

Open in app ↗

Sign up

Sign in

Medium



Search



Onion Architecture, CQRS ve MediatR ile Sürdürülebilir ve Test Edilebilir Backend Mimarisi

1. Giriş

6 min read · Jul 10, 2026

B

Mahmut Bora Cihangir

Follow



Share

Bir yazılım projesi başlangıçta birkaç basit kod ile başlar. Ancak zaman içinde yazılım projeleri giderek büyür. Bu nedenle onlarca hatta yüzlerce sayfa ve dosyadan oluşan bir şekle bürünür. Bu durum projelerin karmaşıklığını arttırırken geliştirme aşamasının da proje ilerledikçe zorlaşmasına sebep olur. Eğer proje başında doğru mimari seçilmezse bu karışıklık giderek büyür. Kullanılan monolit mimari veya diğer mimariler bu karmaşıklığı giderme gibi durumlarda eksik kalabilir. Zamanla kullanıcı arayüzüne yapılan değişiklikler de, veri tabanı ile iletişim de bir yük haline gelir.

Bu durumlara kullanılmak için ve bu sorunlara çözüm olmak için Onion Architecture (Soğan Mimarisi) geliştirilmiştir. Bu mimari iş kurallarını yapının en merkezine alarak dış dünyaya yani veri tabanı, framework ve kullanıcı arayüzü gibi özelliklere olan bağımlılığı tersine çevirir. Bu mimari genellikle CQRS(Command Query Responsibility Segregation) deseni ile birlikte kullanılır. CQRS, okuma ve yazma işlemlerinin görevlerini birbirlerinden ayırarak kodun daha anlaşılır ve ölçeklenebilir olmasını sağlar. .NET ekosisteminde bu ikisini birlikte kullanmanın en yaygın yöntemi ise MediatR kütüphanesidir. MediatR, mediator tasarım deseni ile birlikte Command ve Query Handler sınıflarının temiz bir şekilde yönlendirilmesini sağlar.

2. Onion Architecture Nedir?

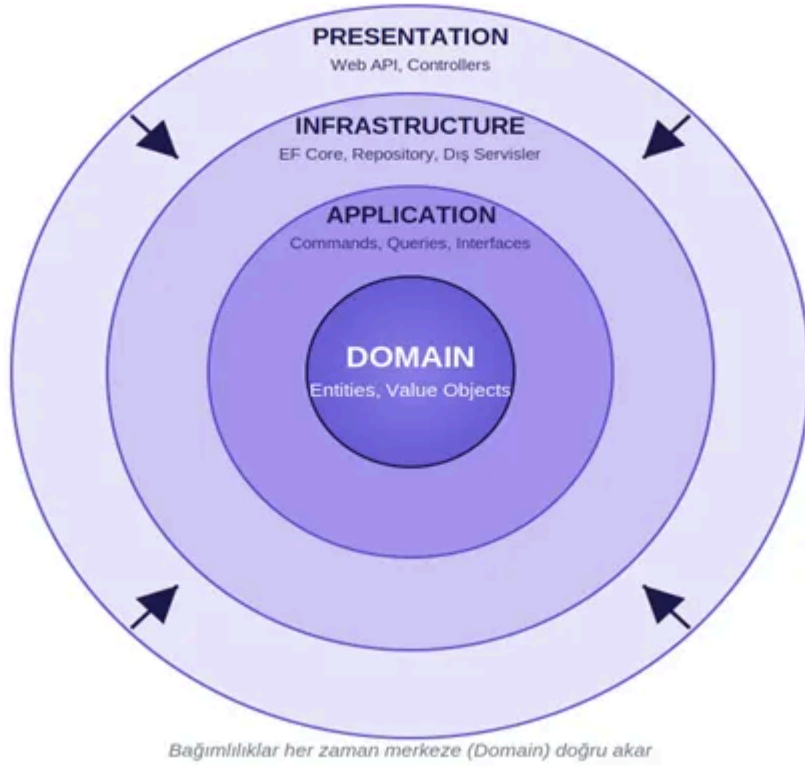
Onion Architecture, ilk olarak 2008 yılında Jeffrey Palermo tarafından tanıtılan bir yazılım mimarisidir. Palermo bu mimariyi tanıtmalarının en büyük sebebi N-tier(N katmanlı) mimarilerde sıkça karşılaşılan iş kurallarının veri tabanı ve dış servislerle doğrudan bağımlı olma sorunu çözmek istemesidir. Bu mimari Hexagonal Architecture ve Clean Architecture ile aynı felsefeye sahiptir. Merkezde iş kuralları yer almalı ve diğer detaylar bu merkeze bağımlı olmalıdır, merkez dışarıya bağımlı olmamalıdır.

Onion Architecture dört ana katmandan oluşur. Bunlar sırasıyla domain, application, infrastructure ve presentation olarak bilinir. Domain katmanı, entity'lerin, value object'lerin, domain servislerin ve domain exception'ların bulunduğu katmandır. Bu katman hiçbir katmana veya pakete bağımlı değildir. Sadece saf C# kodlarından oluşur.

Application katmanı, iş akışlarını tanımlar. CQRS kullanılıyorsa sınıfları ve handler'ları, DTO'lar ve dış dünya ile iletişime geçen arayüzler bu katmanda yer almaktadır. Bu katman domain katmanına bağımlıdır.

Infrastructure katmanı, application katmanında tanımlanan arayüzlerin somut implementasyonlarını bulundurur. Entity Framework Core ile veri tabanı bağlantısı, dış api entegrasyonları ve diğer teknik işlemler bu katmanda gerçekleştirilir.

Presentation katmanı, kullanıcıdan veya dış sistemlerden gelen istekleri karşılayan katmandır. Genellikle bir ASP.NET Core Web API projesi şeklinde yapılır. Controller'lar isteği doğrudan işlemek yerine MediatR üzerinden Application katmanına yönlendirir.



Şekil 1: Onion Architecture katmanları ve bağımlılık yönü

2.1 Onion Architecture Avantaj ve Dezavantajları

Onion Architecture birçok avantaj ve dezavantaj ile birlikte gelir. Bunlar arasında en önemli avantajlardan biri test edilebilir olmasıdır. Domain ve Application katmanlarının dışa bağımlılığı yoktur. Bu nedenle testler hızlı ve güvenli bir şekilde yapılabilmektedir. Bunların yanında teknolojik olarak bağımsız olması da büyük avantaj sağlar. Örneğin entity framework ile dapper arasında yapılacak geçişlerde bağımsız olmasından dolayı her şeyi değiştirmek yerine sadece Infrastructure katmanında değişiklik yapmak yeterlidir. İş kuralları etkilenmez. Ayrıca bağımlılık az olduğundan dolayı ekipler arası sorumluluk ayrımı nettir.

Dezavantajlarına gelecek olursak daha küçük çaplı projelerde gereksiz karmaşıklık ve maliyet ortaya çıkartır. Ayrıca yapısı karışık olduğundan dolayı mimariyi tam olarak bilmeyen bir ekibin bu mimariye alışıp proje yapması zaman alır.

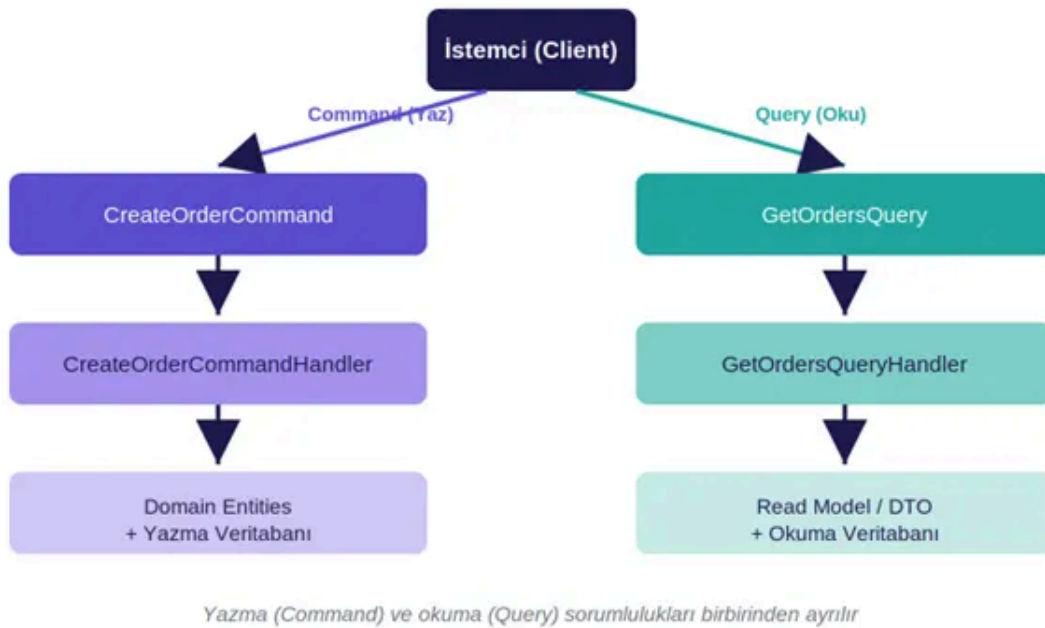
2.2 Onion Architecture ile Clean Architecture Arasındaki Fark

Clean Architecture 2012 yılında Robert C. Martin tarafından tanıtılmış olup Onion Architecture ve Hexagonal Architecture gibi mimarilerin ortak noktasını daha genel ve soyut bir şekilde yansıtmalarıyla ortaya çıkmıştır. İkisi de aynı temel felsefeye sahiptir. Bu bağımlılıkların içe doğru, iş kurallarına doğru akması gerektiğini savunan bir anlayıştır. Aralarındaki farklar daha çok sunumda ortaya çıkar. Onion

Architecture direkt olarak somut ve isimlendirilmiş 4 katmanı belirtirken Clean Architecture ise direkt olarak katman adı ve sayılarını tamamen belirtmeyip daha çok entities, use cases, interface adapters, frameworks & Drivers gibi daha soyut şekilde tanımlar. Clean Architecture'da use case kavramı çok belirgindir ve üzerine çok durulur. Her iş akışı ayrı bir use case sınıfı olarak modellenir ve katmanlar arası veri geçişi daha katı kurallarla belirlenir. Onion Architecture ise bu geçişi daha esnek tutar. Interface ve dependency injection ile bu geçişi sağlar. Günümüzde birçok kişi hala ikisini anlatırken karıştırmaya devam etmektedir.

3. CQRS (Command Query Responsibility Segregation)

CQRS açılımını Türkçeye çevirecek olursak komut sorgu sorumluluk ayrımı olarak çevrilebilir. Bu desen Bertrand Meyer'in nesne yönelimli programlama için tanımladığı Command Query Separation (CQS) prensibinden esinlenmektedir. CQS prensibine göre bir method ya bir işlem yapmalı (command) ya da bir veri döndürmelidir (query) ama ikisini aynı anda yapmamalıdır. Greg Young ise bu prensibi mimariye uygulanmasını sağlayarak CQRS deseninin yaygınlaşmasını sağlamıştır.



Şekil 2: CQRS deseninde Command ve Query akışlarının ayrılması

CQRS'in temel fikri, bir yazılımdaki okuma (query) ve yazma (command) işlemlerinin farklı sorumluluklara sahip olmasıdır. Normalde eski yöntem ve geleneksel yöntemlerde bir sınıf hem okuma hem yazma işlemlerini yapar. Belli bir süre sonra bu sınıf şişer. Bunun sonucunda bakım ve geliştirme yapmak zorlaşır. CQRS ile birlikte her iş akışı kendi command ya da query handler'ı ile birlikte izole

hale gelir ve bu sorunun önüne geçilir. Bu ayrımın sağladığı faydalar arasında büyük projelerde büyük kodların küçük parçalara bölünmesi, okuma tarafının optimize edilerek daha iyi performans sağlaması ve büyük çalışmalarda okuma ve yazmanın birbirinden ayrılması olarak söylenebilir. CQRS daha çok kod organizasyonu sağlayan bir desen olarak kullanılır.

Get Mahmut Bora Cihangir's stories in your inbox

Join Medium for free to get updates from this writer.

Enter your email

Subscribe

☐ Remember me for faster sign in

Command sistemde ekleme, silme ve güncelleme gibi durum değişiklikleri yapar. Genellikle geriye sonuç döndürmez ya da bir sonuç verir. Sorgulama işlemi yapılmaz. Query ise sistemde veri okur yani listeleme gibi işlemleri yapar ve bu da veri tabanında herhangi bir değişikliğe sebep olmaz.

```
// Command
0 references
public record CreateOrderCommand(string CustomerName, decimal TotalAmount)
|   : IRequest<Guid>;
// Query
0 references
public record GetOrderByIdQuery(Guid OrderId)
|   : IRequest<OrderDto>;
1 reference
public record OrderDto(Guid Id, string CustomerName, decimal TotalAmount, string Status);
```

Şekil 3: Basit Command ve Query Sınıfı Örneği

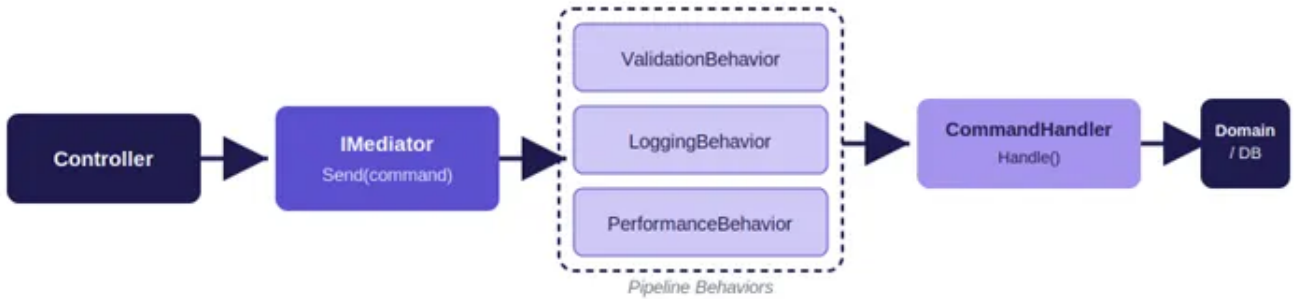
3.3 CQRS ile Onion Architecture Arasındaki İlişki

CQRS, Onion Architecture içinde Application katmanında doğal bir organizasyon biçimi sağlar. Projede bulunan bütün özellikler kendi command, query, handler ve validator sınıflarıyla birlikte gruplanır. Bu sayede domain katmanını CQRS yapısından haberdar olmaz. CQRS tamamen application katmanında kullanılan bir yapıdır ve domain yapısını etkilemez.

İleri seviye senaryolarda yani daha büyük projelerde CQRS, event sourcing ile birlikte kullanılarak okuma ve yazma için birbirinden ayrı veri tabanları kullanılabilir. Ancak bu, CQRS'in zorunluluklarından biri değildir. Günümüzde çoğu projede tek bir veri tabanı üzerinden kullanılmaktadır. CQRS sadece kod organizasyonu ve sorumluluk ayrımı olarak kullanılmaktadır.

4. MediatR Nedir?

MediatR, Jimmy Bogard tarafından geliştirilen, .NET için hafif ve açık kaynaklı bir kütüphanedir. Mediator tasarım desenini uygulayarak, nesneler arasındaki doğrudan referansları ortadan kaldırır. Tüm istek ve yanıt akışını merkezi bir yapı üzerinden yönetir. CQRS ve MediatR birlikte kullanıldığında, controller'lar servis sınıflarını değil IMediator arayüzüne bağımlı hale gelir.



İstek, Handler'a ulaşmadan önce ortak davranışlardan (pipeline) geçer

Şekil 4: MediatR üzerinden bir isteğin pipeline behaviors'lardan geçerek Handler'a ulaşması

MediatR tasarım deseni, nesneler arasındaki çoklu, karmaşık ilişkileri tek bir merkez ile yöneterek daha basit bir yapı haline getirmeyi amaçlar. MediatR açısından bu hangi handler ile işlem yapılacağını bilmek zorunda olmadığı anlamına gelir. Controller command ya da query oluşturur ve send metoduna gönderir. MediatR gönderilen metodun tipine bakarak ilgili handler'ı çalıştırır. Örnek verecek olursak burada uçakların kontrol eden kuleyi örnek verebiliriz. Bir uçak gelir ve diğer uçaklarla iletişime geçmek için kuleyi kullanır. Bu kule işte mediatR'ı temsil eder.

CQRS ve MediatR desenleri, uygulama üzerindeki her işlemi birer command veya query nesnesine dönüştürerek yönetmemizi sağlar. Gelen istek doğrudan veri tabanına ulaşmaz. İlk olarak command'e iletilir. Daha sonra bu komut, handler tarafından işlenerek veri tabanına kaydedilir. Controller katmanını bu akışta sadece bir yönlendirici olarak çalışır. Herhangi bir iş mantığı barındırmaz. Bu nedenle veri

erişim katmanına doğrudan bağımlılığı kalmaz. Bu durum, projenin test edilebilirliğini ve bakım kolaylığını ciddi oranda artırır.

MediatR'ın Pipeline Behavior özelliği ise doğrulama (validation), loglama ve performans ölçümü gibi ortak ihtiyaçları tek merkezden yönetmeye yarar. İstek handler sınıfına ulaşmadan önce bu arayüz devreye girerek verilerin kurallara uygunluğunu kontrol eder. Olası bir hata durumunda akış kesilerek doğrudan yanıt döner. Böylece hatalı isteklerin handler'a ulaşamaz. Bu yapı sayesinde geliştiriciler, handler sınıflarında yalnızca uygulamanın ana iş kurallarına odaklanır.

5. Onion Architecture, CQRS ve MediatR Birlikte Kullanımı?

Onion Architecture, CQRS ve MediatR entegrasyonu, uygulamalarda katmanlı ve sürdürülebilir bir mimari kurmanın en etkili yollarından biridir. Bu modelde istekler Controller'dan alınıp MediatR ile doğrudan Command/Query handler'larına iletilir. Domain katmanı tamamen merkezde konumlanır. Domain katmanı, dış katmanlardan veya veri tabanı işlemlerinden habersiz şekilde yalnızca iş kurallarını barındırır. Bu izolasyon bağımlılıkları minimize ederken, test edilebilirliği ve kod kalitesini artırır.

Sistemin verimli çalışması için Domain katmanının bağımsızlığı korunmalıdır. CQRS prensiplerine sadık kalınarak her handler tek bir sorumluluk üstlenmelidir. Ortak ihtiyaçlar pipeline yapısıyla çözülmelidir. Son olarak günümüzde var olan mimariler incelendiğinde hiçbir mimariye en iyisi diyemeyiz. Her mimari farklı ihtiyaçlara göre farklı sonuçlar verir. Bu nedenle ihtiyaçlarımızı doğru belirleyip bize en uygun mimariyi seçmeliyiz.

6. Kaynaklar

<https://www.gencayyildiz.com/blog/nedir-bu-onion-architecture-tam-teferruatli-inceleylim/>

<https://server78135.medium.com/onion-architecture-nedir-aade61f04b8e>

<https://roshancloudarchitect.me/comparing-onion-architecture-vs-1ab4863419e6>

<https://dev.to/godofgeeks/onion-architecture-vs-clean-architecture-584h>

<https://medium.com/sahibinden-technology/cqrs-design-pattern-nedir-neden-kullan%C4%B1%C4%B1r-bir-cqrs-design-pattern-i%C4%B1n-incelemesi-ced6713abc6e>

<https://eryilmaz0.medium.com/mediatr-nedir-4913b99784dd>

<https://hakantopuz.medium.com/net-core-ve-mediatr-ile-cQRS-pattern-kullan%C4%B1m%C4%B1-d2e5edc4fd07>

Onionarchitecture

Backend Development

API

CQRS

Mediatr



Follow

Written by Mahmut Bora Cihangir

0 followers · 1 following

