<epam>

Чистая архитектура на практике

Андрей и Денис Цветцих

Lead Developers

О нас

- Практикуем чистую архитектуру
- Помогаем коллегам чистить проекты

Зачем?

- Книга
- Видео / статьи / habr
- Github – проекты с 4k звезд
- Курсы

Как оказалось:

- Книга отличная, но абстрактная
  Как раскладывать код по папкам – не ясно ☹

- Все остальные материалы:
  - Просто пиар
  - Поверхностный пересказ книги
  - Pet project
  - Как раскладывать код по папкам – все равно не ясно ☹

О чем поговорим:

- Что такое чистая архитектура?
- Чем отличается от других архитектур?
  - Слоистая
  - Луковая
  - Порты-адаптеры
- Возьмем все самое лучшее из разных архитектур
- Соберем шаблон нового проекта

Чистая архитектура

Роберт Мартин
“Uncle Bob”

Чистая архитектура

Чего хотелось?

- Стандартизации
- Быстрого старта новых проектов
- Быстрого переключения разработчиком между проектами
- Меньше боли при поддержке существующих проектов
- Гордиться своей работой

Что такое архитектура?

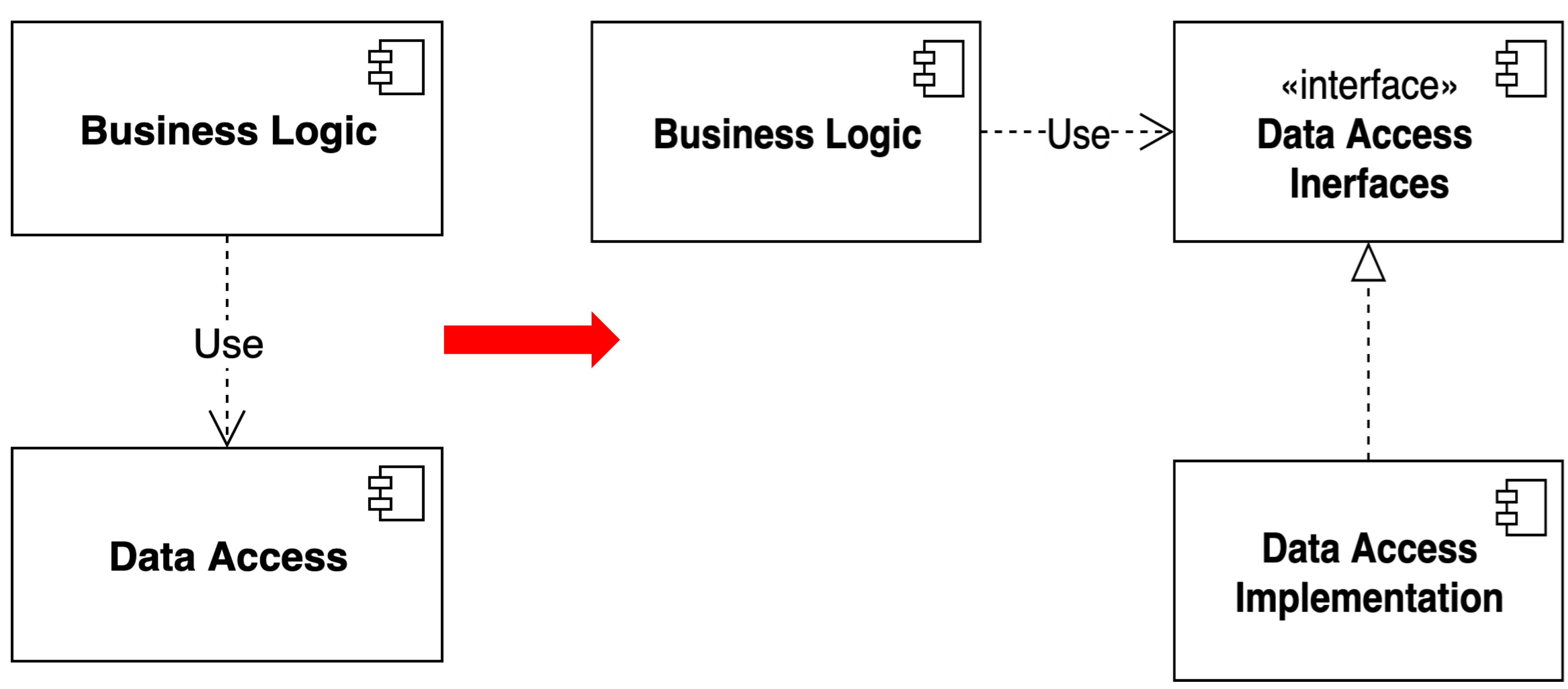
Святая троица:

- Компоненты системы
- Их публичные контракты
- Связи между компонентами

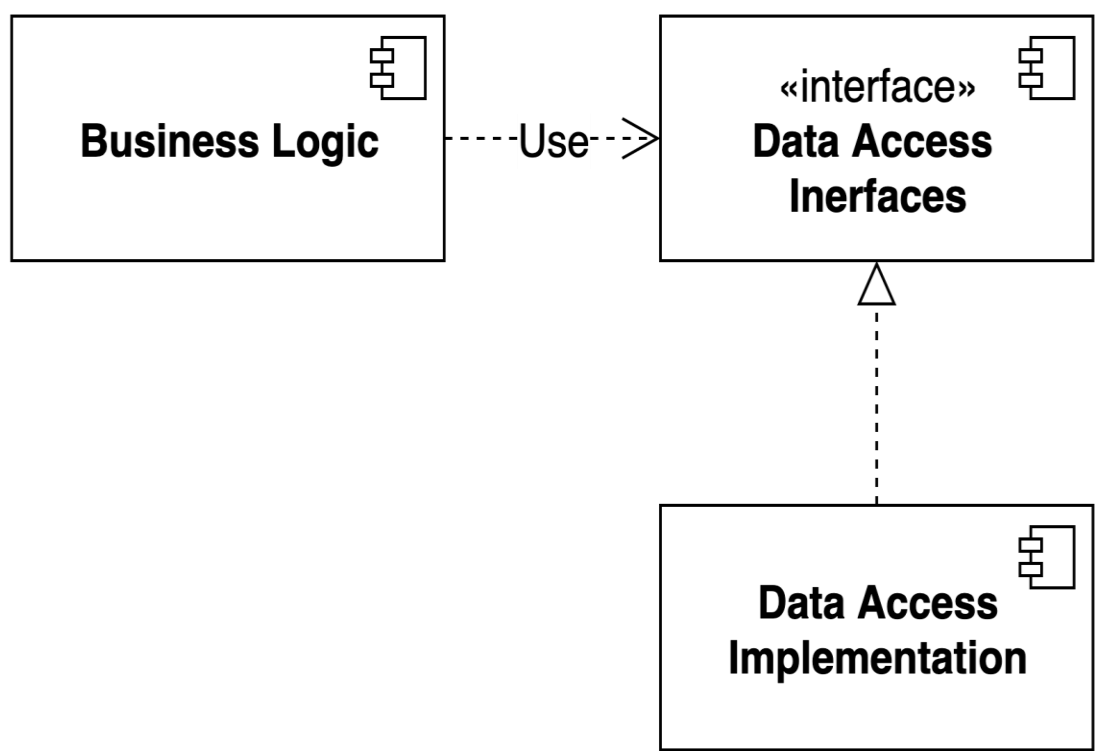
Что получим:

- Как раскладывать файлы по папкам:
    - Какие компоненты нужно создать
    - Что конкретно положить в каждый компонент
- Что является контрактом каждого компонента
- Как правильно настроить связи между компонентами

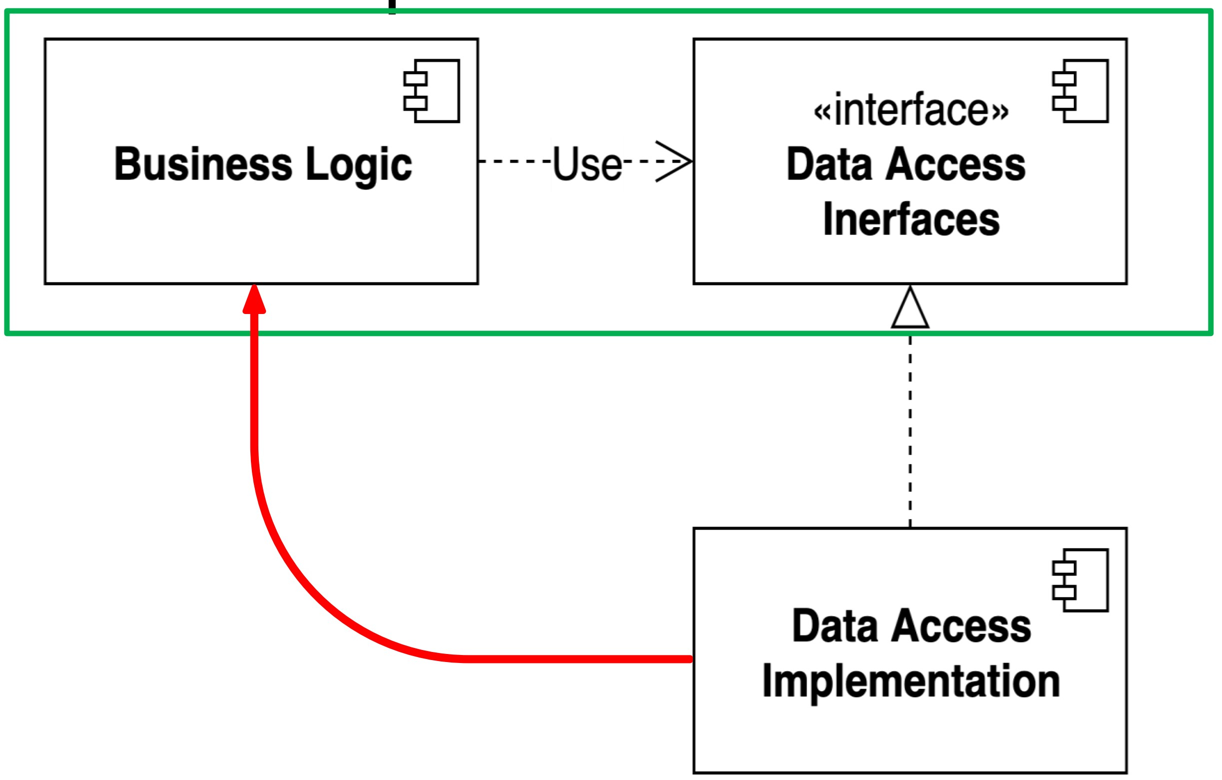
Инверсия зависимостей



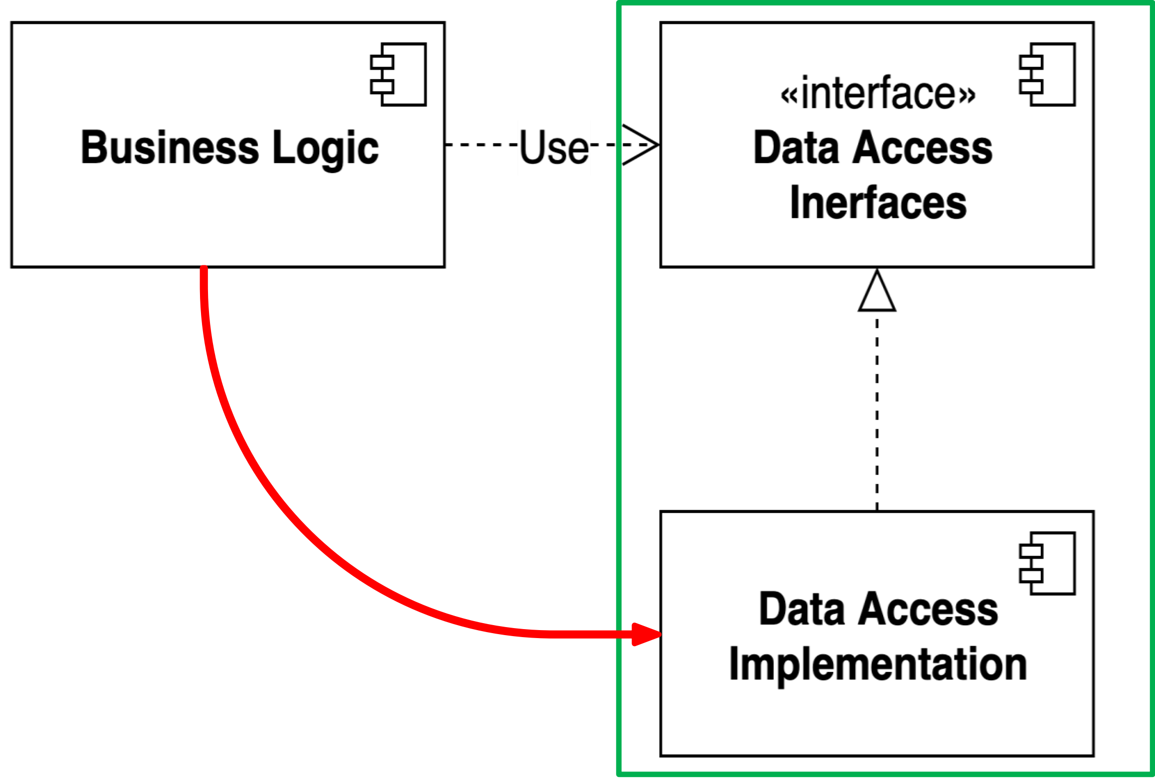
Упаковка по проектам 1



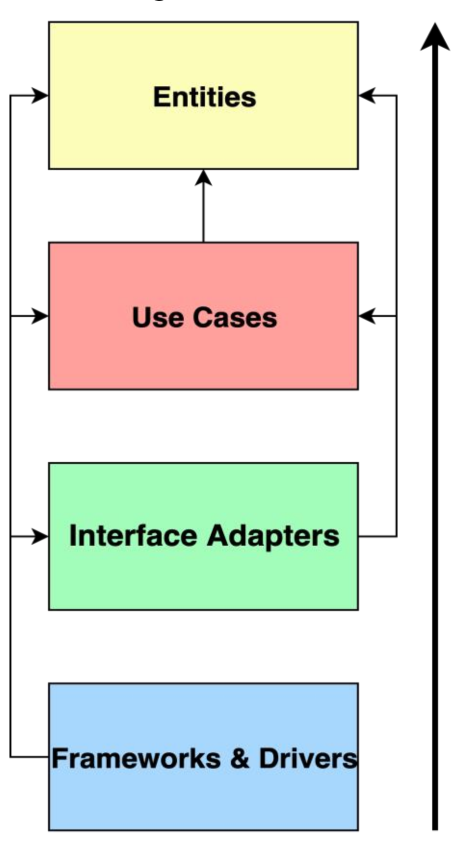
Упаковка по проектам 2



Упаковка по проектам 3



Чистая Архитектура



Controllers, Gateways, Presenters

DB, Web, UI, Devices, External Interfaces

Entities

- Бизнес логика
- Данные и правила их обработки
  - Rich model
  - Anemic Model + Domain Services
- Нет интерфейсов репозиториев
- Не обязательно классы

Use Cases

- Логика приложения
- Не зависят от фреймворков (ORM, Web)
- Не знают откуда вызываются
- Сервисы или CQRS

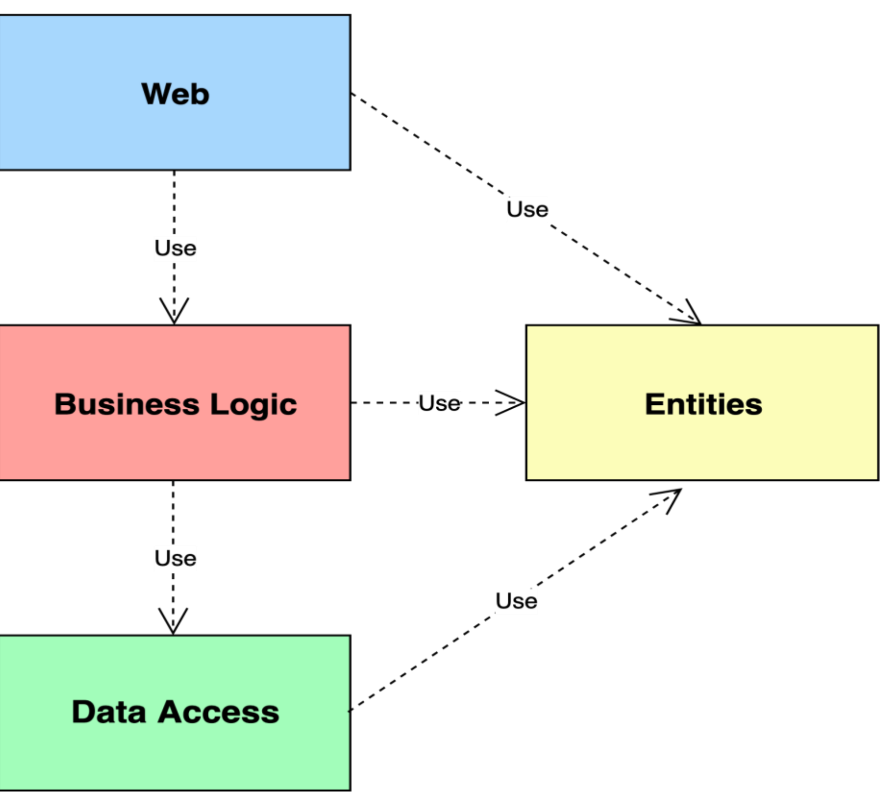
Interface Adapters

- Связывают Use Cases с внешним миром
- Контроллеры
- Доступ к данным
- Внешние сервисы
- Timer / Service Bus workers

Frameworks

- База данных
- Web
- Swagger
- Тесты

Слоистая Архитектура



Слоистая Архитектура: Проекты

Solution 'Dotnext.CleanArchitecture' (4 of 4 projects)

- 1-Dotnext.Entities
- 2-Dotnext.DataAccess.Sqlite
- 3-Dotnext.BusinessLogic
- **4-Dotnext.Web**

Слоистая Архитектура: Entities

- 1-Dotnext.Entities
  - Dependencies
  - Order.cs
  - OrderItem.cs
  - Product.cs

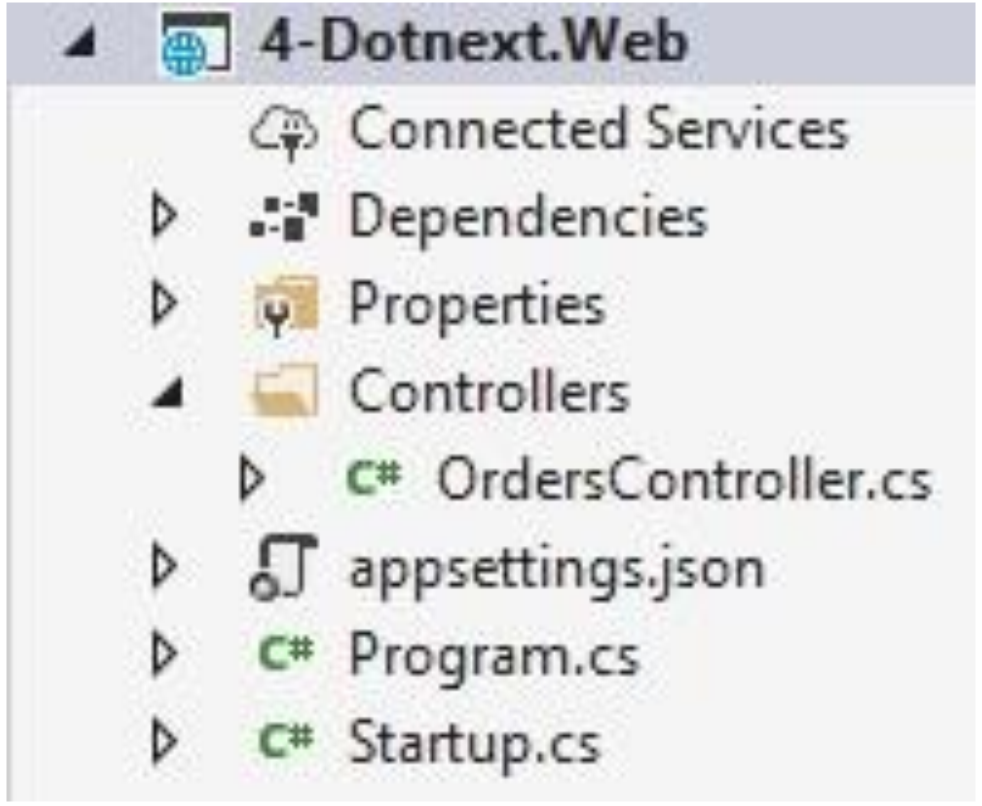
Слоистая Архитектура: Data Access

- 2-Dotnext.DataAccess.Sqlite
  - Dependencies
  - Configurations
  - Migrations
  - SqliteDbContext.cs

Слоистая Архитектура: Business Logic

3-Dotnext.BusinessLogic
- Dependencies
- IOrdersService.cs
- OrdersService.cs

Слоистая Архитектура: Web



Shared компонент

- Dotnext.Shared
  - Dependencies
  - Extentions
    - CollectionExtensions.cs
    - DateTimeExtensions.cs
    - StringExtensions.cs
  - Pagination
  - Services
    - IDateTime.cs
  - Utils
    - EnumUtils.cs

Shared компонент

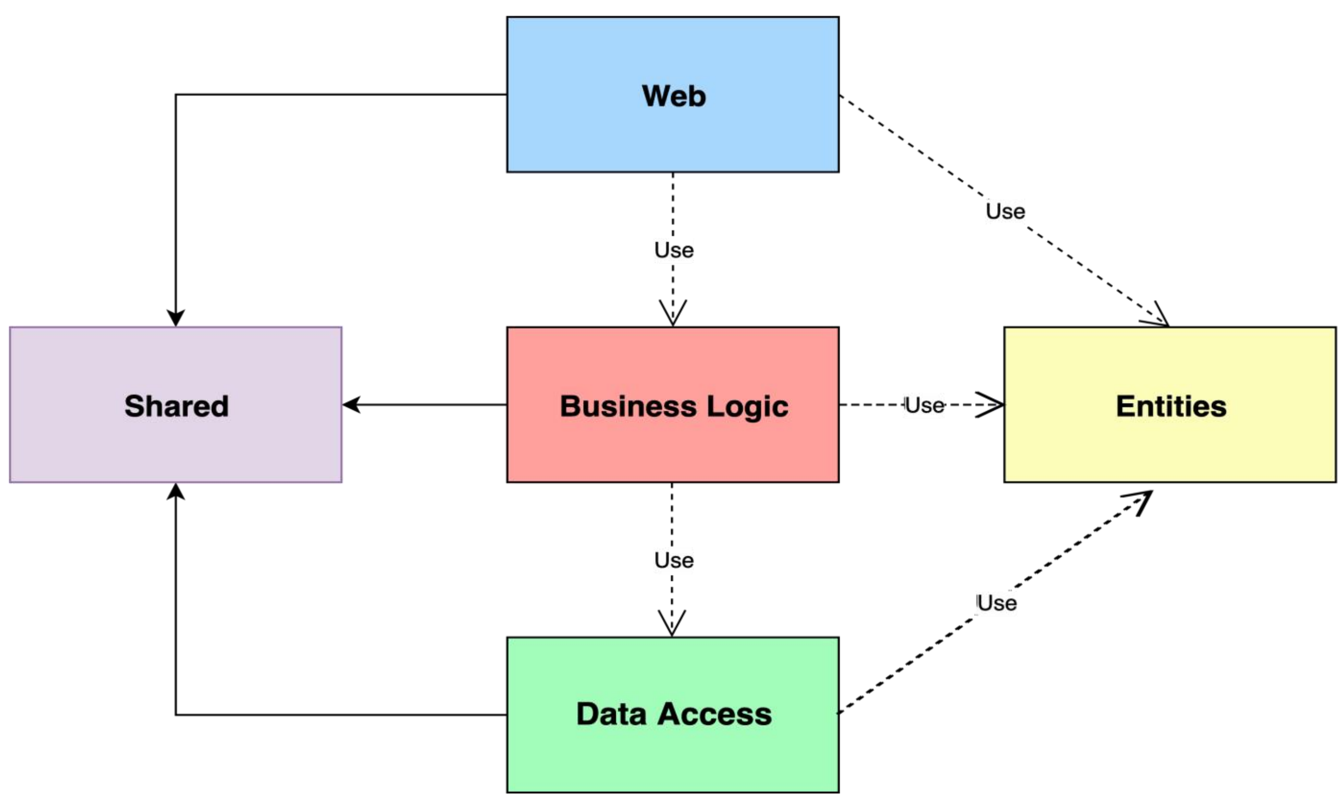
- IDateTime
  - UtcNow

- Extensions / Utils
  - Date
  - String
  - …

- Pagination

- Validation

Shared компонент



Shared компонент

Solution 'Dotnext.CleanArchitecture' (5 of 5 projects)
- 0-Dotnext.Shared
- 1-Dotnext.Entities
- 2-Dotnext.DataAccess.Sqlite
- 3-Dotnext.BusinessLogic
- **4-Dotnext.Web**

Shared компонент

- Epam.DotNet.Shared
- Epam.DotNet.Shared.EF6
- Epam.DotNet.Shared.EFCore
- …

Shared компонент - резюме

- Контракт:
  Все классы публичные

- Связи:
  - Не зависит от других проектов
  - От него может зависеть любой другой проект

Видишь Shared проект?
А он есть!

Entities

- 1-Dotnext.Entities
  - Dependencies
  - Enums
    - OrderType.cs
  - Exceptions
    - DomainException.cs
  - Models
    - Order.cs
    - OrderItem.cs
    - Product.cs

Entities

- Сущности
- Перечисления
- Исключения
- Доменные события
- Базовые классы – супертип слоя

Entities

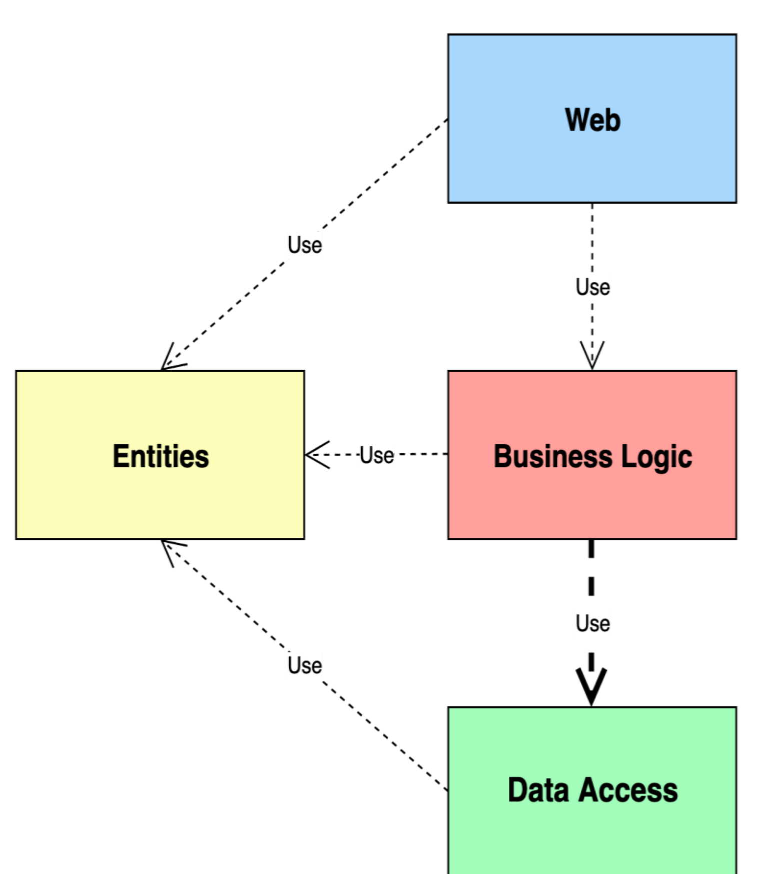
```
public class Order : Entity
{
    public OrderType Type { get; set; }

    public string ExternalId { get; set; }

    public ICollection<OrderItem> Items { get; set; }

    public decimal CalculateTotal() =>
        Items.Sum(x => x.Price * x.Count);
}
```

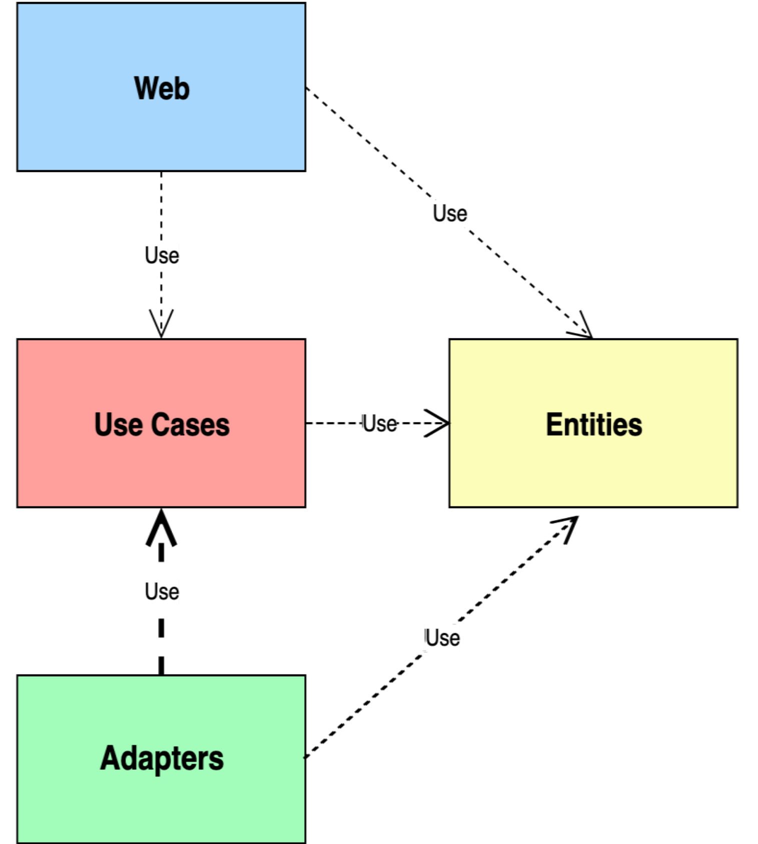
Entities - резюме

- Контракт:
  Все классы публичные

- Связи:
  - Не зависит от других проектов (Shared не в счет)
  - От него может зависеть любой другой проект

Слоистая
Чистая


Слоистая
Web
Use
Use
Entities
Use
Business Logic
Use
Use
Data Access


Чистая
Web
Use
Use
Use Cases
Use
Entities
Use
Use
Adapters

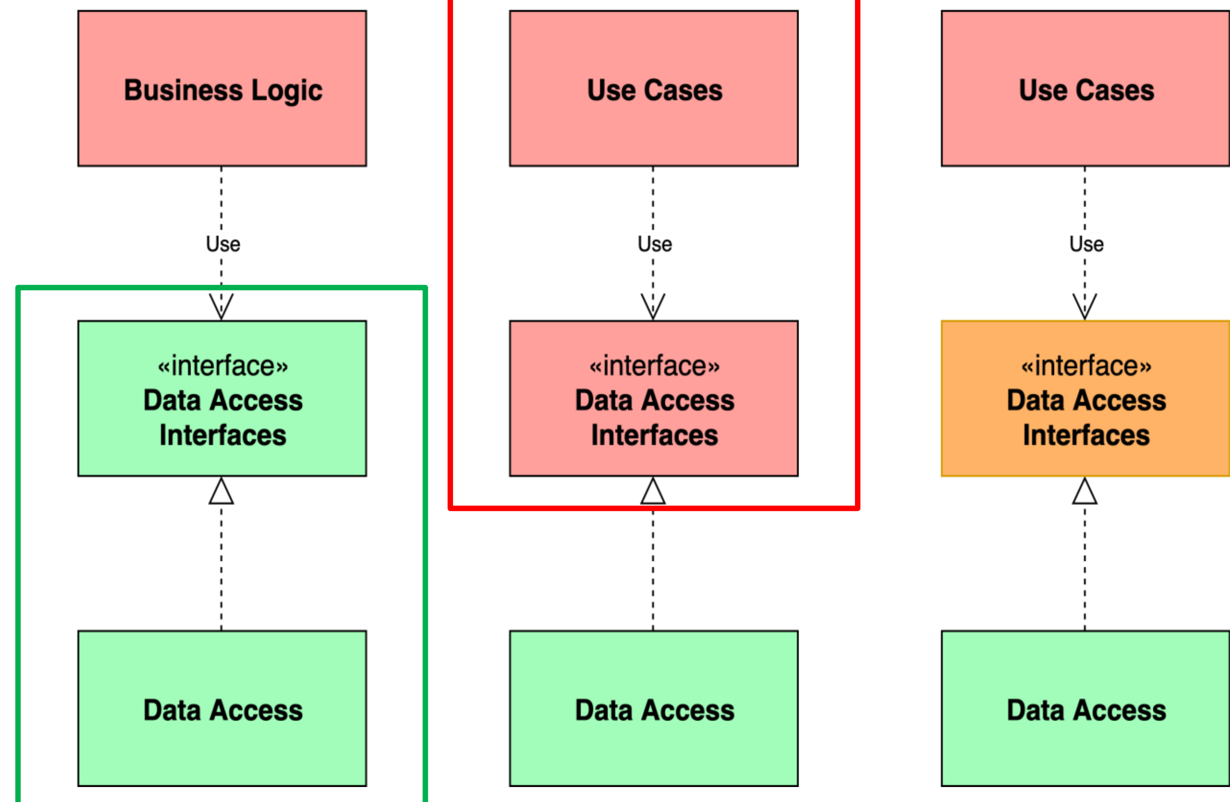
Интерфейс для DB контекста

```
public interface IDbContext
{
    DbSet<Order> Orders { get; }
    …

    Task<int> SaveChangesAsync();
}
```

Интерфейс для DB контекста

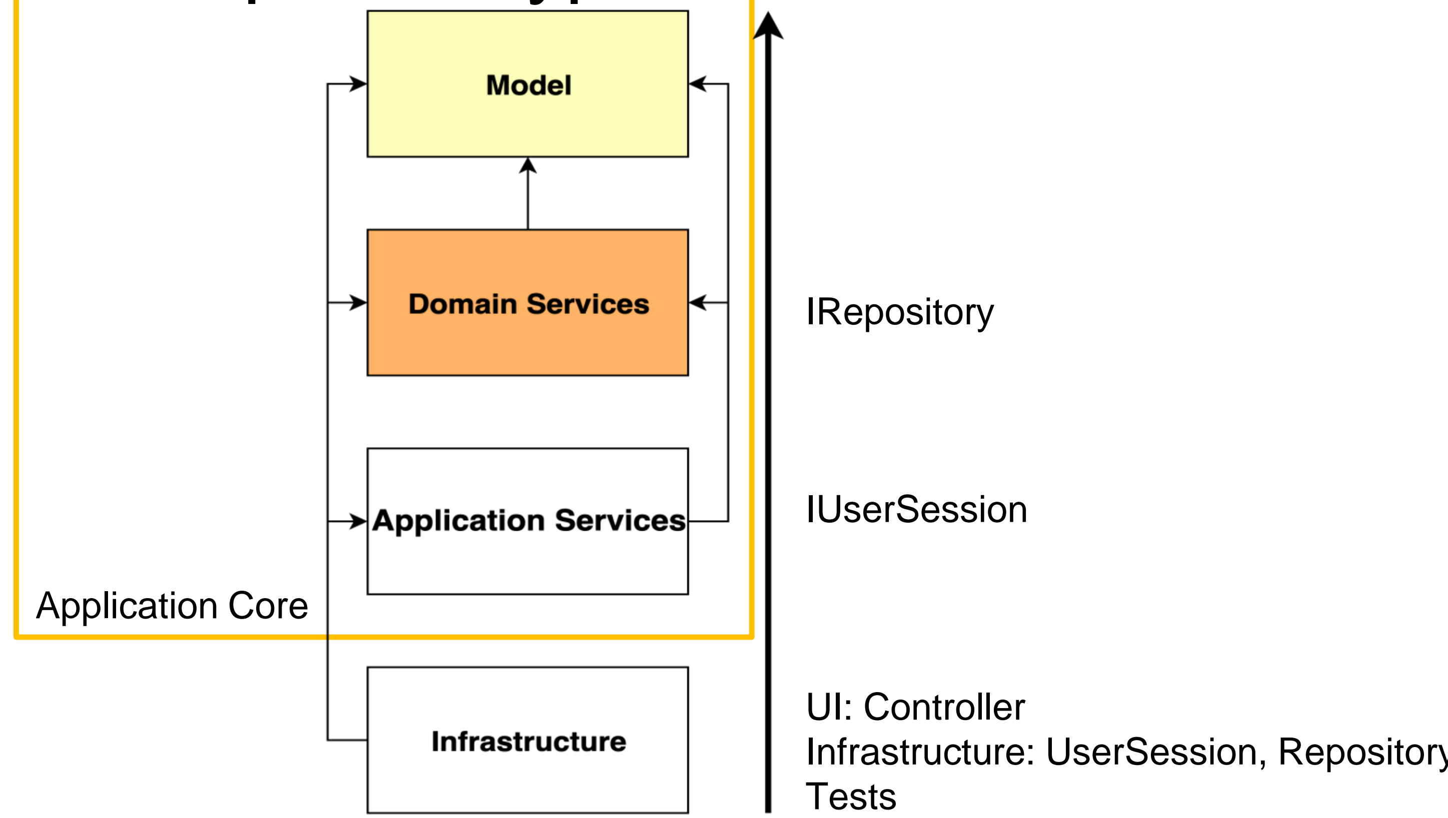
```
public class SqliteDbContext : DbContext, IDbContext
{
}

internal class OrdersService: IOrdersService
{
    private readonly IDbContext _context;
    public OrdersService(IDbContext context)
    {
        _context = context;
    }
}
```

Куда его положить интерфейс?

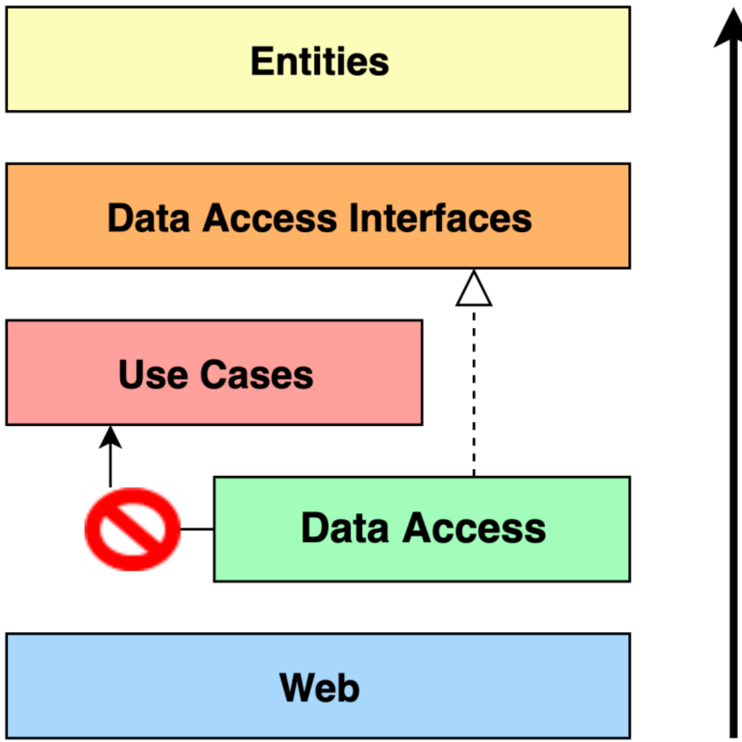


Луковая
Чистая

Луковая Архитектура



Чистая Архитектура



Проекты:

Solution 'Dotnext.CleanArchitecture' (6 of 6 projects)

- 0-Dotnext.Shared
- 1-Dotnext.Entities
- 2-Dotnext.DataAccess.Interfaces
- 3-Dotnext.UseCases
- 4-Dotnext.DataAccess.Sqlite
- **5-Dotnext.Web**

Доступ к данным - резюме

- **Интерфейсы**:
  - Контракт – интерфейс IDbContext
  - Зависимости – Entities

- **Адаптеры**:
  - Контракт: модуль для DI контейнера
  - Зависимости:
    - Entities
    - Реализуемый интерфейс
    - DI контейнер

Почему не работает правило зависимостей?

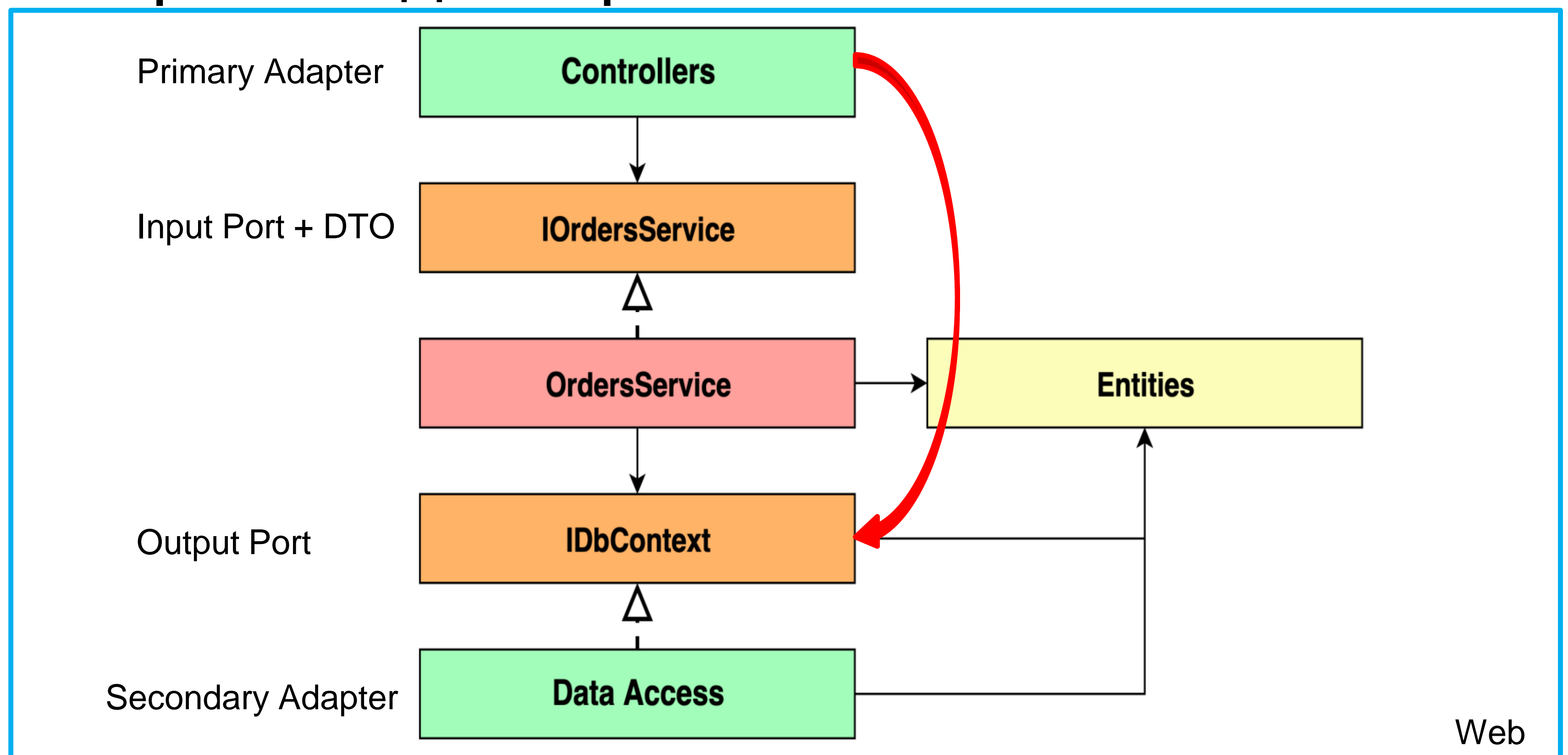
Вызов сервиса из контроллера

```
private readonly IOrdersService _ordersService;

[HttpGet("{id}")]
public async Task<OrderDto> GetById(int id)
{
    return await _ordersService.GetById(id);
}
```

Вызов DB контекста из контроллера

```
private readonly IDbContext _context;

[HttpGet("{id}")]
public async Task<OrderDto> GetById(int id)
{
    var order = await _context.Orders.FindAsync(id);
    return new OrderDto(order);
}
```

Порты и адаптеры



Контроллеры – в отдельный проект

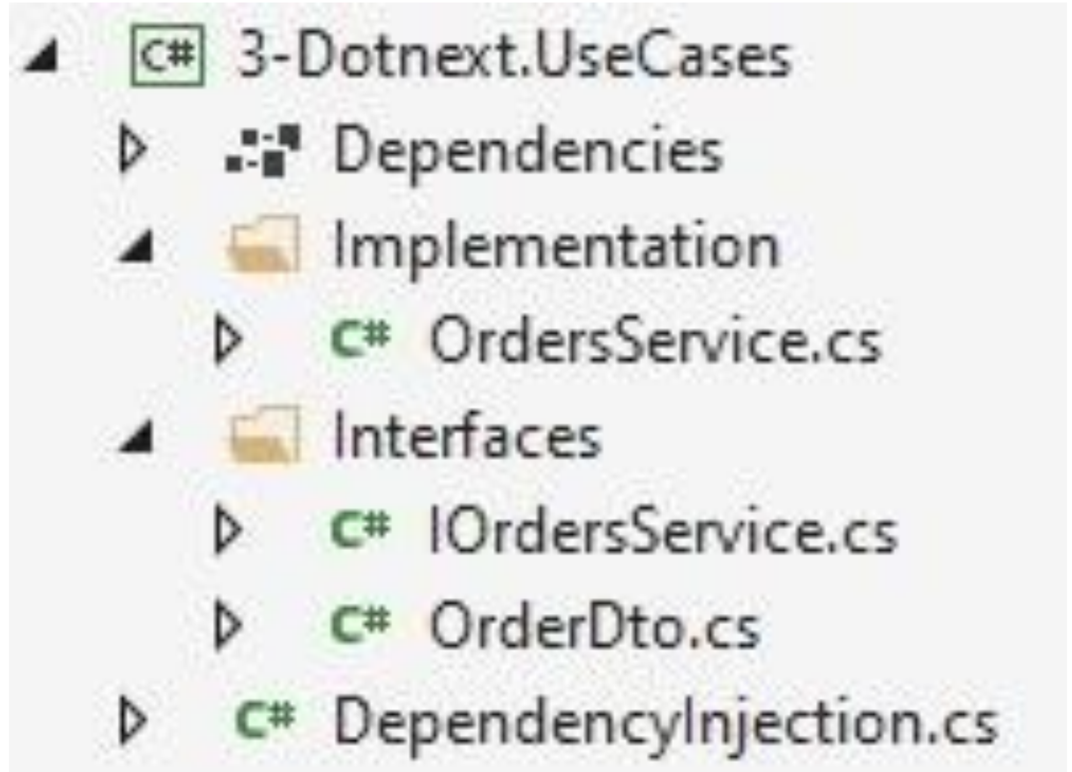
- 4-Dontext.Controllers
  - Dependencies
  - OrdersController.cs
- **5-Dotnext.Web**
  - Connected Services
  - Dependencies
  - Properties
  - appsettings.json
  - Program.cs
  - Startup.cs

Контроллеры - резюме

- Контракт:
    - Все классы публичные
    - Это не важно т.к. на него ссылается только Web

- Связи:
    - Ссылается на контракт Use Cases
        - Input Port – IOrdersService + OrderDto
    - Не зависит от Entities

Use Cases



Use Cases

- Input Port
    - Data Transfer Object
    - IOrdersService

- Use Cases
    - OrdersService

- Mappings (DTO -> Entities)

- DI module

Интерактор это:

- Сервис
    - метод сервиса

- CQRS handler

Интерактор

```
public async Task<OrderDto> CreateOrder(CreateOrderDto dto)
{
    var order = CreateOrderFromDto(dto);
    order.CalculateTotal();
    _context.Orders.Add(order);
    await _context.SaveChangesAsync();

    var externalId = _externalOrdersSerice.Create(order);
    order.ExternalId = externalId;
    await _context.SaveChangesAsync();

    return new OrderDto(order);
}
```

Интерактор

- Взаимодействие:
  - Логики
  - Базы
  - Сторонних сервисов

- Конвертация данных:
  - DTO -> Entity -> Email

Use Cases - резюме

- Контракт:
  - Input Port = интерфейсы сервисов + DTO
  - DI модуль

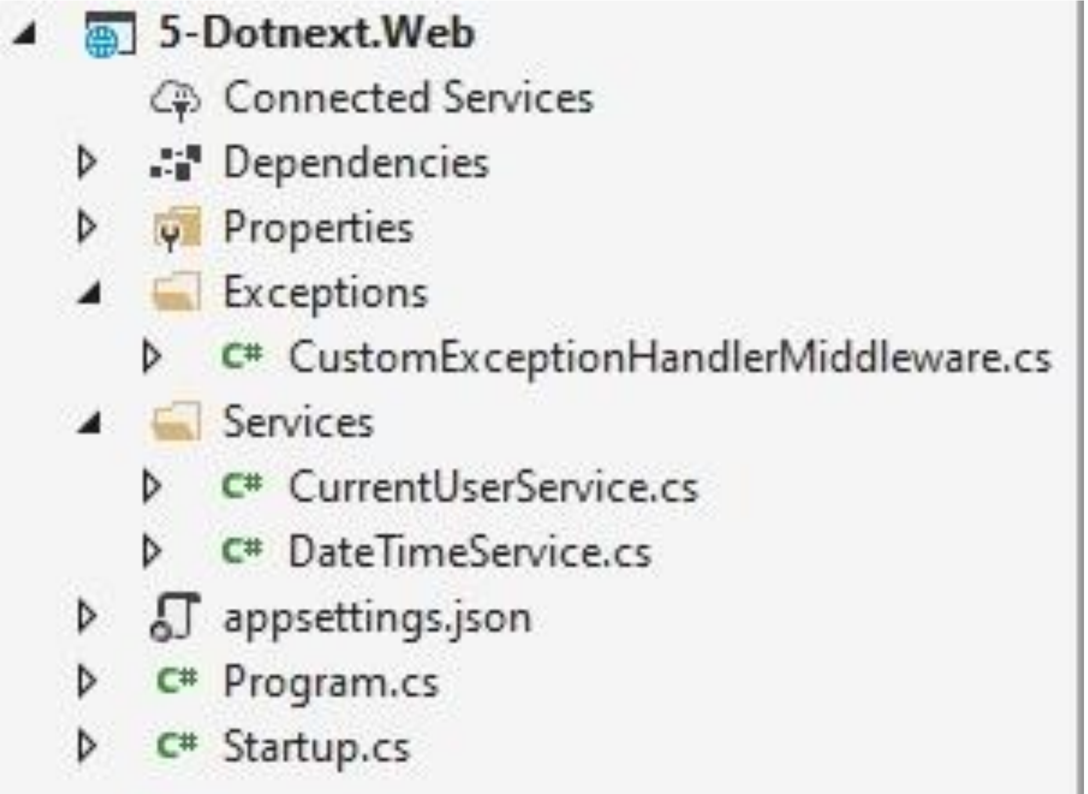
- Связи:
  - Entities
  - Интерфейсы вторичных адаптеров
  - Инфраструктура: Automapper и др.

Use Cases - резюме

- Контракт и реализация вместе?

- Если да:
  - Нужно следить за контрактом
    public -> internal
  - Транзитивность зависимостей:
    Use Cases -> Entities
    Controllers -> UseCases -> Entities

Web – корень композиции



Web – корень композиции

```
builder.RegisterModule(
    new SqlServerDataAccessModule(connectionString));
builder.RegisterModule(
    new UseCasesModule());
builder.RegisterModule(
    useSendGrid ? new SendGridEmailModule() : new EmailModule());
```

Web – корень композиции

- Корень композиции
- Инфраструктура хоста
    - Текущий пользователь
- Обработка исключений
- Swagger

Web - резюме

- Контракт:
  - Не важно

- Связи:
  - Знает о всех остальных проектах

Базовая часть. Проекты:

- Shared
- Entities
- Data Access Interfaces
- Use Cases Interfaces
- Use Cases Implementation
- Data Access Sqlite
- Controllers
- Web

Entities – тип модели

- Rich
  - Логика внутри доменных объектов

- Anemic
  - Логика внутри доменных сервисов

Защита Rich Model

```
public Order Handle(CreateOrderCommand command)
{
    var order = CreateOrderDromCommand(command);

    var totalPrice = order.CalculateTotalPrice();
    order.CalculateTax(totalPrice);

    return order;
}
```

Защита Rich Model

```
public decimal CalculateTax(decimal total)
{
    var tax = CalculateDefaultTax();
    if (total + tax > LOW_TAXABLE_BOUND)
    {
        var additionalTax = _externalService.GetAdditionalTax(total);
        tax += additionalTax;
    }
    return tax;
}
```

Защита Rich Model

```
public delegate decimal AdditionalTaxCalculator(decimal price);

public decimal CalculateTax(decimal totalPrice, AdditionalTaxCalculator calculator)
{
    var tax = CalculateDefaultTax();
    if (totalPrice + tax > LOW_TAXABLE_BOUND)
    {
        var additionalTax = calculator.Invoke(totalPrice);
        tax += additionalTax;
    }
    return tax;
}
```

Защита Rich Model

```
public Order Handle(CreateOrderCommand command)
{
    var order = CreateOrderDromCommand(command);

    var totalPrice = order.CalculateTotalPrice();
    order.CalculateTax(totalPrice, _externalService.GetAdditionalTax);

    return order;
}
```

Защита Rich Model – резюме

- Вместо одного универсального интерфейса – много специализированных делегатов

- Вместо IRepository
    - SearchAdditionalTaxes delegate
    - SearchPopularProducts delegate

- Делегаты создаются:
    - Только те что нужно
    - Только тогда, когда нужно

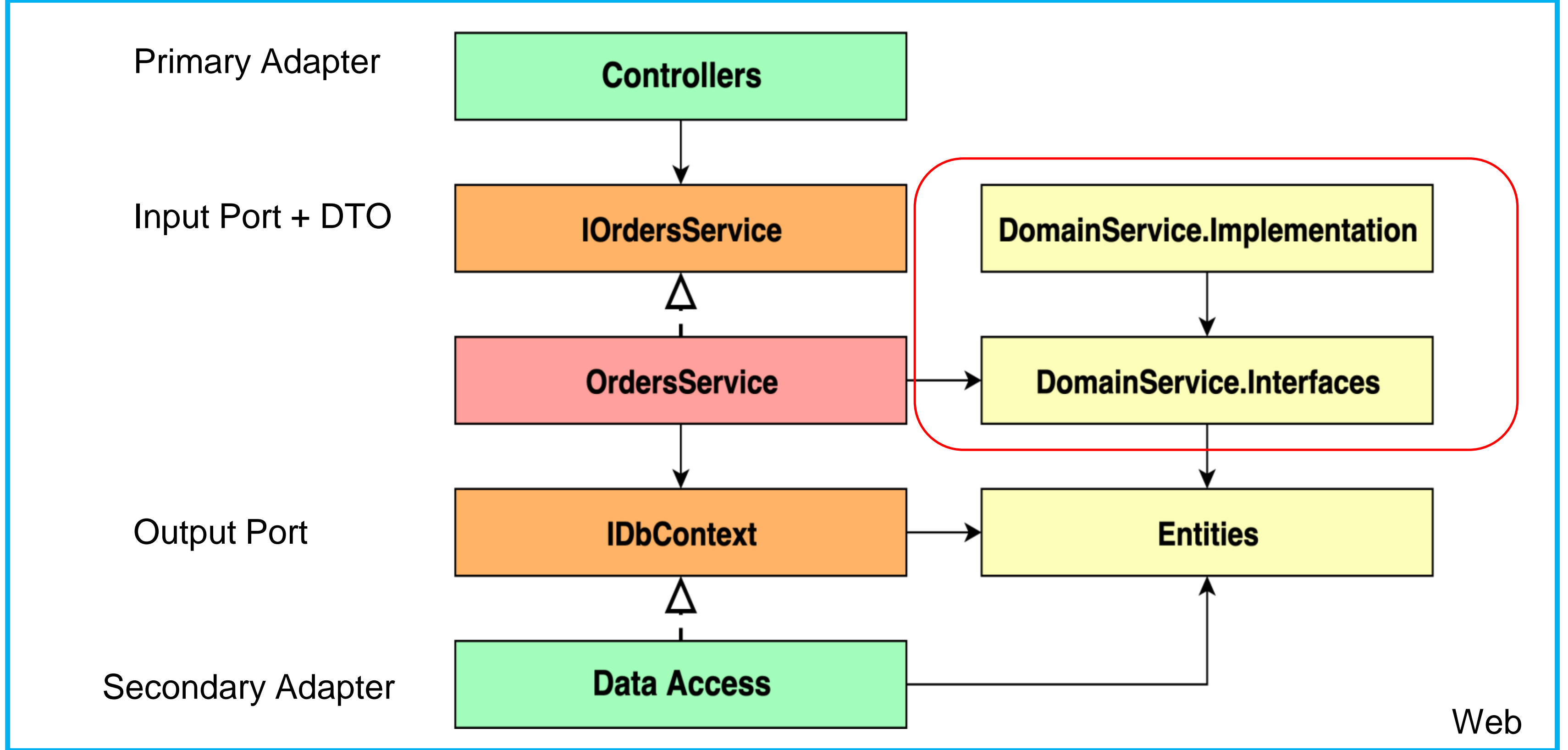
Доменные сервисы

- 1.0-Dotnext.Entities
- 1.1-Dotnext.DomainServices.Interfaces
  - Dependencies
  - IOrdersCalculator.cs
- 1.2-Dotnext.DomainServices.Implementation
  - Dependencies
  - DependencyInjection.cs
  - OrdersCalculator.cs

Доменные сервисы

```
private readonly IOrdersCalculator _ordersCalculator;

public async Task<OrderDto> CreateOrder(CreateOrderDto dto)
{
    var order = CreateOrderFromDto(dto);

    // order.CalculateTotal();
    var total = _ordersCalculator.CalculateToal(order);
```

Доменные сервисы



Доменные сервисы - резюме

- Анемичная модель закрывает доступ к операциям из вторичных адаптеров?

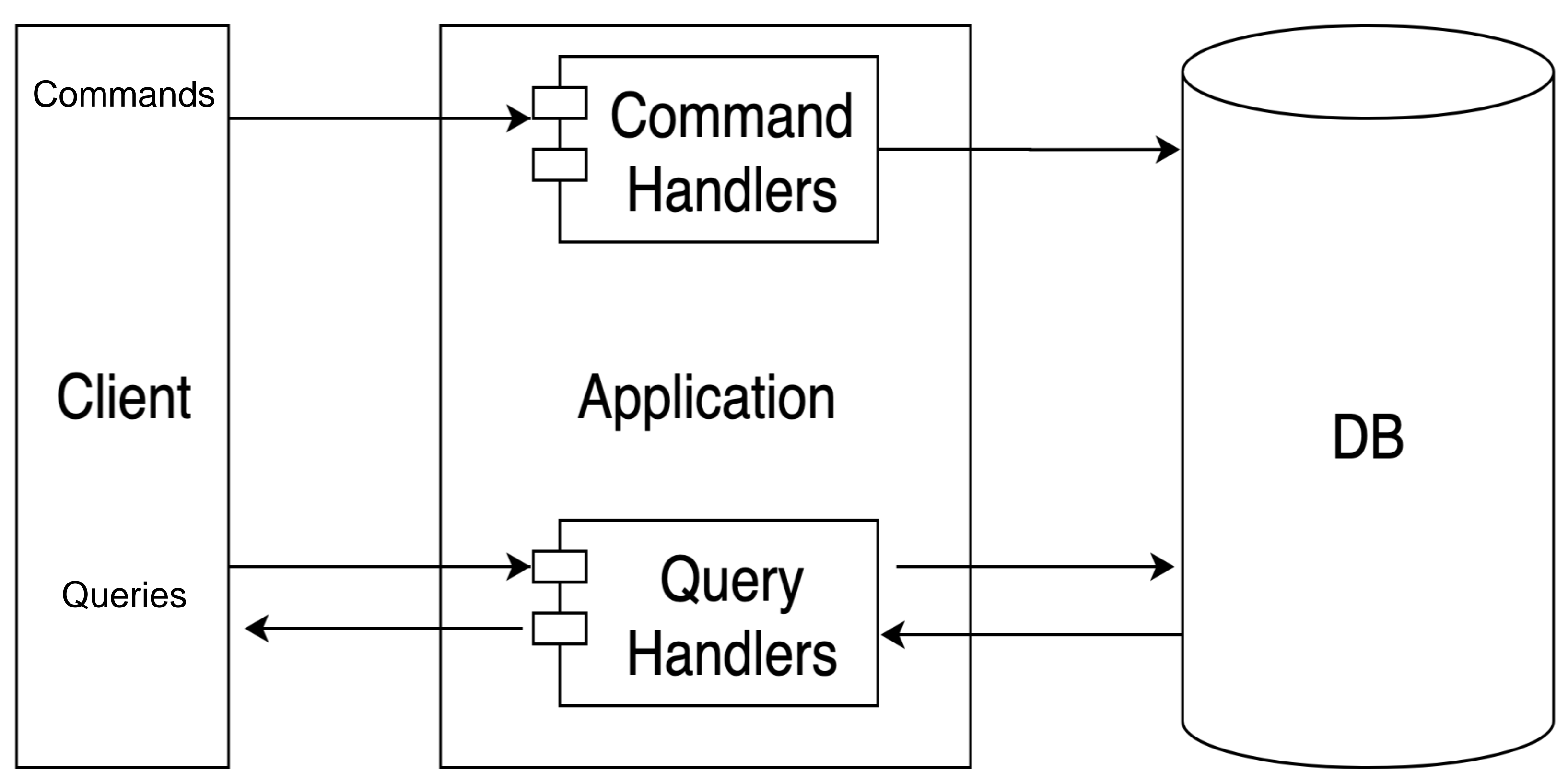
- Нет, с точки зрения архитектуры – тип модели не важен!

Доменные сервисы - резюме

- **Интерфейсы**:
  - Контракт – все классы
  - Зависимости – Entities

- **Реализация**:
  - Контракт: модуль для DI контейнера
  - Зависимости:
    - Entities
    - Реализуемый интерфейс
    - DI контейнер

CQRS lite



CQRS lite

```
◢ C# 3-Dotnext.UseCases
  ▷ Dependencies
  ◢ Orders
    ◢ Commands
      ◢ CreateOrder
        ▷ C# CreateOrderCommand.cs
        ▷ C# CreateOrderCommandHandler.cs
        ▷ C# CreateOrderDto.cs
    ◢ Mappings
      ▷ C# OrdersMappingProfile.cs
    ◢ Queries
      ◢ GetOrderByIdQuery
        ▷ C# GetOrderByIdQuery.cs
        ▷ C# GetOrderByIdQueryHandler.cs
        ▷ C# OrderDto.cs
  ▷ Products
  ▷ C# DependencyInjection.cs
```

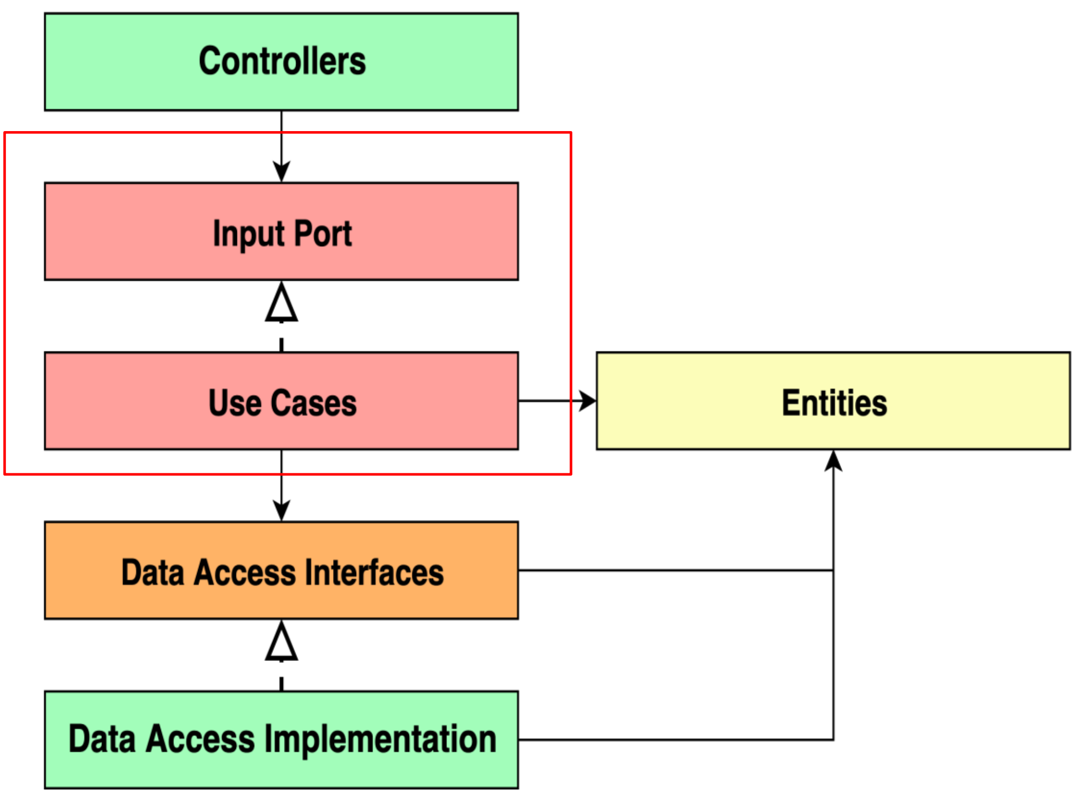
CQRS lite

- OrdersService
  - GetById
  - Create

- GetByIdQuery
  - Handler
  - OrderDto

- CreateOrderCommand
  - CreateOrderDto
  - Handler

CQRS lite



CQRS lite - резюме

- Вместо одного универсального сервиса – много специализированных handlers
- Контракт
  - Commands, Queries, DTOs
- Связи:
  - Entities, Data Access Interfaces
- Контракт и реализация вместе
- MediatR

Use Cases – где искать общую логику?

- Сервисы:
  - приватные методы

- CQRS
  - репозитории поверх EF!

CQRS – что с общей логикой?

- Спецификация, Extensions
- Pipeline
- Наследование handlers
  - Множественное наследование?
- Вызов Handler1 из Handler2
  - Обычно не подходит, т.к. используется pipeline
- Вынесем общую логику в новый класс!

CQRS – что с общей логикой?

- 3.0-Dotnext.UseCases
- 3.1-Dotnext.ApplicationServices.Interfaces
  - Dependencies
  - AccessLevel.cs
  - IPermissionsManager.cs
  - ObjectAccessLevel.cs
  - ObjectType.cs
- 3.2-Dotnext.ApplicationServices.Implementation
  - Dependencies
  - DependencyInjection.cs
  - PermissionsManager.cs

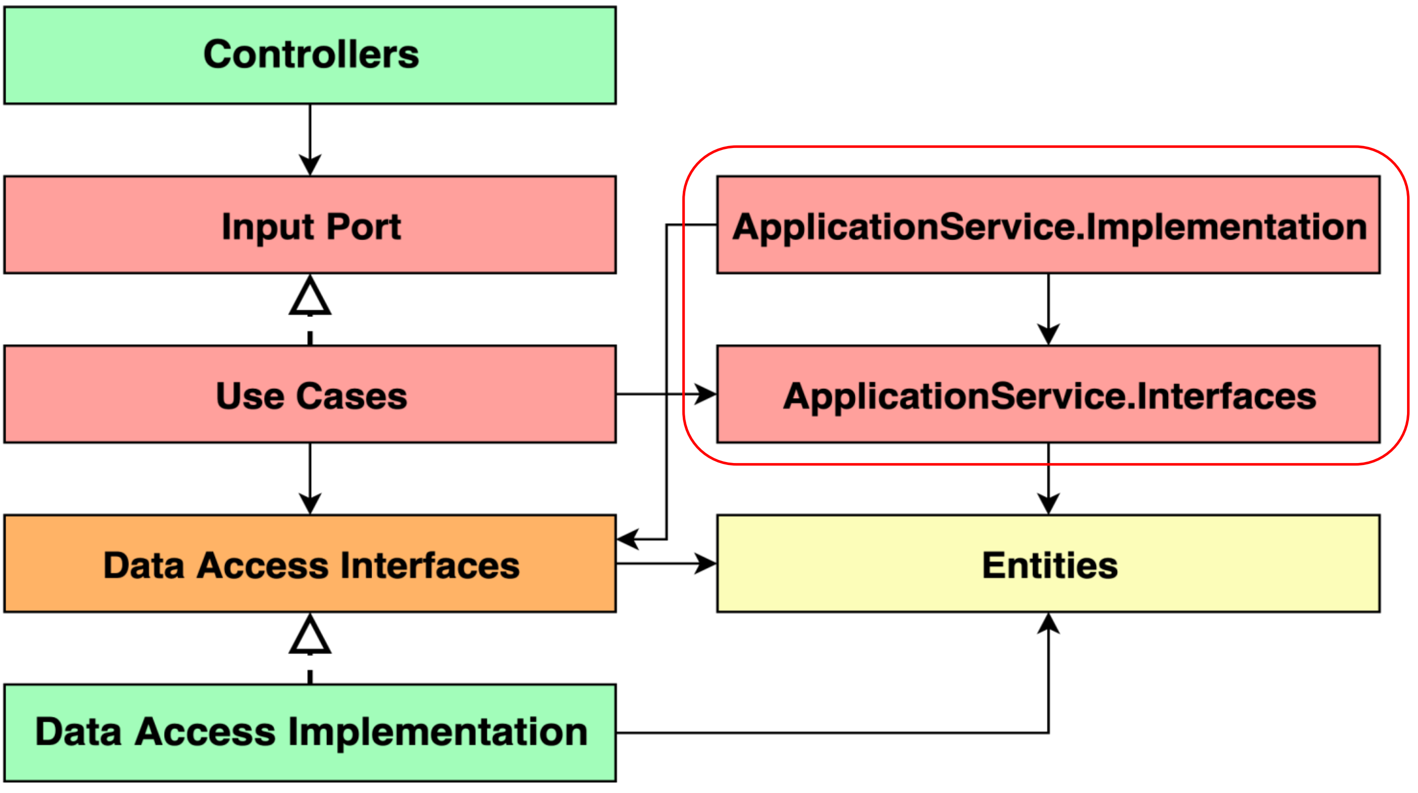
CQRS – что с общей логикой?

- AccessLevel
  - View
  - Register, Sign, Archive, …

- ObjectAccessLevel
  - UserId
  - ObjectId
  - AccessLevel

- PermissionsManager
  - CheckAccess

Application сервисы - резюме



Application сервисы - резюме

- **Интерфейсы**:
    - Контракт – все классы
    - Зависимости – Entities, Data Access Interfaces

- **Реализация**:
    - Контракт: модуль для DI контейнера
    - Зависимости:
        - Entities, Data Access Interfaces
        - Реализуемый интерфейс
        - DI контейнер
- Use Cases зависит от Application Service Interfaces

Расширенная часть:

- IRepository -> delegates
- Domain Services
- CQRS
- Application Services

Общие принципы

- Инверсия зависимостей:
  - отделить интерфейс от реализации
  - положить их в разные проекты
  - сделать неправильное использование невозможными

- Много маленьких специализированных решений лучше одного универсального:
  - Application: Services -> CQRS
  - Entities: IRepository -> Delegates

Компромиссы

- Все компоненты с реализацией зависят от DI контейнера
- Компонент Use Cases содержит и контракт и реализацию
- Архитектура должна соответствовать текущему уровню развития проекта
- Чтобы у вашего нового проекта был шанс стать кровавым – иногда нужны быстрые результаты на старте

Что получили

- Создание базовой структуры для нового проекта
    - 4 часа

- Переключение разработчика между проектами
    - Структура проекта - мгновенно

- Понятно что и где искать
- И куда новый код положить

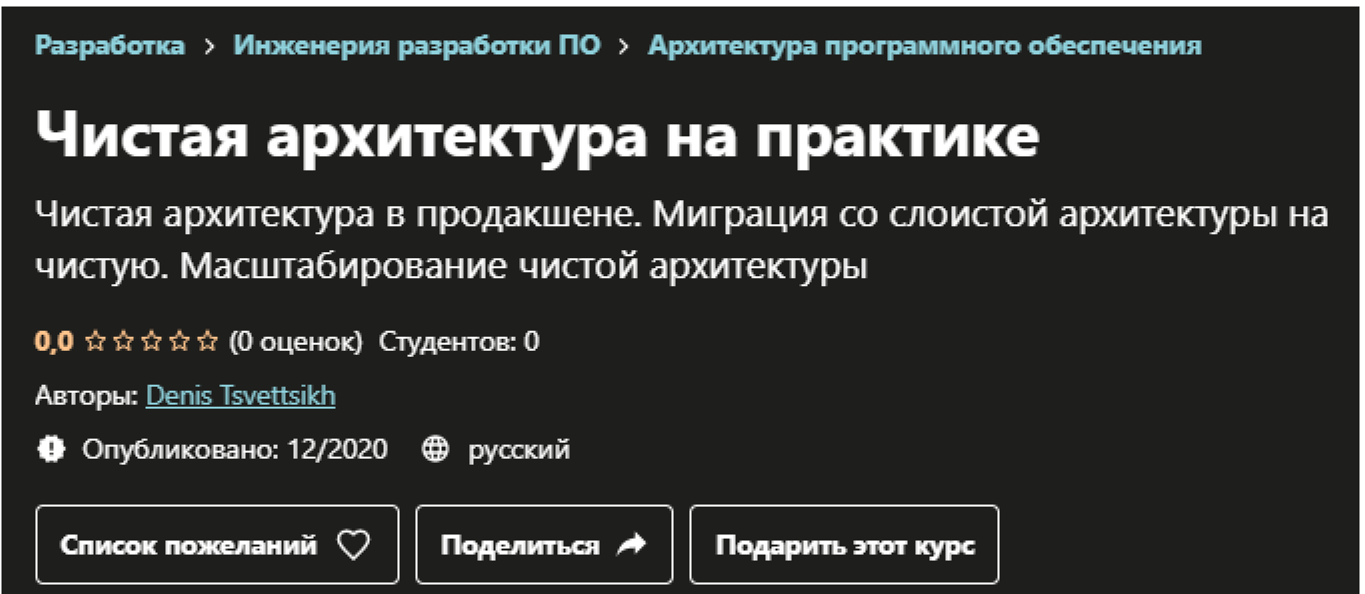
Это не бесплатно

- Тактические споры еще продолжаются
- Нужно обучать сотрудников
- Шаблон нового проекта с модулями бесполезен
    - Сразу после создания он будет правиться
    - Трудозатраты на создание выше экономии

Литература

- Роберт Мартин – Чистая Архитектура
- Jeffrey Palermo - The Onion Architecture
- Alistair Cockburn – The Hexagonal Architecture
- Jason Taylor - Clean Architecture
- Ben Morris – NetArchTest

Что посмотреть:



https://www.udemy.com/course/clean-architecture-csharp-ru/

Вопросы?

@AndrewTsw

andrew.tsw@google.com

https://github.com/andrewtsw

Андрей Цветцих
Lead Developer
EPAM

@den_tsvettsikh

denis.tsvettsih@yandex.ru

https://github.com/denis-tsv

Денис Цветцих
Lead Developer
EPAM