SALES PROJE İNCELEMESİ

Sales projesi üzerinde çalışırken Category Controllerımızı oluşturduktan sonra boş olmayan bir View oluşturduk ve daha sonra hata aldık. Aynı projenin, düzeltilmiş ve çalışan haliyle, hatalı hali karşılaştırılıp farklılıklar incelenmiştir. Raporda, bu farklılıklar anlatılacaktır.

STARTUP

Anlatıma projemizin başlangıcı olan Startuptan başlanacak. ConfigureServices methoduna servisler eklenmektedir.

- ***services.AddDbContext<SalesDbContext>(options=>options.UseSqlServer(Configuration.GetConnectionString("SalesConn")));***

SqlServer bağlantımızın yapılabilmesi için bu servis ekleniyor. Appsettings.json'da yazılan connection stringimiz bunun sayesinde okunabilmekte ve sql ile bağlantı yapılabilmektedir. Bu her iki projede de bulunmaktadır. İncelenen projede fazladan 2 servis daha bulunmaktadır.

Bunlar;

- ***services.AddScoped<ICategoryRepository, CategoryDAL>();***
- ***services.AddScoped<IProductRepository ,ProductDAL>();***

Servisimizi dependency injection yöntemi ile controllerımızda kullanabileceğiz. Bunun için Controllerda kullanacağımız servisin nesnesinin private olarak tanımlanması gerekmektedir. Controller constructor methodunda ise inject edeceğimiz servisi tanımladıktan sonra, constructor üzerine gelen değeri private olarak yarattığımız nesne içerisine atamalıyız. Bu sayede en az bağımlılık ile seviş içerisindeki tüm methodlara ulaşabiliyoruz.

Bu gereklilikler incelenen projede CategoriesController ve ProductsControllerda yapılmıştır. Controllerların anlatıldığı kısımda gösterilecektir.

Modelin Oluşturulması

Her iki projede de tabloda görünecek olan özelliklerinin bulunacağı classlar daha sonra SalesDbContext oluşturulmuştur.

Entities klasöründe her bir varlık için class oluşturulup proportileri eklenir. SalesDbContextin EF Core içinde yer alan DbContext classından türetilmesi sağlanmıştır. Bu class aslında her bir varlık için DbSet bu eklemeler de yapıldıktan sonra kodun son hali aşağıda bulunmaktadır.

- ```
  public class SalesDbContext : DbContext
  {
      public SalesDbContext(DbContextOptions options):base(options)
      {
          //constructor oluşturuldu.
      }
      public DbSet<Category> Categories { get; set; }
      public DbSet<Product> Products { get; set; }
  }
  ```
  ```

Bu işlemlerin sonunda model hazırdır. Migration ve update işlemlerinden sonra veritabanı oluşmuş olacaktır.

Hatalı projemizde farklı olarak OnConfiguring metodu ile kullanılacak olan sql bağlantı ismi tanımlanmıştı.

- ***protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)***
  {
  optionsBuilder.UseSqlServer("Name=ConnectionStrings:SalesConn");
  }

IRepoistory

Önceki projelerde CRUD işlemleri için Controllerda bütün işlemleri uzun uzun yazıyorduk. Bunun önüne geçilebilmesi için IRepository interface'i oluşturulmuştur. Hatalı proje ile arasındaki tek fark sonuna new() eklenmesi.

- ***public interface IRepository<T> where T:class, new()***
  {
  void Add(T entity);
  void Update(T entity);
  void Delete(T entity);
  T GetSingle(int id);
  List<T> GetAll();
  }

BaseRepository

BaseRepository classımız IRepositoryden kalıtım almıştır. Add, Update, Delete gibi CRUD mothodlarımızın işlemlerini burada dolduruyoruz.

İncelenen projenin BaseRepositorysinde;

- ***protected readonly DbContext _context;***
- ***public BaseRepository(DbContext context)***
  {
  _context = context;
  }

Tanımlaması yapılmış. Hatalı projeden farklı olarak;

GetSingle methodu id ye göre bulunup getiriliyor. Ve Update methodu diğer yapılan projelerdeki gibi değil aşağıdaki kod kullanılmıştır;

- ***_context.Entry(entity).State = EntityState.Modified;***

Hatalı projede ise methodlar aşağıdaki gibi yazılmıştır.

- ```
  public void Add(T entity)
  {
          using (var _context = new SalesDbContext())
          {
              _context.Add(entity);
              _context.SaveChanges();
          }
  }
  ```

CategoryDAL

Projede CategoryDAL kısmına hiçbir şey yazılmamıştı ancak incelenen projede bu bölüme eklemeler yapılmış.

- ```
  public CategoryDAL(SalesDbContext context):base(context)
      {

      }
  ```

- ```
  private SalesDbContext Context
     {
         get { return _context as SalesDbContext; }
     }
  ```

- ```
  public List<Category> GetPopularCategories()
   {
         throw new NotImplementedException();
   }
  ```

ICategoryRepository

Bu şekilde interface oluşturulmamıştı. İncelenen projede ise oluşturulmuş ve GetPopularCategories methodu çağrılmış.

CategoryController

Controllerımızda Indeximiz aşağıdaki gibi yazılmıştı.

- ```
  public IActionResult Index()
   {
      var categories = new CategoryDAL();
     return View(categories.GetAll());
  }
  ```
  ```

var yazarak türü belirtmemiz gerekmemektedir**.** Nesneyi oluşturduğumuzda CategoryDAL çağrılacak fakat bizim CategoryDAL sınıfımız boş.

İncelenen projede;

- ***private readonly ICategoryRepository _categoryRepository;***
- ***public CategoriesController(ICategoryRepository categoriesController)***
  {
  _categoryRepository = categoriesController;
  }
- ***public IActionResult Index()***
  {
  return View(_categoryRepository.GetAll());
  }

Startup ile ilgili bölümde services.AddScoped<ICategoryRepository.CategoryDAL>(); servisinden dolayı nesnesinin private olarak tanımlanması gerekmektedir. Controller constructor methodunda ise inject edeceğimiz servisi tanımladıktan sonra, constructor üzerine gelen değeri private olarak yarattığımız nesne içerisine atamalıyız demiştik. İlk 5 satırdaki kod bunun için yazılmıştır. Index içerisinde GetAll methodu ile listeleme yapılacak.

Tüm bu işlemler bittikten sonra Indexin View'i oluşturulunca proje çalışmaktadır.