N-Layer Proje Yapısı:Katmanlar

Bir yazılım projesini, adeta bir şehir inşa eder gibi hayal edin. Sanki bir şehirde yeni bir bina inşa ediliyormuş gibi düşünebiliriz. Bu bina, sağlam temeller üzerine kurulmuş bir yapı olmalı ki gelecekteki zorluklara karşı dayanıklı olsun. İşte, yazılım dünyasındaki projeler de benzer bir prensibe dayanır. **N-Layer (Katmanlı)** proje yapısı da bu bağlamda, yazılım dünyasında bir projenin sağlam temeller üzerine inşa edilmesini sağlayan bir yapıdır.

Şehrimizin giriş kapısı, projenin de giriş katmanını temsil eder. Bu katman, projenin dış dünya ile iletişimini sağlar. Kullanıcılar, bu kapıdan içeri adım atarlar ve projenin temelini oluşturan **Presentation Katmanı** ile karşılaşırlar. Burada, kullanıcı girişleri, istekler ve projenin dış dünya ile etkileşimiyle ilgili işlemler gerçekleşir.

Şehrin ana caddeleri ve bulvarları, projenin iş mantığının yürütüldüğü **İş Katmanı**'nı temsil eder. **İş Katmanı**, projenin kalbidir ve burada temel iş süreçleri, veri işleme ve iş kuralları uygulanır. Şehirdeki binaların birbiriyle iletişim kurmasını sağlayan bu ana arterler, yazılım projelerinde farklı modüllerin birbirleriyle etkileşimini yönetir.

Şehirdeki altyapı ve enerji sistemleri, **veritabanları ve veri depolama katmanını** simgeler. Veri Katmanı, projenin belleğidir. Burada, projeye ait veriler depolanır, yönetilir ve gerektiğinde çeşitli katmanlara dağıtılır. Veri Katmanı, projenin geçmişini ve anını koruyan bir arşiv gibidir.

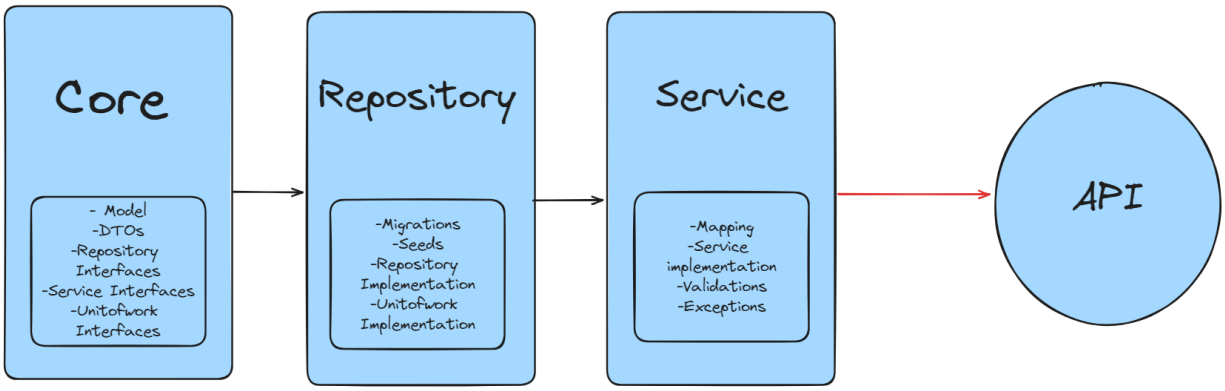
Son olarak, şehirdeki sokak lambaları ve güvenlik kameraları gibi detaylar, projemizin **Web Application** yani **UI** tarafını temsil eder. Kullanıcıya görünür olan arayüz, ön yüz aracılığıyla tasarlanır ve yönetilir. Kullanıcılar, bu katman sayesinde projenin görünür yüzüyle etkileşime geçerler. Buradaki kısımda katmanlardan bir tanesi ön yüz projesiyle birlikte içerisinde servisleri de barındaran bir Web Application da olabilir, sadece endpointlerden oluşan bir API projesi de olabilir. İhtiyaçlarınıza istinaden bu değişebilir.

İşte böyle bir şehir hikayesi içerisinde, **N-Layer** proje yapısı da bir şehrin katmanları gibi birbirine entegre, ama bir o kadar da bağımsız katmanlardan oluşan bir yapı sunar. Her katman, projenin sağlam temeller üzerine inşa edilmesine katkıda bulunur ve projenin gelecekteki büyümesine, değişimine ve gelişimine uygun bir altyapı sunar.

Biz bugün detaylı olarak bir N-Layer projedeki katmanların içerisinde ne gibi alanların olacağını inceleyeceğiz. Umuyorum yazıdan hem keyif alırsınız hem de sizler için faydalı olur. O halde başlayalım!

Gelişme

Yazılım projeleri, katmanlı mimari ile düzenlenerek daha sürdürülebilir ve yönetilebilir hale getirilebilir. Her katmanın kendine göre ayrı bir görevi vardır.



*Techcareer.net eğitimlerinden öğrencim Şevval Özdemir der ki ;**"Rotasız gemi olmaz!"** O nedenle kendimize yukarıdaki gibi rota oluşturduk.*

En Az Üç Katman!

Katmanlı mimaride en az üç katman bulunmalıdır. Bir projenin temel yapı taşlarını oluşturmak ve bu temelde geliştirmeler yapmak için minimum üç katmanın oluşturulmasını önerir. Bu katmanlar, genellikle temel işlevselliği sağlamak için kullanılır ve projenin karmaşıklığına göre daha fazla katman eklenerek geliştirilebilir.Projelerin gereksinimleri farklılık gösterebilir ve bu nedenle katman sayısı ihtiyaca göre artırılabilir. Örneğin, sistemdeki performansı artırmak için bir önbellekleme (caching) katmanı eklenmesi veya sistemdeki olayları kaydetmek için bir loglama katmanı eklenmesi gibi durumlar düşünülebilir. Bu ek katmanlar, projenin özel ihtiyaçlarına ve gereksinimlerine yönelik olarak tasarlanabilir.Bu prensipler, bir yazılım veya sistem tasarımının düzenli, ölçeklenebilir ve bakımı kolay bir yapıya sahip olmasını sağlamak amacıyla kullanılır. Katmanlı mimari, her katmanın belirli bir sorumluluğa sahip olduğu ve katmanların birbirinden bağımsız olduğu bir yapı sunarak yazılım geliştirmeyi daha etkili hale getirir.

Temel Katmanlar ve İsimlendirmeleri

İsimlendirme olarak "**Core**," "**Repository**," ve "**Service**" isimlendirmeleri oldukça yaygın ve anlamlı katman adları olduğu için biz de bunları kullanabiliriz. Bu üç katmanlı yapı, temel işlevselliği düzenler ve sorumlulukları açık bir şekilde tanımlar. Her katman, kendi işlevselliğine odaklanır ve diğer katmanlarla minimum bağımlılık içinde olmaya çalışır.Alternatif olarak, bazı projelerde **"Business"** (İş) ve "Data Access" (Veri Erişimi) gibi isimlendirmeler de kullanılabilir:

- **Business Katmanı:** İş mantığı kurallarını içerir.

- **Data Access Katmanı:** Veri erişimi ve veritabanı işlemlerini içerir.

Bu isimlendirmeler, temel prensipleri benzer şekilde yansıtar ve genellikle projenin gereksinimlerine, ekibin tercihlerine veya endüstri standardına bağlı olarak seçilir. Önemli olan, projede kullanılan isimlendirmenin tutarlı olması ve ekibin projenin gereksinimlerine uygun bir yapı oluşturmasını sağlamasıdır.

Core Katmanı Detayları

Temel Detaylar

- Genel iş mantığı ve temel işlevselliği içerir.
- Projedeki temel modelleri, yardımcı sınıfları ve ortak araçları içerir.
- Bu katman, projenin genel yapısını oluşturan temel bileşenleri içerir.

Proje Kapsamındaki Detayları

- En içteki katmandır ve temel işlevlere odaklanır.
- İçerisinde iş kuralları veya iş mantığı barındırmaz.
- Projenin genelini ilgilendiren sınıfları içerir: **Modeller**, **DTO nesneleri,Service ve Repository interfaceleri**, **Unitofwork Design Pattern'i için interfaceler**.

Peki Modellerden kastımız nedir ?

"Modeller" terimi genellikle bir yazılım projesinde veri yapılarını temsil eden sınıfları ifade eder. Bu sınıflar, genellikle projenin mantığına ve iş gereksinimlerine uygun olarak tasarlanır ve veritabanındaki tablolar, dış servislerden alınan veri formatları veya başka kaynaklardan gelen verilerle ilişkilendirilir.Modeller, genellikle veri taşıma amaçları için kullanılır ve proje içindeki verileri temsil ederler. Örneğin, bir e-ticaret uygulamasında "Ürün" adında bir model, bir ürünün adını, fiyatını, stok durumunu ve diğer ilgili bilgileri içerebilir. Bu modeller, genellikle projede kullanılan diğer bileşenler arasında veri alışverişi yapmak için kullanılır.

```
public class Product
{
   public int Id { get; set; }
   public string Name { get; set; }
   public decimal Price { get; set; }
   public int StockQuantity { get; set; }
   // Diğer özellikler...
}
```

Peki DTO'dan kastımız nedir ?

DTO, "Data Transfer Object" teriminin kısaltmasıdır. DTO'lar, genellikle veri transferi amacıyla kullanılan nesnelerdir. Bu nesneler, genellikle veritabanı varlıkları veya diğer modellerle ilgili verileri taşımak için tasarlanmıştır. DTO'lar, veri taşıma işlemleri sırasında kullanılan bir araç olarak düşünülebilir.

Veri Transferi: İki farklı katman veya bileşen arasında veri transferi yapmak için kullanılırlar. Örneğin, bir web servisi çağrısı sırasında, istemcinin ve sunucunun arasında veri taşımak için DTO'lar kullanılabilir.

İstemci ve Sunucu Arasında Veri İletimi: Web uygulamalarında veya mikro servis mimarilerinde, istemci ve sunucu arasında veri alışverişi için kullanılır. Bu, genellikle farklı veri ihtiyaçlarına sahip olan istemci ve sunucu arasında etkili bir veri transferi sağlar.

Veri Filtreleme ve Gizleme: DTO'lar, yalnızca belirli alanları veya özellikleri içerecek şekilde tasarlanabilir. Bu, gereksiz veri aktarımını önler ve ağ trafiğini azaltabilir.

İş Katmanı ve Sunum Katmanı Arasındaki İletişim: Bir iş katmanındaki servisler ve sunum katmanındaki kullanıcı arayüzü arasında veri iletimi sırasında, iş katmanındaki karmaşık nesnelerin basitleştirilmiş bir temsilini sağlamak için DTO'lar kullanılabilir.

```
public class ProductDTO
{
  public string Name { get; set; }
  public decimal Price { get; set; }
  // Diğer özellikler...
}
```

Peki Service ve Repository Interfacelerinden kastımız nedir ?

“Service” ve “Repository” terimleri, genellikle yazılım uygulamalarının tasarımında kullanılan iki farklı konsepti ifade eder. Bu kavramlar, genellikle iş mantığı, veritabanı etkileşimi ve diğer temel işlevselliğin düzenli ve modüler bir şekilde yönetilmesine yardımcı olmak amacıyla kullanılır.

- *Service (Hizmet) İnterfaceleri*

Servisler, genellikle iş mantığı veya uygulama seviyesindeki belirli görevleri yerine getiren bileşenlerdir.

```
public interface IProductService
{
   ProductDTO GetProductById(int productId);
   void AddProduct(ProductDTO productDTO);
   void UpdateProduct(ProductDTO productDTO);
   void DeleteProduct(int productId);
   // Diğer işlevler...
}
```

- *Repository (Depo) İnterfaceleri*

Repository, genellikle veritabanı işlemlerini yöneten bir tasarım desenidir. Repository'ler, veritabanına erişimi soyutlar ve veri tabanı işlemlerini gerçekleştiren metotları içerir. Bu sayede uygulama katmanları, veritabanı işlemleriyle doğrudan etkileşimde bulunmak yerine bu soyutlama aracılığıyla işlemlerini gerçekleştirir.

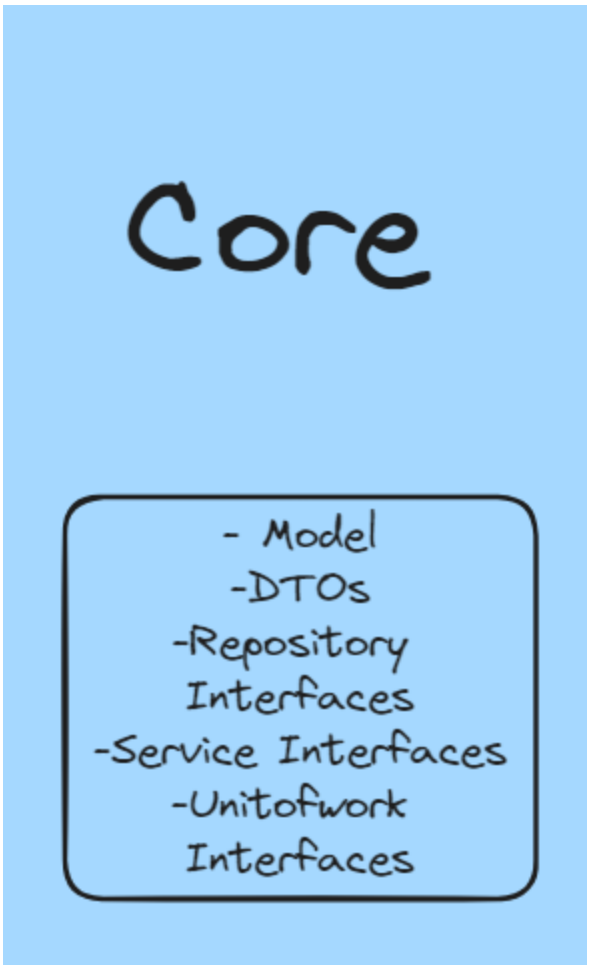
```
public interface IProductRepository
{
  ProductDTO GetProductById(int productId);
  void AddProduct(ProductDTO productDTO);
  void UpdateProduct(ProductDTO productDTO);
  void DeleteProduct(int productId);
  // Diğer işlevler...
}
```

Peki peki UnitOfWork Design Pattern nedir?

"Unit of Work" (UnitOfWork), bir tasarım deseni olup genellikle bir veritabanı işlem sürecini tek bir işlem (transaction) olarak ele almak ve bu süreci yönetmek için kullanılır. Bu desen, birden çok işlem adımını birleştirerek ya hepsini başarılı bir şekilde gerçekleştirir ya da hiçbirini gerçekleştirmez. Unit of Work, genellikle bir iş katmanı ile veritabanı arasındaki etkileşimi soyutlamak ve yönetmek için kullanılır.

```
public interface IUnitOfWork : IDisposable
{
  void Commit();
  void CommitAsync();
}
```

Bu arayüzü implemente eden bir sınıf, ilgili repository'leri döndüren metotları sağlar ve aynı zamanda işlemleri yönetir. Unit of Work sınıfı, iş katmanının tek bir işlem içinde birden çok veritabanı işlemi gerçekleştirmesine ve bunları koordine etmesine olanak tanır.



Repository Katmanı Detayları

Temel Detaylar

- Veritabanı veya diğer veri kaynaklarına erişimi yönetir.
- Veri modeli ile ilgili işlemleri (okuma, yazma, güncelleme, silme) gerçekleştirir.
- Veritabanı sorgularını ve işlemlerini içerir.

Proje Kapsamındaki Detayları

- Core katmanını referans alır.
- **Migrations**: Veri tabanı ile senkronizasyonu sağlar.
- **Seeds**: Veri tabanına varsayılan verilerin eklenmesini yönetir.
- **Repository Implementasyonu:** Core katmanındaki Repository interfacelerinin implementasyon işlemi burada gerçekleşir..

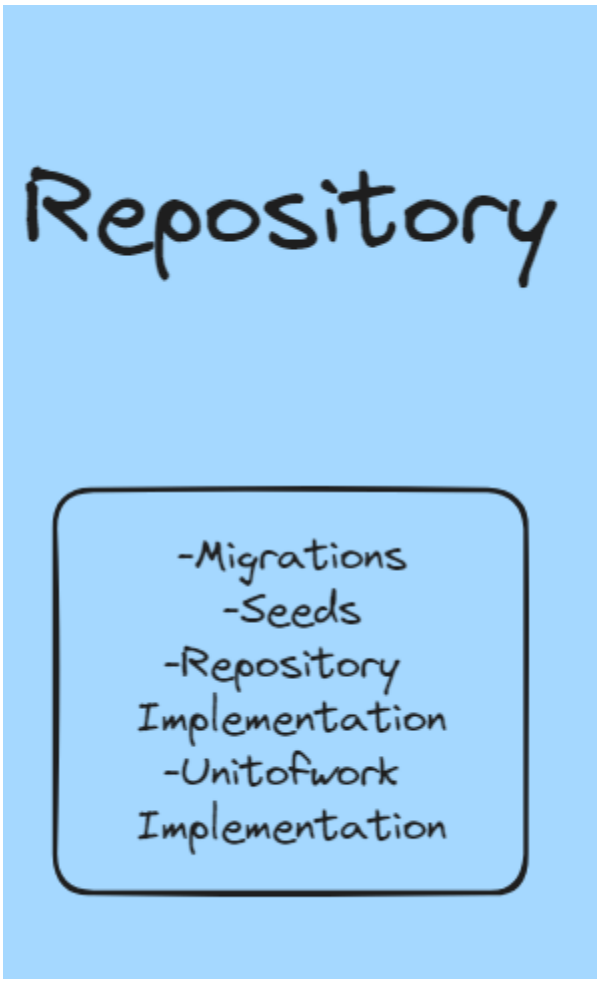
- **UnitOfwork Implementasyonu:** Core katmanındaki **UnitOfWork** Interface'sinin implementasyon işlemi burada gerçekleşir.

Migrationlara dair biraz daha detay mı versek?

Migrations, bir yazılım uygulamasının veritabanı şemasını evrimleştirmek ve güncellemek için kullanılan bir konsepttir. Bu süreç, veritabanındaki değişiklikleri belgeleyen ve uygulayan otomatikleştirilmiş bir yöntem sunar. Migrations, veritabanı şemasındaki değişiklikleri adım adım takip etmenizi sağlar, bu da uygulamanın gelişimine ve sürdürülebilirliğine katkıda bulunur. Bu süreç aynı zamanda farklı uygulama sürümleri arasında uyumluluğu sağlar, test ve geri alabilirlik imkanı sunar, ve uygulamanın ilk kurulumunu kolaylaştırır. Migrations genellikle mevcut durumu belirleme, değişiklikleri belgeleme ve bu değişiklikleri uygulama adımlarını içerir. Bu yöntem, yazılım geliştirme sürecinde ekipler arasında işbirliğini ve veritabanı yönetimini kolaylaştırır.

Seeds kavramına biraz daha detay mı versek?

"Seeds," bir yazılım uygulamasında kullanılacak varsayılan verilerin veritabanına eklenmesini yöneten bir kavramdır. Bu süreç, uygulamanın başlangıcında veya belirli senaryolarda kullanılacak temel verilerin belirlenmesi, bir "Seed" dosyasında tanımlanması ve uygulama başladığında veya belirli durumlar gerçekleştiğinde bu verilerin otomatik olarak veritabanına eklenmesini içerir. Seeds, uygulamanın başlangıç durumunu oluşturmak, test senaryolarında tutarlılık sağlamak ve geliştirme süreçlerini kolaylaştırmak için önemlidir.



Service Katmanı Detayları

Temel Detaylar

- İş mantığı kurallarını uygular.
- Repository katmanından alınan verileri kullanarak iş mantığı işlemlerini gerçekleştirir.
- Dış dünya ile iletişimi sağlar ve genellikle kullanıcı arayüzü veya diğer sistemlerle etkileşimde bulunur.

Proje Kapsamındaki Detayları

- Repository katmanını referans alır.
- Business kodlarını içerir.
- **Mapping**: Entity'leri DTO nesnelerine maplemek için kullanılan dosyaları içerir (örneğin, AutoMapper).

- **Service Implementasyonu:** Core katmanındaki Service Interfaceleri'nin implementasyonunu içerir.
- **Validations**: Giriş doğrulamalarını içerir.
- **Exceptions**: Hata yönetimi ile ilgili kodları içerir.

Mapping'e biraz daha detay mı versek?

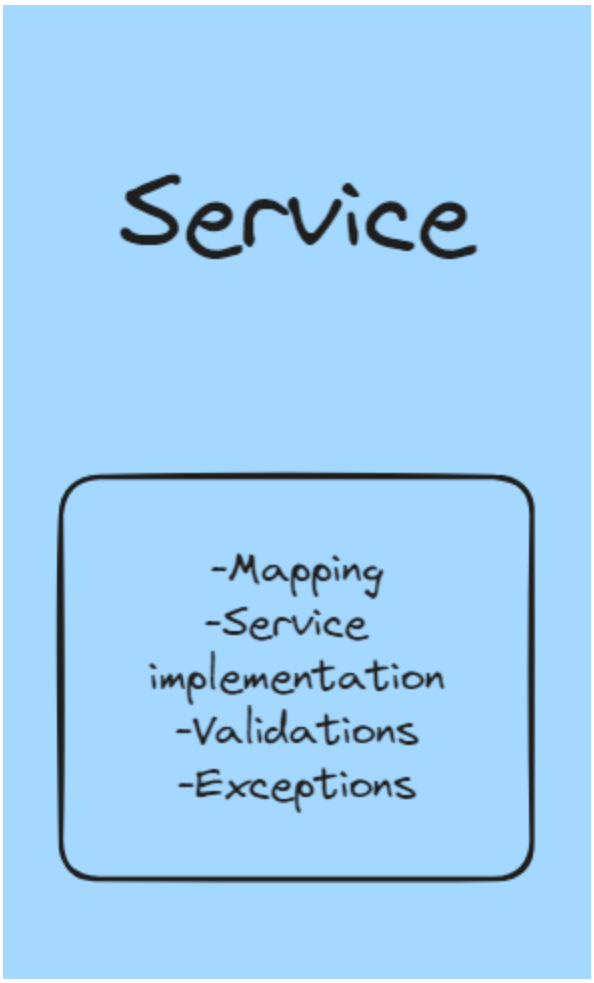
"**Mapping,"** genellikle **AutoMapper** gibi araçlar kullanılarak, uygulama içindeki varlık (entity) nesnelerinin, **DTO** (Data Transfer Object) nesnelerine dönüştürülmesini sağlayan dosyaları içerir. Bu süreç, veritabanındaki varlık nesneleri ile kullanıcı arayüzü veya dış servislerle iletişim için kullanılan DTO'lar arasında uyumluluk sağlar. Bu dosyalar, genellikle varlık ve DTO arasındaki alan eşleştirmelerini tanımlar ve bu dönüşümü otomatize eder. Bu sayede, uygulamanın farklı katmanları arasında veri transferini kolaylaştırarak, kod tekrarını azaltır ve geliştirme süreçlerini daha sürdürülebilir kılar.

Validasyonlara biraz daha detay mı versek?

"Validations," genellikle **Fluent Validation** gibi araçlar kullanılarak, giriş verilerinin doğrulanması için kullanılan dosyaları içerir. Örneğin, .NET Core üzerinde Fluent Validation kullanılarak, DTO'lar üzerindeki giriş verileri kolayca doğrulanabilir. Bu doğrulamalar, giriş parametrelerinin boş olmamasından, belirli bir uzunlukta olmasına, sayısal veya alfabetik olmasına kadar geniş bir yelpazede kuralları kapsayabilir. Fluent Validation, solid prensiplerine uyumu kolaylaştırır çünkü kurallar açık ve ayrı dosyalarda tanımlanabilir, bu da tek sorumluluk ilkesine uyumu destekler. Ayrıca, doğrulama kurallarının değişmesi durumunda kodun daha az değiştirilmesini sağlar, bu da açık/kapalı(Open Close) ilkesine uyumu güçlendirir.

Exceptionslara biraz dahada detay mı versek?

"Exceptions," genellikle servis katmanında hata yönetimi ile ilgili kodları içerir. Bu dosyalar, uygulamanın hata durumlarıyla başa çıkmasını sağlar. Hata yönetimi, özellikle servislerin dış kaynaklarla (veritabanı, dış API'lar vb.) etkileşimde bulunduğu durumlarda kritiktir. Hataların uygun şekilde işlenmesi ve kullanıcıya anlamlı geri bildirimlerin sağlanması, uygulamanın güvenilir ve kullanıcı dostu olmasını sağlar. Servis katmanında hata yönetimi, özel istisna sınıfları, hata kodları, günlükleme (logging) ve gerekirse geriye hata mesajları döndürme gibi tekniklerle uygulanabilir. Solid prensiplerine uyum sağlamak için, bu kodlar genellikle tek sorumluluk ilkesi doğrultusunda tasarlanır, böylece her bir dosya belirli bir tür hatayla ilgilenir ve değişiklikler minimal olur.



API(Presentation) Katmanı Detayları

Presentation (API) katmanı, bir yazılım projesinde kullanıcı ve sistem arasındaki etkileşimi yöneten ana katmandır. Bu katman, genellikle web API'leri veya kullanıcı arayüzleri ile ilgili işlevleri içerir. API katmanında, genellikle **Middlwares** ve **Filterlar** gibi yapılar kullanılarak gelen isteklerin ve cevapların işlenmesi yönetilir. **Middleware'**ler, gelen isteği ve giden cevabı manipüle etmek, ek işlemler eklemek veya hata yönetimini ele almak için kullanılır. **Filterlar** ise genellikle isteğin veya cevabın belirli koşullara göre filtrelenmesi veya değiştirilmesi amacıyla kullanılır. Bu katman, kullanıcı arayüzleri veya dış sistemlerle etkileşimde bulunarak uygulamanın dış dünya ile iletişimini sağlar.

Sonuç

Sonuç olarak, yazılım projelerini bir şehir inşa sürecine benzetmek, N-Layer proje yapısının önemini anlamamıza yardımcı olabileceğini düşündüğüm için öncelikle konuya girişi bu şekilde gerçekleştirdim. Sonrasında Core, Repository , Service ve Presentation katmanlarında ne gibi özellikler olabileceğini açıklamaya çalıştım.

Umuyorum faydalı olmuştur,

Bir sonraki yazıda görüşmek üzere!

Kaynakça

- https://www.youtube.com/watch?v=r-RUY2caw3s
- https://www.youtube.com/watch?v=xJC7ItRoEbw
- https://www.youtube.com/watch?v=Srp1iyZu-ww&pp=ygUNbi1sYXllciBwcm9qZQ%3D%3D
- https://www.udemy.com/course/asp-net-core-api-web-cok-katmanli-mimari-api-best-practices/
- https://www.gencayyildiz.com/blog/c-ta-n-tier-architecturecokn-katmanli-mimari/