ASP.NET CORE 8.0 MVC DERS NOTLARI-(7.HAFTA)

İçindekiler

VERİTABANI İŞLEMLERİ

A. Veritabanı Yönetimi için Entity Framework (EF) Kütüphanesinin Tanıtımı

Tüm veritabanları üzerinde işlem yapabilen veri erişim teknolojisi olarak **EntityFrameworkCore** kütüphanesini kullanalım. Bundan başka "dapper" yada "Adonet" kütüphaneleri de kullanılabilir.

Entity Framework (EF), Microsoft tarafından geliştirilen ve .NET uygulamalarında veritabanı işlemlerini kolaylaştıran bir araçtır. SQL sorguları yazmadan, C# gibi dillerle doğrudan veritabanı işlemleri gerçekleştirebiliriz.

Entity Framework'ün Avantajları:

-Veritabanı işlemleri için SQL yazmaya gerek kalmaz. SQL Kodlarını Otomatik Üretir. LINQ (Language Integrated Query) kullanarak sorgular yazılabilir.

– Veritabanından Bağımsızdır. **SQL Server**, MySQL, PostgreSQL, Oracle gibi farklı veritabanlarıyla çalışabilir.

– Kod tarafında yapılan değişiklikler, doğrudan veritabanına yansıtılabilir (Code-First yaklaşımı). Code-First (koddan veritabanını oluşturma) kullanımında Önce C# sınıfları (entity'ler-Tablolara karşılık gelir) tanımlanır, ardından veritabanı oluşturulur. Veritabanı yoksa entity'lerden otomatik olarak oluşturur.

– Bakım ve Geliştirme Kolaylığı: Veritabanı işlemlerini bir katman olarak soyutladığı için kod daha temiz olur.

---------- 000----------

Örnek kod ve açıklaması şu şekildedir.

```
public class Urun
{
    public int Id { get; set; }
    public string Ad { get; set; }
    public decimal Fiyat { get; set; }
}

public class MyDbContext : DbContext
{
```

```
    public DbSet<Urun> Urunler { get; set; }
}
```

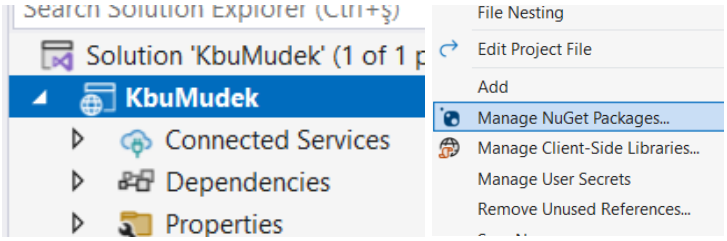
Burada

- **Urun** adında bir sınıf tanımlanıyor (Bu, veritabanında bir tabloya karşılık gelir).
- **MyDbContext**, Entity Framework'ün veritabanı işlemlerini yönettiği bağlantı sınıfıdır.
- **DbSet<Urun>** ile **Urunler** adında bir veritabanı tablosu temsil ediliyor.

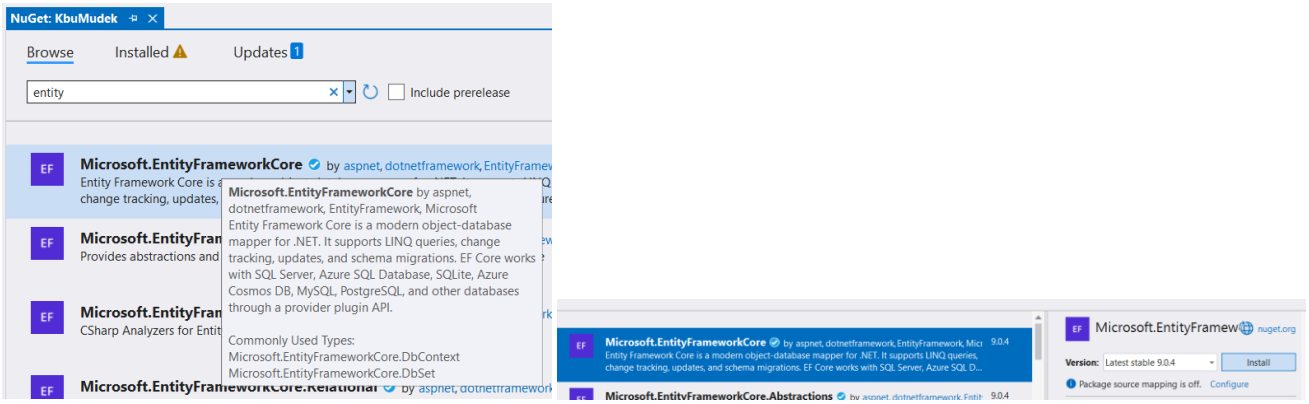
Entity Framework sayesinde manuel SQL kodları yazmadan veritabanı işlemleri gerçekleştirebiliriz. Entity Framework, yazılım projelerinde veritabanı işlemlerini kolaylaştırmak için oldukça faydalıdır.

B. Entity Framework'ün Kurulumu

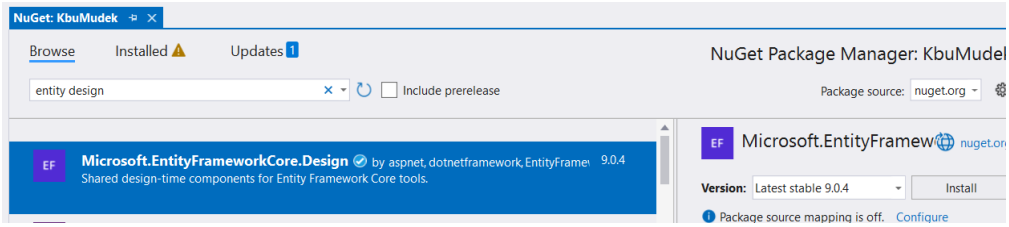
EF için 4 tane paketin projemiz içerisine kurulması gerekmektedir. Bunun için projenin üzerine sağ tuşa tıklayalım ve "Manage NuGet Packages" linkden, paketlerimizi yükleyecek olan NuGet ekranının açılmasını sağlayalım.



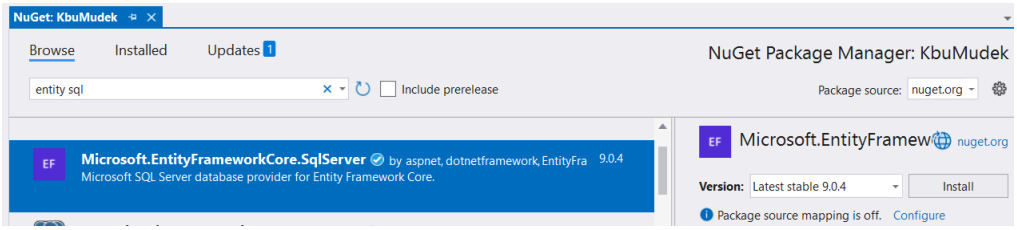
Açılan ekrandan Browse kısmına geçelim ve arama çubuğuna Entity yazalım. Gelen listeden "Microsoft.**EntityFrameworkCore**" paketini seçelim ve yanda çıkan "Install" düğmesine tıklayarak yükleyelim. Çıkacak lisans sözleşmelerini kabul edelim ("I accept").



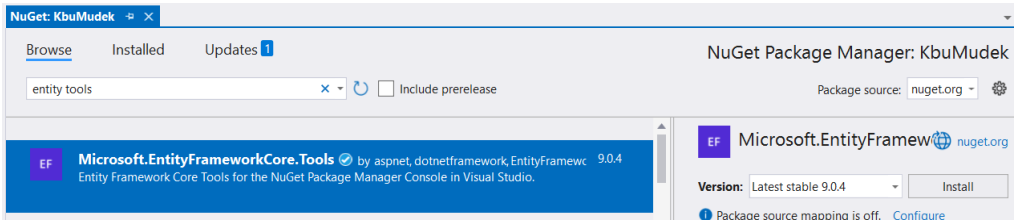
İkinci indirip kuracağımız paket "Microsoft.EntityFrameworkCore.**Design**" paketidir.



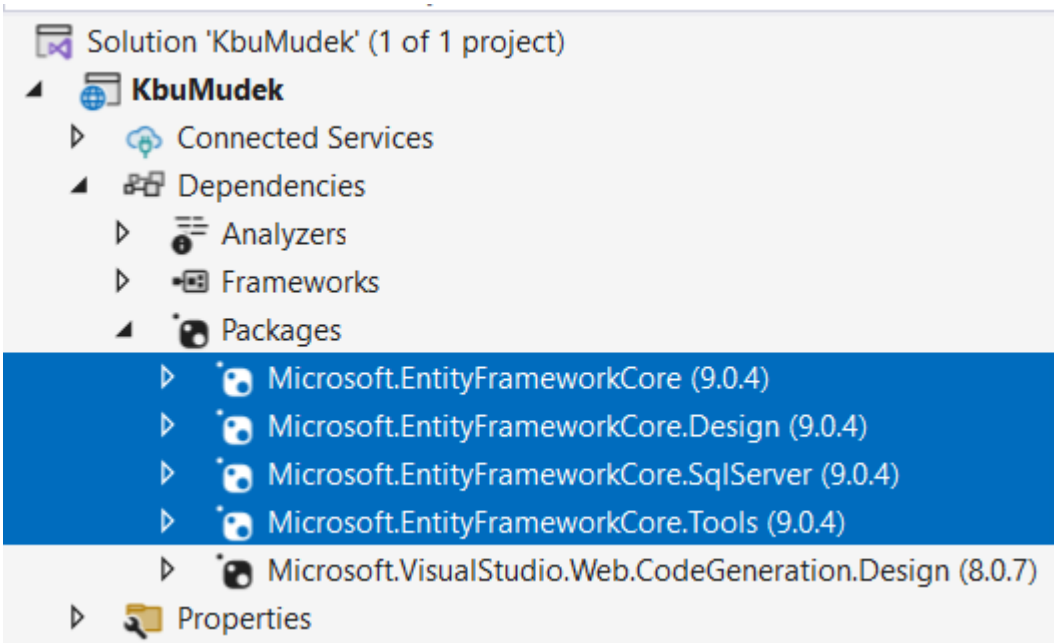
Üçüncü indireceğimiz paket "Microsoft.EntityFrameworkCore.**SqlServer**" paketidir.



Dördüncü indireceğimiz paket "Microsoft.EntityFrameworkCore.**Tools**" paketidir.



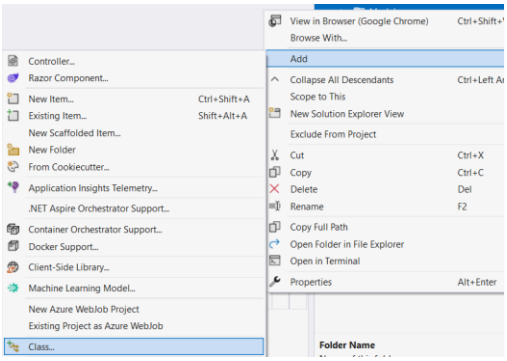
4 paketi kurduk. Kurduğumuz bu paketler projemizin içerisinde nerde kontrol edelim. Projemizde "Dependencies>Packages" kısmında görebiliriz.

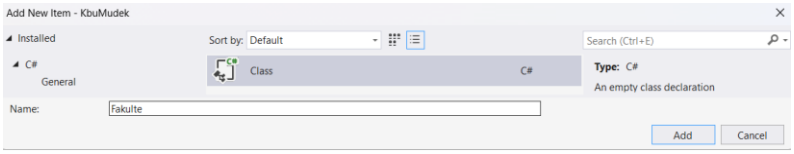


C. Model Sınıflarının Oluşturulması

Veritabanımızı oluşturacağımız ve yöneteceğimiz EF alt yapımızı kurduk ve hazır ettik. Şimdi veritabanımızdaki tablolara karşılık gelen ve bilgileri onun üzerinden taşıyacağımız Model yapılarımızı oluşturalım. Daha önce 4. notlarda sayfalar arasında bilgi taşımak için Model yapısına ihtiyaç duymuştuk. Kısa bir uygulama orda yapmıştık. Burda artık hem bilgi taşıma hem de Veritabanı tablolarımızı oluşturmak için gerekli olan model yapılarını oluşturmaya devam edelim.

Model klasörünün üzerine sağ tuşa tıklayıp Model>Add>Class yolunu izleyerek sınıfımızı oluşturalım. Oluşturacağımız model sınıfı "Fakulte" olsun. Yani üniversitede kurulacak olan ilk birim Fakültedir. Onu oluşturalım. Ardından "Bölüm" "Kullanici" vs diğer bilgiler için modellerimizi benzer şekilde oluştururuz.





"Fakulte" sınıfımızı aşağıdaki gibi oluşturalım. Sınıfın içindeki property'leri hızlı bir şekilde oluşturmak için "prop" yazıp iki defa Tab tuşuna basarsak hazır bir format oluşturacaktır. Bize sadece düzenlemesi kalacaktır. Burada **class** larımız, veritabanında **tabloya** karşılık gelir, **property** ler ise tablonun **sütun**larına karşılık gelir. FakulteId sütunu veritabanımızda primary key (tekrarsız otomatik numara atanacak) olacağından bunu tarif etmek için üzerindeki satıra [Key] yazdık. Key yazısının altında kırmızı çizgili uyarı çizgisi gözükecektir. Bunun ne olduğunu anlayacak olan yada onu kullanacak olan "...DataAnnotations" kütüphanesini de eklemeliyiz (Key'in üzerine tıkladığımızda hangi kütüphane ile çalışacağını gösterir, onu seçersek üste ekleyecektir).

```
using System.ComponentModel.DataAnnotations;

namespace KbuMudek.Models
{
    public class Fakulte
    {
        [Key]
        public int FakulteID { get; set; }
        public required string FakulteAdi { get; set; } //required doldurulmasını zorunlu
kılar
    }
}
```

Benzer şekilde "Bolum", "Program", "Kullanici", "Personel", "Ogrenci" vs projemizde ne kadar Model sınıfı varsa onları oluştururuz.

Ek Bilgi: Sınıf oluştururken "FakulteID" şeklinde yazdığımızda yani sınıf adının sonuna bir ID ifadesi eklediğimizde [Key] ifadesini yazmasak ta olur. Veritabanı oluşturulurken sonunda ID olan alanı otomatik olarak Primary Key olarak atayacaktır.

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace KbuMudek.Models
{
    public class Ogrenci
    {
        [Key]
        public int OgrenciId { get; set; }

        [Required(ErrorMessage = "Öğrenci numarası zorunludur.")]
        public int OgrenciNo { get; set; }

        [Required(ErrorMessage = "Ad alanı zorunludur.")]
        [StringLength(50)]
        public string Ad { get; set; }

        [Required(ErrorMessage = "Soyad alanı zorunludur.")]
```

```
        [StringLength(50)]
        public string Soyad { get; set; }

        // Opsiyonel alan
        public DateTime? DogumTarihi { get; set; }
    }
}
```

Diğer Model sınıfları da bu şekilde eklenir.

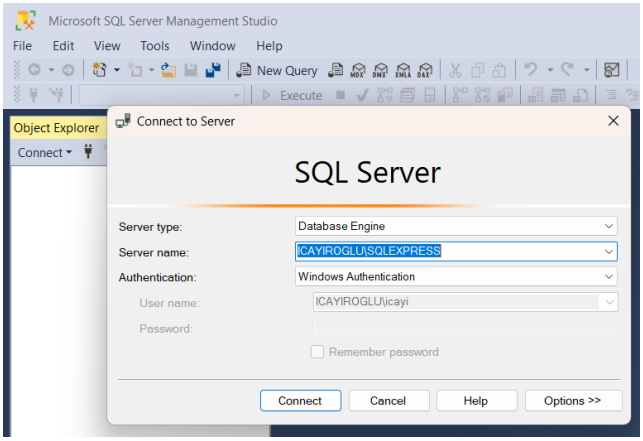
D. Context Sınıfının Oluşturulması

Bunların haricinde bize özel bir Model sınıfı daha ihtiyaçtır. Bu sınıflarımızı kullanarak Veritabanındaki hangi tabloları oluşturacağımızı gösteren bir sınıftır. İçerisinde **Context** sınıfı (Veritabanın tamamına karşılık gelir) ve **DbSet** olarak bilinen property ler (tablolara karşılık gelir) vardır. Context sınıfını kullanarak hangi modellerin hangi veritabanı tablosunu oluşturacağını belirleriz. Context sınıfı içerisinde bazı tabloları aşağıdaki şekilde tanımlamış olalım. Bir projede çok daha fazla sayıda tablo olabilir. Örnek olması için birkaç tanesini aşağıda gösterdik. Bizim Context ismini verdiğimiz sınıf adı aslında DbContext yapısını kullanmaktadır. Yani orada hazırlanan özellikleri davranışları bu sınıfta miras almaktadır.

```
using Microsoft.EntityFrameworkCore;

namespace KbuMudek.Data
{
    public class Context:DbContext
    {
        public DbSet<Fakulte> Fakulteler { get; set; }
        public DbSet<Bolum> Bolumler { get; set; }
        public DbSet<Program> Programlar { get; set; }
        public DbSet<Personel> Personeller { get; set; }
        public DbSet<Ogrenci> Ogrenciler { get; set; }
    }
}
```

E. Context Sınıfı İçerisinde Bağlantı Adresini Tanımlama

Veritabanına bağlantı kurmak için **Context** sınıfı içerisinde adresini vermemiz gerekiyor. Bunun için **DbContext** sınıfı içinde bulunan **OnConfiguring** metodunu yeniden tanımlayarak kullanacağız. Yani üzerine onu ezecek şekilde (**override** = ezme olarak tercüme edildi) yeniden tanımlayacağız. **OnConfiguring** metodu bizden **DbContextOptionsBuilder** tipindeki **optionsBuilder** parametresini isteyecektir. Bu parametre içerisinde Veritabanı adresimizi verebiliriz. SQL Server programını açtığımızda (Sql Server Management Studio programı bu servere açan programın adıdır. Bu program da yüklü olmalıdır) burada geçen **Server Name = ICAYIROGLU\SQLEXPRESS** ifadesi birim Server'ın adı olmuş olacak.



Aşağıdaki gibi bağlantı adresinin optionsBuilder parametresini kullanarak veriyoruz. Burada **server** bizim bilgisayarımız da kurulu olan SqlServer'ın adıdır. Yukarıdaki gibi adını alıyoruz. KbuMudek yazan yer ise bu server içerisinde oluşturacağımı **database**'in adıdır. Bu ismi biz belirliyoruz. **integrated security**=true ifadesi ise Windows Authentication (Windows'un kimlik doğrulamasını) kullanacağımızı bildirir. Eğer integrated security=false dersek yada hiçbir şey yazmazsak bu sefer Windows'un kimlik doğrulamasını kullanamayacağından bizden kullanıcı adı ve şifre isteyecektir. Bunu bu link içerisinde vermemiz gerekir. Bu database'i açarken bu şifreyi kullanacaktır. Yani bu ifade şu şekilde olur.

```
optionsBuilder.UseSqlServer("server=ICAYIROGLU\\SQLEXPRESS; database=KbuMudek; integrated security=false; user id=ali; password=1234");
```

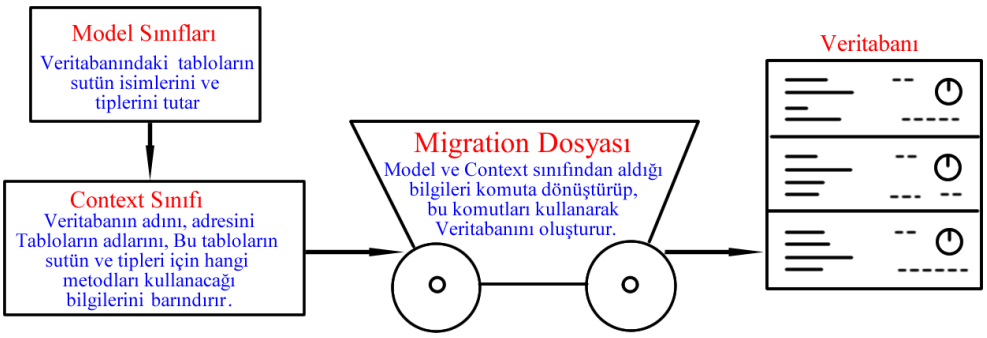
```
using Microsoft.EntityFrameworkCore;
using WEB_UYGULAMASI_B.Models;

namespace KbuMudek.Data
{
    public class Context:DbContext
    {
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer("Server=ICAYIROGLU\\SQLEXPRESS; Database=ProjeB; Integrated Security=True; TrustServerCertificate=True;");
        }


        public DbSet<Fakulte> Fakulteler { get; set; }
        public DbSet<Bolum> Bolumler { get; set; }
        public DbSet<Hoca> Hocalar { get; set; }
        public DbSet<Ogrenci> Ogrenciler { get; set; }

    }
}
```

Böylece Context sınıfı içerisinde veritabanı adresimizi tanımlamış olduk.

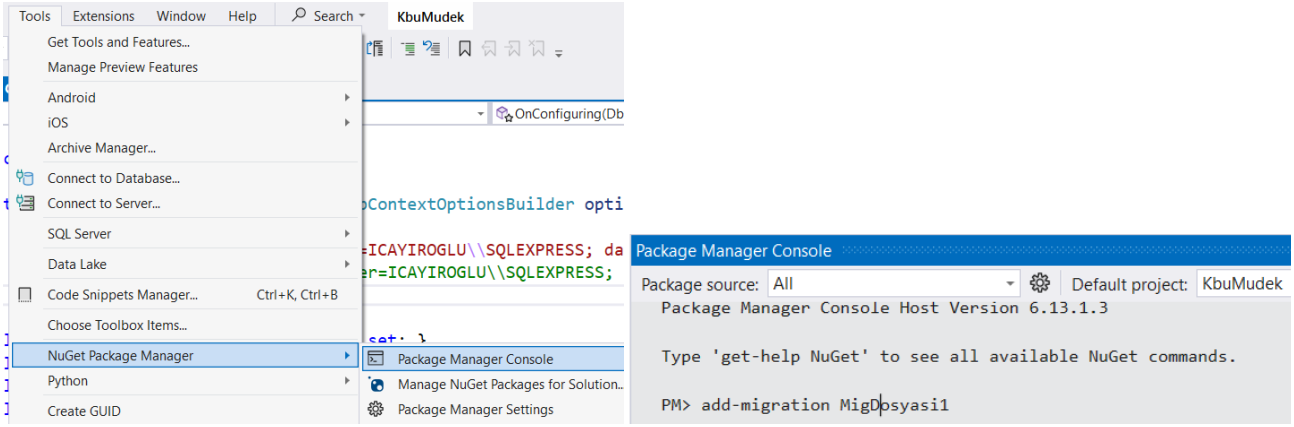
F. Migration Dosyasını Oluşturma

Oluşturduğumuz Context sınıfı içinde kullandığımız DbSet ile veritabanında kullanacağımız tabloları (Örnek: Fakulteler) ve bu tabloları oluştururken gerekli olacak sütun adlarını ise içerisinde belirttiğimiz Model isimleri ile verdik (Örnek: <Fakulte>). Peki bu bilgileri kullanarak Sql Server da tabloları nasıl oluşturacağız. Bu bilgileri kullanarak veritabanını oluşturacak komut ve yapıların içerisinde olduğu ayrı bir dosyaya ihtiyacımız vardır. Bu dosyaya Migration Dosyası diyoruz. Bu dosya Contex ve Model sınıfklarındaki bilgileri Veritabanına

taşıyacak (göç ettirecek=migration yapacak) özel bir dosyadır. Bu dosyanın içeriğini program bize kendisi otomatik oluşturuverecek. Ardından bu dosyayı çalıştırdığımızda Veritabanı yapısı otomatik oluşmuş olacak.



Migration oluşturmak için “Package Manager Consule” penceresini açalım. Bu pencerede “add-migration migdosyasi1” ismini vererek ilk migration dosyamızı oluşturalım.

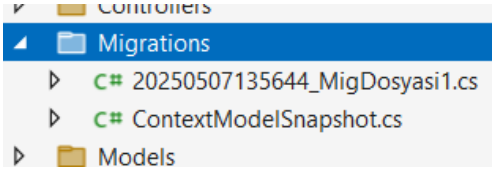


Entera bastığımızda “Build started” diyerek migration oluşturmayı başlatacaktır. Eğer Model sınıflarımızda ve Context sınıfımızda hatalı yada eksik bir tanımlama yapmadıysak Migration dosyası oluşacaktır. Aşağıda “KullaniciRol” metodunda Primary key tanımlanmadığından hata verdi. Bu hatayı düzeltip tekrar deneyelim.

```
PM> add-migration MigDosyasi1
Build started...
Build succeeded.
Unable to create a 'DbContext' of type 'Context'. The exception 'The entity type 'KullaniciRol' requires a primary key to be defined. If you intended to use a keyless entity type, call 'HasNoKey' in 'OnModelCreating'. For more information on keyless entity types, see https://go.microsoft.com/fwlink/?linkid=2141943.' was thrown while attempting to create an instance. For the different patterns supported at design time, see https://go.microsoft.com/fwlink/?linkid=851728
PM>
```

Hatayı düzeltip tekrar denediğimizde aşağıdaki gibi bir sonuç alırız. Projemiz içerisinde Migrations klasörünün ve içerisinde dosyaların oluştuğunu görebiliriz.

```
PM> add-migration MigDosyasi1
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
PM>
```



MigDosyasi1 açtığımızda içerisinde tüm tabloları oluşturacak tanımlamaları görebiliriz.

```
KbuMudek.Migrations.MigDosyasi1                    Up(MigrationBuilder migrationBuilder)
migrationBuilder.CreateTable(
    name: "Dersler",
    columns: table => new
    {
        DersID = table.Column<int>(type: "int", nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        ProgramID = table.Column<int>(type: "int", nullable: false),
        DersKodu = table.Column<string>(type: "nvarchar(max)", nullable: false),
        DersAdi = table.Column<string>(type: "nvarchar(max)", nullable: false),
        DersinTuruID = table.Column<int>(type: "int", nullable: false),
        Teorik = table.Column<int>(type: "int", nullable: false),
        Uygulama = table.Column<int>(type: "int", nullable: false),
        Kredi = table.Column<int>(type: "int", nullable: false),
        Akts = table.Column<int>(type: "int", nullable: false),
        AcildigiDonem = table.Column<int>(type: "int", nullable: false),
        OğretimElemaniID = table.Column<int>(type: "int", nullable: false)
    },
```

Şu ana kadar daha veritabanını oluşturacak dosyayı oluşturduk. Şimdi bunu çalıştırarak Veritabanımızı ve içerisindeki tabloları oluşturalım.

G. Veritabanını Oluşturma

Migration dosyamızı “Package Manager Console” penceresinde veritabanını oluşturması için çalıştıralım. Consol satırına “update-database” komutunu yazıp entera tıklayalım. Server ile bağlantıyı kurdu fakat giriş işlemi esnasında kullanılan sertifika zincirinde bir hata aldık.

```
PM> update-database
Build started...
Build succeeded.
Microsoft.Data.SqlClient.SqlException (0x80131904): A connection was successfully established with the server, but then an error occurred
during the login process. (provider: SSL Provider, error: 0 - Sertifika zinciri güvenilmeyen bir yetkili tarafından verildi.)
```

```
Error Number:-2146893019,State:0,Class:20
A connection was successfully established with the server, but then an error occurred during the login process. (provider: SSL Provider,
error: 0 - Sertifika zinciri güvenilmeyen bir yetkili tarafından verildi.)
PM>
```

Bu hatanın sebebi; ASP.NET Core uygulama, SQL Server'a bağlanırken şifreli (SSL) bağlantı kullanmak istiyor, ancak SQL Server'ın sunduğu sertifika güvenilir değil ya da sistem tarafından doğrulanamadığından hata oluşuyor. Bu hatayı çözmek için bağlantı adresinin içerisine “**TrustServerCertificate=true**” ifadesini ekleyelim. Bunun anlamı "sertifika güvenilir olmasa bile bağlanmaya çalış" anlamına gelecektir. Yeni bağlantı adresimiz şu şekilde oldu (Context sınıfı içerindedir).

```
optionsBuilder.UseSqlServer("server=ICAYIROGLU\\SQLEXPRESS; database=KbuMudek; integrated security=true; TrustServerCertificate=true");
```

Yeni durumda Migration çalıştı. Gidip Veritabanı ve tabloları oluşmuş mu kontrol edelim.

```
PM> update-database
Build started...
Build succeeded.
Acquiring an exclusive lock for migration application. See https://aka.ms/efcore-docs-migrations-lock for more information if this takes
too long.
Applying migration '20250507135644_MigDosyasi1'.
Done.
PM>
```

“KbuMudek” veritabanımız oluşmuş. İçerisindeki tablolara baktığımızda bu tablolarında oluştuğunu görüyoruz.