1-Rapor Giriş

Bu projede Onion mimarisi ve CQRS'e dair birçok şey öğrendim. Bir mimariyle çalışmanın avantajı olarak, değişiklik yapmam gerektiğinde bunu koda rahatça yansıtabildim. Database olarak MySQL kullanıldı.

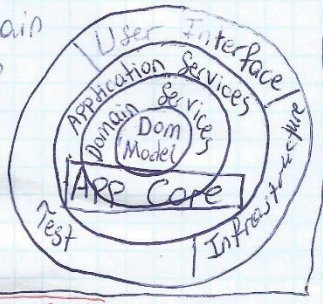
Çalışmamın temelini attığım youtube videolarının linkini ve çalışırken aldığım notları aşağıdaki bölüme ekledim. Ayrıca youtube videolarına paralel olarak kendim kodladığım ve sonuncusunu referans proje olarak kullandığım 3 adet alıştırma projesini Alıştırmalar klasörüne ekledim.

2- Defter

ONION ARCHITECTURE

Tech Buddy (Link 0)

★ Domain + App = Core



★ Amaç loose coupling (Gevşek Bağlılık) oluşturabilmek. Domain Driven Desing, CQRS, Onion Arch, Clean Arch gibi kavramların tamamının amacı Loose coupling.

★ Eğer uygulamayı katmanlara ayırabilirsek ve bu katmanlar arasındaki bağlantıyı minimize edersek, bir değişiklik olduğunda (Db değişikliği, UI değişikliği gibi) değişiklikler büyük olsa da bunları uygulamaya kolayca yansıtabiliriz.

★ SOLID:

- S: Single responsibility Principle - Her katmanın kendi görevini yapması.
- O: Open/closed principle - O katman üzerinde değişikliğe kapalı, gelişime açık bir yapı kurulması. Yani var olanı değiştirme ihtiyacı doğmasın ancak ekstra özellikler eklemek istediğimizde genel yapı bundan etkilenmesin.
- L: Liskov substitution principle - Kodlarımızda herhangi bir değişiklik yapmaya gerek duymadan, alt sınıflar türedikleri (üst) sınıfların yerine kullanabilmeliyiz.
- I: Interface Segregation Principle - Bir sınıfın tüm işlemlerini tek bir arayüze yüklemek yerine, ihtiyaçlara göre birden çok arayüz oluşturmalıyız. (Link 1)
- D: Dependency Inversion Principle - Üst seviye sınıflar, alt seviye sınıflara doğrudan bağımlı olmamalıdır. Bağımlılıklar, abstraksiyonlar üzerinden sağlanmalıdır. (Link 1)

★ Onion'daki temel fikir, içteki'nin dıştakine değil, dıştakinin içtekine bağımlı olmasıdır. Yani UI → App Ser'e App Ser → Dom Ser'e bağımlı olma. Amaç, değişiklik gerektiğinde en az zahmete girmek.

★ Yani Dom. Serv. den Dom Model'e ulaşabilmelisin ancak tersini yapamamalısın.

★ Onion Arch, Clean Arch'ı uygulamak için bir yöntemdir. (S1)

★ Domain tarafında veritabanı objeleri, veritabanına gidip gelirken kullanacağımız entity'ler ve bu entity'lere hizmet edecek (içerisinde kod bulunmayan) başka objeler bulunur. Application katmanı DTO'ları içerir, mapping ve conversion burada yapılır. Application'da tanımlanan repository'ler ve interface'ler Infrastructure'da tanımlanır. Yani kurallar Application'da belirtilir ve Infrastructure'da uygulanır.

★ Yan yana olan katmanlar birbirleriyle haberleşebiliyor.
★ Persistence katmanı, Infrastructure katmanının içinde bulunur.
★ İstekler dışarıdan içeriye gelir. Yani, infrastructure application'a gidip senin interface'ini kullanarak içeriyi doldurdum, der. Presentation'dan Application Rule'larına ulaşmana gerek kalmıyor, Presentation Infra'ya ulaşarak, infra o datayı Application'a iletilmesi ve oradan Application Rule'ların çalıştırılması işlemlerini yapacak. Presentation'da yalnızca göstermekle ilgilenmiş olacağız.

Tech Buddy-2 (Link 2)

★ Bu videoda Onion mimarisine uyduramadığımız noktalar olacak. CQRS ve MediatR Pattern ile bu sorunu gözeceğiz.

★ En dıştan da en içe erişebilirsin, sıkıntı yaratmaz. Yani Infra'dan Domain'e erişebilirsin. Ama bunu yapmasan daha iyi olur.
- Bunu yapmayarak daha temiz şitmenin yolu CQRS + MediatR

Tech Buddy-3 (Link 3)

★ CQRS'in temel amacı, sistemimizde oluşacak isteklerin durumlarının kontrolü. Sistem üzerinde oluşan istekler 2'ye ayrılıyor. Veriyi manipüle eden istekler Command, veri üzerinde değişiklik yapmadan veriyi dış dünyaya servis eden Query. Burada tek veritabanı da kullanılabilir: Read ve Write'e olacak 2 Db'de kullanılabilir. 2 Db kullanmak daha uygun ancak Db'lerin senkronize olması gerekiyor. Senkronizasyon için genellikle Projections Pattern kullanılır. S2

★ S3'te event store Db'sinin bulunmasının sebebi, eğer Event Bus ve sonraki adımlarda bir kopukluk olursa, bu eventin tekrar event-store'dan fırlatılmasını sağlamak. S4'te daha ayrıntılı bir şema var.

★ Query Handlerları ve Command Handlerları oluşturabilmek ve yakalayabilmek için de MediatR isminde bir MediatR patterni kullanıyoruz.

★ Havaalanına inecek olan veya pistten kalkacak olan uçaklar birbirleriyle doğrudan haberleşemezler. Hepsi kule aracılığıyla birbirine mesaj gönderebilir, durumlarını oraya bildirebilir. MediatR'da da olay bu. Modeller birbirleriyle konuşamıyor, herkes kendini MediatR'a anlatıyor, MediatR, gelen bir command ise Command Handlera, query ise Query Handlera yönlendirir.

★ Controller'da yarattığımız query'i mediatr ile send ederiz. Send metodu ilgili query'nin handler'ini bulacak ve handler içindeki handler'i çağıracak ve buradan bir viewmodel geriye dönecek. Sonuçta Controller buradan gelen datayı dışarı verecek.

★ Program.cs'teki 'builder.Services.AddMediatR(...)' IRequest'ten türetilen ve IRequestHandler'den türetilen classların taranmasını sağlar.

★ Bir sonraki videoda Onion Arch'a MediatR patterni ve CQRS'i, Onion Arch ile kurduğumuz ilk yapıya giydireceğiz.

⊛ Bu kodda Post sadece Guid döndürüyor. Yani post işlemi yapmıyor, sadece çalıştığını görmek için yazdık.

Tech Buddy - 4 (Link 4)

★P1 /WebApi/Controllers/ProductController deki GetU metodunun temel problemi; entity katmanının repository üzerinden controller tarafına ulaşması ve ve bizim dışarı dönmek için hazırlamış olduğumuz ProductViewDto yerine Controller tarafında entity modelin elimizde olması. Burada mappingi controller üzerinde yapmıştık ancak bu mapping işlemlerinin Application içerisinde olması gerekiyor.

★Command ve Query leri Features klasöründe ~~yazacağız~~. bulunduracağız.

★GetAllProducts Query 'deki Handle 'da S4 yaptık. Buna mapping denir. Bunu değiştireceğiz ve AutoMapper adlı paketi bu amaçla kullanacağız. Bunu GeneralMapping.cs de yapacağız. S5

★Artık Controller 'da Repository 'e değil Imediatr 'a erişmemiz gerekiyor. Controller 'da artık Features/Query 'deki GetAllProductsQuery 'e ulaşacağız.

★Buradaki loose coupling 'e örnek: Evet Application ile Controller bağlı ancak sadece Features (Queries-Commands) vasıtasıyla bağlı, işte bu sadecenin varlığı loose coupling.

★InMemoryDb her kapatıp açtığımızda sıfırlanır.

★Clean Arch: Yaptığın uygulama içerisinde bağlılıkları azaltarak geliştirme yapmayı kolaylaştırmak. Onion Arch bir örneğidir.

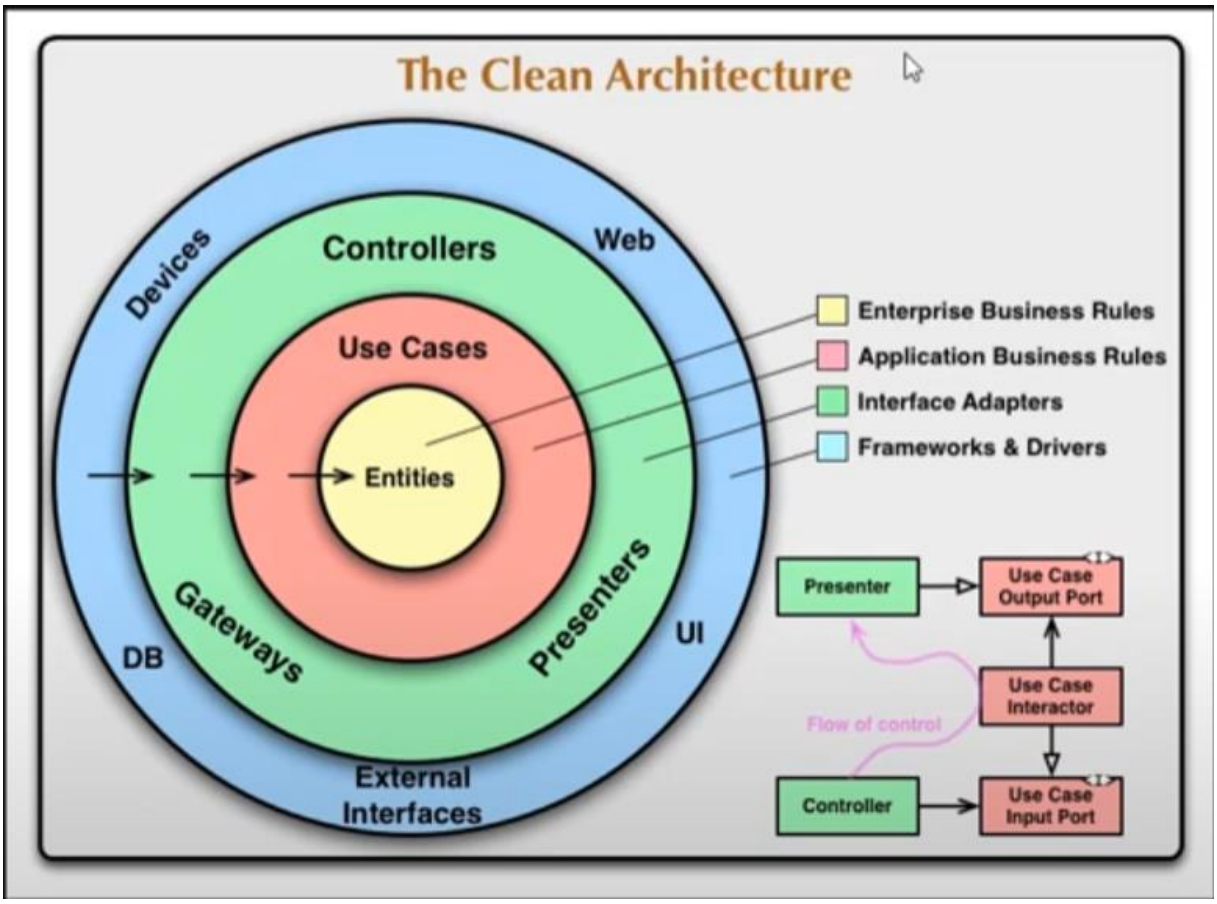
★Domain + Application bizim için Core oldu. Infrastructure tarafında sadece Persistence 'i kullanıyoruz verilerin kalıcı olması için. Ancak external bir API 'ye de gitmeniz gerektiğinde veya identity server 'a gitmemiz gerektiğinde o projeyi de yine infrastructure altında konumlandırırız. Önemli olan, en baştaki grafikte her zaman dışarıdan içeriye gitmek.

★Yeni bir class ekleyeceksen ve bunu nereye ekleyeceğini bilmiyorsan, bunu eklediğinde dıştan içe kuralını bozup bozmayacağını düşün.

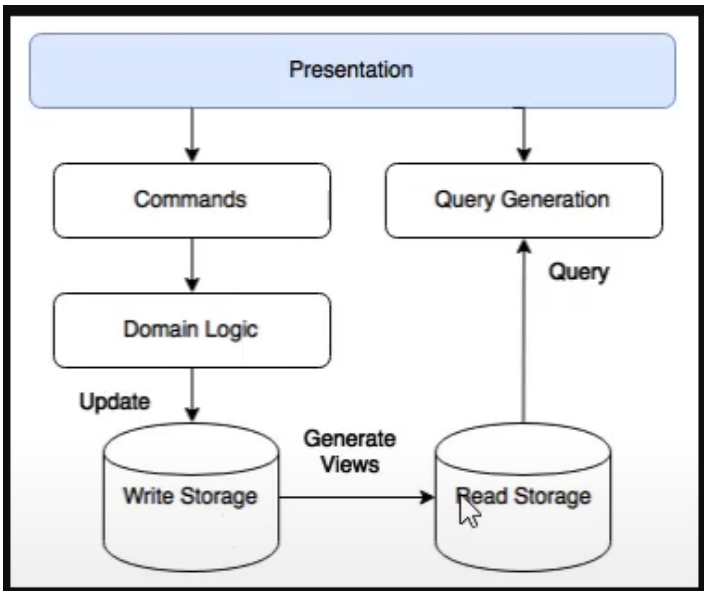
3-Defter Kaynakça

0- Tech Buddy: https://www.youtube.com/watch?v=Q1XyDTmm4tw
1- https://medium.com/@fdikmen/solid-prensipleri-7eeca8dbb499
2- Tech Buddy-2: https://www.youtube.com/watch?v=CCWTITXALGo
3- Tech Buddy-3: https://www.youtube.com/watch?v=GDKy2xZsZhs
4- Tech Buddy-4: https://www.youtube.com/watch?v=KIhMeJ-jYME

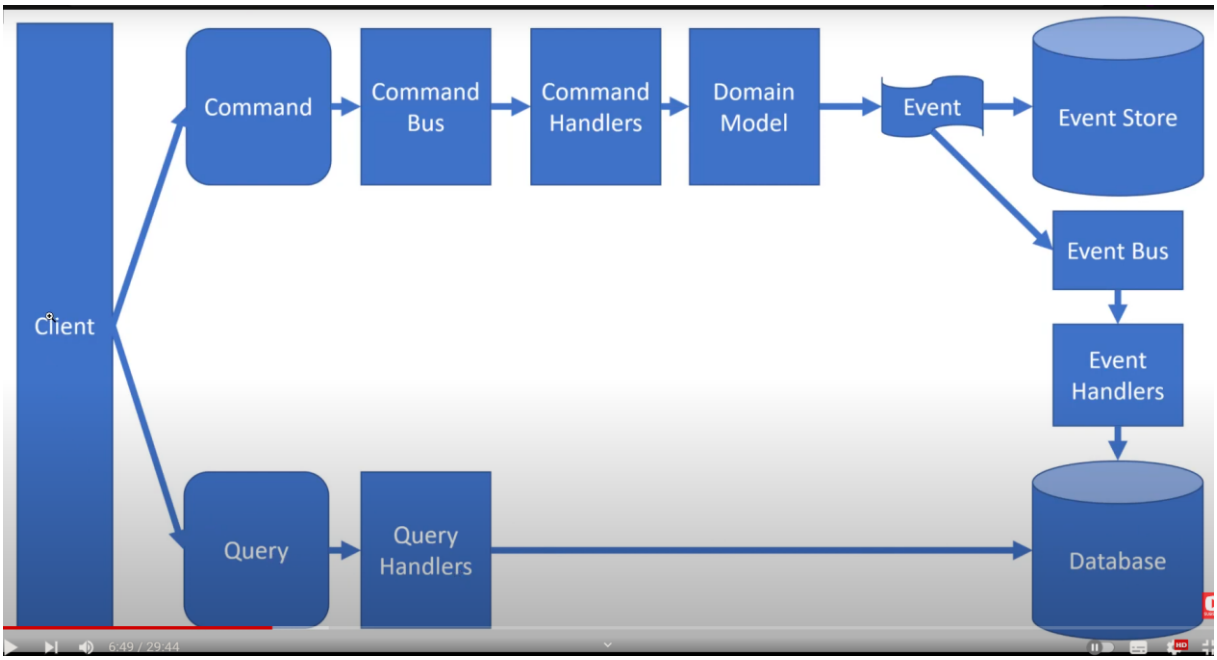
S1-



S2-



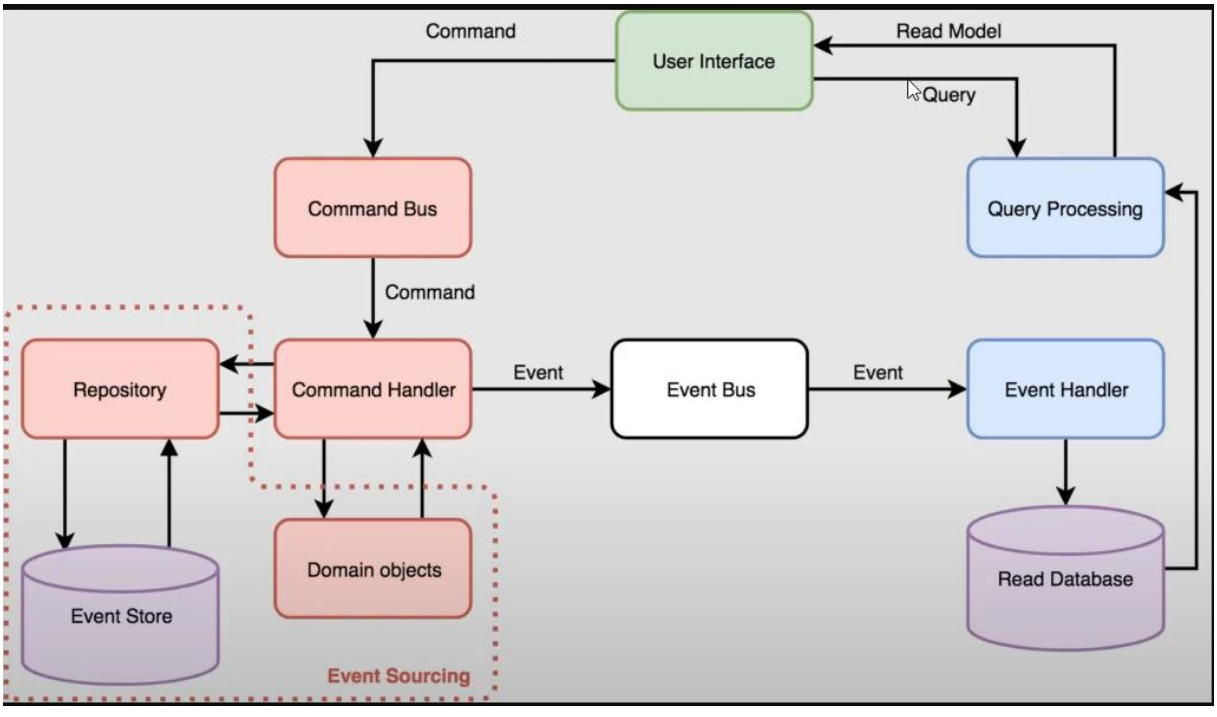
S3-



S4-

```
0 references
public async Task<List<ProductViewDto>> Handle(GetAllProductsQuery request, CancellationToken cancellationToken)
{
    var products = await productRepository.GetAllAsync();

    return products.Select(i => new ProductViewDto()
    {
        Id = i.Id,
        Name = i.Name
    }).ToList();
}
```



S5-

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using ProductApp.Application.Dto;
using ProductApp.Application.Interfaces.Repository;

namespace ProductApp.WebApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 reference
    public class ProductController : ControllerBase
    {
        private readonly IProductRepository productRepository;

        0 references
        public ProductController(IProductRepository productRepository)
        {
            this.productRepository = productRepository;
        }
        [HttpGet]
        0 references
        public async Task<IActionResult> Get()
        {
            var allList = await productRepository.GetAllAsync();
            var result = allList.Select(i => new ProductViewDto
            {
                Id = i.Id,
                Name = i.Name
            }).ToList();

            return Ok(result);
        }
    }
```

```
using MediatR;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using ProductApp.Application.Dto;
using ProductApp.Application.Features.Queries.GetAllProducts;
using ProductApp.Application.Interfaces.Repository;

namespace ProductApp.WebApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 reference
    public class ProductController : ControllerBase
    {
        private readonly IMediator mediator;

        0 references
        public ProductController(IMediator mediator)
        {
            this.mediator = mediator;
        }
        [HttpGet]
        0 references
        public async Task<IActionResult> Get()
        {
            var query=new GetAllProductsQuery();

            return Ok(await mediator.Send(query));
        }
    }
```