Onion Architecture Katmanları

Bu mirari Dependency Inversion ilkesine dayanır.Aşagıdaki Şekilde Gösterilmişitir.

1.Model Layers ( Model Katmanı )

2. Data Layer ( Data Katmanı )

3.Service Layer ( Servis Katmanı)

4.UI (Web) Layer ( UI Katmanı )

**Model Katmanı**

Model katmanı entity sınıflarımızı içeren bir katmandır. Her entity ayrı bir class dosyasında yazılır. Enum’lar da aynı şekilde ayrı dosyalarda yer alır. Entity’lerin neredeyse tamamının miras aldığı BaseEntity adında bir sınıf bulunur. Sadece many-to-many ilişkileri kurmaya yarayan ara entityler BaseEntity’i miras almazlar. Entitylerde Data Annotation attribute’ları kullanılmaz. Bu attribute’lar yerine Data katmanında Builder class’ları ile fluent-api kodları yazılacaktır.

**Data Katmanı**

Data katmanında model katmanında tanımlanan entitylerin veritabanı işlemlerini yapmaya yarayan repository’ler bulunur. Aynı zamanda projenin kalbi olan Unit Of Work, Generic Respository sınıf ve interface’leri de bu katmanda bulunur. Her repository için aynı .cs uzantılı dosyada hem interface hem de class tanımlaması yapılacaktır ve repository’ler eager loading yapabilmek için Entity Framework’teki Include metotlarını da destekleyecektir.

Data katmanında aynı zamanda DbContext’imiz ve Seed extension metodumuz da bulunmaktadır. DbContext’in çalışabilmesi için bu katmanı Entity Framework Core nuget pakedini yüklememiz gerekecektir.

**Service Katmanı**

Data katmanındaki repository’lerin bir ya da bir kaçını kullanan bussiness logic katmanımız olan service katmanında ProductService ya da CategoryService gibi class’lar tanımlanır. Bu serviceler genellikle kayıt ekleme, güncelleme, arama ve silme işlemleri yapmak için repository’leri kullanır ve değişiklikleri veritabanına aynı transaction ve context ile uygulayabilmek için unit of work’tan yararlanır.

**UI Katmanı**

MVC projemiz bu katmandadır. Startup sınıfında .NET Core ile birlikte gelen gömülü dependency injection özelliğinden yararlanarak DbContext ve Unit Of Work scoped service olarak eklenir. Repository ve Service’ler ise Transient service olarak eklenir.

# Onion Architecture Proje Yapısı

# İlk olarak File/New Project/Other Projet Types/Blank Solution Seçilik Proje adı verilir. ( Örn : OnionArchitectureSample )

# Oluşan Projeye Sağ tılayaıp Add/New Project – Class Library seçilir ve Projeadı.Model katmanı eklenir ( Örn : OnionArchitectureSample.Model)

# Oluşan Projeye Sağ tılayıp Add/New Project – Class Library seçilir ve Projeadı.Data katmanı eklenir ( Örn : OnionArchitectureSample.Data)

# Oluşan Projeye Sağ tılayıp Add/New Project – Class Library seçilir ve Projeadı.Service katmanı eklenir ( Örn : OnionArchitectureSample.Service)

# Oluşan Projeye Sağ tılayıp Add/New Project – .Net Core Web Application seçilir ve Projeadı.UI yada Projeadı.Wen katmanı eklenir ve Sonraki Pencerede Web Appication Seçlilir. Üyelik Sistemi varsa Change Authentication dan eklenir( Örn : OnionArchitectureSample.Web)

# 

# Onion Architecture Uygulanışı

# 1.Model Katmanı

Model katmanı , Mimarinin Çekirdek ve Merkez parçasıdır.İlk olarak Bu Katmanı oluşturmak için “OnionArchutectureSample.Model katmanını oluşturduk.”EntitiyFrameworkCore “ Ve “EntitiyFrameworkCore.sqlServer” yüklenir.

# Base Entity Oluşturulması

1. **public** **class** BaseEntity
2. {
3. **public** **long** Id { **get**; **set**; }
4. **public** DateTime AddedDate { **get**; **set**; }
5. **public** DateTime ModifiedDate { **get**; **set**; }
6. **public** **string** AddedBy { **get**; **set**; }
7. **public** **string** ModifiedBy { **get**; **set**; }
8. }

# BaseEntity : Tüm Entity’ler için ortak olan alanlar için oluşturulur. Daha Sonra oluşturulan Entityler , BaseEntity’den bu Property’leri miras alır.Örnegin Id alanı Tüm Entity’ler için ortaktır.

# Entity’lerin Eklenmesi

1. **public** **class** Category:BaseEntity
2. {
3. **public** **string** Name { **get**; **set**; }
4. **public** **virtual** ICollection<Product> Products { **get**; **set**; }
5. }

# Category Entity’si BaseEntityden Miras olarak , BaseEntity içindeki Property’leri miras alır.

1. **public** **class** Product:BaseEntity
2. {
3. **public** **string** Title { **get**; **set**; }
4. **public** **string** Description { **get**;**set**; }
5. **public** **decimal** Price { **get**; **set**; }
6. **public** **long** CategotyId { **get**; **set**; }
7. **public** Category Category { **get**; **set**; }
8. }

# Product Entity’si BaseEntityden Miras olarak , BaseEntity içindeki Property’leri miras alır.

# 2.Data Katmanı

”EntitiyFrameworkCore “ Ve “EntitiyFrameworkCore.sqlServer” yüklenir.

# Builder’lerin Eklenmesi

# Data Katmanına Sağ tıklayıp yeni bir klasör ekliyip Bu klasöre “ Builders” Adını veriyoruz.Daha Sonra Klasöre Entity’lerin Builderlerini Oluşturuyoruz.

# Category Entity 🡺 CategoryBuilder

# Product Entity 🡺 ProductBuilder

# CategoryBuilder

1. **public** CategoryBuilder(EntityTypeBuilder<Category> entitybuilder)
2. {
3. // Id 'in Key oldugunu Belirttik
4. entitybuilder.HasKey(a => a.Id);
6. //Name alanının Required ve max 200 Karakter alabilecegini Belirttik
7. entitybuilder.Property(p => p.Name).IsRequired().HasMaxLength(200);
8. }

# ProductBuilder

1. **public** ProductBuilder(EntityTypeBuilder<Product> entitybuilder)
2. {
3. // Id alanı Key
4. entitybuilder.HasKey(a => a.Id);
6. // Title Alanı Zorunlu ve Max 200 Karakter
7. entitybuilder.Property(p => p.Title).IsRequired().HasMaxLength(200);
9. //Category ve Product arasında 1-n ilişki var ve ForegnKey CategoryId.
10. entitybuilder.HasOne(p => p.Category).WithMany(c => c.Products).HasForeignKey(p => p.CategotyId);
11. }

# ApplicationDbContext Eklenmesi

# Data Katmanına Sağ tıklayıp yeni bir Class Ekliyoruz. Bu class’a “ApplicationDbContext “ adını veriyoruz.

1. **public** **class** ApplicationDbContext:DbContext
2. {
4. // ApplicationDbContext'e DbContextOptions olarak Parametreli constructor eklenir.
5. **public** ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : **base**(options)
6. {
8. }
9. // Ayrıca Düzgün Çalışması için Parametresiz Constructor eklenir.
10. **public** ApplicationDbContext()
11. {
13. }
14. //  Entity'lerin Veritabanıyla Bağlantılı çalışması için DBsetler eklenmeli
15. **public** DbSet<Product> Products { **get**; **set**; }
16. **public** DbSet<Category> Categories { **get**; **set**; }

19. // Oluşturduğumuz Builderlerin Entitiy ile Builder arasında Bağlantılı olark çalışması için Eklenir.
20. // Eklenmediği Takdirde ForeginKeyler , Requiredler ... Çalışmaz.
21. **protected** **override** **void** OnModelCreating(ModelBuilder builder)
22. {
23. **base**.OnModelCreating(builder);
24. **new** ProductBuilder(builder.Entity<Product>());
25. **new** CategoryBuilder(builder.Entity<Category>());
26. }

# ApplicationDbContextSeed Eklenmesi

1. **public** **static** **class** ApplicationDbContextSeed
2. {
3. // Başlangıçta Migrata yapar.
4. // Migration varsa aynen devam edeer. YoKsa Product ve Category migrationlarını yapar.
5. **public** **static** **void** Seed (**this** ApplicationDbContext context)
6. {
7. context.Database.Migrate();
8. **if** (context.Categories.Any())
9. {
10. **return**;
11. }
12. AddCategories(context);
13. AddProducts(context);
14. }
15. //Başlangıçta 4 adet Kategori ekler .
16. **public** **static** **void** AddCategories (ApplicationDbContext context)
17. {
18. context.AddRange(
19. **new** Category { Name = "Telefon" },  **new** Category { Name = "Tablet" },  **new** Category { Name = "Bilgisayar" },  **new** Category { Name = "Televizyon" }  );
20. context.SaveChanges();
21. }
22. //Başlangıçta ~~iPHONE OLAN 4000 PRİCE'LI Categoryid 1 ~~ olan bir ürünü otomatik ekler.
23. **public** **static** **void** AddProducts(ApplicationDbContext context)
24. {
25. context.AddRange(  **new** Product { Title = "İphone" , Price=4000 , CategotyId=1} );
26. context.SaveChanges();
27. }

# Infrastructure Eklenmesi

# Data Katmanına Sağ tıklayıp yeni bir klasör ekliyip Bu klasöre “Infrastructure” Adını veriyoruz.Daha Sonra Bu klasöre 4 Ader Class Eklenir.

# RepositoryBase

# IRepository

# UnitOfWork

# IUnitOfWork

# RepositoryBase : Her Entity için Ortak olan Sil,Güncelle,oluştur.. vb Metodları içeren Class’dır.Ortak Oldugu için Entity’lerin Repository’leri RepositoryBase’ı miras alır. RepositoryBase’ı Kullanmak için IRepository Oluşturulur

1. **public** **abstract** **class** RepositoryBase<T> where T : BaseEntity
2. {
3. #region Properties
4. **private** ApplicationDbContext dataContext;
5. **private** **readonly** DbSet<T> dbSet;
6. **protected** ApplicationDbContext DbContext
7. {
8. **get** { **return** dataContext; }
9. }
10. #endregion
11. **protected** RepositoryBase(ApplicationDbContext dbContext)
12. {
13. dataContext = dbContext;
14. dbSet = DbContext.Set<T>();
15. }
16. #region Implementation
17. **public** **virtual** **void** Add(T entity)
18. {
19. // Base Entity'den Miras alarak oluşturulan Herhangi bir Entity'i Add Metodu ile Dbcontext'e Ekler.
20. dbSet.Add(entity);
21. }
22. **public** **virtual** **void** Update(T entity)
23. {
24. // Base Entity'den Miras alarak oluşturulan Herhangi bir Entity'i  Update Metodu ile Dbcontext'e Günceller.
26. dbSet.Attach(entity);
27. dataContext.Entry(entity).State = EntityState.Modified;
28. }
29. // Base Entity'den Miras alarak oluşturulan Herhangi bir Entity'i Delete Metodu ile Dbcontext'den Siler..
31. **public** **virtual** **void** Delete(T entity)
32. {
33. dbSet.Remove(entity);
34. }
35. // Base Entity'den Miras alarak oluşturulan Herhangi bir Koşullu Entity'i Delete Metodu ile Dbcontext'e Ekler.
37. **public** **virtual** **void** Delete(Expression<Func<T, **bool**>> where)
38. {
39. IEnumerable<T> objects = dbSet.Where<T>(where).AsEnumerable();
40. **foreach** (T obj **in** objects)
41. dbSet.Remove(obj);
42. }
43. **public** **virtual** T GetById(**long** id, **params** **string**[] navigations)
44. {
45. var **set** = dbSet.AsQueryable();
46. **foreach** (**string** nav **in** navigations)
47. **set** = **set**.Include(nav);
48. **return** **set**.FirstOrDefault(f => f.Id == id);
49. }
50. **public** **virtual** IEnumerable<T> GetAll(**params** **string**[] navigations)
51. {
52. var **set** = dbSet.AsQueryable();
53. **foreach** (**string** nav **in** navigations)
54. **set** = **set**.Include(nav);
55. **return** **set**.AsEnumerable();
56. }
58. **public** **virtual** IEnumerable<T> GetMany(Expression<Func<T, **bool**>> where, **params** **string**[] navigations)
59. {
60. var **set** = dbSet.AsQueryable();
61. **foreach** (**string** nav **in** navigations)
62. **set** = **set**.Include(nav);
63. **return** **set**.Where(where).AsEnumerable();
64. }
66. **public** T Get(Expression<Func<T, **bool**>> where, **params** **string**[] navigations)
67. {
68. var **set** = dbSet.AsQueryable();
69. **foreach** (**string** nav **in** navigations)
70. **set** = **set**.Include(nav);
71. **return** **set**.Where(where).FirstOrDefault<T>();
72. }
73. #endregion
75. }

# IRepository : RepositoryBase’ı Kullanmak için IRepository Oluşturulur

1. **public** **interface** IRepository<T> where T : **class**
2. {
3. // Yeni bir Entity İşaretler
4. **void** Add(T entity);
6. // Entity Güncellemesini İşaretler
7. **void** Update(T entity);
9. // Entity Silmesini İşaret
10. **void** Delete(T entity);
11. **void** Delete(Expression<Func<T, **bool**>> where);
13. // İd kullanarak Entity Çek
14. T GetById(**long** id, **params** **string**[] navigations);
16. // Koşul kullanarak Entity Çek
17. T Get(Expression<Func<T, **bool**>> where, **params** **string**[] navigations);
19. // T tipi Entities Hepsini çeker.
20. IEnumerable<T> GetAll(**params** **string**[] navigations);
22. // Koşulla bağlı tüm Entitileri çek
23. IEnumerable<T> GetMany(Expression<Func<T, **bool**>> where, **params** **string**[] navigations);
25. }

# UnitOfWork :Service içinde kullanılan Repository’lerde Aynı DbContext’den Yararlanmasını sağlıyor. UnitOfWork’u kullanmak için IUnitOfWork Oluşturulur.

1. **public** **class** UnitOfWork : IUnitOfWork
2. {
3. **private** ApplicationDbContext dbContext;
5. //Reposity'lerin Aynı DbContext i kullanmasını sağlıyor.
6. **public** UnitOfWork(ApplicationDbContext dbContext)
7. {
8. **this**.dbContext = dbContext;
9. }
11. **public** ApplicationDbContext DbContext
12. {
13. **get** { **return** dbContext; }
14. }
16. // DbContext deki degişiklerde hata oluşursa Geri alır.
17. **public** **void** Commit()
18. {
19. **try**
20. {
21. DbContext.SaveChanges();
22. }
23. **catch** (Exception ex)
24. {
26. **throw** ex;
27. }
29. }
30. }

# IUnitOfWork : UnitOfWork’u kullanmak için IUnitOfWork Oluşturulur.

1. **public** **interface** IUnitOfWork
2. {
3. **void** Commit();
4. }

# Repositories Eklenmesi

# Data Katmanına Sağ tıklayıp yeni bir klasör ekliyip Bu klasöre “Repositories” Adını veriyoruz.Daha Sonra Klasöre sağ tıklayıp , Her bir Entity için Repository oluşturulur.

# Category Entity 🡺 Category Builder 🡺 Category Repository

# Product Entity 🡺 Product Builder 🡺 Product Repository

# CategoryRepository

1. **public** **class** CategoryRepository : RepositoryBase<Category>, ICategoryRepository
2. {
3. **public** CategoryRepository(ApplicationDbContext dbContext)
4. : **base**(dbContext) { }
6. // İsme göre ilk sıradaki kateoriyi getirir.
7. **public** Category GetCategoryByName(**string** categoryName)
8. {
9. var category = **this**.DbContext.Categories.Where(c => c.Name == categoryName).FirstOrDefault();
11. **return** category;
12. }
13. }
15. **public** **interface** ICategoryRepository : IRepository<Category>
16. {
17. Category GetCategoryByName(**string** categoryName);
18. }

# 

# ProductRepository

# 3.Service Katmanı

# Category Entity 🡺 Category Builder 🡺 Category Repository 🡺 Category Service

# Product Entity 🡺 Product Builder 🡺 Product Repository 🡺 Product Service

# Not!! : Services Katmanı Referans olarak Data ve Model katmanını kullanır.Bunun için Projeye sağ tıklayıp Add/Reference seçip Data ve Model Katmanları işaretlenmelidir.

# Service Eklenmesi

# Service Katmanına Sağ tıklayıp yeni bir Class Ekliyoruz. Bu class’a “CategoryService “ adını veriyoruz.

# CategoryService

1. **public** **interface** ICategoryService
2. {
3. Category GetCategory(**long** id); // Tek Kategori çagırma
4. IEnumerable<Category> GetCategories(); // Birden Çok Kategori çagırma
5. **void** CreateCategory(Category category); // Kategori oluşturma
6. **void** UpdateCategory(Category category); // Kategori güncelleme
7. **void** DeleteCategory(**long** id); // Kategori SİLME
8. **void** SaveCategory(); //Kategori kaydetme
9. }
10. **public** **class** CategoryService : ICategoryService
11. {
12. **private** **readonly** ICategoryRepository categoryRepository;
13. **private** **readonly** IUnitOfWork unitOfWork;
14. **public** CategoryService(ICategoryRepository categoryRepository, IUnitOfWork unitOfWork)
15. {
16. **this**.categoryRepository = categoryRepository;
17. **this**.unitOfWork = unitOfWork;
18. }
19. **public** IEnumerable<Category> GetCategories()
20. {
21. var categories = categoryRepository.GetAll();
22. **return** categories;
23. }
24. **public** Category GetCategory(**long** id)
25. {
26. var category = categoryRepository.GetById(id);
27. **return** category;
28. }
29. **public** **void** CreateCategory(Category category)
30. {
31. categoryRepository.Add(category);
32. }
33. **public** **void** UpdateCategory(Category category)
34. {
35. categoryRepository.Update(category);
36. }
37. **public** **void** DeleteCategory(**long** id)
38. {
39. categoryRepository.Delete(x => x.Id == id);
40. }
41. **public** **void** SaveCategory()
42. {  unitOfWork.Commit();  }
43. }

# 4.UI (Web) Katmanı

# Not!! : UI Katmanı Referans olarak Data,Model Ve Service katmanını kullanır.Bunun için Projeye sağ tıklayıp Add/Reference seçip Data,Model Ve Service Katmanları işaretlenmelidir.

# StartUp Düzelnemesi

# 1.Seed Methodunun Çalışması için app.UseStaticFiles(); altına Şu kod Eklenmeli

1. // Seed Methofunu kullanması için bu komutu ekliyoruz.
2. app.ApplicationServices.GetRequiredService<ApplicationDbContext>().Seed();

# 2.UnitOfWork ve Repository’lerin Kullanılabilmesi için services.AddMvc(); altına şu kodlar eklenir.

1. // IUnitOfWork Kullanılması için Eklenir.
2. services.AddScoped<IUnitOfWork, UnitOfWork>();
4. // ICategoryRepository Kullanılan yerlerde Geçici olarak CategoryRepository Kullanır.
5. // Her Talepte yeniden Oluşturulur.
6. services.AddTransient<ICategoryRepository, CategoryRepository>();
7. services.AddTransient<ICategoryService, CategoryService>();