**ÇOK KATMANLI MİMARİ YAPISI**

İçerisinde bulunduğumuz veya kendimiz ilerlettiğimiz yazılım projelerimizin ölçeği küçük olduğu sürece projeye hakimiyet kolay olduğundan projenin mimari yapısını / tasarımını pek önemsemeyiz. Ancak proje büyüyüp dallanıp budaklandığı zaman işler değişir. Kodlar birbirine girer, kodları takip etmek zorlaşır, hata ayıklamak git gide bir eziyet halini alır. Hele ki herhangi bir sistemi değiştirip yerine başka bir sistem koymak istediğinizde veya projenizin çok büyüdüğünü düşünüp başka yazılımcıları da projeye dahil etmek istediğinizde işler iyice içerisinden çıkılmaz bir hal alır. Projenin belli bir mimari yapıya oturtulması burada önem kazanır.

Çok katmanlı mimari yaklaşım da bu karmaşık yapıyı sorumluluklara , fonksiyonelliklere , İşlere ve görevlere göre birbirinden ayırarak daha basit ve yönetilebilir uygulamalar yapılmasını sağlar. Aslında çok katmanlı mimariyle yapılmak istenen, kendi içerisinde aynı amaca hizmet eden aynı mantık için oluşturulmuş kısımları ayrı bir katmanda (solutionda) bir araya getirmektir. Örneğin , veri tabanı bağlantısı için yazılan kodlar bir katmana , işin fonksiyonelliğini içeren kodlar başka bir katmana gibi.

Peki çok katmanlı mimari kullanmak bize ne gibi avantajlar sağlar ?

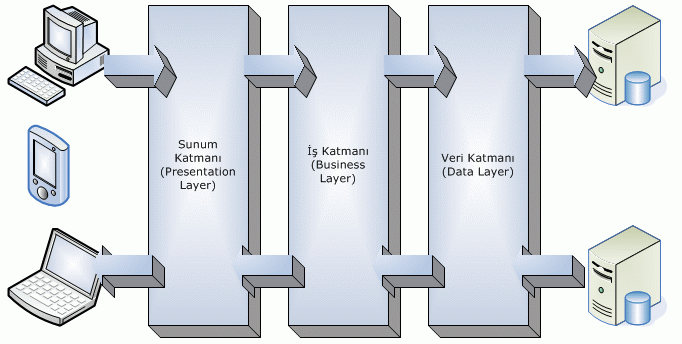
* Kompleks ve gitgide karmaşıklaşan projelere daha kolay hakimiyet.
* Hata ayıklamanın kolaylaşması.
* Projede ekip olarak çalışmayı kolaylaştırır.
* Değişim sürecini hızlandırır.
* Esnek bir yapı sağlar.

Çok katmanlı mimarinin dezavantajları ise

* Deployment süresinin uzunluğu

Çok katmanlı mimari yapısı oluştururken , katmanlar , katmanların yapısı ve sayısı projenin büyüklüğü ve ihtiyaçlara göre belirlenir. Ancak genel olarak kabul görmüş bir katmanlı mimari kalıbı vardır. Bu katmanlı mimari kalıbı üç katmandan oluşur.

* **Veri Katmanı (Data Layer) :** Veri tabanı bağlantı kodlarının bulunduğu katman.
* **İş Katmanı (Business Layer) :** Sunum katmanından gelen bilgileri gerekli koşullara göre işleyerek veya denetleyerek veri katmanının sağladığı metotları kullanarak veritabanına gönderen aynı şekilde veritabanından da bilgileri alarak gerekli süreçlerden geçirip presentation katmanına gönderen katman.
* **Sunum Katmanı (Presentation Layer) :** Kullanıcıyla direkt olarak etkileşimde bulunulan katman.



**1.Veri Katmanı(Data Katmanı)**

  Veri tabanı bağlantı kodlarının bulunduğu katman. Bu katmanda genellikle her veritabanına karşılık gelen veritabanı CRUD işlemlerini yapan kodlar vardır. Yani her veritabanına karşılık gelen Repository classları mevcuttur. Dolayısıyla birden fazla repository bulunmasıyla beraber bir de repositoryleri tek elden yönetmeyi sağlayan UnitOfWork classı bulunur. Bunlara ilaveten yapı olarak istenirse DbContext ile veritabanına bağlanan sınıf da bu katmanda ele alınabilir.  Bu katmanda yazılan veri erişim sınıfı biraz daha soyut ve daha generic bir yapıda olmasına dikkat edilmelidir.

Bu katmanın en önemli özelliği, bu katman veritabanına direkt olarak entegre olan veritabanıyla direkt konuşan katman olduğundan, kendisine gelen bilgileri veritabanına create,update,delete veya read işlemi yapacağıdan, bu bilgileri direkt olarak doğru ve uygun kabul eder.

Yani bu bilgilerin ilgili veritablosuna yapısal olarak uygunluğunu inclemek gerekli validasyonları yapmak veri katmanının görevi değildir. Diğer katmanlarda, veri katmanına bilgi ulaşana kadar gerekli validasyonlar yapılıp bilgi uygunluğu kontrol edilmiş olması gerekir.

Bu katman karşımıza DataLayer,DataAccessLayer gibi isimlerle de çıkabilir.

**2.İş Katmanı (Business/Implementation Layer)**

Sunum katmanından gelen bilgileri gerekli koşullara göre işleyerek veya denetleyerek veri katmanının sağladığı metotları kullanarak veritabanına gönderen aynı şekilde veritabanından da bilgileri alarak gerekli süreçlerden geçirip presentation katmanına gönderen katman. Veri katmanıyla sadece veriler uygulama ortamına çekilmiştir fakat üstünde yapılacak olan işlemler hala belli değildir. Bunun için çekilmiş olan verilerin yapılan uygulamaya uyarlama işlemi yerine getirilmelidir. Bu da iş katmanı (Business Layer) ile sağlanmaktadır. İş katmanı ile oluşturulan programa uyarlanmış veriler artık hazır hale gelmiştir.Ayrıca kullanıcı rolleri,yönetimi,yetkilendirmeler bu katmanda yapılan projelerin sorumluluğundadır.

**3.Sunum Katmanı (Presentation Layer)**

   Kullanıcıyla direkt olarak etkileşimde bulunulan katman. Bu katman veri katmanının direkt olarak veri tabanıyla bağlantıda olması gibi direkt olarak kullanıcıyla bağlantıdadır. Bu katman aracılığıyla kullanıcı tarafından girilen bilgiler alınır.

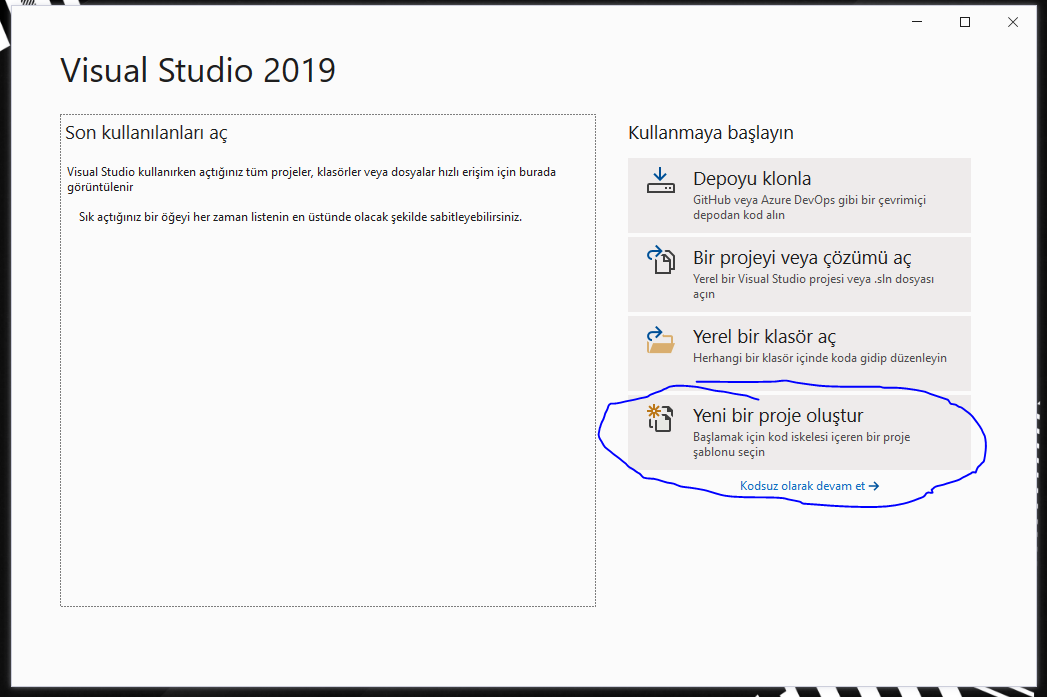
Sunum katmanı; Shared UI (User Interface) kodları, Code behind ve kullanıcıya bilgiyi göstermek için tasarımın(görselliğin) kullanılması gibi bileşenleri içerir. Normal şartlarda Burada hiçbir işlem yapılmaz. Yalnızca kullanıcıdan alınan istek parametreleri iş katmanına iletilir. İşlem sonunda da iş katmanından gelen sonuçları kullanıcıya sunar.  doğrudan veriye erişimi yoktur ve iş mantığı ile ilgilenmez.

Üç katmanlı mimarinin birbirileri arasında bir etkileşim olduğu gözükmektedir.Önce veri katmanı (Data Layer) ile veriler veritabanından çekilir, iş katmanı (Business Layer) ile bu verilerin uygunluğu sağlanır ve sunum katmanı (Presentation Layer) ile bir arayüz oluşturulup kullanıcıya veriler gösterilir. Bu işlem tam tersi yönde de yapılabilir, yani kullanıcı arayüz vasıtasıyla veri girişi yapar (Sunum katmanı), girilen veri veritabanına uygun hale getirilir (İş katmanı) ve son olarak veri tabanına aktarılır (Veri katmanı).

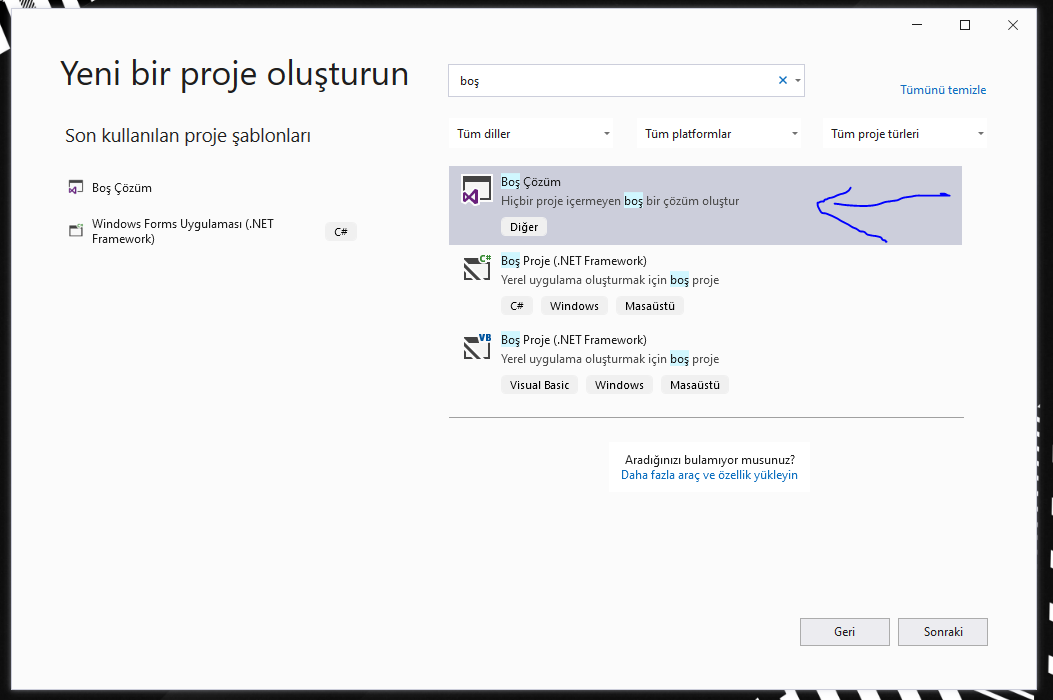
Elbette her fırsatta söylediğimiz gibi, katmanlı mimarideki katmanların yapısı ve oluşumu projenin ihtiyaçlarına göre belirlenir. İhtiyaçlar doğrultusunda proje katman sayısı artırılıp azaltılabilir.

**UYGULAMANIN YAPIM AŞAMALARI**

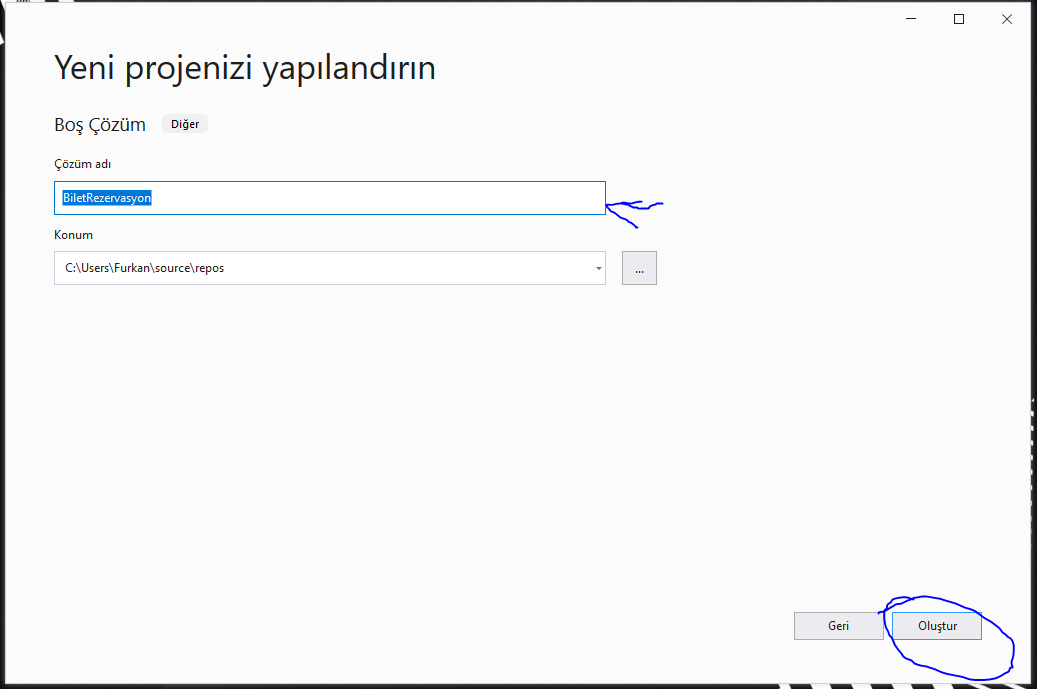
* 1. Adım: Visual Studio 2019 uygulamımızı açıyoruz daha sonra “Yeni bir proje oluştur” alanına tıklıyoruz.



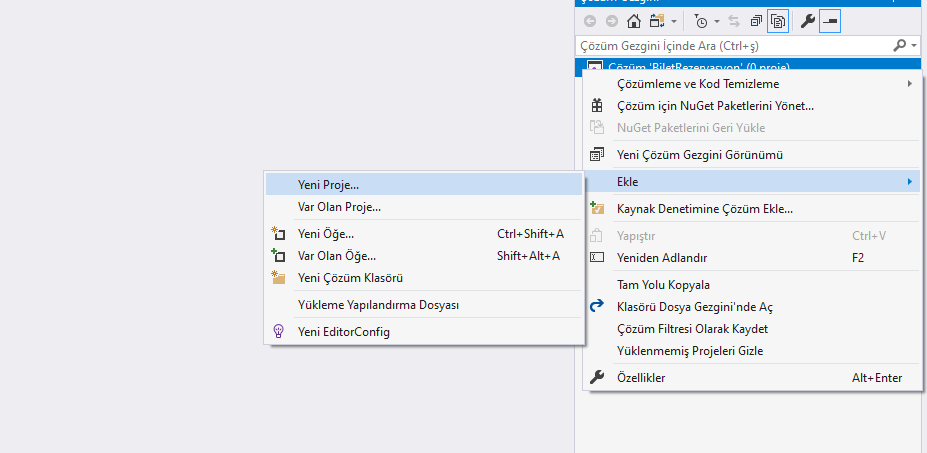
* 1. Adım: Uygulamamızı yazmaya başlamak için “Boş Çözüm” seçip “Sonraki” butonuna basıyoruz.



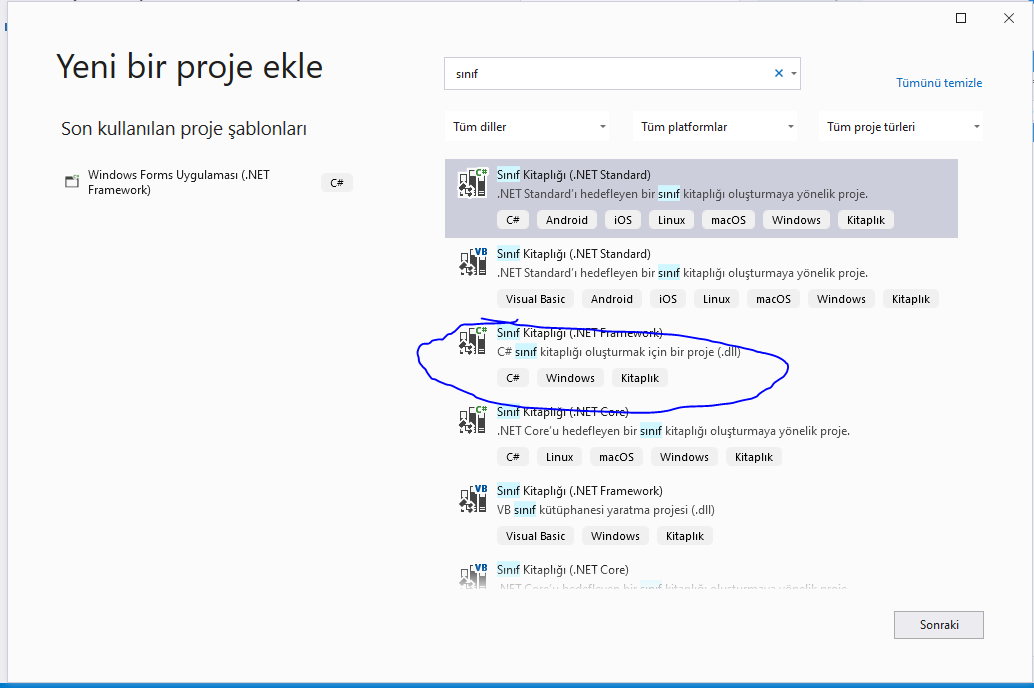
* 1. Adım: Çözüm adı alanına “BiletRezervasyon” yazıp “Oluştur” butonuna basıyoruz.



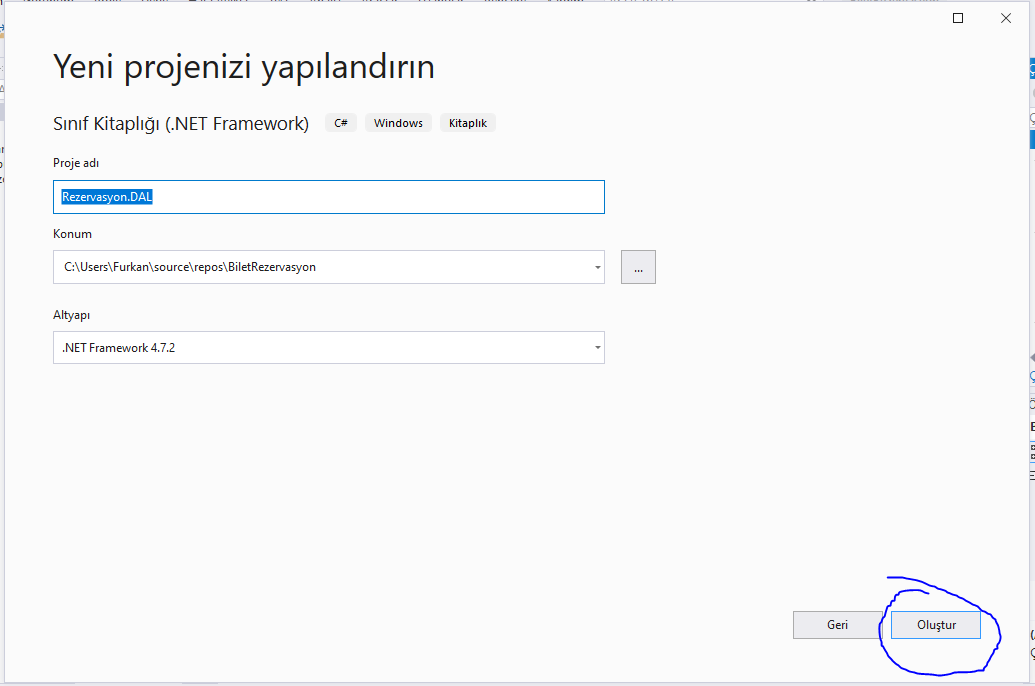
* 1. Adım: “Çözüm: ‘BiletRezervasyon’ “ ‘a sağ tıklıyoruz ve Ekle üzerine gelip Yeni Projeye Tıklıyoruz.



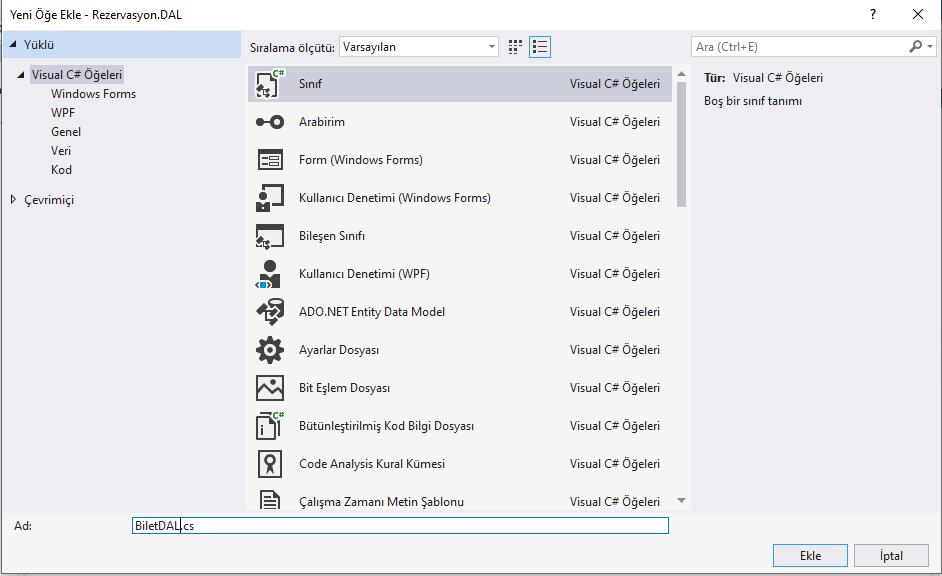
* 1. Adım: Aramaya “sınıf” yazıyoruz ve Sınıf Kitaplığı(.Net Framework) seçiyoruz ve Sonraki seçiyoruz.



* 1. Adım: Proje adı alanına “Rezervasyon.DAL” yazıp. Data Access Layer bölümünü tanımlıyoruz. İsimlendirme standartı önemli burada ben upper casing standartına uyacağım. Paket mantığını göz önünde bulundurarak oluşturmuş olduk.



* 1. Adım: Daha önceden oluşturmuş olduğumuz Tablomuzun Data Acces Layer işlemlerini yürütecek olan Class’ın tanımlamışını yapıyoruz.

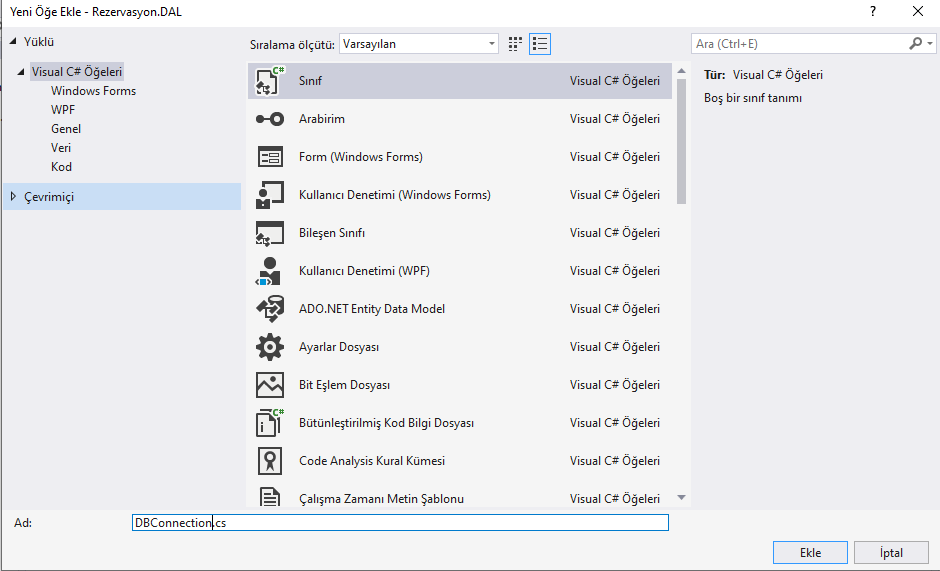


7.1 Ek: Daha Önce Oluşturduğumuz MS Access tablom:

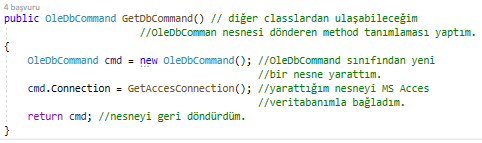
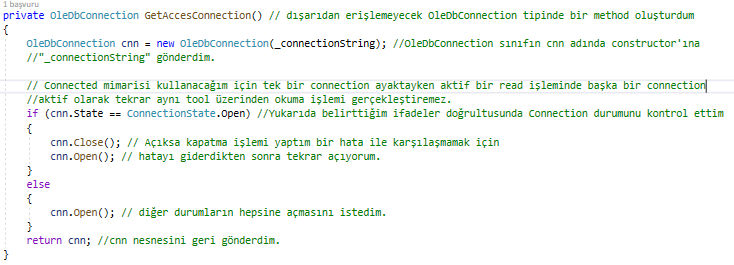
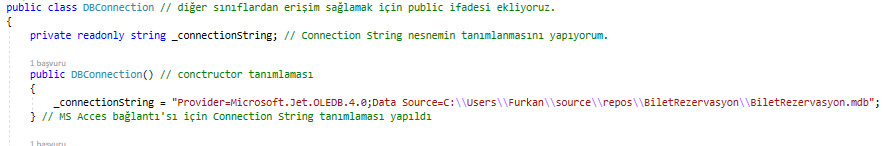
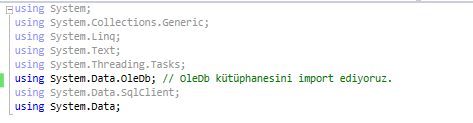




* 1. Adım: Data Access mekanizması olarak en ilkel yöntemlerden biri olan Ado.net connected kullanıcam. Bunun için MS Access’e bağlanmamızı sağlayacak açık bir command nesnesi verecek class yazmamız gerek bunun için. Yeni bir Class oluşturuyoruz adını “DBConnection.cs” Yapıyoruz

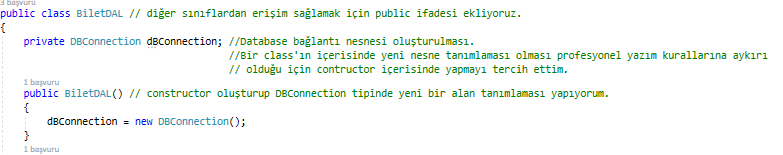
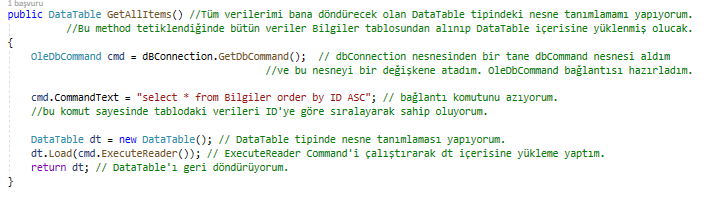
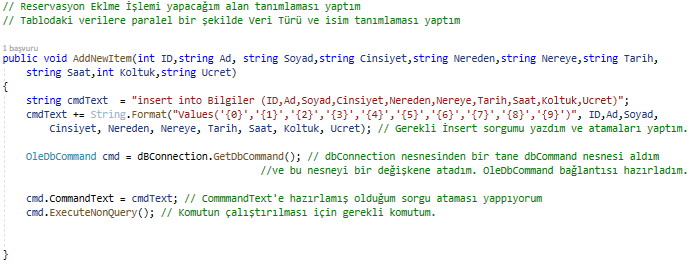
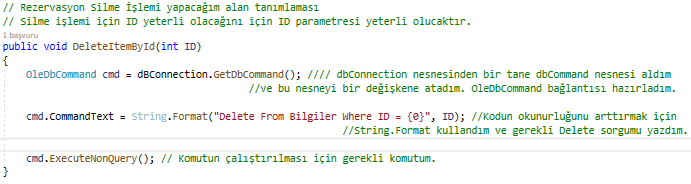
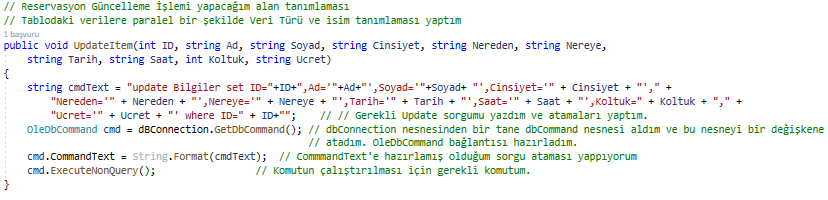


* 1. Adım: Daha önceki adımda yaratmış olduğum DBConnection Classının gerekli kodlarını yazıyoruz.

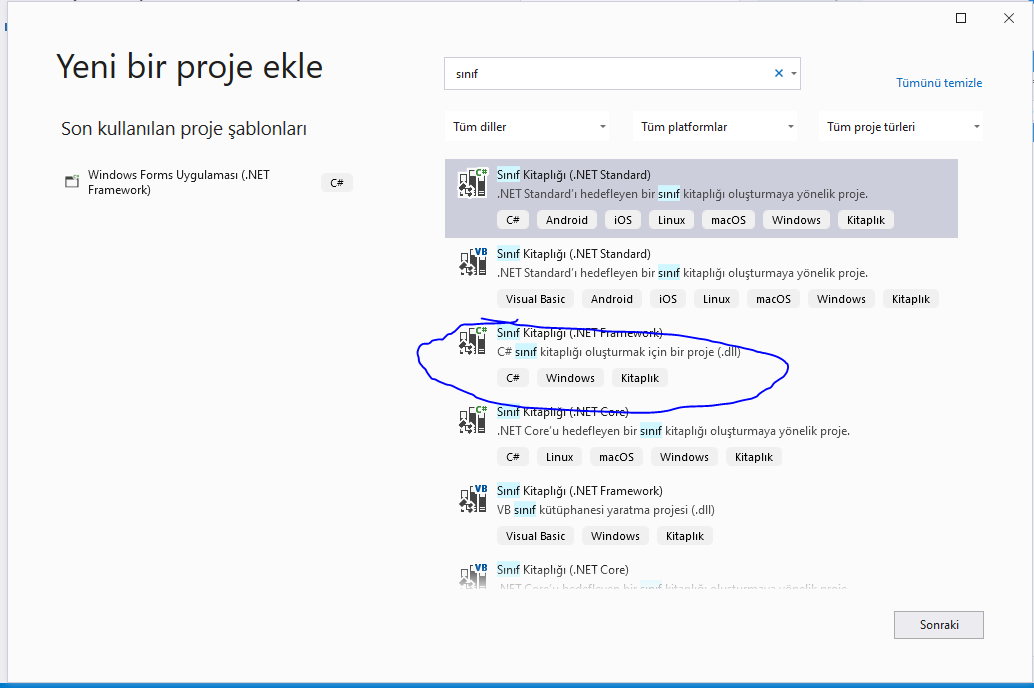


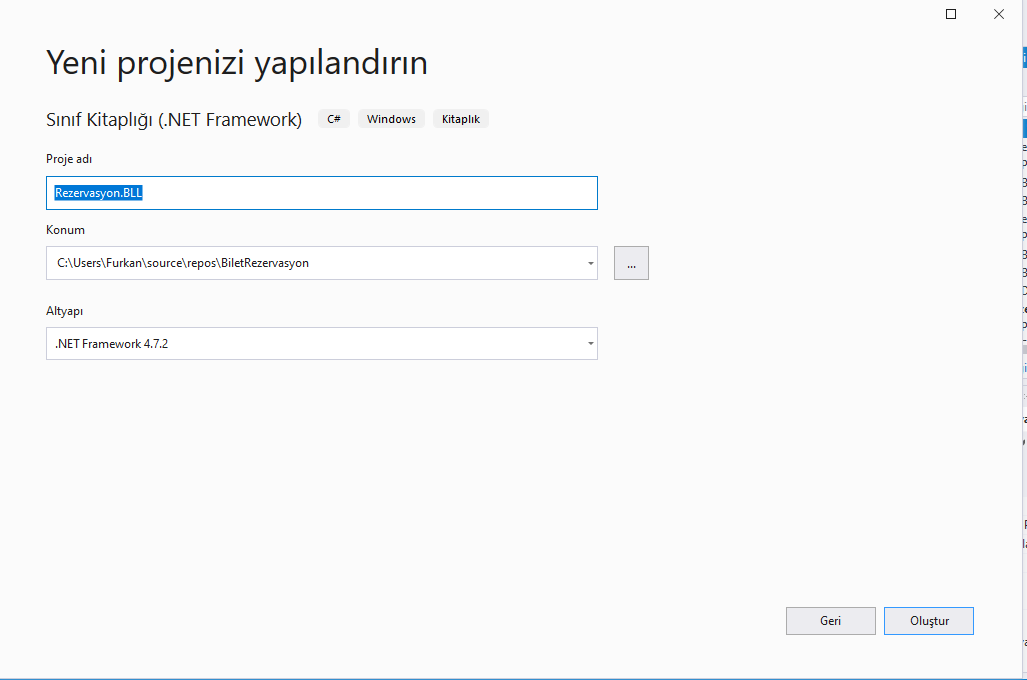
* 1. Adım: Daha önce tanımlamış olduğumuz BiletDAL.cs’e giriş yapıyoruz. Burada repository mantığı kullanacağım. Bilet Rezervasyon’u ile ilgili ekleme silme güncelleme gibi yapıları burada oturtacağım. DataTable kullanarak verilerime çekmeye çalışacağım.

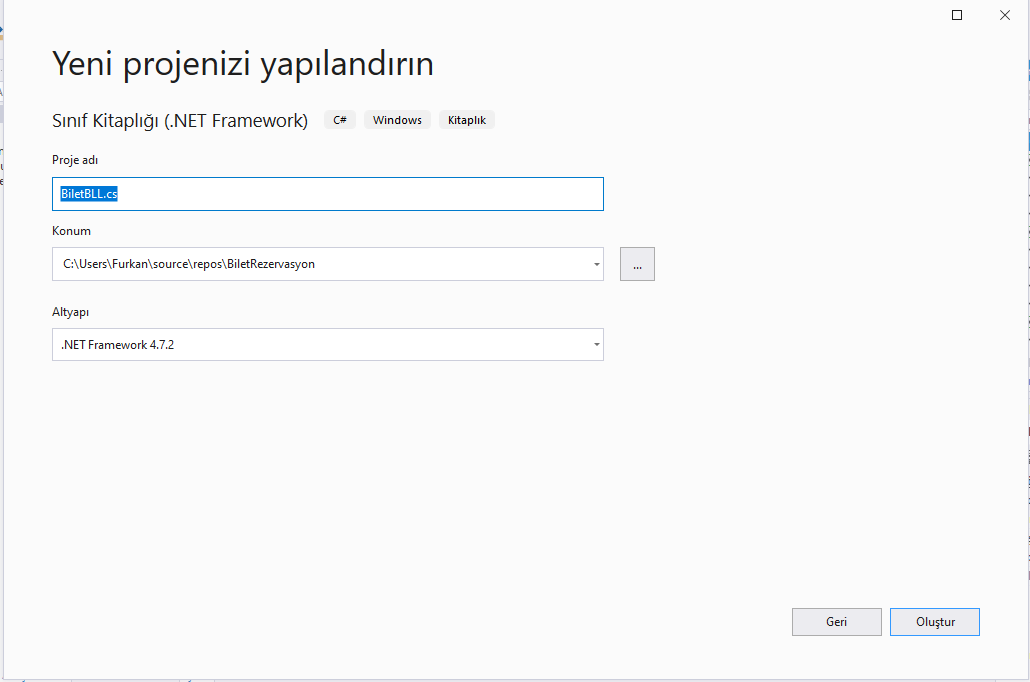
using System.Data.Oledb; // import etmeyi unutmayalım.

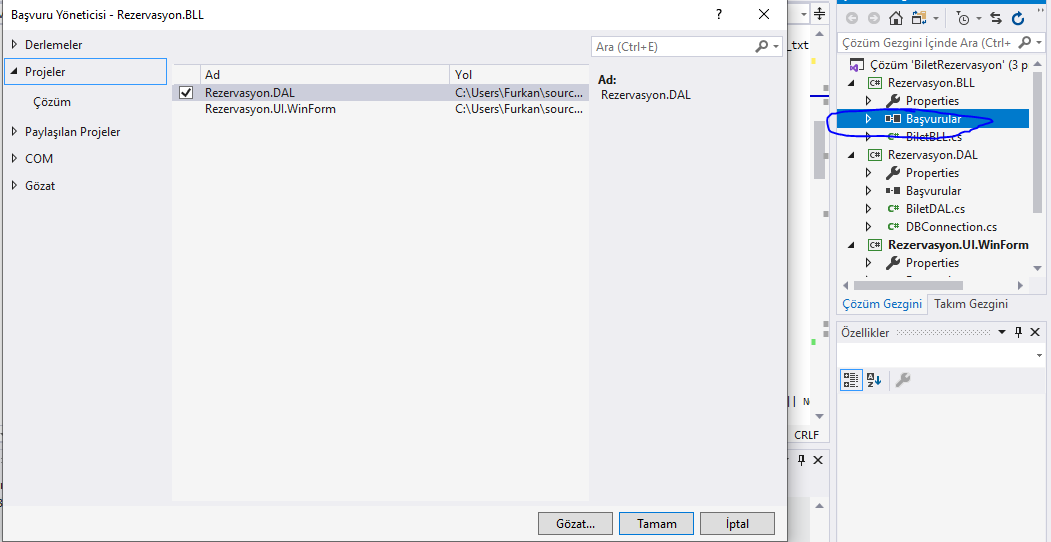


* 1. Adım: Aramaya “sınıf” yazıyoruz ve Sınıf Kitaplığı(.Net Framework) seçiyoruz ve Sonraki seçiyoruz.

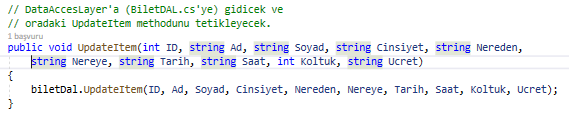
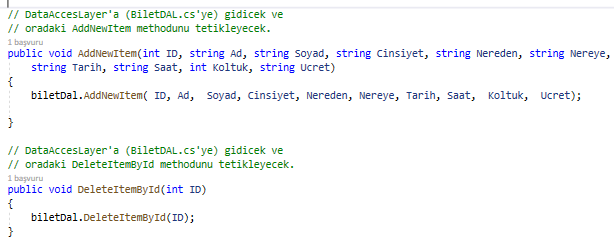
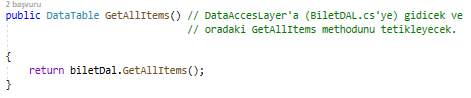
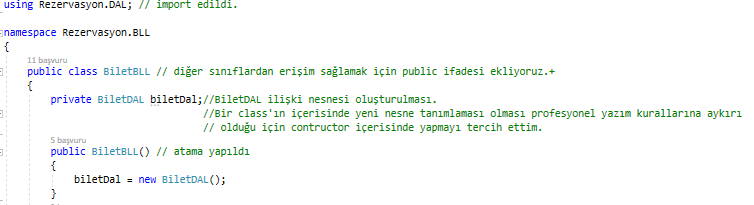
* 1. Adım: **Business Logic Layer** katmanımz olan “Rezervasyon.BLL” adını yazıp oluştur butonuna basıyoruz.



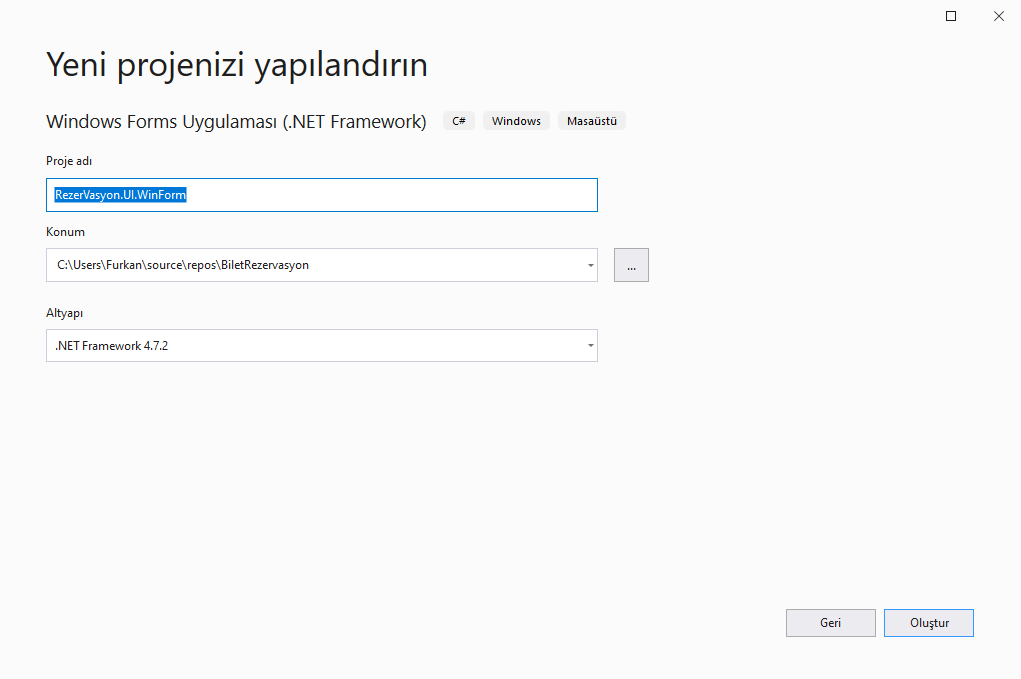
* 1.  Adım: Rezervasyon.BLL’e sağ tıklayıp altına Rezervasyonun mantıksal işlerine halledicek olan “BiletBLL.cs” adını yazıyoruz ve .cs oluştuyoruz.. BiletBLL.cs ‘de temel Data Access repository karşılayacak ve bunların kontrolünü yapacak methodlar olacak.
  2. Adım: BiletBLL burada gidikcek BiletDAL’a gidicek ve orada tetiklenilmesini istenen methodu tetikleyecek. Ama ilk önce o BiletDAL’ı referans almamız gerekiyor. Çünkü bir proje başka bir projenin içerisinde kendisine izin verilen yani enkapsülasyon yapısının dışında kalıp kendisine izin verilen memberları kullanmak için Referans alması gerekiyor. Yani BLL –> DAL’ı referans alıcak.



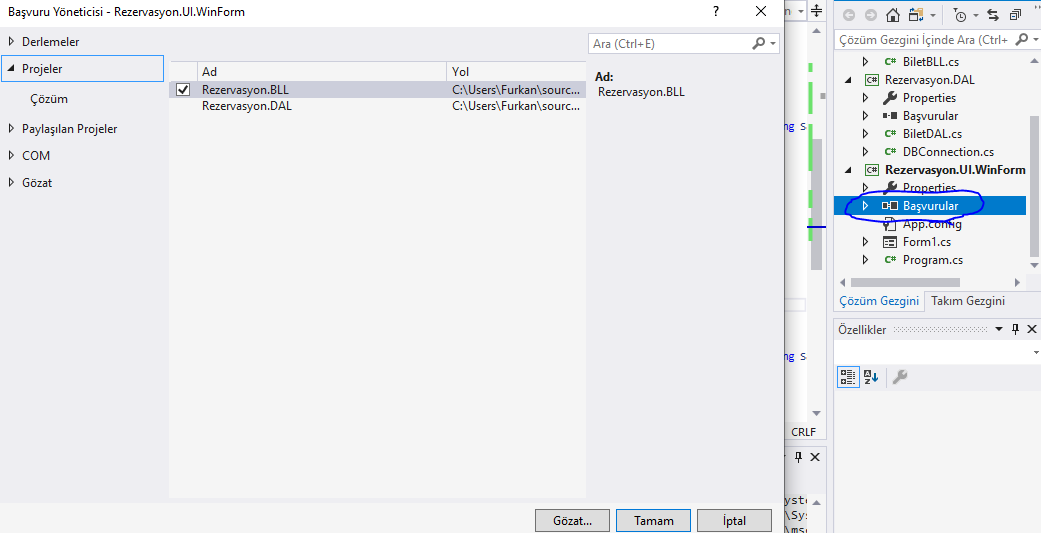
* 1. Adım : BLL içerisindeki kodların yazımı.



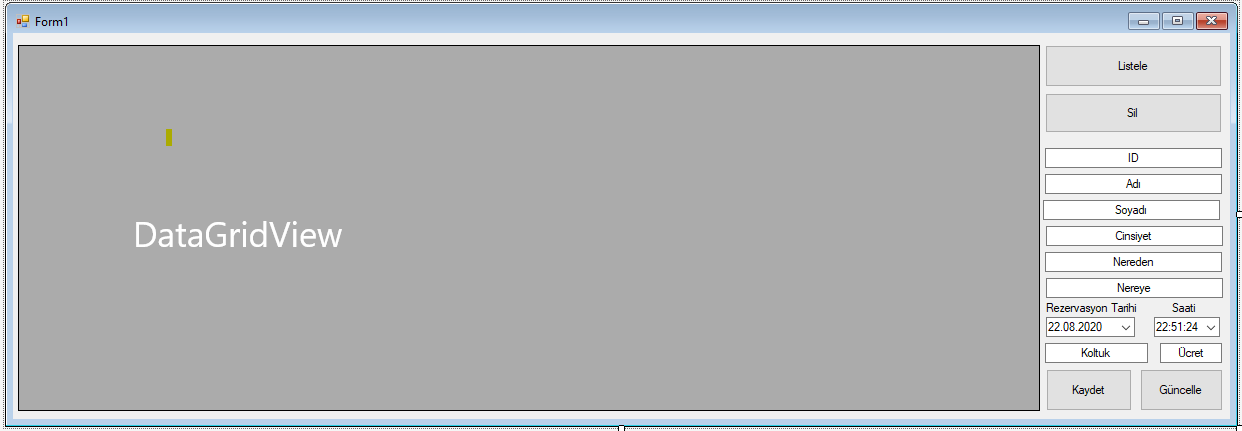
* 1. Adım: Son Katmanımız olan UI katmanının oluşturulması. BiletRezervasyon Çözüm’e sağ tıklıyoruz Ekle->Yeni Proje->Windows Forms Uygulaması seçip Proje adı alanına “Rezervasyon.UI.WinForm” yazıp oluştur diyoruz.



* 1. Adım: Hiyerarşi’ye göre sadece BLL ile haberleşmemiz gerek. Bu yüzden Bilet.BLL’i - > UI’a referans almalıyız.



* 1. Adım: Windows Form Tasarımı yaptım.



* 1. Adım: Form.cs kodlaması yapılması.

Uygulama tasarımını yaptıktan sonra ekrandaki elementslerden herhangi birine tıklayarak Form.cs ekranına ulaşabiliriz.

