçimlendirmek için kullanılır.
   * Özellikle "SonOdemeTarih" ve "OdemeDurum" sütunlarını kontrol eder.
   * Ödeme tarihi geçmiş ve ödeme durumu false ise hücre arka planını kırmızı yapar.
   * Ödeme durumu true ise hücre arka planını yeşil yapar.
   * Eğer ödeme tarihine 5 gün kala ise hücre arka planını sarı yapar.

Bu şekilde, kullanıcıya ödeme durumlarına göre filtreleme yapma ve DataGridView'de belirli durumlarda renkleri değiştirme yetenekleri sunulmaktadır.

Formun Üstü

using BusinessLayer;

using System;

using System.Collections.Generic;

using System.Data;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Employee.MenuLayer.DuesLayer.MenuLayer

{

public partial class SetDues : Form

{

BlAidat \_blAidat;

public SetDues()

{

InitializeComponent();

\_blAidat = new BlAidat();

}

private void AidatBelirle\_Load(object sender, EventArgs e)

{

goster();

this.Height = eskiUzunluk;

}

public void goster()

{

tablo.DataSource = \_blAidat.TabloSorgu();

tablo.AutoResizeColumns();

}

private void cbEkle\_CheckedChanged(object sender, EventArgs e)

{

Secildi();

}

private void cbDuzenle\_CheckedChanged(object sender, EventArgs e)

{

Secildi();

}

private void cbSil\_CheckedChanged(object sender, EventArgs e)

{

Secildi();

}

public int eskiUzunluk = 300; public int yeniUzunluk = 515;

private void Secildi()

{

if (cbEkle.Checked && !cbDuzenle.Checked && !cbSil.Checked)

{

gbEkle.Visible = true;

gbDuzenle.Visible = false;

gbSil.Visible = false;

this.Height = yeniUzunluk;

}

if (!cbEkle.Checked && cbDuzenle.Checked && !cbSil.Checked)

{

gbEkle.Visible = false;

gbDuzenle.Visible = true;

gbSil.Visible = false;

this.Height = yeniUzunluk;

}

if (!cbEkle.Checked && !cbDuzenle.Checked && cbSil.Checked)

{

gbEkle.Visible = false;

gbDuzenle.Visible = false;

gbSil.Visible = true;

this.Height = yeniUzunluk;

}

if (cbEkle.Checked && cbDuzenle.Checked && !cbSil.Checked)

{

gbEkle.Visible = true;

gbDuzenle.Visible = true;

gbSil.Visible = false;

this.Height = yeniUzunluk;

}

if (cbEkle.Checked && !cbDuzenle.Checked && cbSil.Checked)

{

gbEkle.Visible = true;

gbDuzenle.Visible = false;

gbSil.Visible = true;

this.Height = yeniUzunluk;

}

if (!cbEkle.Checked && cbDuzenle.Checked && cbSil.Checked)

{

gbEkle.Visible = false;

gbDuzenle.Visible = true;

gbSil.Visible = true;

this.Height = yeniUzunluk;

}

if (cbEkle.Checked && cbDuzenle.Checked && cbSil.Checked)

{

gbEkle.Visible = true;

gbDuzenle.Visible = true;

gbSil.Visible = true;

this.Height = yeniUzunluk;

}

if (!cbEkle.Checked && !cbDuzenle.Checked && !cbSil.Checked)

{

gbEkle.Visible = false;

gbDuzenle.Visible = false;

gbSil.Visible = false;

this.Height = eskiUzunluk;

}

}

public int kimlik;

private void btnEkle\_Click(object sender, EventArgs e)

{

if (cmEkleYil != null && cmEkleAy != null && txtEkleTutar != null)

{

string yil = cmEkleYil.Text;

string ay = cmEkleAy.SelectedItem.ToString();

int tutar;

if (txtEkleTutar.Text != "")

{

tutar = Convert.ToInt32(txtEkleTutar.Text);

}

else

{

MessageBox.Show("Lütfen tutar bilgisi giriniz.");

return;

}

string yilNumarasi = yil.Substring(2);

int ayNumarasi = cmEkleAy.Items.IndexOf(ay) + 1;

string duzenlenmisAyNumarasi = ayNumarasi < 10 ? "0" + ayNumarasi.ToString() : ayNumarasi.ToString();

string ayyil = yilNumarasi + duzenlenmisAyNumarasi;

kimlik = Convert.ToInt32(ayyil);

bool insertsonuc = \_blAidat.EklemeSorgu(kimlik, yil, duzenlenmisAyNumarasi, tutar);

if (insertsonuc)

{

MessageBox.Show("Kayıt Başarılı !");

goster();

BorcAtama(tutar);

}

else

{

MessageBox.Show("Kayıt Başarısız !!!");

}

}

else if (cmEkleYil == null)

{

MessageBox.Show("Lütfen Yıl Seçiniz.");

}

else if (cmEkleAy == null)

{

MessageBox.Show("Lütfen Ay Seçiniz.");

}

else if (txtEkleTutar == null)

{

MessageBox.Show("Lütfen tutar Giriniz.");

}

else

{

MessageBox.Show("Belirlenemeyen bir hata oluştu.");

}

}

private void btnDuzenle\_Click(object sender, EventArgs e)

{

if (txtDuzenleTutar.Text != null)

{

string yil = cmDuzenleYil.Text;

string ay = cmDuzenleAy.SelectedItem.ToString();

int tutar = Convert.ToInt32(txtDuzenleTutar.Text);

string yilNumarasi = yil.Substring(2);

int ayNumarasi = cmDuzenleAy.Items.IndexOf(ay) + 1;

string duzenlenmisAyNumarasi = ayNumarasi < 10 ? "0" + ayNumarasi.ToString() : ayNumarasi.ToString();

string ayyil = yilNumarasi + duzenlenmisAyNumarasi;

kimlik = Convert.ToInt32(ayyil);

bool updatesonuc = \_blAidat.GuncellemeSorgu(kimlik, yil, duzenlenmisAyNumarasi, tutar);

if (updatesonuc)

{

MessageBox.Show("Güncelleme Başarılı.");

goster();

BorcGuncelleme(tutar);

}

else

{

MessageBox.Show("Güncelleme Başarısız.");

}

}

else

{

MessageBox.Show("Lütfen tutar giriniz.");

}

}

private void btnSil\_Click(object sender, EventArgs e)

{

if (cmSilYil != null && cmSilAy != null)

{

string yil = cmSilYil.Text;

string ay = cmSilAy.SelectedItem.ToString();

string yilNumarasi = yil.Substring(2);

int ayNumarasi = cmSilAy.Items.IndexOf(ay) + 1;

string duzenlenmisAyNumarasi = ayNumarasi < 10 ? "0" + ayNumarasi.ToString() : ayNumarasi.ToString();

string ayyil = yilNumarasi + duzenlenmisAyNumarasi;

kimlik = Convert.ToInt32(ayyil);

bool insertsonuc = \_blAidat.SilmeSorgu(kimlik);

if (insertsonuc)

{

MessageBox.Show("Silme Başarılı !");

goster();

}

else

{

MessageBox.Show("Silme Başarısız !!!");

}

}

else

{

MessageBox.Show("Belirlenemeyen bir hata oluştu.");

}

}

public void BorcAtama(int tutar)

{

DataTable dataTable = new DataTable();

dataTable = \_blAidat.TcSorgu();

if (dataTable.Columns.Contains("Tc\_No") && dataTable.Columns["Tc\_No"].DataType == typeof(string))

{

List<string> Tcler = new List<string>();

foreach (DataRow row in dataTable.Rows)

{

string value = row.Field<string>("Tc\_No");

Tcler.Add(value);

}

foreach (string Tc\_No in Tcler)

{

\_blAidat.BorcEkle(Tc\_No, kimlik, tutar);

}

}

}

public void BorcGuncelleme(int tutar)

{

DataTable dataTable = new DataTable();

dataTable = \_blAidat.TcSorgu();

if (dataTable.Columns.Contains("Tc\_No") && dataTable.Columns["Tc\_No"].DataType == typeof(string))

{

List<string> Tcler2 = new List<string>();

foreach (DataRow row in dataTable.Rows)

{

string value = row.Field<string>("Tc\_No");

Tcler2.Add(value);

}

foreach (string Tc\_No in Tcler2)

{

\_blAidat.BorcGuncelle(Tc\_No, kimlik, tutar);

}

}

}

public string SecilenID; public string SecilenYil; public string SecilenAy; public string SecilenTutar;

private void tablo\_CellClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex >= 0)

{

DataGridViewRow selectedRow = tablo.Rows[e.RowIndex];

if (tablo.Columns.Contains("AidatID"))

{

int aidatIDColumnIndex = tablo.Columns["AidatID"].Index;

object aidatIDValue = selectedRow.Cells[aidatIDColumnIndex].Value;

int aidatYilColumnIndex = tablo.Columns["AidatYil"].Index;

object aidatYilValue = selectedRow.Cells[aidatYilColumnIndex].Value;

int aidatAyColumnIndex = tablo.Columns["AidatAy"].Index;

object aidatAyValue = selectedRow.Cells[aidatAyColumnIndex].Value;

int aidatTutarColumnIndex = tablo.Columns["AidatTutar"].Index;

object aidatTutarValue = selectedRow.Cells[aidatTutarColumnIndex].Value;

SecilenID = aidatIDValue.ToString();

SecilenYil = aidatYilValue.ToString();

SecilenAy = aidatAyValue.ToString();

SecilenTutar = aidatTutarValue.ToString();

int ay = Convert.ToInt32(SecilenAy) - 1;

//SİLME İŞLEMİ İÇİN GEREKLİ ATAMALAR.

cmSilYil.Text = SecilenYil;

cmSilAy.SelectedIndex = ay;

//GUNCELLEME İÇİN GEREKLİ ATAMALAR.

cmDuzenleYil.Text = SecilenYil;

cmDuzenleAy.SelectedIndex = ay;

txtDuzenleTutar.Text = SecilenTutar;

}

}

}

}

}

Bu kod, aidat işlemlerini gerçekleştiren bir Windows Forms uygulaması formunu açıklar. İşte kodun temel mantığı:

1. **Form Yükleme (AidatBelirle\_Load):**
   * Form yüklendiğinde çağrılır.
   * **goster** fonksiyonu ile DataGridView'e aidat verileri yüklenir.
2. **CheckBox Olayları (cbEkle\_CheckedChanged, cbDuzenle\_CheckedChanged, cbSil\_CheckedChanged):**
   * Kullanıcı, eklemek, düzenlemek veya silmek istediğini seçer.
   * Seçime göre ilgili grup kutuları (**gbEkle**, **gbDuzenle**, **gbSil**) görünür veya görünmez hale gelir.
   * Form yüksekliği (**this.Height**) duruma göre ayarlanır.
3. **Veri Gösterme (goster fonksiyonu):**
   * **\_blAidat** nesnesi üzerinden aidat tablosu sorgulanır ve DataGridView'e yüklenir.
4. **Ekleme, Düzenleme ve Silme İşlemleri:**
   * Kullanıcı, Ekle, Düzenle veya Sil butonlarına tıkladığında çağrılır.
   * Ekleme: Yeni bir aidat kaydı oluşturur. **BorcAtama** fonksiyonu ile borçlar güncellenir.
   * Düzenleme: Var olan bir aidat kaydını günceller. **BorcGuncelleme** fonksiyonu ile borçlar güncellenir.
   * Silme: Bir aidat kaydını siler.
5. **Borç Atama (BorcAtama fonksiyonu):**
   * Veritabanındaki tüm üyeler için, yeni eklenen aidatın bilgileriyle bir borç kaydı oluşturur.
6. **Borç Güncelleme (BorcGuncelleme fonksiyonu):**
   * Veritabanındaki tüm üyeler için, güncellenen aidatın bilgileriyle borç kaydını günceller.
7. **DataGridView Hücre Tıklama Olayı (tablo\_CellClick):**
   * DataGridView'de bir hücreye tıklandığında çağrılır.
   * Seçilen hücrenin değerleri, düzenleme veya silme işlemleri için ilgili alanlara atanır.

Bu şekilde, kullanıcı aidat ekleyebilir, düzenleyebilir, silebilir ve bu işlemler sonucunda borçlar güncellenir. Formun yüksekliği, kullanıcının yapmak istediği işleme göre dinamik olarak değişir.

using DernekUyeTakipSistemi.Admin;

using DernekUyeTakipSistemi.Employee.MenuLayer.DuesLayer.MenuLayer;

using System;

using System.Data;

using System.Linq;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Employee

{

public partial class DuesMenu : Form

{

private bool uygulamaKapatildi = false;

public DuesMenu()

{

InitializeComponent();

}

private void btnGeri\_Click(object sender, EventArgs e)

{

uygulamaKapatildi = true;

FormlariKapa();

AMenu aMenu = Application.OpenForms["AMenu"] as AMenu;

EMenu eMenu = Application.OpenForms["EMenu"] as EMenu;

if (aMenu != null)

{

this.Close();

aMenu.Show();

}

if (eMenu != null)

{

this.Close();

eMenu.Show();

}

}

private void FormlariKapa()

{

foreach (Form form in Application.OpenForms.Cast<Form>().ToList())

{

if (form.Name == "DuesMenu" || form.Name == "SetDues" || form.Name == "PaymentDues")

{

form.Close();

}

}

}

private void btnBelirle\_Click(object sender, EventArgs e)

{

if (Application.OpenForms["SetDues"] == null)

{

SetDues aidatBelirle = new SetDues();

aidatBelirle.Show();

}

}

private void btnOdeme\_Click(object sender, EventArgs e)

{

if (Application.OpenForms["PaymentDues"] == null)

{

PaymentDues aidatOdeme = new PaymentDues();

aidatOdeme.Show();

}

}

private void DuesMenu\_FormClosing(object sender, FormClosingEventArgs e)

{

if (!uygulamaKapatildi)

{

DialogResult cikis = MessageBox.Show("Çıkış yapmak istediğinize emin misiniz ?", "", MessageBoxButtons.YesNo);

if (cikis == DialogResult.Yes)

{

uygulamaKapatildi = true;

Application.Exit();

}

else

{

e.Cancel = true;

}

}

}

}

}

Bu kod, "Aidat Menüsü" adlı bir Windows Forms uygulamasını temsil eder ve aşağıdaki temel özelliklere sahiptir:

1. **Geri Dönme (btnGeri\_Click):**
   * Kullanıcı, "Geri" butonuna tıkladığında bu olay tetiklenir.
   * Açık olan "DuesMenu", "SetDues" ve "PaymentDues" formları kapatılır.
   * Kullanıcı daha önce açılmış olan "AMenu" veya "EMenu" formuna geri dönebilir.
2. **Formları Kapatma (FormlariKapa fonksiyonu):**
   * Açık olan formların isimleri kontrol edilerek, "DuesMenu", "SetDues" ve "PaymentDues" formları kapatılır.
3. **Aidat Belirleme (btnBelirle\_Click):**
   * "Belirle" butonuna tıklandığında, "SetDues" formu açılır. Ancak, bu form zaten açıksa tekrar açılmaz.
4. **Aidat Ödeme (btnOdeme\_Click):**
   * "Ödeme" butonuna tıklandığında, "PaymentDues" formu açılır. Ancak, bu form zaten açıksa tekrar açılmaz.
5. **Form Kapatılma Olayı (DuesMenu\_FormClosing):**
   * Form kapatılmak istendiğinde bu olay tetiklenir.
   * Kullanıcıya çıkış yapmak isteyip istemediği sorulur.
   * Eğer "Evet" (Yes) seçeneği seçilirse, uygulama kapatılır. Aksi takdirde, form kapatılmaz.

Bu kod, bir aidat takip sistemine ait menü arayüzünün temel işlevselliğini sağlar. Kullanıcı aidat belirleme ve ödeme işlemlerini gerçekleştirebilir, menüler arasında geçiş yapabilir ve uygulamayı güvenli bir şekilde kapatabilir.

using BusinessLayer;

using System;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Employee.MenuLayer.EmployeeInfoLayer.MenuLayer

{

public partial class EmployeeInfoLogin : Form

{

int gvnlkkod;

public string Tc\_no;

BlUserLogin \_blUserLogin;

public EmployeeInfoLogin()

{

InitializeComponent();

txtKAd1.MaxLength = 13;

txtKAd2.MaxLength = 13;

txtSifre1.MaxLength = 13;

txtSifre2.MaxLength = 13;

txtGuvenlik.MaxLength = 6;

\_blUserLogin = new BlUserLogin();

this.AcceptButton = btnGuncelle;

}

private void UserName\_Load(object sender, System.EventArgs e)

{

this.Size = new System.Drawing.Size(310, 105);

GuvenlikKodYenile();

mvctKAd.Visible = false;

mvctParola.Visible = false;

gbKAd.Visible = false;

gbParola.Visible = false;

GuvenlikKod.Visible = false;

txtGuvenlik.Visible = false;

lb5.Visible = false;

btnGuncelle.Visible = false;

}

private void txtGuvenlik\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsDigit(e.KeyChar) && e.KeyChar != (char)Keys.Back)

{

e.Handled = true;

}

}

private void GuvenlikKodYenile()

{

Random random = new Random();

gvnlkkod = random.Next(10000, 99999);

GuvenlikKod.Text = gvnlkkod.ToString();

}

public void TcAtama(string tc)

{

Tc\_no = tc;

}

public void MevcutKAd()

{

mvctKAd.Visible = true;

mvctKAd.Text = \_blUserLogin.KAdGetir(Tc\_no);

}

public void MevcutParola()

{

mvctParola.Visible = true;

mvctParola.Text = \_blUserLogin.ParolaGetir(Tc\_no);

}

private void HangisiSecili()

{

if (cbKAd.Checked && !cbParola.Checked)

{

//Sadece kullanıcı adı seçili.

MevcutKAd();

mvctParola.Visible = false;

this.Size = new System.Drawing.Size(310, 300);

gbKAd.Visible = true;

gbParola.Visible = false;

GuvenlikKod.Visible = true;

txtGuvenlik.Visible = true;

lb5.Visible = true;

btnGuncelle.Visible = true;

gbKAd.Location = new System.Drawing.Point(25, 70);

GuvenlikKod.Location = new System.Drawing.Point(159, 154);

lb5.Location = new System.Drawing.Point(65, 183);

txtGuvenlik.Location = new System.Drawing.Point(148, 183);

btnGuncelle.Location = new System.Drawing.Point(207, 209);

}

if (!cbKAd.Checked && cbParola.Checked)

{

//Sadece parola seçili.

MevcutParola();

mvctKAd.Visible = false;

this.Size = new System.Drawing.Size(310, 300);

gbParola.Visible = true;

gbKAd.Visible = false;

GuvenlikKod.Visible = true;

txtGuvenlik.Visible = true;

lb5.Visible = true;

btnGuncelle.Visible = true;

gbParola.Location = new System.Drawing.Point(25, 70);

GuvenlikKod.Location = new System.Drawing.Point(159, 154);

lb5.Location = new System.Drawing.Point(65, 183);

txtGuvenlik.Location = new System.Drawing.Point(148, 183);

btnGuncelle.Location = new System.Drawing.Point(207, 209);

}

if (cbKAd.Checked && cbParola.Checked)

{

//İkiside seçiliyse.

MevcutKAd();

MevcutParola();

this.Size = new System.Drawing.Size(310, 385);

gbKAd.Visible = true;

gbParola.Visible = true;

GuvenlikKod.Visible = true;

txtGuvenlik.Visible = true;

lb5.Visible = true;

btnGuncelle.Visible = true;

gbKAd.Location = new System.Drawing.Point(25, 70);

gbParola.Location = new System.Drawing.Point(25, 150);

GuvenlikKod.Location = new System.Drawing.Point(159, 231);

lb5.Location = new System.Drawing.Point(65, 263);

txtGuvenlik.Location = new System.Drawing.Point(148, 263);

btnGuncelle.Location = new System.Drawing.Point(208, 295);

}

if (!cbKAd.Checked && !cbParola.Checked)

{

mvctKAd.Visible = false;

mvctParola.Visible = false;

this.Size = new System.Drawing.Size(310, 100);

gbKAd.Visible = false;

gbParola.Visible = false;

GuvenlikKod.Visible = false;

txtGuvenlik.Visible = false;

lb5.Visible = false;

btnGuncelle.Visible = false;

}

}

private void cbKAd\_CheckedChanged(object sender, EventArgs e)

{

HangisiSecili();

}

private void cbParola\_CheckedChanged(object sender, EventArgs e)

{

HangisiSecili();

}  
 private bool KontrollerGecerliMi()

{

if (cbKAd.Checked && cbParola.Checked)

{

if (txtKAd1.Text == txtKAd2.Text)

{

if (txtSifre1.Text == txtSifre2.Text)

{

if (txtGuvenlik.Text == GuvenlikKod.Text)

{

bool onay = \_blUserLogin.KullaniciAdSifreDegistir2(Tc\_no, txtKAd1.Text, txtSifre1.Text);

return onay;

}

else

{

MessageBox.Show("Güvenlik kodunu yanlış girdiniz! \nLütfen Güvenlik kodunu kontrol ediniz.");

txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

else if (txtSifre1.Text != txtSifre2.Text && txtGuvenlik.Text != GuvenlikKod.Text)

{

MessageBox.Show("Güvenlik Kodu YANLIŞ! \nŞifreler Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtKAd1.Text = null; txtKAd2.Text = null; txtSifre1.Text = null;

txtSifre2.Text = null; txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

else

{

MessageBox.Show("Şifreler Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtSifre1.Text = null; txtSifre2.Text = null;

txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

else if (txtKAd1.Text != txtKAd2.Text && txtGuvenlik.Text != GuvenlikKod.Text)

{

MessageBox.Show("Güvenlik Kodu YANLIŞ! \nKullanıcı Adları Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtKAd1.Text = null; txtKAd2.Text = null; txtSifre1.Text = null;

txtSifre2.Text = null; txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

else

{

MessageBox.Show("Kullanıcı Adları Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtKAd1.Text = null; txtKAd2.Text = null; txtSifre1.Text = null;

txtSifre2.Text = null; txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

if (!cbKAd.Checked && cbParola.Checked)

{

if (txtSifre1.Text == txtSifre2.Text)

{

if (txtGuvenlik.Text == GuvenlikKod.Text)

{

bool onay = \_blUserLogin.SifreDegistir2(Tc\_no, txtSifre1.Text);

return onay;

}

else

{

MessageBox.Show("Güvenlik kodunu yanlış girdiniz! \nLütfen Güvenlik kodunu kontrol ediniz.");

txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

else if (txtSifre1.Text != txtSifre2.Text && txtGuvenlik.Text != GuvenlikKod.Text)

{

MessageBox.Show("Güvenlik Kodu YANLIŞ! \nŞifreler Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtSifre1.Text = null; txtSifre2.Text = null; txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

else

{

MessageBox.Show("Şifreler Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtSifre1.Text = null; txtSifre2.Text = null; txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

if (cbKAd.Checked && !cbParola.Checked)

{

if (txtKAd1.Text == txtKAd2.Text)

{

if (txtGuvenlik.Text == GuvenlikKod.Text)

{

bool onay = \_blUserLogin.KullaniciAdDegistir2(Tc\_no, txtKAd1.Text);

return onay;

}

else

{

MessageBox.Show("Güvenlik kodunu yanlış girdiniz! \nLütfen Güvenlik kodunu kontrol ediniz.");

txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

else if (txtKAd1.Text != txtKAd2.Text && txtGuvenlik.Text != GuvenlikKod.Text)

{

MessageBox.Show("Güvenlik Kodu YANLIŞ! \nKullanıcı Adları Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtKAd1.Text = null; txtKAd2.Text = null; txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

else

{

MessageBox.Show("Kullanıcı Adları Aynı DEĞİL! \nLütfen kontrol edip yeniden deneyiniz.");

txtKAd1.Text = null; txtKAd2.Text = null; txtGuvenlik.Text = null; GuvenlikKodYenile();

return false;

}

}

if (!cbKAd.Checked && !cbParola.Checked)

{

return false;

}

else { return false; }

}

private void btnGuncelle\_Click(object sender, EventArgs e)

{

if (KontrollerGecerliMi())

{

//Veri tabanına veri yollanacak ve update edilecek.

MessageBox.Show("Başarılı işlem.");

this.Size = new System.Drawing.Size(310, 105);

GuvenlikKodYenile();

gbKAd.Visible = false; txtKAd1.Clear(); txtKAd2.Clear();

gbParola.Visible = false; txtSifre1.Clear(); txtSifre2.Clear();

GuvenlikKod.Visible = false;

txtGuvenlik.Visible = false; txtGuvenlik.Clear();

lb5.Visible = false;

btnGuncelle.Visible = false;

cbKAd.Checked = false;

cbParola.Checked = false;

}

}

}

}

Bu kod, "Çalışan Bilgi Girişi ve Güncelleme" işlevselliğini sağlayan bir Windows Forms uygulamasını temsil eder ve aşağıdaki temel özelliklere sahiptir:

1. **Formun Başlatılması (EmployeeInfoLogin):**
   * Form yüklenirken, gerekli başlangıç ayarları yapılır.
   * Kullanıcı adı, şifre, güvenlik kodu alanlarına maksimum karakter uzunlukları belirlenir.
   * **\_blUserLogin** adlı **BlUserLogin** sınıfından bir nesne oluşturulur.
2. **Mevcut Kullanıcı Adı ve Şifre Gösterme (MevcutKAd ve MevcutParola):**
   * Kullanıcının mevcut kullanıcı adını ve şifresini göstermek için **MevcutKAd** ve **MevcutParola** metodları kullanılır.
3. **Güvenlik Kodu Yenileme (GuvenlikKodYenile):**
   * Güvenlik kodunu yenilemek için **GuvenlikKodYenile** metodu kullanılır.
4. **Hangi Bilgilerin Güncelleneceğini Belirleme (HangisiSecili):**
   * Kullanıcı, form üzerinde hangi bilgileri güncellemek istediğini seçebilir.
   * Seçilen bilgilere göre formun boyutu ve görünürlük ayarları değiştirilir.
5. **Kontrollerin Geçerliliğini Kontrol Etme (KontrollerGecerliMi):**
   * Kullanıcının girdiği bilgilerin geçerliliğini kontrol eder.
   * Seçilen bilgileri ve güvenlik kodunu kullanarak uygun güncelleme işlemlerini yapar.
   * Hatalı giriş durumunda kullanıcıya uygun hata mesajları gösterilir.
6. **Güncelleme İşlemi (btnGuncelle\_Click):**
   * Kullanıcı, güncelleme butonuna tıkladığında bu metod çalışır.
   * **KontrollerGecerliMi** metodu ile giriş bilgilerinin geçerliliği kontrol edilir.
   * Geçerliyse, güncelleme işlemi gerçekleştirilir ve kullanıcı bilgilendirilir.
   * Formun görünürlük durumu ve içeriği sıfırlanarak başlangıç durumuna getirilir.

Bu kod, kullanıcı bilgilerinin güncellenmesi amacıyla geliştirilmiş bir formu temsil eder. Kullanıcı, form üzerinden hangi bilgilerini güncellemek istediğini seçebilir ve bu bilgileri güncellerken güvenlik kontrolleri yapılır.

using BusinessLayer;

using DernekUyeTakipSistemi.Customer;

using System;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Employee.MenuLayer.EmployeeInfoLayer.MenuLayer

{

public partial class EmployeeInfoPersonal : Form

{

public string Tc\_no;

BlUser \_blUser;

BlUserLogin \_blUserLogin;

DataTable dt;

public EmployeeInfoPersonal()

{

InitializeComponent();

\_blUser = new BlUser();

\_blUserLogin = new BlUserLogin();

dt = new DataTable();

txtTc.MaxLength = 11;

txtAd.MaxLength = 15;

txtSoyad.MaxLength = 15;

txtTel.MaxLength = 15;

txtMail.MaxLength = 60;

txtAdres.MaxLength = 60;

FillIl();

this.AcceptButton = btnGuncelle;

}

public void TcGetir(string tc)

{

Tc\_no = tc;

FormGuncelle();

}

private void ClearForm()

{

// Formdaki tüm kontrol değerlerini temizleme işlemi

foreach (Control control in Controls)

{

if (control is System.Windows.Forms.TextBox textBox)

{

textBox.Clear();

}

else if (control is System.Windows.Forms.ComboBox comboBox)

{

comboBox.SelectedIndex = -1;

}

}

}

private void FormGuncelle()

{

dt = \_blUserLogin.TcArama(Tc\_no);

if (dt.Rows.Count > 0)

{

txtTc.Text = dt.Rows[0]["Tc\_No"].ToString();

txtAd.Text = dt.Rows[0]["Ad"].ToString();

txtSoyad.Text = dt.Rows[0]["Soyad"].ToString();

cmCins.Text = dt.Rows[0]["Cinsiyet"].ToString();

cmKan.Text = dt.Rows[0]["Kan\_Grup"].ToString();

txtTel.Text = dt.Rows[0]["Telefon"].ToString();

txtMail.Text = dt.Rows[0]["Mail"].ToString();

txtAdres.Text = dt.Rows[0]["Adres"].ToString();

cmil.SelectedValue = dt.Rows[0]["IlID"].ToString();

cmilce.SelectedValue = dt.Rows[0]["IlceID"].ToString();

}

else

{

ClearForm();

}

}

private void txtAd\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsLetter(e.KeyChar) && e.KeyChar != (char)Keys.Back && e.KeyChar != (char)Keys.Space)

{

e.Handled = true;

}

}

private void txtSoyad\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsLetter(e.KeyChar) && e.KeyChar != (char)Keys.Back && e.KeyChar != (char)Keys.Space)

{

e.Handled = true;

}

}

private void imlec(int position)

{

txtTel.SelectionStart = position;

txtTel.Focus();

}

private int eskiuzunluk = 0;

private void txtTel\_TextChanged(object sender, EventArgs e)

{

string text = txtTel.Text.Replace("(", "").Replace(")", "").Replace(" ", "");

StringBuilder formattedNumber = new StringBuilder();

int yeniuzunluk = text.Length;

if (yeniuzunluk > eskiuzunluk)

{

if (text.Length > 0)

{

formattedNumber.Append("(" + text.Substring(0, Math.Min(3, text.Length)) + ")");

yeniuzunluk = text.Length;

}

if (text.Length > 3)

{

formattedNumber.Append(" " + text.Substring(3, Math.Min(3, text.Length - 3)));

yeniuzunluk = text.Length;

}

if (text.Length > 6)

{

formattedNumber.Append(" " + text.Substring(6, Math.Min(2, text.Length - 6)));

yeniuzunluk = text.Length;

}

if (text.Length > 8)

{

formattedNumber.Append(" " + text.Substring(8, Math.Max(0, text.Length - 8)));

yeniuzunluk = text.Length;

}

txtTel.Text = formattedNumber.ToString();

txtTel.SelectionStart = txtTel.Text.Length;

}

if (yeniuzunluk < eskiuzunluk)

{

if (text.Length == 3)

{

imlec(4);

}

if (text.Length == 0)

{

txtTel.Text = string.Empty;

}

}

eskiuzunluk = yeniuzunluk;

}

private void txtTel\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsDigit(e.KeyChar) && e.KeyChar != (char)Keys.Back)

{

e.Handled = true;

}

}

private void txtMail\_TextChanged(object sender, EventArgs e)

{

KontrolEt();

}

private int KontrolEt()

{

string email = txtMail.Text;

if (IsValidEmail(email))

{

txtMail.ForeColor = Color.Black;

return 1;

}

else

{

txtMail.ForeColor = Color.Red;

return 0;

}

}

private bool IsValidEmail(string email)

{

string emailPattern = @"^[a-zA-Z0-9.\_-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$";

return System.Text.RegularExpressions.Regex.IsMatch(email, emailPattern);

}

private void FillIl()

{

cmil.DataSource = \_blUser.IlSorgu();

cmil.DisplayMember = "IlAdi";

cmil.ValueMember = "IlID";

cmil.SelectedItem = -1;

}

private void cmil\_SelectedIndexChanged(object sender, EventArgs e)

{

if (cmil.SelectedValue != null)

{

DataRowView selectedRow = cmil.SelectedItem as DataRowView;

if (selectedRow != null)

{

int selectedIlID = Convert.ToInt32(selectedRow["IlID"]);

FillIlce(selectedIlID);

}

}

}

private void FillIlce(int ilId)

{

cmilce.DataSource = \_blUser.IlceSorgu(ilId);

cmilce.DisplayMember = "IlceAdi";

cmilce.ValueMember = "IlceID";

cmilce.SelectedItem = -1;

}

private bool KontrolleriGecerliMi()

{

if (txtAd.Text.Length <= 2 || txtSoyad.Text.Length <= 2)

{

MessageBox.Show("Lütfen Gerçek Adınızı ve Soyadınızı Giriniz.");

return false;

}

if (txtTel.Text.Length != 15)

{

MessageBox.Show("Lütfen Telefon numaranızı uygun formatta giriniz.");

return false;

}

if (KontrolEt() != 1)

{

MessageBox.Show("Lütfen MAİL adresinizi uygun formatta giriniz.");

return false;

}

if (cmil.SelectedIndex == -1 || cmilce.SelectedIndex == -1)

{

MessageBox.Show("Lütfen İl ve İlçe Seçiniz.");

return false;

}

if (txtAdres.Text.Length <= 7)

{

MessageBox.Show("Lütfen Açık Adres Bilginizi Giriniz.");

return false;

}

return true;

}

private void btnGuncelle\_Click(object sender, EventArgs e)

{

if (!KontrolleriGecerliMi())

{

return;

}

int ilID = cmil.SelectedIndex >= 0 ? Convert.ToInt32(cmil.SelectedValue) : -1;

int ilceID = cmilce.SelectedIndex >= 0 ? Convert.ToInt32(cmilce.SelectedValue) : -1;

bool insertSonuc = \_blUser.GuncellemeSorgu(txtTc.Text, txtAd.Text, txtSoyad.Text, txtTel.Text, txtMail.Text, txtAdres.Text, ilID, ilceID);

if (insertSonuc)

{

MessageBox.Show("Güncelleme Başarılı!");

ClearForm();

FormGuncelle();

txtAd.Focus();

CMenu cMenu = Application.OpenForms["CMenu"] as CMenu;

cMenu.SetLabelText();

}

else

{

MessageBox.Show("Kayıt gerçekleşirken bir hata oluştu.");

}

}

}

}

Bu kod, "Çalışan Bilgi Güncelleme" işlevselliğini gerçekleştiren bir Windows Forms uygulamasını temsil eder ve aşağıdaki temel özelliklere sahiptir:

1. **Formun Başlatılması (EmployeeInfoPersonal):**
   * Form yüklenirken, gerekli başlangıç ayarları yapılır.
   * Kullanıcı giriş yapmışsa, ilgili kullanıcının TC kimlik numarası alınarak formun güncellenmesi sağlanır.
   * İlgili iş mantığı sınıfları (**BlUser** ve **BlUserLogin**) kullanılarak ilgili bilgilerin getirilmesi sağlanır.
2. **Formu Güncelleme (FormGuncelle):**
   * TC kimlik numarasına göre kullanıcının kişisel bilgileri form üzerinde görüntülenir.
   * Bu işlem, kullanıcının daha önce kayıt yapmış olması durumunda geçerli bilgileri formda göstermek amacıyla gerçekleştirilir.
3. **Telefon Numarası Formatlama (txtTel\_TextChanged ve txtTel\_KeyPress):**
   * Telefon numarasının girildiği TextBox'un **TextChanged** olayı dinlenir.
   * Girilen telefon numarası, belirli bir formata otomatik olarak dönüştürülerek ekranda gösterilir.
   * Kullanıcının sadece sayı girmesine izin verilir.
4. **E-posta Geçerliliği Kontrolü (KontrolEt ve IsValidEmail):**
   * E-posta adresinin geçerliliğini kontrol eden metodlar bulunur.
   * Eğer geçerli bir e-posta değilse, e-posta TextBox'ı kırmızı renkte gösterilir.
5. **İl ve İlçe Seçimleri (FillIl, cmil\_SelectedIndexChanged, ve FillIlce):**
   * İl ve ilçe ComboBox'ları ilgili iş mantığı sınıflarından alınan verilerle doldurulur.
   * İl seçildiğinde ilçe ComboBox'ı güncellenir.
6. **Kontrollerin Geçerliliğini Kontrol Etme (KontrolleriGecerliMi):**
   * Girilen bilgilerin uygunluğunu kontrol eder.
   * Uygun olmayan bir durum varsa kullanıcıya uygun bir hata mesajı gösterilir.
7. **Güncelleme İşlemi (btnGuncelle\_Click):**
   * Kullanıcının girdiği bilgilerin geçerli olup olmadığını kontrol eder (**KontrolleriGecerliMi**).
   * Geçerliyse, güncelleme işlemi yapılır ve kullanıcı bilgilendirilir.
   * Form temizlenir ve tekrar güncellenmiş bilgilerle doldurulur.
   * İlgili menü formu (**CMenu**) üzerindeki bir etiket güncellenir.

Bu kod, çalışanın kişisel bilgilerini güncelleyebileceği bir formu temsil eder. Girilen bilgilerin uygunluğu kontrol edilir ve ardından güncelleme işlemi gerçekleştirilir.

using BusinessLayer;

using DernekUyeTakipSistemi.Employee.MenuLayer.EmployeeInfoLayer.MenuLayer;

using System;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Employee.MenuLayer.EmployeeInfoLayer

{

public partial class EmployeeInfoMenu : Form

{

private EmployeeInfoLogin UserName;

private EmployeeInfoPersonal UserUpdate;

private bool uygulamaKapatildi = false;

public static string Tc\_no { get; set; }

public EmployeeInfoMenu()

{

InitializeComponent();

}

private void btnUserName\_Click(object sender, EventArgs e)

{

if (UserName == null || UserName.IsDisposed)

{

UserName = new EmployeeInfoLogin();

UserName.FormClosed += (s, args) => this.Focus();

UserName.Show();

UserName.TcAtama(Tc\_no);

}

}

private void btnUpdate\_Click(object sender, EventArgs e)

{

if (UserUpdate == null || UserUpdate.IsDisposed)

{

UserUpdate = new EmployeeInfoPersonal();

UserUpdate.FormClosed += (s, args) => this.Focus();

UserUpdate.Show();

UserUpdate.TcGetir(Tc\_no);

}

}

private void btnGeri\_Click(object sender, EventArgs e)

{

FormlariKapa();

uygulamaKapatildi = true;

this.Close();

EMenu eMenu = Application.OpenForms["EMenu"] as EMenu;

eMenu.Show();

}

public void FormlariKapa()

{

if (UserName != null)

{

UserName.Close();

}

if (UserUpdate != null)

{

UserUpdate.Close();

}

}

private void EmployeeInfoMenu\_FormClosing(object sender, FormClosingEventArgs e)

{

if (!uygulamaKapatildi)

{

DialogResult cikis = MessageBox.Show("Çıkış yapmak istediğinize emin misiniz ?", "", MessageBoxButtons.YesNo);

if (cikis == DialogResult.Yes)

{

uygulamaKapatildi = true;

Application.Exit();

}

else

{

e.Cancel = true;

}

}

}

}

}

Bu kod, "Çalışan Bilgi Menüsü"nü temsil eden bir Windows Forms uygulamasını oluşturan bir sınıftır ve aşağıdaki temel özelliklere sahiptir:

1. **Bağlantılı Form Nesneleri:**
   * **EmployeeInfoLogin** ve **EmployeeInfoPersonal** adında iki bağlantılı form nesnesi (**UserName** ve **UserUpdate**) bulunur.
   * Bu nesneler, kullanıcının giriş bilgilerini güncelleyebileceği ve kişisel bilgilerini güncelleyebileceği alt formları temsil eder.
2. **TC Kimlik Numarası Bilgisi:**
   * **Tc\_no** adında bir static özellik, bu form sınıfı üzerinden ulaşılabilir bir şekilde tanımlanmıştır.
   * Bu özellik, kullanıcının TC kimlik numarasını saklamak için kullanılır.
3. **Kullanıcı Adı Bilgileri Formunu Açma (btnUserName\_Click):**
   * "Kullanıcı Adı Bilgileri" butonuna tıklandığında, **EmployeeInfoLogin** formu oluşturulur ve gösterilir.
   * Eğer bu form daha önce kapatılmışsa veya hiç oluşturulmamışsa, yeni bir örneği oluşturulur.
4. **Kişisel Bilgileri Güncelleme Formunu Açma (btnUpdate\_Click):**
   * "Bilgileri Güncelle" butonuna tıklandığında, **EmployeeInfoPersonal** formu oluşturulur ve gösterilir.
   * Eğer bu form daha önce kapatılmışsa veya hiç oluşturulmamışsa, yeni bir örneği oluşturulur.
   * Bu formun **TcGetir** metodu aracılığıyla, kullanıcının TC kimlik numarası bu forma iletilir.
5. **Geri Butonu ve Form Kapatma (btnGeri\_Click ve EmployeeInfoMenu\_FormClosing):**
   * "Geri" butonuna tıklandığında, diğer alt formların kapatılması ve ana menüye dönülmesi sağlanır.
   * Form kapatılmak istendiğinde, bir onay mesajı gösterilir ve kullanıcı onay verirse uygulama kapatılır.
6. **Formları Kapatma (FormlariKapa):**
   * Diğer alt formları kapatmak için kullanılan bir metottur.
   * **UserName** ve **UserUpdate** form nesneleri varsa, bunlar kapatılır.

Bu kod, çalışanın kendi bilgilerini güncelleyebileceği bir menüyü temsil eder. Menüde, kullanıcı adı bilgilerini güncelleyebileceği bir alt form ve kişisel bilgilerini güncelleyebileceği bir diğer alt form bulunmaktadır. Kullanıcının işlem yapmak istediği formu seçmesi sağlanır.

Formun Üstü

using BusinessLayer;

using DernekUyeTakipSistemi.Admin;

using DernekUyeTakipSistemi.Employee.MenuLayer.ListLayer.ListMenu;

using DernekUyeTakipSistemi.Employee;

using iTextSharp.text;

using iTextSharp.text.pdf;

using System;

using System.Data;

using System.IO;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Employee.MenuLayer.ListLayer.ListMenu

{

public partial class ListMenu : Form

{

BlList \_blList;

private bool uygulamaKapatildi = false;

public ListMenu()

{

InitializeComponent();

\_blList = new BlList();

ekran.DataSource = null;

cbOrtak1.Visible = false;

lbOrtak1.Visible = false;

btnMail.Visible = false;

btnPDF.Visible = false;

cbOrtak1.SelectedIndex = -1;

cbGenel.SelectedIndex = 0;

}

private void KanlariComboBoxaEkle()

{

cbOrtak1.DataSource = null;

cbOrtak1.Items.Clear();

cbOrtak1.Items.Add("Tümü");

cbOrtak1.Items.Add("0 Rh +");

cbOrtak1.Items.Add("0 Rh -");

cbOrtak1.Items.Add("A Rh +");

cbOrtak1.Items.Add("A Rh -");

cbOrtak1.Items.Add("B Rh +");

cbOrtak1.Items.Add("B Rh -");

cbOrtak1.Items.Add("AB Rh +");

cbOrtak1.Items.Add("AB Rh -");

}

private void IlleriComboBoxaEkle()

{

cbOrtak1.DataSource = \_blList.IlSorgu();

cbOrtak1.DisplayMember = "IlAdi";

cbOrtak1.ValueMember = "IlID";

cbOrtak1.SelectedIndex = -1;

}

private void StatuleriComboBoxaEkle()

{

cbOrtak1.DataSource = null;

cbOrtak1.Items.Clear();

cbOrtak1.Items.Add("Tümü");

cbOrtak1.Items.Add("Aktif Üyeler");

cbOrtak1.Items.Add("Pasif Üyeler");

}

private void BorclariComboBoxaEkle()

{

cbOrtak1.DataSource = null;

cbOrtak1.Items.Clear();

cbOrtak1.Items.Add("Tümü");

cbOrtak1.Items.Add("Borçlu Üyeler");

cbOrtak1.Items.Add("Borcu OLMAYAN Üyeler");

}

private void cbGenel\_SelectedIndexChanged(object sender, EventArgs e)

{

ekran.DataSource = null;

cbOrtak1.Visible = false;

lbOrtak1.Visible = false;

btnMail.Visible = false;

btnPDF.Visible = false;

dateTimePicker1.Visible = false;

dateTimePicker2.Visible = false;

if (cbGenel.SelectedIndex == 1)

{

lbOrtak1.Visible = true;

cbOrtak1.Visible = true;

lbOrtak1.Text = "Kan Grubu";

KanlariComboBoxaEkle();

}

else if (cbGenel.SelectedIndex == 2)

{

lbOrtak1.Visible = true;

cbOrtak1.Visible = true;

lbOrtak1.Text = "İl";

IlleriComboBoxaEkle();

}

else if (cbGenel.SelectedIndex == 3)

{

lbOrtak1.Visible = true;

cbOrtak1.Visible = true;

lbOrtak1.Text = "Üye Durumu";

StatuleriComboBoxaEkle();

}

else if (cbGenel.SelectedIndex == 4)

{

lbOrtak1.Visible = true;

cbOrtak1.Visible = true;

btnPDF.Visible = true;

lbOrtak1.Text = "Borç Bilgisi";

BorclariComboBoxaEkle();

}

else if (cbGenel.SelectedIndex == 5)

{

dateTimePicker1.Visible = true;

dateTimePicker2.Visible = true;

}

}

private void cbOrtak1\_SelectedIndexChanged(object sender, EventArgs e)

{

if (cbGenel.SelectedIndex == 1)

{

if (cbOrtak1.SelectedIndex != -1)

{

string KanGrubu;

if (cbOrtak1.SelectedIndex == 0)

{

ekran.DataSource = \_blList.KanSorgu1();

}

if (cbOrtak1.SelectedIndex > 0)

{

KanGrubu = cbOrtak1.SelectedItem?.ToString() ?? string.Empty;

ekran.DataSource = \_blList.KanSorgu2(KanGrubu);

}

}

}

if (cbGenel.SelectedIndex == 2)

{

if (cbOrtak1.SelectedIndex != 1)

{

if (cbOrtak1.SelectedIndex == 0)

{

ekran.DataSource = \_blList.KanSorgu1();

}

if (cbOrtak1.SelectedIndex > 0)

{

int ilID = cbOrtak1.SelectedIndex >= 0 ? Convert.ToInt32(cbOrtak1.SelectedValue) : -1;

ekran.DataSource = \_blList.IlIlceTablo(ilID);

}

}

}

if (cbGenel.SelectedIndex == 3)

{

if (cbOrtak1.SelectedIndex != -1)

{

string statu;

if (cbOrtak1.SelectedIndex == 0)

{

ekran.DataSource = \_blList.StatuSorgu1();

}

if (cbOrtak1.SelectedIndex == 1)

{

statu = "A";

ekran.DataSource = \_blList.StatuSorgu2(statu);

}

if (cbOrtak1.SelectedIndex == 2)

{

statu = "D";

ekran.DataSource = \_blList.StatuSorgu2(statu);

}

}

}

if (cbGenel.SelectedIndex == 4)

{

if (cbOrtak1.SelectedIndex != -1)

{

bool borcBilgi;

if (cbOrtak1.SelectedIndex == 0)

{

btnMail.Visible = false;

ekran.DataSource = \_blList.BorcSorgu1();

}

if (cbOrtak1.SelectedIndex == 1)

{

btnMail.Visible = true;

borcBilgi = true;

ekran.DataSource = \_blList.BorcSorgu2(borcBilgi);

}

if (cbOrtak1.SelectedIndex == 2)

{

btnMail.Visible = false;

borcBilgi = false;

ekran.DataSource = \_blList.BorcSorgu2(borcBilgi);

}

}

}

}

private void btnGeri\_Click(object sender, EventArgs e)

{

AMenu aMenu = Application.OpenForms["AMenu"] as AMenu;

EMenu eMenu = Application.OpenForms["EMenu"] as EMenu;

if (aMenu != null)

{

uygulamaKapatildi = true;

this.Close();

aMenu.Show();

}

if (eMenu != null)

{

uygulamaKapatildi = true;

this.Close();

eMenu.Show();

}

}

private void OpenDetailsForm(DataGridViewRow selectedRow, string[] emailList)

{

// Yeni bir detay formu oluştur

MailPage detailsForm = new MailPage();

// Detay formuna tıklanan satırın bilgilerini iletiyoruz

detailsForm.SetDetails(selectedRow.Cells["Ad"].Value.ToString(),

selectedRow.Cells["Soyad"].Value.ToString(),

selectedRow.Cells["Mail"].Value.ToString(),

selectedRow.Cells["ToplamBorc"].Value.ToString(), emailList);

// Detay formunu gösteriyoruz

detailsForm.ShowDialog();

}

private void btnMail\_Click(object sender, EventArgs e)

{

if (ekran.SelectedCells.Count > 0)

{

// Seçili hücrenin bulunduğu satırın indeksi

int rowIndex = ekran.SelectedCells[0].RowIndex;

// Seçili satırın bilgilerini al

DataGridViewRow selectedRow = ekran.Rows[rowIndex];

string[] emailList = new string[ekran.Rows.Count];

for (int i = 0; i < ekran.Rows.Count; i++)

{

// Her satırın Mail hücresinden değeri al ve diziye ekle

if (ekran.Rows[i].Cells["Mail"].Value != null)

{

emailList[i] = ekran.Rows[i].Cells["Mail"].Value.ToString();

}

else

{

emailList[i] = ""; // Eğer Mail hücresi boşsa, diziye boş bir değer ekleyebilirsiniz

}

}

OpenDetailsForm(selectedRow, emailList);

}

else

{

MessageBox.Show("Lütfen bir satır seçin.");

}

}

private void btnPDF\_Click(object sender, EventArgs e)

{

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.Filter = "PDF Files|\*.pdf";

saveFileDialog.Title = "Save PDF File";

saveFileDialog.FileName = "DataGridExport";

if (saveFileDialog.ShowDialog() == DialogResult.OK)

{

ExportDataGridToPdf(ekran, saveFileDialog.FileName);

}

}

private void ExportDataGridToPdf(DataGridView dataGridView, string filePath)

{

try

{

using (FileStream fs = new FileStream(filePath, FileMode.Create))

{

Document document = new Document(PageSize.A4.Rotate(), 10f, 10f, 10f, 0f);

PdfWriter.GetInstance(document, fs);

document.Open();

PdfPTable table = new PdfPTable(dataGridView.Columns.Count);

table.WidthPercentage = 100;

// Add Headers

foreach (DataGridViewColumn column in dataGridView.Columns)

{

PdfPCell cell = new PdfPCell(new Phrase(column.HeaderText));

table.AddCell(cell);

}

// Add Data

foreach (DataGridViewRow row in dataGridView.Rows)

{

foreach (DataGridViewCell cell in row.Cells)

{

PdfPCell pdfCell = new PdfPCell(new Phrase(cell.Value?.ToString()));

table.AddCell(pdfCell);

}

}

document.Add(table);

document.Close();

MessageBox.Show("PDF file saved successfully!", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

catch (Exception ex)

{

MessageBox.Show("An error occurred: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void ListMenu\_FormClosing(object sender, FormClosingEventArgs e)

{

if (!uygulamaKapatildi)

{

DialogResult cikis = MessageBox.Show("Çıkış yapmak istediğinize emin misiniz ?", "", MessageBoxButtons.YesNo);

if (cikis == DialogResult.Yes)

{

uygulamaKapatildi = true;

Application.Exit();

}

else

{

e.Cancel = true;

}

}

}

private void dateTimePicker2\_ValueChanged(object sender, EventArgs e)

{

DateTime dt1 = dateTimePicker1.Value;

DateTime dt2 = dateTimePicker2.Value;

DataTable sorgu = \_blList.TariheGore(dt1, dt2);

if (sorgu != null)

{

ekran.DataSource = sorgu;

}

}

private void label1\_Click(object sender, EventArgs e)

{

}

}

}

Bu kod, "Liste Menüsü" adlı bir Windows Forms uygulamasını oluşturan bir sınıftır ve aşağıdaki temel özelliklere sahiptir:

1. **Bağlantılı Business Layer:**
   * **BlList** adında bir Business Layer sınıfı (**\_blList**) kullanılarak veritabanı işlemleri gerçekleştirilir.
2. **ComboBox Veri Bağlama Metotları:**
   * **KanlariComboBoxaEkle()**, **IlleriComboBoxaEkle()**, **StatuleriComboBoxaEkle()**, **BorclariComboBoxaEkle()** adında metotlar ile ComboBox'lara ilgili verileri ekler.
3. **ComboBox Seçim Değişim Eventi (cbGenel\_SelectedIndexChanged):**
   * Genel durumu belirten ComboBox'ın seçim değişimine göre, diğer ComboBox'ların görünürlüğünü ve içeriğini ayarlar.
   * Bu duruma göre "Kan Grubu", "İl", "Üye Durumu", "Borç Bilgisi" gibi filtreleme seçenekleri sunar.
   * Tarih aralığı seçeneği için DateTimePicker'ları görünür hale getirir.
4. **ComboBox Filtreleme Eventi (cbOrtak1\_SelectedIndexChanged):**
   * **cbGenel** ComboBox'ında seçilen filtre türüne göre, **cbOrtak1** ComboBox'ında seçilen özel filtreleme seçeneklerine göre veritabanından sorgu yapar.
   * Bu duruma göre Kan grubuna, İl'e, Üye durumuna ve Borç bilgisine göre filtreleme sağlar.
5. **Geri Butonu Eventi (btnGeri\_Click):**
   * Bu butona tıklandığında, programın ana menüsüne dönülmesini sağlar.
6. **Detay Formunu Açma Eventi (btnMail\_Click):**
   * Seçilen bir hücredeki verilere dayanarak, bir detay formunu (**MailPage**) açar. Bu form, seçilen satırdaki kullanıcıya ait detay bilgileri ve mail bilgilerini gösterir.
7. **PDF İhraç Etme Eventi (btnPDF\_Click):**
   * DataGridView'daki verileri bir PDF dosyasına dışa aktarır.
8. **Form Kapatma Eventi (ListMenu\_FormClosing):**
   * Form kapatılmak istendiğinde bir onay mesajı gösterir ve kullanıcı onay verirse uygulamayı kapatır.
9. **Tarih Aralığına Göre Sorgulama Eventi (dateTimePicker2\_ValueChanged):**
   * DateTimePicker'lar aracılığıyla seçilen tarih aralığına göre veritabanında sorgulama yapar ve sonuçları gösterir.

Bu kod, kullanıcıya çeşitli filtreleme seçenekleri sunan bir liste menüsü uygulamasını temsil eder. Kullanıcı, kan grubu, il, üye durumu, borç bilgisi veya belirli bir tarih aralığına göre filtreleme yapabilir ve sonuçları görüntüleyebilir. Ayrıca, seçilen verileri PDF olarak dışa aktarabilir veya detayları içeren bir mail formunu açabilir.

using System.Net;

using System.Net.Mail;

using System.Windows.Forms;

namespace DernekUyeTakipSistemi.Employee.MenuLayer.ListLayer.ListMenu

{

public partial class MailPage : Form

{

private string email, konu, icerik1;

private string[] emailList;

public MailPage()

{

InitializeComponent();

}

public void SetDetails(string a, string b, string c, string d, string[] emailListe)

{

emailList = emailListe;

email = c;

konu = "Borc Bilgisi Hk.";

icerik1 = "Sayın (" + a + " " + b + "),\r\n\r\nGEDİK DERNEK TAKİP SİSTEMİMİZDE Kayıtlı Hesabınızda" +

"\r\nToplam (" + d + ")TL borcunuz bulunmaktadır." +

"\r\nLütfen son ödeme tarihi geçmeden borçlarınızı ÖDEYİNİZ! " +

"\r\nAksi durumda hakkınızda yasal işlem başlatılacaktır.\r\n\r\n" +

"\r\nİyi Günler Dileriz, " +

"\r\nGedik Dernek Takip Sistemi İnsan Kaynakları.";

radioButton2.Checked = true;

txtPosta.Visible = true;

txtPosta.Text = email;

txtKonu.Text = konu;

txtIcerik.Text = icerik1;

}

private void radioButton1\_CheckedChanged(object sender, System.EventArgs

{

if (radioButton1.Checked)

{

txtPosta.Visible = false;

txtKonu.Text = "Kullanıcılara Gönderilecek Mail Konusunu Belirtiniz.";

txtIcerik.Text = "Kullanıcılara Gönderilecek Mail İçeriğini Bu alana yazınız.\r\nGedik Dernek Takip Sistemi,\r\nİnsan Kaynakları";

}

}

private void radioButton2\_CheckedChanged(object sender, System.EventArgs

{

if (radioButton2.Checked)

{

txtPosta.Visible = true;

txtPosta.Text = email;

txtKonu.Text = konu;

txtIcerik.Text = icerik1;

}

}

private void btnGonder\_Click(object sender, System.EventArgs e)

{

if (radioButton1.Checked)

{

// E-posta konu ve içeriğini alın

string subject = txtKonu.Text;

string body = txtIcerik.Text;

// Her bir e-posta adresi için döngü

foreach (string toEmail in emailList)

{

// E-postayı gönder

if (IsValidEmail(toEmail))

{

SendEmail(toEmail, subject, body);

// E-posta gönderildikten sonra isteğe bağlı olarak başarı mesajı gösterebilirsiniz

MessageBox.Show("E-posta başarıyla gönderildi!", "Başarı", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show("Geçersiz e-posta adresi: " + toEmail, "Hata", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

// Tüm e-postalar gönderildikten sonra isteğe bağlı olarak bir başarı mesajı gösterebilir veya başka eylemler gerçekleştirebilirsiniz

MessageBox.Show("Tüm e-postalar başarıyla gönderildi!", "Başarı", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

if (radioButton2.Checked)

{

// Text kutularından e-posta detaylarını al

string toEmail = txtPosta.Text;

string subject = txtKonu.Text;

string body = txtIcerik.Text;

// E-posta adresinin geçerli olup olmadığını kontrol et

if (IsValidEmail(toEmail))

{

// E-postayı gönder

SendEmail(toEmail, subject, body);

// İsteğe bağlı olarak, e-posta gönderildikten sonra başarı mesajı gösterebilir veya başka eylemler gerçekleştirebilirsiniz

MessageBox.Show("E-posta başarıyla gönderildi!", "Başarı", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show("Geçersiz e-posta adresi!", "Hata", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

private bool IsValidEmail(string email)

{

try

{

var addr = new MailAddress(email);

return addr.Address == email;

}

catch

{

return false;

}

}

private void SendEmail(string to, string subject, string body)

{

// SMTP sunucu detaylarınız ile değiştirin

string smtpServer = "smtp-mail.outlook.com";

int smtpPort = 587;

string smtpUsername = "gedikdrnk.69@hotmail.com";

string smtpPassword = "15975365469asd+";

using (MailMessage mail = new MailMessage())

{

mail.From = new MailAddress(smtpUsername);

mail.To.Add(to);

mail.Subject = subject;

mail.Body = body;

mail.IsBodyHtml = false;

using (SmtpClient smtp = new SmtpClient(smtpServer, smtpPort))

{

smtp.Credentials = new NetworkCredential(smtpUsername, smtpPassword);

smtp.EnableSsl = true;

smtp.Send(mail);

}

}

}

}

}

Bu kod, "MailPage" adlı bir Windows Forms uygulamasını oluşturan bir sınıftır ve aşağıdaki temel özelliklere sahiptir:

1. **Mail Detayları ve Oluşturma Metodu (SetDetails):**
   * **SetDetails** metodu, detay formunun oluşturulmasını ve içeriğinin belirlenmesini sağlar.
   * Kullanıcı adı (**a** ve **b**), e-posta adresi (**c**), toplam borç miktarı (**d**) ve e-posta listesi (**emailListe**) gibi bilgileri alarak, mail içeriğini oluşturur.
2. **RadioButton'lar ile Mail Şablon Seçimi:**
   * Kullanıcıya, önceden belirlenmiş bir şablon kullanarak tüm kullanıcılara mail gönderme veya özel bir e-posta adresine mail gönderme seçenekleri sunar.
   * **radioButton1** seçildiğinde genel bir şablon, **radioButton2** seçildiğinde özel bir e-posta adresi belirlenir.
3. **Mail Gönderme Metodu (btnGonder\_Click):**
   * Kullanıcı, belirlediği şablona ve e-posta adresine göre mail gönderebilir.
   * Eğer genel bir şablon seçilmişse, belirlenen e-posta listesine döngü ile mail gönderir.
   * Eğer özel bir e-posta seçilmişse, tek bir e-posta adresine mail gönderir.
   * Geçerli olmayan e-posta adresleri veya başarılı gönderimler sonrasında uygun mesajları gösterir.
4. **E-posta Adresi Doğrulama Metodu (IsValidEmail):**
   * Verilen bir e-posta adresinin geçerli olup olmadığını kontrol eder.
5. **E-posta Gönderme Metodu (SendEmail):**
   * **SmtpClient** sınıfını kullanarak belirtilen SMTP sunucu detayları ile e-posta gönderir.
   * Gönderim başarılı olursa, başarı mesajı gösterir; aksi takdirde, hata mesajı gösterir.

Bu kod, kullanıcıya belirli bir şablona göre mail gönderme imkanı tanıyan bir arayüz sunan bir Windows Forms uygulamasını temsil eder. Kullanıcı, genel bir şablon seçerek belirli bir e-posta listesine veya özel bir e-posta adresine mail gönderebilir. E-posta gönderim işlemleri için SMTP protokolü kullanılarak, gerekli güvenlik ve kimlik bilgileri ile e-posta gönderimi gerçekleştirilir.

Formun Üstü

Formun Üstü

1. **UYGULAMA GÖRÜNTÜLERİ**

metin, ekran görüntüsü, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, marka içeren bir resim

Açıklama otomatik olarak oluşturuldu



