# **Çok Katmanlı Mimari**

Bir proje dahilindeki class içindeki kod satırlarını azaltmayı hedefleme durumu ve bakım kolaylığı sağlama işlemi katman ekleyerek gerçekleştirilir. Her şeyi controller içinde yapmayı hedefledik fakat işler arttıkça controller katmanında diyelim ki 1000 satır kod oluştu. Bakım sırasında çok fazla zaman ve dikkat sorunu oluşturacak. Bu sorunu çözmek için kodu azaltarak Servis Katmanı oluşturduk. Servis Katmanı dahilinde ise bir DB üzerinde işlem göreceğini ve her bir depo işlemi için bir katman daha eklenebilir. Bunun için de Repository Katmanı, Servis Katmanı altına tanımlanır.

diyagram, plan, çizgi, metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Birçok katman eklenebilir fakat temel olarak 3 katman belirlenmiştir.

* Veri Katmanı: Veri tabanı bağlantı kodlarının bulunduğu katman.
* Sunum Katmanı: Kullanıcıyla direkt iletişimde bulunan katman.
* İş Katmanı: Sunum katmanından gelen isteklere göre veri katmanına gönderen katmandır.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

## Veri Katmanı (Data Layer-DataAccessLayer)

Veri tabanı bağlantısı sağlanarak verileri işleme yapılmaktadır. Silme, güncelleme, ekleme veya veri çekme gibi işlemeler yapılır. İş kodları yapılmaz.

Bilginin uyumluluğuna bakılmaz. Bu katmana gelene kadar daha önceden yapılmış olduğu kabul edilir.

## İş Katmanı (Business/Implementation Layer)

Sunum katmanından gelen verileri işleyerek veya denetleyerek veri katmanına uygun hale getirilir. Aynı şekilde veri katmanından gelen verileri gerekli işlemleri yaparak sunum katmanına ileten ara katmandır.

## Sunum Katmanı (Presentation Layer)

Kullanıcıya bir arayüz sunan projelerdeki katmandır. Kullanıcı ile iletişime geçilir. Kısaca kullanıcıdan Request alınır.

Katman sayısı projedeki ihtiyaçlara göre seçilir. Katman sayısının fazlalığı çalışma zamanını hızlandırma gibi bir durum gerçekleştirmez. Sadece **Clean Code** ve **Bakım Kolaylığı** sağlar.

## Servis Katmanı (Service Layer)

Uygulamanın servis edildiği katmandır. İş katmanı üzerinde yapılan işlemlerin hangi platformlarda olacağı gösterilmektedir.

RestAPI’nin dış dünya ile iletişim kurduğu yerdir. Kullanıcı isteklerini alır, işler, gerekli verileri Repository katmanından çeker veya işler.

İş mantığını ve işlemleri bir yerde toplar, tekrar kullanımı kolaylaştırır. Kodun daha organize olmasını sağlar ve servislerin test işlemlerini kolaylaştırır. Fakat ek bir katman eklendiği için kodun karmaşıklığını arttırabilir.

## Repository Katmanı (Repository Layer)

Veri tabanı veya başka bir veri deposu ile etkileşime geçen katmandır. Veri tabanı sorgularını yürütür, veri depolama işlemlerini gerçekleştirir ver Servis’e gerekli bilgilendirmeleri yapar.

* Veri tabanı işlemlerini merkezi yere depolar ve tekrar kullanımı kolaylaştırır.
* Veri tabanı işlemlerini soyutlar ve farklı veri tabanları ile etkileşimi kolaylaştırır.
* Test edilebilirlik ve bakım kolaylığı sağlar.
* Fakat veri tabanı bağlantıları ve sorguları yönetmede ekstra kod karmaşıklığı getirebilir.

# **Data Transfer Object**

Veri aktarımını kolaylaştırmak ve veri alışverişi sırasında karmaşıklığını azaltmak için kullanılır.

* **Veri Aktarımı:** Farklı katmanlar arasında veri aktarımını kolaylaştırmak için kullanılır.
* **Veri Kapsülleme:** Birçok veriyi tek bir nesne içinde tutar. Verileri daha iyi organize etme ve iletişimi kolaylaştırmaya yarar.
* **Taşınabilirlik ve Bağımsızlık:** Uygulamanın farklı katmanları arasında veri taşımayı sağlar. Sonradan eklenebilecek katmanlara da aktarım kolaylığı sağlayacaktır.

Sadece gerekli bilgileri taşıyarak veri transferini optimize ederek **performansı** arttırır, veri yapısında değişiklik yapılırsa sadece DTO katmanında değişim yapılarak **esneklik** arttırılır ve yalnızca gerekli verilerin taşınmasından dolayı hassas verilerin ifşası engelleyerek **güvenlik** sağlanabilir.

Bunların aksine, büyük projelerde birden fazla DTO olabilir ve **yönetimini zorlaştırabilir** ve fazla DTO olmasından kaynaklı kod karmaşıklığı oluşabilir ve **bakım zorlaşabilir**. Uygulama içinde farklı veri kopyaları içerebilir ve bundan kaynaklı **veri tutarsızlığı** oluşabilir.

diyagram, ekran görüntüsü, yazı tipi, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Genellikle **Servis ve Veri Erişim** katmanlarında kullanılırlar. Özellikle mikro servis mimarisinde farklı hizmetler arasında iletişimi kolaylaştırmak için kullanılır.

# **Entity Nedir?**

Yazılımda gerçek dünya nesne veya kavramlarını temsil eden yapılardır. Bu yapılar genellikle yazılımın iş mantığı ile ilgilidir ve kullanıcılar tarafından tanınabilir.

* **Veri Alanları (Properties):** Gerçek dünya nesnelerini anlatan veri alanlarına sahiptir.
* **İlişkiler (Relationship):** Entityler arasında ilişkiler kurulabilir.
* **Bilgi (Behavior):** Veriyi işleme, erişim ve değiştirme yetkisine sahiptirler.

Entityler uygulamanın yapı taşı olduğundan düzgünce tasarlanmalıdır. Tasarlanırken genellikle **“Domain Driven Design”** (Alan Odaklı Tasarım) prensibini kullanır.

Genellikle uygulamanın veri katmanında bulunur. **Veri Erişim Katmanı** entitylere ulaşır ve gerekli işlemi yapar. **İş Katmanı** ise bu entityleri kullanılacak işlemleri gönderir ve dönütleri kullanıcıya gösterir.

**Avantajlar:**

* Gerçek dünya sorunları olduğundan kodun anlaşılabilirliği artar.
* Farklı katmanlarda kullanılabilir ve yeniden kullanılabilirliği artar.
* DDD prensibine göre tasarlanırsa: daha esnek, ölçekli ve bakım kolaylığı olur.

**Dezavantajları:**

* Kötü tasarlanırsa uygulamanın karmaşıklığı artar ve bakımı zorlaşır.
* İlişkisel veri tabanında büyük entityler performans sorununa yol açar.
* İyi tasarlanmamışlarsa uygulamadaki veri bütünlüğünü ve tutarlılığını olumsuz etkiler.

# **ORM Nedir?**

**Persistence,** objenin durumunu kalıcı bir alana depolama işlemidir.

**ORM,** nesneleri bir veri tabanına bağlama işlemidir. Veri tabanı işlemlerini **SQL Sorgu** yerine direkt programlama dili ile yapılması olarak tanınır.

**ORM,** veri tabanı ile uygulama arasındaki **“Data Layer”** olarak yer alır.

**Avantajları:**

* Uygulama ile veri tabanının bağımlılığı yoktur ve duruma göre başka veri tabanları kullanılabilir.
* SQL Sorgularını nesne yönelimli olarak yapar, bu sayede koda anlaşılır ve bakım kolaylığı sağlar.
* SQL enjeksiyon saldırılarını önler ve veri güvenliği sağlar.

**Dezavantajları:**

* Bazı durumlarda karmaşık sorguların performansını etkileyebilir.
* Bazı durumlar için veri tabanı için performans kaybı.
* Bilgi alış-verişi sırasında kontrolün tamamının elde olmaması.
* Kullanılan ORM aracının öğrenme süreci.

## ORM Modelleme Teknikleri

1. **Database First Yaklaşımı**

Var olan bir veri tabanını .EDMX uzantılı dosya olarak oluşturulur. Tablolar üzerindeki değişiklikler management studio aracılığıyla gerçekleştirilir.

**Scaffolding** yöntemi ile tablolar projeye entity class olarak eklenir.

1. **Model First Yaklaşımı**

Öncelikle bir model oluşturmaya daha sonrasında modele göre veri tabanı şeması oluşturmadır.

Öncelikle model .EDMX dosyası olarak saklanır. Dosya içerisinde tablolar, ilişkiler vb. EDMX design olarak belirtilir. Daha sonrasında modelden veri tabanı oluşturulur.

metin, yazı tipi, ekran görüntüsü, logo içeren bir resim

Açıklama otomatik olarak oluşturuldu

1. **Code First Yaklaşımı**

Kod kullanılarak veri tabanı ve entity modeli oluşturma yaklaşımıdır. Entity classları oluşturulur. Classlar tablo olurken içlerindeki propertyler ise sütun olarak aktarılır.

metin, ekran görüntüsü, yazı tipi, meneviş mavisi içeren bir resim

Açıklama otomatik olarak oluşturuldu

# **Controller Katmanı Nedir?**

Kullanıcı arabiriminin (UI) isteklerini alarak uygun işlemleri tekrardan kullanıcıya döndüren katmandır. İsteğe göre veri tabanı işlemlerini (Model Katmanı ile etkileşim), iş mantığını (Servis Katmanı ile etkileşim) ve son olarak sunulacak görüntüyü (View Katmanı ile etkileşim) hazırlar.

Genellikle Servis Katmanı ile ayrı tutulur. Arayüz ile Servis Katmanı arasında köprü görevi yapar. Uygulamayı daha modüler, bakımı kolay ve yeniden kullanılabilir yapar.

**Başlıca Görevi:**

* **İstek Yönlendirme:** Kullanıcıdan istekleri alır ve ilgili işlemdeki controller metotlarına yönlendirir.
* **İş Mantığı Yönetme:** İş mantığını (business logic) gerçekleştirmek üzere gerekli servisleri çağırır ve gelen verileri işler.
* **Veri Sunma:** İşlenmiş verileri kullanıcı arabirimine (UI) sunmak üzere uygun görünümlere (View) aktarır.
* **Hata Yönetimi:** Ortaya çıkan hataları ele alır ve uygun kullanıcı arayüzüne (UI) hata mesajlarını yansıtarak kullanıcıya bilgi verir.

# **Request Response Modelleri Nedir?**

metin, ekran görüntüsü, diyagram, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu model bir istemci (genellikle front-end) ile bir sunucu (back-end) arasındaki iletişim akışını açıklar.

**Request (İstek) Modeli:**

* İşlemci tarafından sunucuya iletilen bilgi ve isteklerin yapısı tanımlanır.
* İstemci, sunucuya belirli bir şey gerçekleştirmek için istekte bulunur.
* İstek türü farklı veri formatında olabilir.
* İstekler genellikle istek türü (HTTP Metotları), istek başlığı (headers) ve istek verileri (body) şeklinde bilgi içerir.

**Response (Cevap) Modeli:**

* Sunucu, istemciye gönderilen cevapların yapısını tanımlar.
* Sunucu, isteğe göre cevap üretir.
* İsteğin yapıldığı hakkında cevaplar döndürülebilir.
* Cevap genellikle durum kodu (HTTP durum kodu), cevap başlıkları (headers) ve cevap verileri (body) şeklinde bilgileri içerir.

**Kullanım alanları:**

* Web Uygulamaları: İstemci (Browser) istenilen kaynakları içeren HTTP yanıtını gönderen sunucuya (Web Sunucusu) istekte bulunur.
* API İletişimi: İstemciler, belirli eylemi gerçekleştirmek için sunucuya API istekleri gönderirler. Dönüt, JSON veya XML biçimindedir.
* Ağ İletişimi: Ağ oluşturmada HTTP, FTP ve SMTP gibi çeşitli protokolleri modeli kullanır.
* RPC (Uzaktan Yordam Çağrıcıları): İstemci, uzak bir sunucuya bir işlev için istek gönderir. Bunu SOAP ve gRPC gibi çerçevelerle yapar.
* Veri Tabanları: Veri tabanı ile etkileşime geçilirken istekte bulunulur ve sorgular gönderilir. Cevap olarak da sorgunun işlevi döndürülür.
* IoT (Nesnelerin İnterneti): IoT cihazları ve sistemleri tarafından kullanılır. Merkezi bir sunucu veya ağ geçidi, IoT cihazına istekte bulunur.
* İnternet Hizmetleri: SOAP ve RESTful web hizmetleri modeli kullanan teknolojilere örnektir.
* Dağınık Sistemler: Dağıtılmış sistemlerde bileşenler, aralarında haberleşme ve bilgi alışverişi için kullanırlar. Veriler arasında tutarlılık ve güvenilirlik sağlamak çok önemlidir.
* Gerçek Zamanlı Uygulamalar: Oyun ve sohbet uygulamaları gibi sistemler durmadan modeli kullanan örneklerdir.

**Dezavantajları:**

* Model gereği istemci ve sunucu arasında senkronize bir iletişim vardır. Büyük ölçekli ve dağınık sistemlerde senkronizasyon sorununa neden olabilir. Özellikle yüksek trafikli sistemlerde bekleyen istek ve cevapların yönetilmesi sorun olabilir.
* İstek ve cevaplar ağ üzerinde iletilir ve işlenir. Ağ trafiğine neden olabilir ve iletişim maliyetini etkileyebilir.
* İsteğin ve cevabın tamamını ilettiği için bant genişliğini etkili şekilde kullanmaz.
* Sunucuya gönderilen isteğin cevaplanması için beklenir. Bunun sonucunda zaman gecikmesi ve performans sorunları olabilir.
* İletim sırasında ağ veya sunucuda sorunlarla karşılaşılabilir. Bu durumda istemcinin sunucu ile iletişimi tekrar kurma ve işlemleri tekrarlaması gerekebilir. Sonucunda işlem güvenilirliği azalabilir.

# **Sorular**

1. Asenkron kodlama nasıl yapılır? C# üzerinde asenc kodunun fonksiyonu nedir?