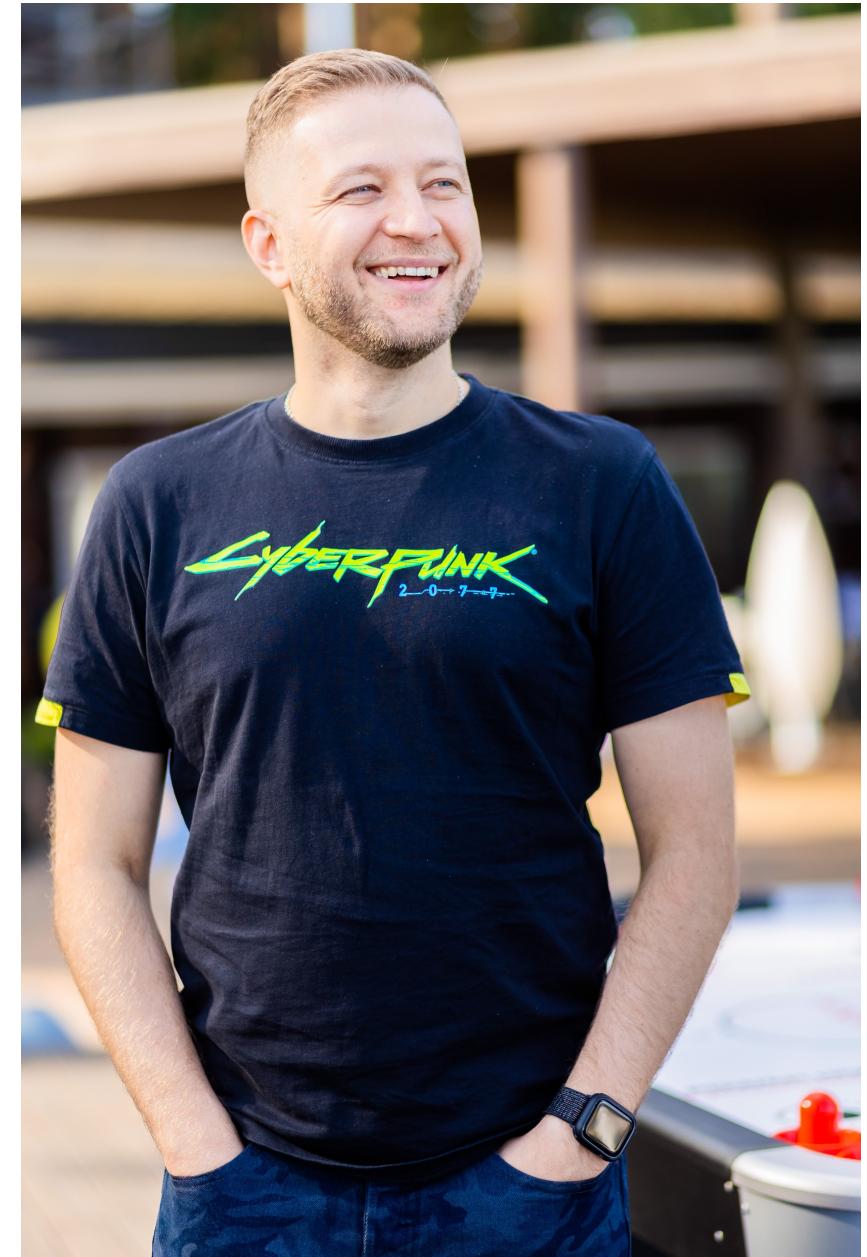


Clean Architecture

Обо мне

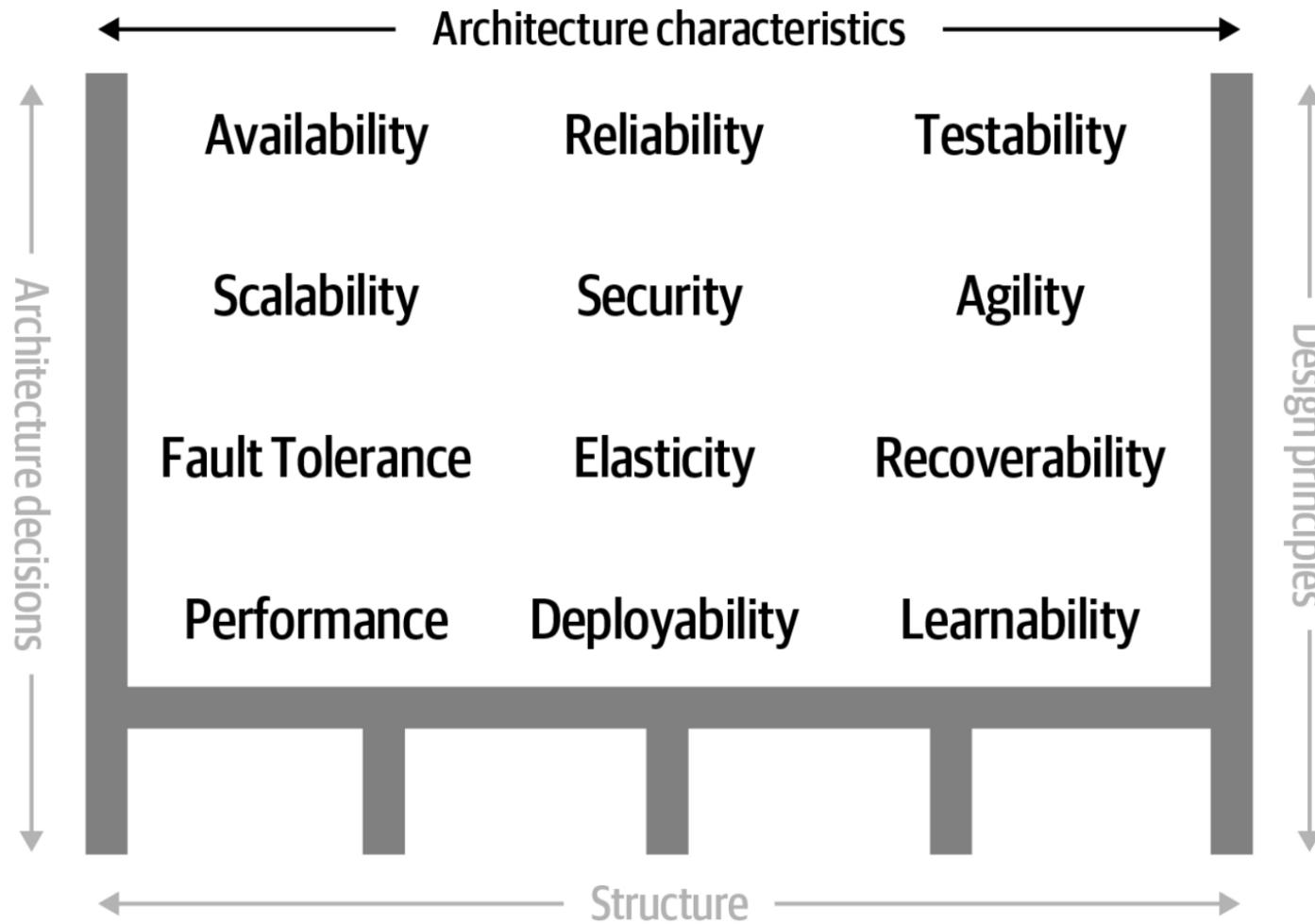
- Игорь Лопушко
- .NET, Golang
- Контакты:
 - <https://www.linkedin.com/in/igor-lopushko/>
 - <https://github.com/igorlopushko/>



Что такое архитектура?

- Выбор структурных элементов и их интерфейсов
- Соединение выбранных элементов структуры и поведения в системе
- Архитектурный стиль

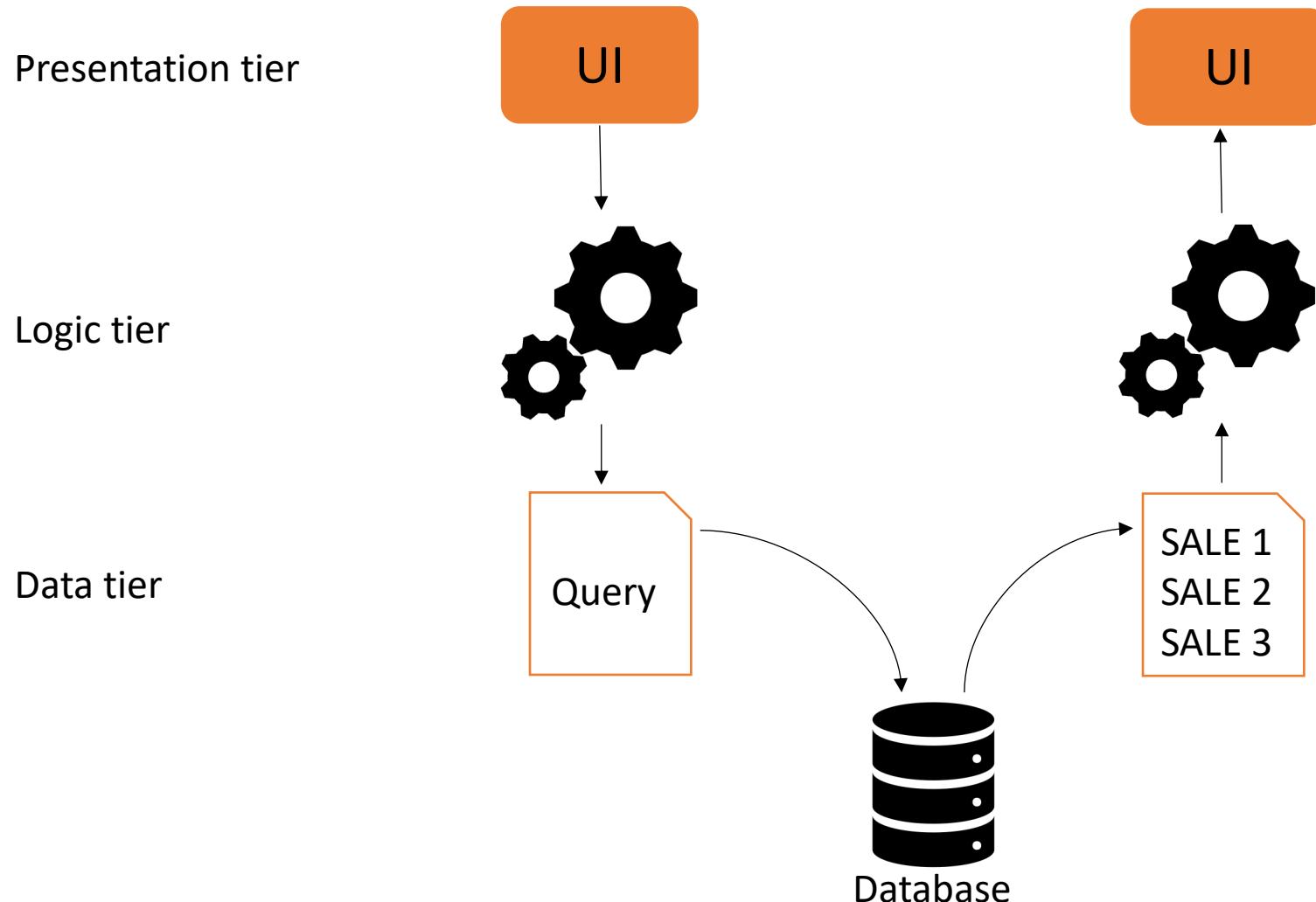
Что такое хорошая архитектура?

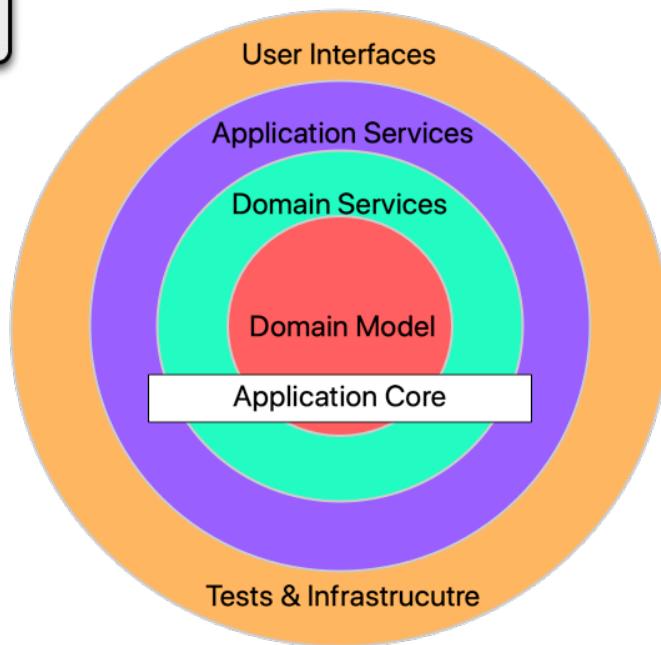
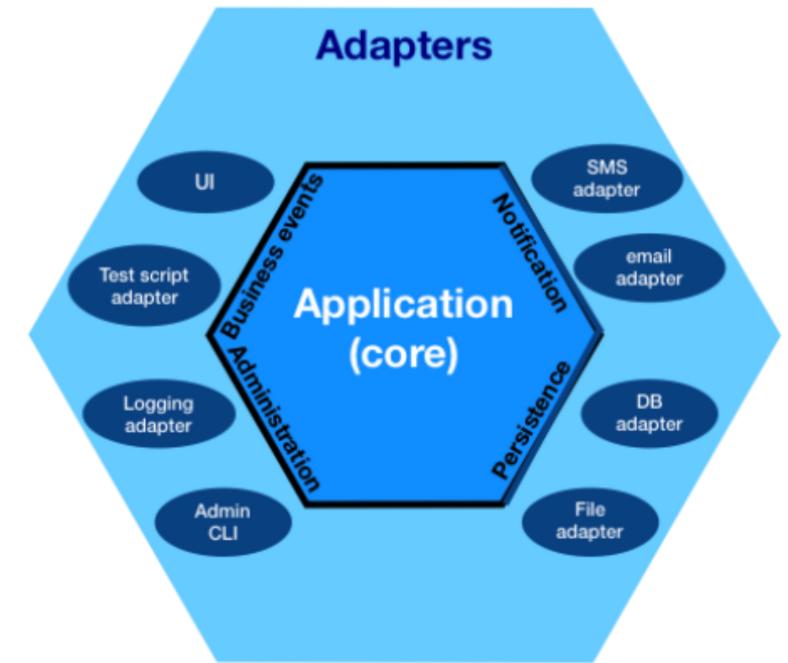
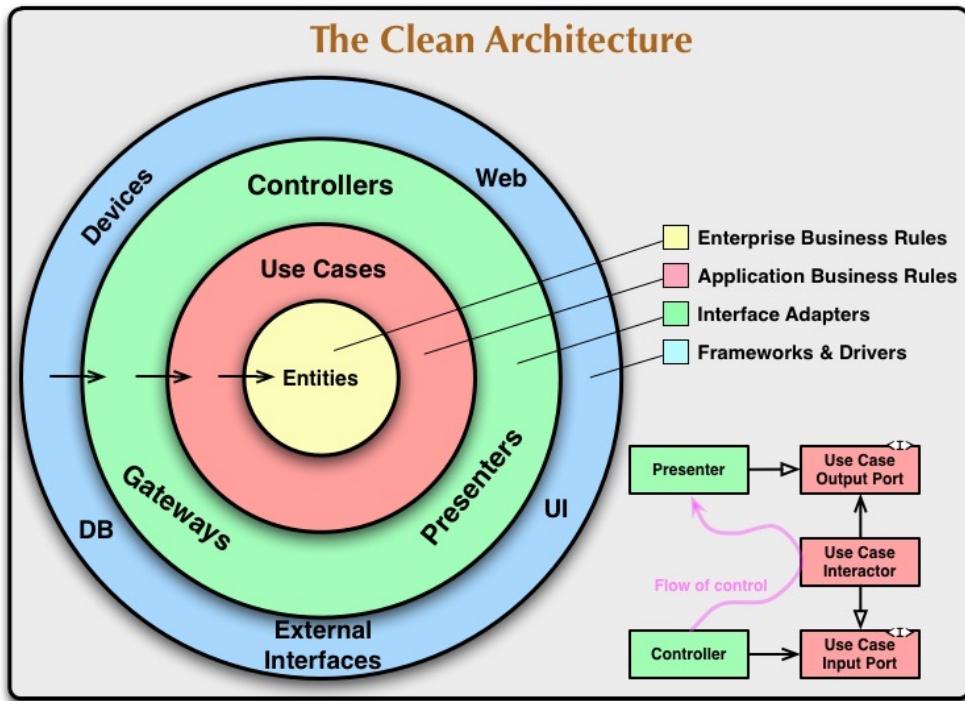


Эволюция архитектуры

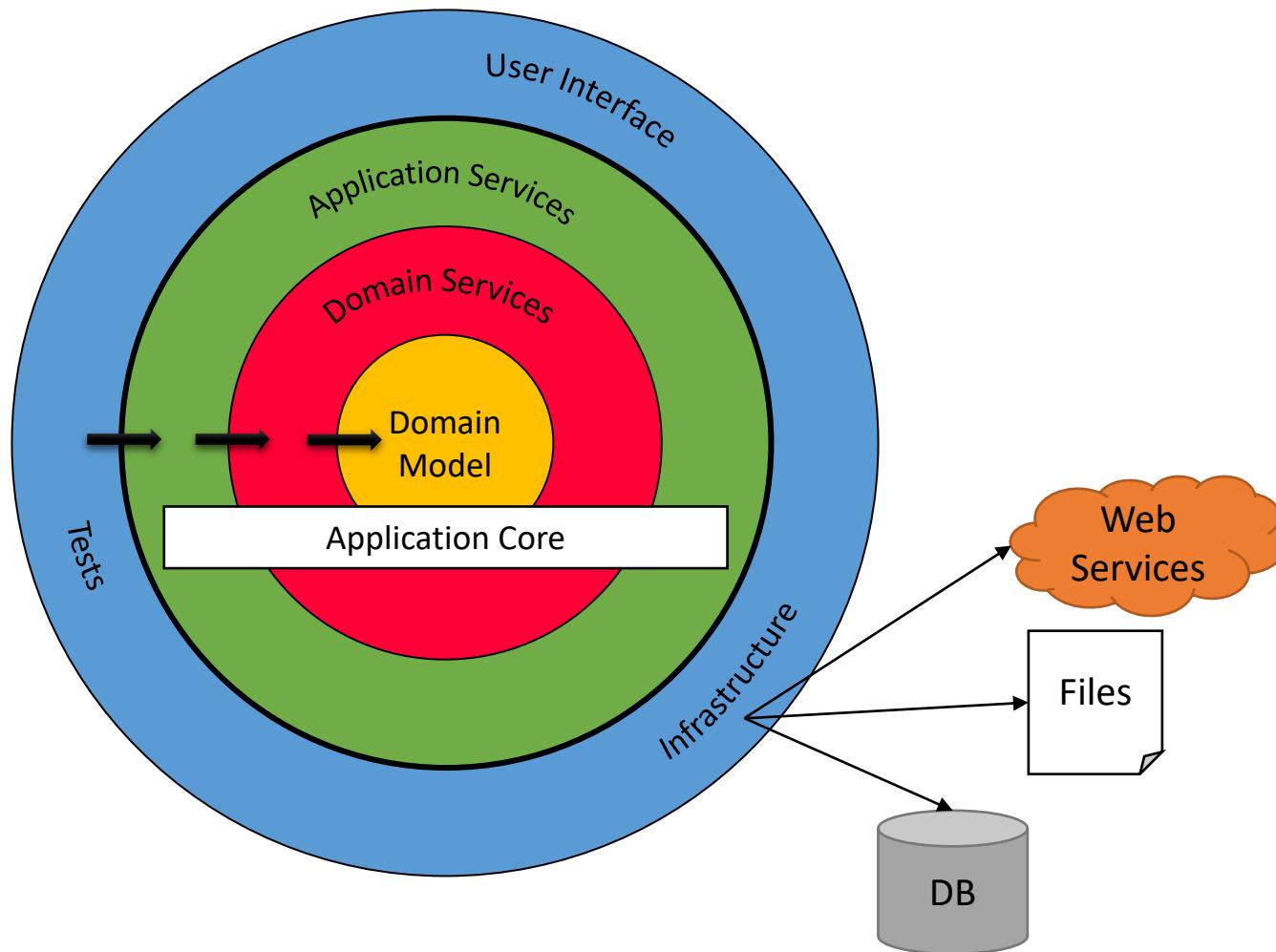


Многослойная архитектура

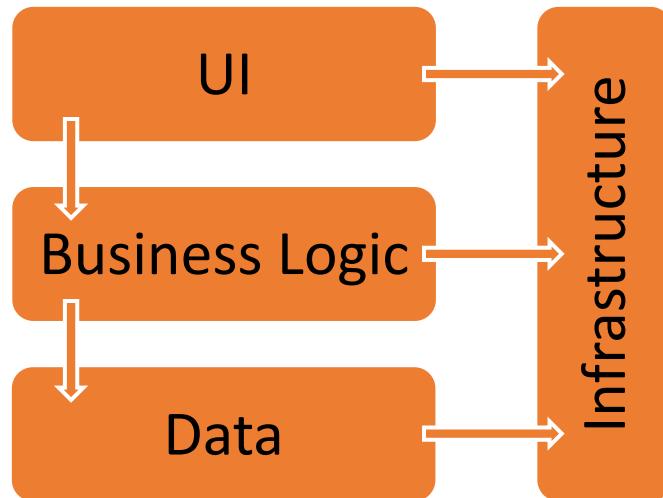




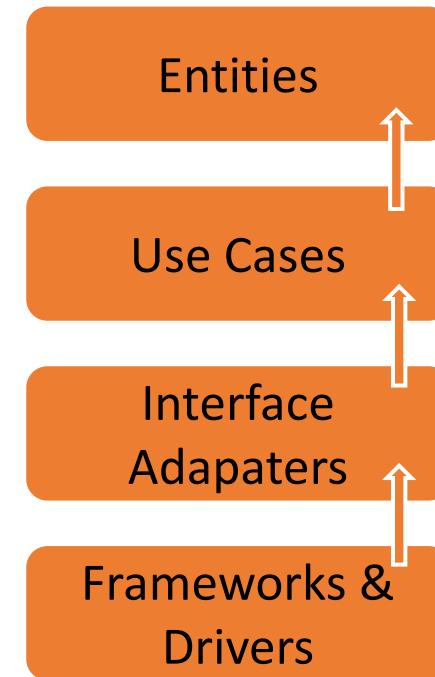
Давайте разбираться



Многослойная или Чистая?



N-tier Architecture



Clean Architecture

Преимущества Чистой Архитектуры

- Независимость от фреймворков
- Большая тестируемость
- Независимость от UI
- Независимость от баз данных
- Независимость от внешних сервисов

Инверсия зависимостей

```
class Book
{
    public string Text { get; set; }
    public ConsolePrinter Printer { get; set; }

    public void Print()
    {
        Printer.Print(Text);
    }
}

class ConsolePrinter
{
    public void Print(string text)
    {
        Console.WriteLine(text);
    }
}
```

Инверсия зависимостей

```
interface IPrinter
{
    void Print(string text);
}

class Book
{
    public string Text { get; set; }
    public IPrinter Printer { get; set; }

    public Book(IPrinter printer)
    {
        this.Printer = printer;
    }

    public void Print()
    {
        Printer.Print(Text);
    }
}
```

```
class ConsolePrinter : IPrinter
{
    public void Print(string text)
    {
        Console.WriteLine("Печать на консоли");
    }
}
```

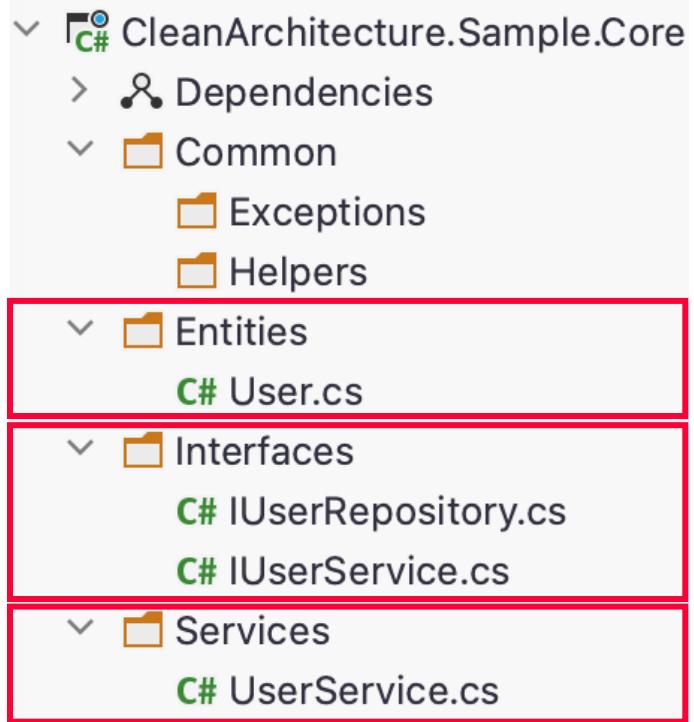
```
class HtmlPrinter : IPrinter
{
    public void Print(string text)
    {
        Console.WriteLine("Печать в html");
    }
}
```



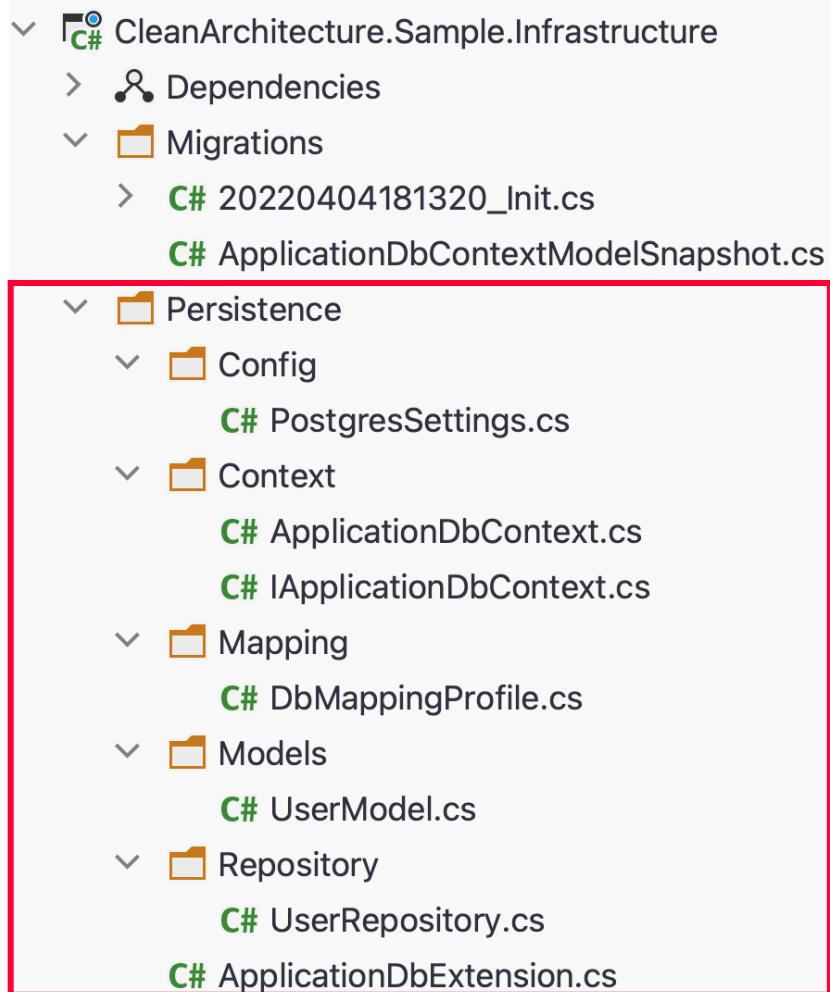
“Talk is
cheap. Show
me the code.”

Linus Torvalds

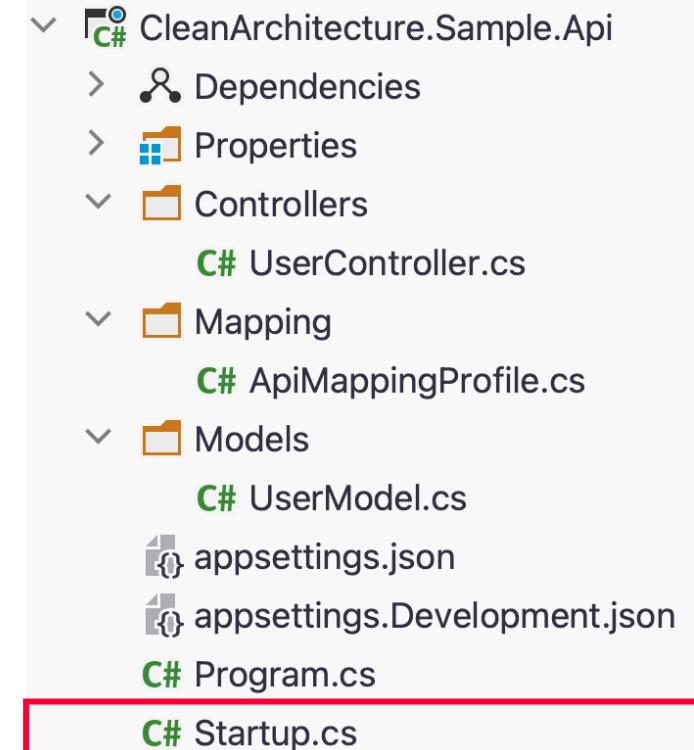
Приимер проекта



Application Core

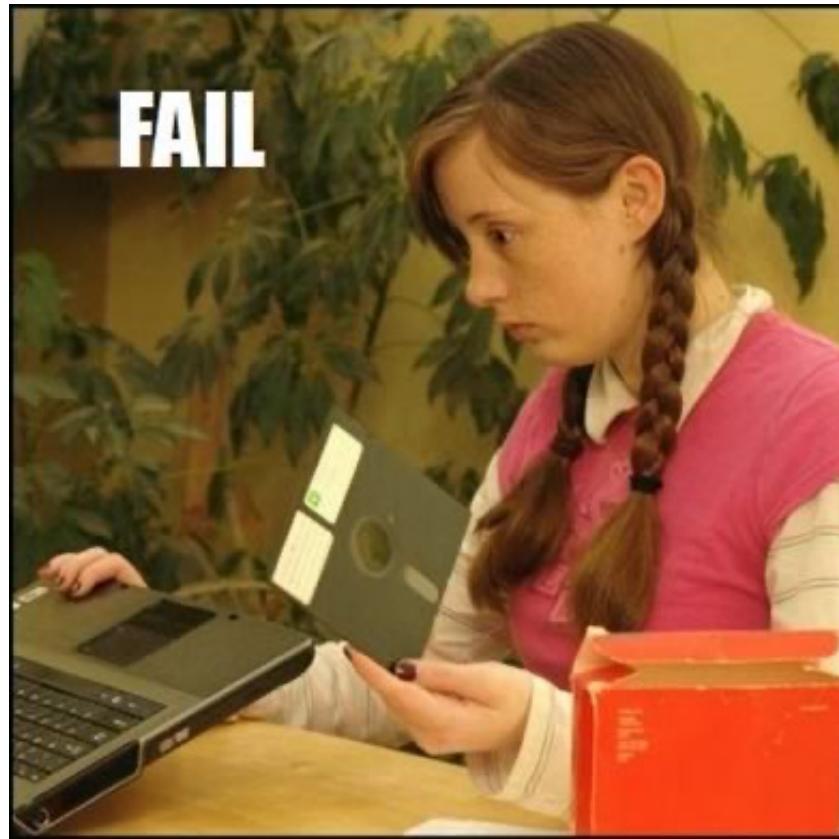


Infrastructure

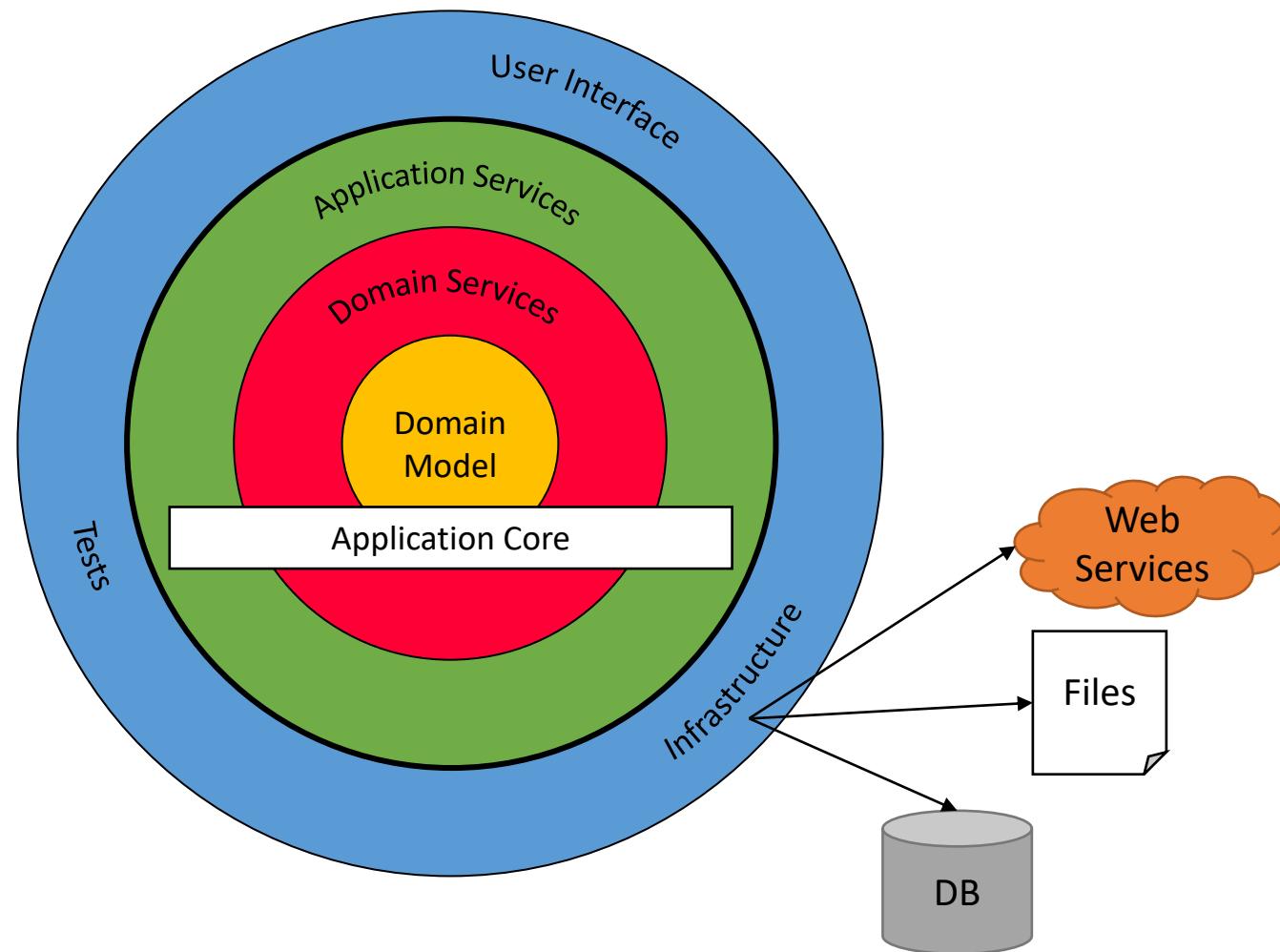


Web API

Сложности на практике



Сколько слоев мне нужно?



Общие принципы и компромиссы

- Инверсия зависимостей
 - Отделить интерфейс от реализации
 - Положить их в разные проекты
 - Сделать неправильное использование невозможным
- Много меленьких решений лучше одного универсального
- Слой Application Core содержит и контракты и реализацию
- Архитектура должна соответствовать текущему уровню развития проекта

Отличие DTO и Entity

- **Data Transfer Object**
 - необходимые для работы и перехода между слоями
- **Entity**
 - Часть бизнес-домена
 - Реализовывает поведение и применяться к различным вариантам использования в домене

Нужно ли поддерживать immutability?

```
using Newtonsoft.Json;

namespace CleanArchitecture.Sample.Api.Models
{
    public class UserModel
    {
        [JsonProperty("id")]
        public int Id { get; set; }

        [JsonProperty("email")]
        public string Email { get; set; }

        [JsonProperty("first_name")]
        public string FirstName { get; set; }

        [JsonProperty("last_name")]
        public string LastName { get; set; }
    }
}
```

```
using Newtonsoft.Json;

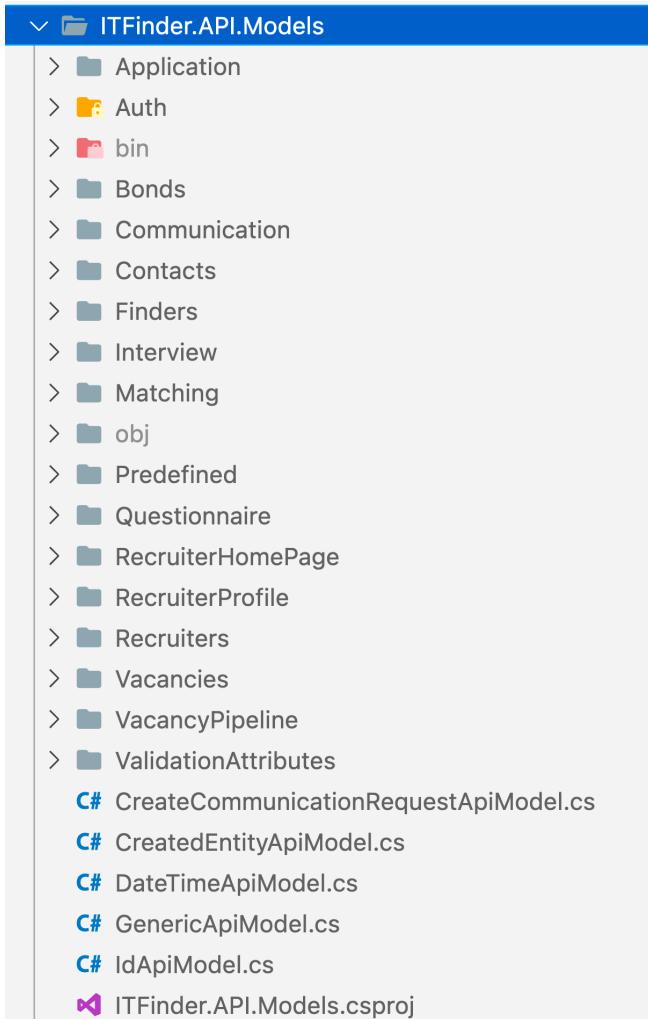
namespace CleanArchitecture.Sample.Api.Models
{
    public class UserModel
    {
        [JsonProperty("id")]
        public int Id { get; init; }

        [JsonProperty("email")]
        public string Email { get; init; }

        [JsonProperty("first_name")]
        public string FirstName { get; init; }

        [JsonProperty("last_name")]
        public string LastName { get; init; }
    }
}
```

Переиспользование моделей



- Большое количество похожих моделей «на все случаи жизни»
- Тяжело декомпозировать

Нужен ли mapping между слоями?

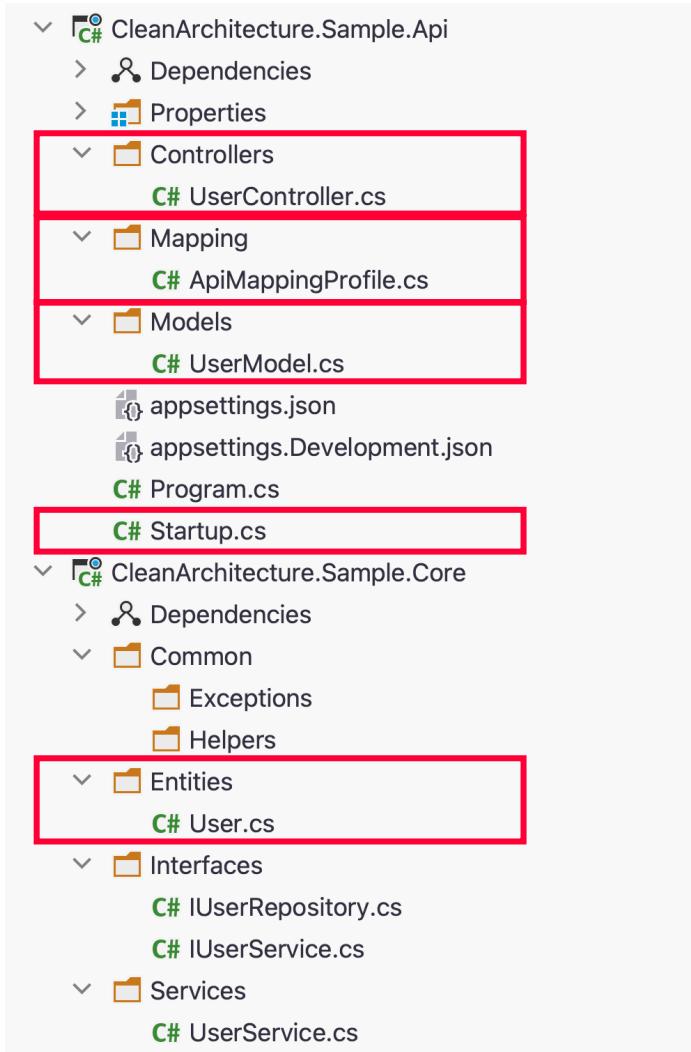
Mapping на каждом слое:

- + Изменение данных в одном слое не затрагивает другой слой
- + Аннотации, нужные для какой-то библиотеки не попадут в другие слои
- Может быть много дублирования
- При изменении данных все равно приходится менять mapper

Использование объектов из слоя Core:

- + Нет дублирования кода
- + Меньше работы
- Присутствие аннотаций, нужных для внешних библиотек на внутреннем слое
- При изменении этого объекта, возможно придется менять код в других слоях

Нужен ли mapping между слоями?



```
[Route(template: "/api/users")]
@Igor Lopushko *
public class UserController : Controller
{
    private readonly IMapper _mapper;
    private readonly IUserService _userService;

    @Igor Lopushko *
    public UserController([Mapper] mapper, [UserService] userService)
    {
        _mapper = mapper;
        _userService = userService;
    }

    [HttpGet]
    [Route(template: "{id}")]
    @Igor Lopushko *
    public async Task< IActionResult> Get(int id)
    {
        var user = await _userService.GetUserAsync(id);
        if (user == null)
        {
            return NotFound();
        }
        return Ok(_mapper.Map< Models.UserModel >(user));
    }
}
```

Нарушение Interface Segregation принципа

```
public interface IVacancyService
{
    void AddKnowledgeArea(int vacancyId, int knowledgeAreaId);
    void AddTechnology(int vacancyId, int technologyId);
    void AddAdditionalTechnology(int vacancyId, int technologies);
    void AddAdditionalKnowledgeArea(int vacancyId, int knowledgeAreaId);
    void RemoveKnowledgeArea(int vacancyId, int knowledgeAreaId);
    void RemoveTechnology(int vacancyId, int technologyId);
    NiceToHaveOption AddVacancyOption(int vacancyId, string text);
    void SetVacancyOption(int vacancyId, int optionId, string text);
    void RemoveVacancyOption(int vacancyId, int optionId);
    Responsibility AddResponsibility(int vacancyId, string text);
    void SetResponsibility(int vacancyId, int responsibilityId, string text);
    void RemoveResponsibility(int vacancyId, int responsibilityId);
    void AddInterviewStep(int vacancyId, int interviewStepId, int order);
    void DeleteInterviewStep(int vacancyId, int interviewStepId);
    Vacancy GetVacancy(int vacancyId);
    Vacancy GetVacancyPreview(int vacancyId);
    Vacancy PatchVacancy(int vacancyId, JsonPatchDocument<Vacancy> vacancyToUpdate);
    Vacancy CreateVacancy(int recruiterId);
    void Publish(int vacancyId);
    void AddWorkType(int vacancyId, int workTypeId);
    void RemoveWorkType(int vacancyId, int workTypeId);
}
```

```
public interface IVacancyKnowledgeAreaService
{
    void AddKnowledgeArea(int vacancyId, int knowledgeAreaId);
    void RemoveKnowledgeArea(int vacancyId, int knowledgeAreaId);
    void AddAdditionalKnowledgeArea(int vacancyId, int knowledgeAreaId);
}

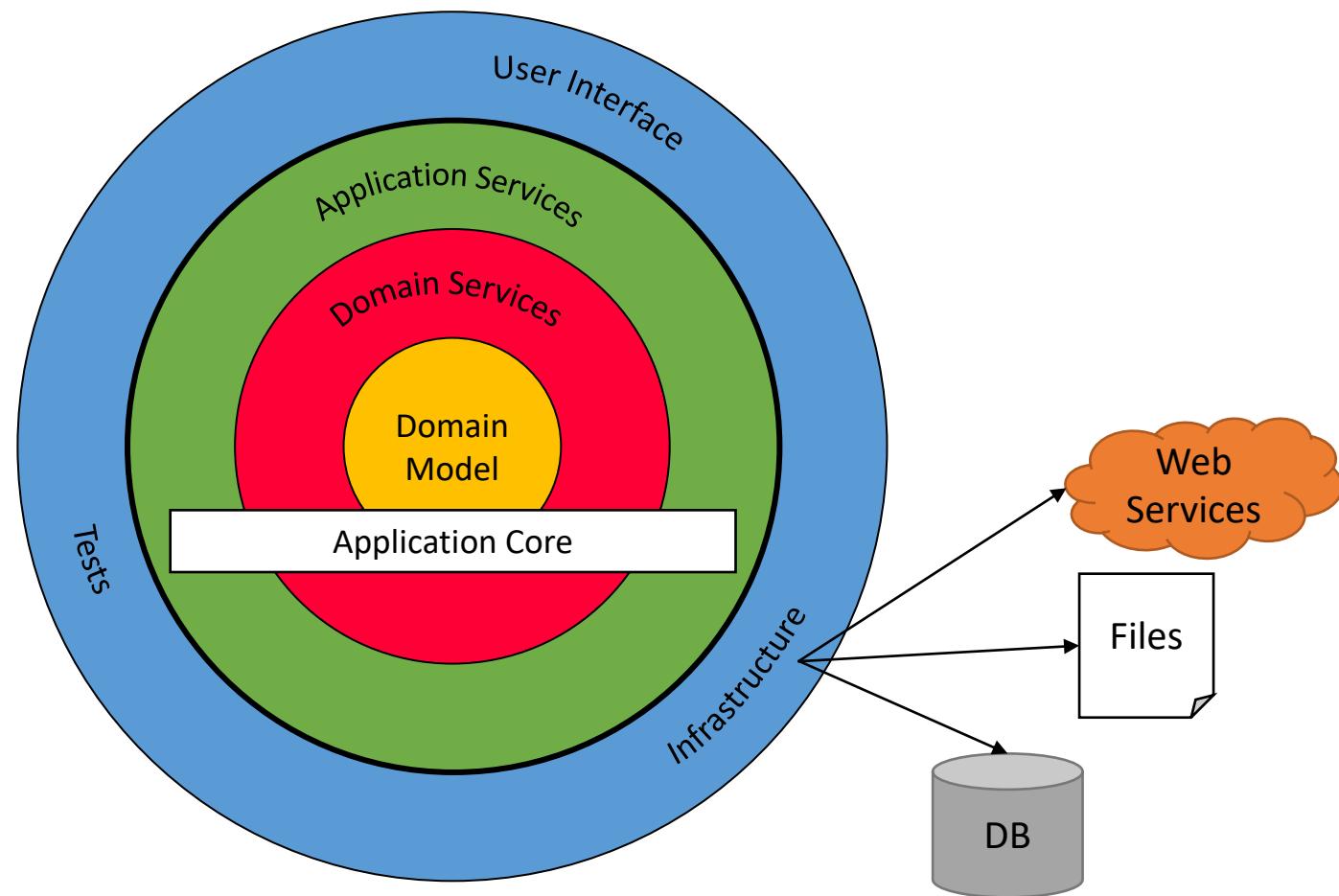
public interface IVacancyTechnologyService
{
    void AddTechnology(int vacancyId, int technologyId);
    void AddAdditionalTechnology(int vacancyId, int technologies);
    void RemoveTechnology(int vacancyId, int technologyId);
}

public interface IVacancyOptionService
{
    NiceToHaveOption AddVacancyOption(int vacancyId, string text);
    void SetVacancyOption(int vacancyId, int optionId, string text);
    void RemoveVacancyOption(int vacancyId, int optionId);
}

public interface IVacancyResponsibilityService
{
    Responsibility AddResponsibility(int vacancyId, string text);
    void SetResponsibility(int vacancyId, int responsibilityId, string text);
    void RemoveResponsibility(int vacancyId, int responsibilityId);
}
```

Вызов Repository только из Application Core?

В идеале использовать Repository нужно только через Service



Вызов Repository только из Application Core?

```
[Route(template: "/api/users")]
public class UserController : Controller
{
    private readonly IUserRepository _userRepository;

    public UserController(IUserRepository userRepository)
    {
        _userRepository = userRepository;
    }

    [HttpGet]
    [Route(template: "{id}")]
    public async Task<IActionResult> Get(int id)
    {
        var user = await _userRepository.GetAsync(id);
        if (user == null)
        {
            return NotFound();
        }
        return Ok(user);
    }
}
```

```
[Route(template: "/api/users")]
public class UserController : Controller
{
    private readonly IUserService _userService;

    public UserController(IUserService userService)
    {
        _userService = userService;
    }

    [HttpGet]
    [Route(template: "{id}")]
    public async Task<IActionResult> Get(int id)
    {
        var user = await _userService.GetUserAsync(id);
        if (user == null)
        {
            return NotFound();
        }
        return Ok(user);
    }
}
```

Entity Framework в сервисах напрямую?

- Нет, они на разных слоях
- Нет, они выполняют различные функции

Entity Framework в сервисах напрямую?

```
public interface IApplicationDbContext
{
    DbSet<User> Users { get; set; }

    DatabaseFacade Database { get; }

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

```
public class UserService : IUserService
{
    private readonly IApplicationDbContext _dbContext;

    public UserService(IApplicationDbContext dbContext)
    {
        _dbContext = dbContext;
    }

    public async Task CreateAsync(User user, CancellationToken token)
    {
        await _dbContext.Users.AddAsync(user, token);
        await _dbContext.SaveChangesAsync(token);
    }

    public async Task<User> GetAsync(int id){...}

    public IEnumerable<User> GetAllAsync(){...}

    public async Task DeleteAsync(int id, CancellationToken token){...}

    public async Task UpdateAsync(User user, CancellationToken token){...}
}
```

Entity Framework в сервисах напрямую?

```
public interface IUserRepository
{
    Task CreateAsync(User user, CancellationToken token);
    Task<User> GetAsync(int id);
    IEnumerable<User> GetAllAsync();
    public Task DeleteAsync(int id, CancellationToken token);
    public Task UpdateAsync(User user, CancellationToken token);
}
```

```
public class UserService : IUserService
{
    private readonly IUserRepository _userRepository;

    public UserService(IUserRepository userRepository)
    {
        _userRepository = userRepository;
    }

    public async Task CreateUserAsync(User user, CancellationToken token)
    {
        await _userRepository.CreateAsync(user, token);
    }

    public async Task<User> GetUserAsync(int id){...}

    public IEnumerable<User> GetAllUsersAsync(){...}

    public async Task DeleteUserAsync(int id, CancellationToken token){...}

    public async Task UpdateUserAsync(User user, CancellationToken token){...}
}
```

Избыточность для простых CRUD операций?

- Дублирование кода
- Много boilerplate кода
- Однострочный вызов



Ресурсы

- Многослойная архитектура:

<https://stackify.com/n-tier-architecture/>

- Чистая архитектура от Дяди Боба:

<https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>

- Джеки Палермо описание Onion архитектуры:

<https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/>

- Пример кода:

<https://www.c-sharpcorner.com/article/onion-architecture-in-asp-net-core-mvc/>

Ваши вопросы?