MERSİN ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ
155-7001 BİTİRME ÖDEVİ - 1

1. Hafta Raporu

Proje Konusu: Akademisyen ve öğrenci arasında ödev ve proje takibini yapan, verilen tarihler arasında ödevin/projenin teslim edilmesi durumunda notlandırmaya izin veren, teslim edilmemesi durumunda not sistemini kapatan, duyuru, sınav notu paylaşımı yapılan, öğrencinin sadece danışmanıyla iletişime geçebileceği web yazılımı.

Geliştiren

İsmet KONUÇ – 17155035

Kullanılacak Teknolojiler

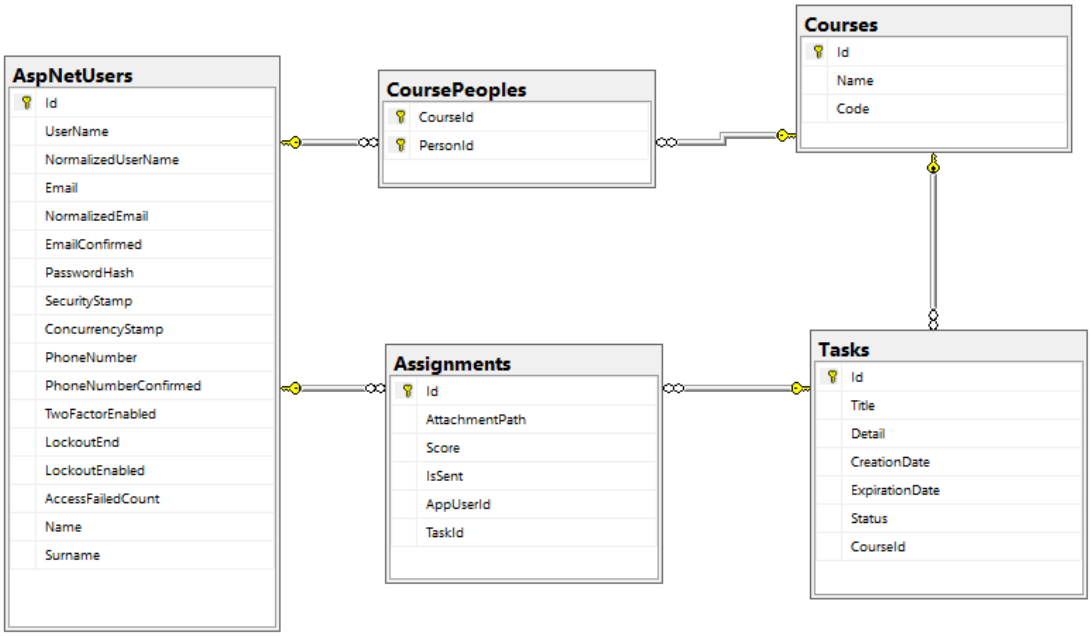
ASP.NET Core 3.1 – Back-End
Microsoft SQL Server – Veri Tabanı
JavaScript – Front-End
Bootstrap – Front-End

Kullanılacak Araçlar

ASP.NET Core Identity Framework – Kullanıcı ve Rol Bazlı İşlemler İçin
Entityframework – ORM

Visual Studio 2019 – Geliştirme Aracı
Visual Studio Code – Geliştirme Aracı

Yaptığım projenin genel adı Learning Management System (LMS). Başta Üniversitemizin kullandığı Moodle sistemi olmak üzere bu kategorideki projeleri inceleyerek veri tabanı tasarımını oluşturmaya çalıştım. Veri tabanını oluştururken Code-First olarak ASP.NET Core tarafında kodlayarak oluşturdum. Ardından Migration kodlarıyla MSSQL tarafında tabloların oluşmasını sağladım.



Görsel 1.1: Veri Tabanı Tasarımı

AspNetUsers Tablosu: Kullanıcı giriş sistemi ver rol sistemi olacağı için ASP.NET Core tarafından sunulan Identity Framework paketinden yararlandım. Bu paket aracılığıyla, Authentication, Authorization ve Role bazlı işlemler hazır olarak sunulduğu için zaman ve güvenlik yönünden faydalı olabileceğini düşündüm. Veri Tabanı tasarımında yer alan AspNetUsers tablosu bu paket tarafından hazır olarak sunuluyor.

CoursePeoples tablosu: Courses tablosu ile AspNetUsers tablosunu birbirine bağlayarak aralarında Many to Many ilişkisi kurulmasını sağlıyor. Bu sayede bir öğrencinin/akademisyenin birden fazla Ders seçebilmesinin yanında bir Dersin birden fazla öğrencisi/akademisyeni olabilmesini sağladım.

Courses Tablosu: Temel olarak bir kursun adını, kodunu, o kursa katıloan kullanıcıları ve o kursa ait açılan konuları içeriyor.

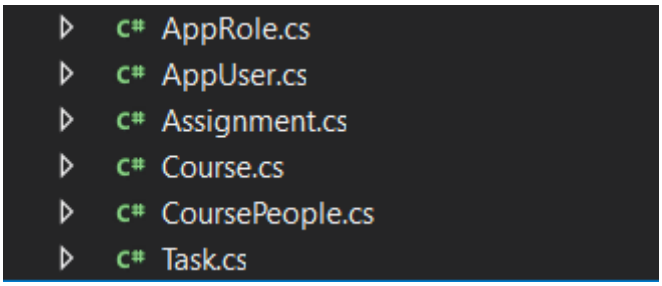
Tasks Tablosu: Başlık, Detay, Oluşturulma Tarihi, Son Teslim Tarihi, Durum (Görevin teslim edilme durumu) ve ait olduğu dersin id'si ve bu görevde verilen yükleme dosyasını içerecek Assignmentlardan oluşuyor.

Assignments Tablosu: Verilen görevde yüklenen dosyanın yolu, o öğrenciye verilen puanı, gönderim durumunu, görevi teslim eden öğrencinin id'sini ve ait olduğu taskid'yi barındırıyor.

Veri tabanı tasarımını tam anlamıyla bitirmedim. Sistemde “Öğretim Elemanı ile İletişime Geçebilme” ve “Not Yayınlama” gibi özellikler bulunacak. Bunların tasarımına tam anlamıyla karar veremediğim için proje ilerledikçe bu tabloları da geliştireceğim.

Veri Tabanının Code-First ile Oluşturulması

Code-First yaklaşımında her tablo bir class tarafından temsil edilirken her property ise bir veri tabanı kolonu tarafından temsil edilir.



Görsel 1.2: Tabloları temsil eden C# Classları

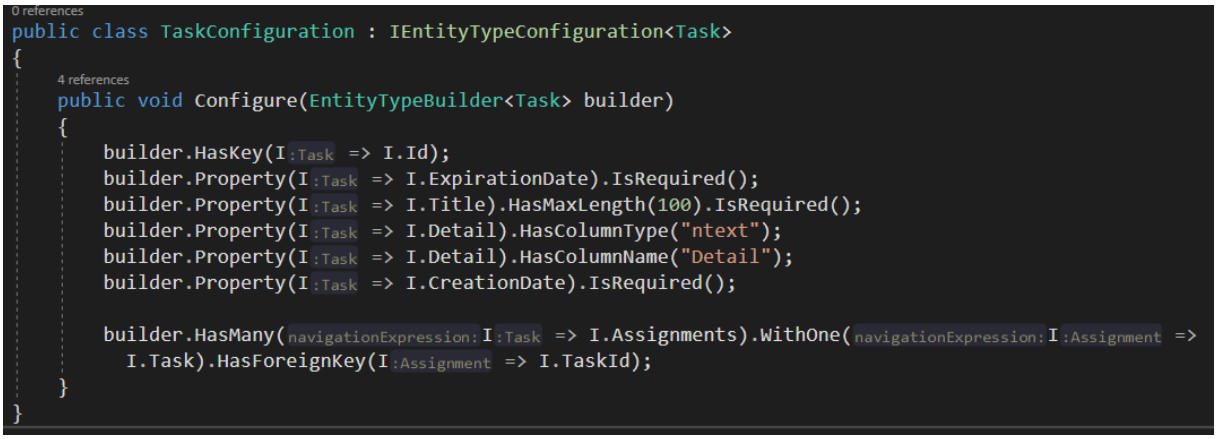
```
29 references
public class Task : ITable
{
    2 references
    public int Id { get; set; }
    1 reference
    public string Title { get; set; }
    2 references
    public string Detail { get; set; }
    1 reference
    public DateTime CreationDate { get; set; } = DateTime.Now;
    1 reference
    public DateTime ExpirationDate { get; set; } = new DateTime(year:2100, month:12, day:31);
    1 reference
    public Status Status { get; set; } = Status.Active;


    2 references
    public List<Assignment> Assignments { get; set; }


    2 references
    public int CourseId { get; set; }
    1 reference
    public Course Course { get; set; }
}
```

Görsel 1.3: Veri Tabanı kolonlarını temsil eden C# propertyleri

Bu şekilde veri tabanının tabloları ve kolonları C# Classları tarafından temsil edilir daha sonra bu classları Veri Tabanı tarafında oluşturmak için ise EntityFramework Core adlı ORM aracından yararlanılır.



```
0 references
public class TaskConfiguration : IEntityTypeConfiguration<Task>
{
    4 references
    public void Configure(EntityTypeBuilder<Task> builder)
    {
        builder.HasKey(I:Task => I.Id);
        builder.Property(I:Task => I.ExpirationDate).IsRequired();
        builder.Property(I:Task => I.Title).HasMaxLength(100).IsRequired();
        builder.Property(I:Task => I.Detail).HasColumnType("ntext");
        builder.Property(I:Task => I.Detail).HasColumnName("Detail");
        builder.Property(I:Task => I.CreationDate).IsRequired();

        builder.HasMany(navigationExpression:I:Task => I.Assignments).WithOne(navigationExpression:I:Assignment =>
          I.Task).HasForeignKey(I:Assignment => I.TaskId);
    }
}
```

Görsel 1.4: Veri Tabanında tutulacak kolonların özelliklerinin ve o tablonun ilişkilerinin belirlenmesi

Her bir tablo için konfigürasyon ayarı geçilir. Bu sayede veri tabanı tarafında oluşturulacak olan kolonların özellikleri ve sınırları belirlenip o tablonunun diğer tablolarla kuracağı ilişkiler belirlenir.

Visual Studio ile Veri Tabanı Arasında İletişimin Kurulması

Entityframework Core adlı ORM aracı veri tabanı ile ilişki kurmamızı sağlar ve bu doğrultuda geliştirilmiş bir C# Class'ı vardır. Bu Class'ın adı DbContext'tir. Fakat projede IdentityFramework kullandığım için bu Class'a benzer IdentityDbContext adlı class geliştirilmiştir. Bu class aracılığıyla veri tabanı ile iletişim kurabilirken, IdentityFramework tarafından sunulan tabloları da (AspNetUsers - AspNetRoles) kullanabiliriz.

```
public class LmsDbContext : IdentityDbContext<AppUser, AppRole,
int>
    {
        protected override void
OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer("Server = DESKTOP-0GDA6G3;
Database = lms; User Id = sa; Password = 1;");
            base.OnConfiguring(optionsBuilder);
        }

        protected override void OnModelCreating(ModelBuilder
builder)
        {
            builder.ApplyConfiguration(new TaskConfiguration());
            builder.ApplyConfiguration(new CourseConfiguration());
            builder.ApplyConfiguration(new AppUserConfiguration());
            builder.ApplyConfiguration(new
CoursePeopleConfiguration());
            builder.ApplyConfiguration(new
AssignmentConfiguration());

            base.OnModelCreating(builder);
        }
```

```
        public DbSet<Task> Tasks { get; set; }
        public DbSet<Course> Courses { get; set; }
        public DbSet<AppUser> AppUsers { get; set; }
        public DbSet<CoursePeople> CoursePeoples { get; set; }
        public DbSet<Assignment> Assignments { get; set; }
    }
```

`OnConfiguring()` metodu ile MSSQL server ile iletişim kurulur. Ardından `OnModelCreating()` metodu ile yapılan konfigürasyon ayarları MSSQL tarafına iletilir bu sayede oluşturulacak olan tabloların sınırları çizlir ve ilişkileri kurulur. Yukarıda yer alan kodlarla da veri tabanı tablolarının isimleri ve karşılığında ait olduğu C# classları belirtilir.

Veri Tabanı Tarafına Sorguların Gönderilmesi

Veri Tabanına CRUD(Create-Read-Update-Delete) işlemleri yapılacağı için yalnızca bu özellikleri sunan C# classlarını geliştirdim. Veri Tabanındaki her tablo için bir repository class'ı oluşturdum. Bu sayede her tablo için CRUD işlemlerinin yanı sıra o tablo için gerekli olan verilerin getirilebilmesini amaçladım. Örneğin Course tablosu için `GetCourseTasks(int courseId)` ve `GetCourseWithId(int courseId)` metodlarını yazdım. Bu metodlar sayesinde bir kursa ait verilen görevler ve bir kursun tüm bilgileri getirilebilir.

```
1 reference
public class EfCourseRepository : EfGenericRepository<Course>, ICourseDal
{
    2 references
    public List<Task> GetCourseTasks(int courseId)
    {
        using var db = new LmsDbContext();

        return db.Courses.Single(I:Course => I.Id == courseId).Tasks;
    }

    2 references
    public Course GetCourseWithId(int courseId)
    {
        using var db = new LmsDbContext();

        return db.Courses.Single(I:Course => I.Id == courseId);
    }
}
```

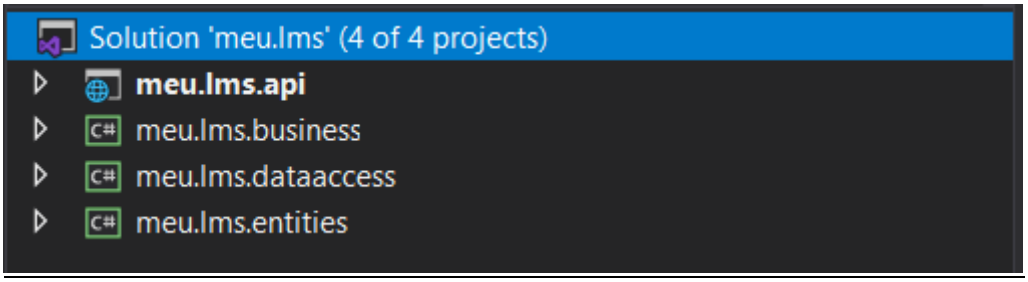
Görsel 1.5: EfCourseRepository.cs

Geliştirme Teknikleri – Design Patternlar – SOLID Prensipleri

Back-End ortamında geliştirme yaparken işlemlerin sürdürülebilir olması açısından SOLID Prensiplerini, Tasarım Kalıplarını ve N Katmanlı Mimariyi kullandım. Her bir tekniğin kendi bünyesinde getirdiği avantajlar mevcut.

N Katmanlı Mimari

Projede, N Katmanlı Mimari tekniğini kullanarak 4 adet katman oluşturdum.



Görsel 1.6 Projede Yer Alan Katmanlar

Entities Katmanı: Veri Tabanı tablolarına karşılık gelen sınıfların bulunduğu katman.

DataAccess Katmanı: Veri Tabanı ve Entites Katmanı ile iletişimde olan katman. Bu katman aracılığıyla Entities katmanında oluşturduğumuz veri tabanı tablolarını SQL Server ile iletişim kurarak veri tabanına yazdırdım ve yine veri tabanına sorgulama yapabilmek için bu katman içerisine çeşitli metotlar yazdım.

Business Katmanı: Data Katmanındaki metotları soyutlayan ve miras alan katman.

Api Katmanı: Projenin arayüzünü oluşturmak için istekte bulunulacak katman. Bu katman aracılığıyla client tarafından gelecek olan istekler entites ve dataaccess katmanlarına iletilecektir ve bu sayede arayüzü oluşturmak için gerekli veriler temin edilecektir. API sayesinde gömülü bir arayüz olmayıp ister Web ister Mobile uygulama rahatça geliştirilebilecektir.

Bu dört katman sayesinde karmaşıklığı azaltıp, her katmana yalnızca belirli işleri yükledim.

Dependency Injection

Proje büyüdükçe bağımlılıklar artar ve projedeki bazı bölgelerde değişiklik yapmak istediğimizde bir çok hatayla karşılaşabiliriz. Örneğin, Dependency Injection kullanmadan yazılan bir projede, veri tabanı teknolojisi değiştirilmek istendiğinde bir çok noktada hata alırız. Bunu önlemek açısından Dependency Injection ile projedeki bağımlılıkları azaltmayı hedefledim.

Repository Design Pattern

Bu tasarım kalıbı sayesinde veri tabanıyla iletişime geçen her servise aynı metotları yazmak yerine bunları soyutlayarak tek bir çatı altında toplayabiliriz. Bu sayede aynı işlemlere ihtiyaç duyan farklı servislere ortak bir servis yazarak (Generic Repository) kod tekrarına düşmeyi ve karışıklığı engellemeye çalıştım.

```
public class EfGenericRepository<Table> : IGenericDal<Table> where Table : class, ITable, new()
{
    4 references
    public void Save(Table table)
    {
        using var db = new LmsDbContext();

        db.Set<Table>().Add(table);
        db.SaveChanges();
    }

    4 references
    public void Delete(Table table)
    {
        using var db = new LmsDbContext();
        db.Set<Table>().Remove(table);
        db.SaveChanges();
    }

    4 references
    public void Update(Table table)
    {
        using var db = new LmsDbContext();
        db.Set<Table>().Update(table);
        db.SaveChanges();
    }

    6 references
    public Table GetTaskWithId(int id)
    {
        using var db = new LmsDbContext();

        return db.Set<Table>().Find(id);
    }

    4 references
    public List<Table> GetAllTasks()
    {
        using var db = new LmsDbContext();

        return db.Set<Table>().ToList();
    }
```

Görsel 1.7: EfGenericRepository.cs

Tüm tablolar tarafından sorgulama yapmak için kullanılabilecek metotları GenericRepository sınıfına yazarız ve bu tablolara ihtiyaç duyan her Repository servisi bu katmanı miras alır bu sayede kod tekrarı oluşmaz.

Özet

İlk hafta bu projenin iskeletini oluşturarak temel olarak veri tabanı tablolarını oluşturdum, veri tabanı ile iletişim kurdum çeşitli teknikler kullanarak projenin karmaşıklaşmasını engellemeye çalıştım. Gelecek haftalarda ise API katmanına yoğunlaşarak projeye fonksiyonellik katacağım.