

# DDD на практике

Анисов Дмитрий



**GENERAL  
SATELLITE**





- Ведущий Backend-разработчик в компании GS Labs
- Основные языки разработки – Go/Python
- Смог сократить время разработки/поддержки новых сервисов с помощью DDD

Контактная информация:

Telegram: @anisovd

VK: <https://vk.com/anisovd>

Mail: [dimaanisov24@gmail.com](mailto:dimaanisov24@gmail.com)



# Цель доклада

- Тактическое проектирование. Как проектировать внутреннюю архитектуру
- Организация кода
- Ответы на самые частые вопросы
- Общая информация

# Что это такое?

- Это **подход** к разработке программного обеспечения
- Данный подход объединяет экспертов в предметной области и разработчиков
- Позволяет писать правильно спроектированное программное обеспечение
- **Необходимо отталкиваться от бизнес-требований, а не от технических деталей**

# Когда стоит использовать

- DDD используется в наиболее важных областях бизнеса
- Где требования часто меняются
- Для упрощения работы с предметной областью
- Для уверенности соответствия кода бизнесу
- Для дешёвого онбординга

# Проблемы в DDD

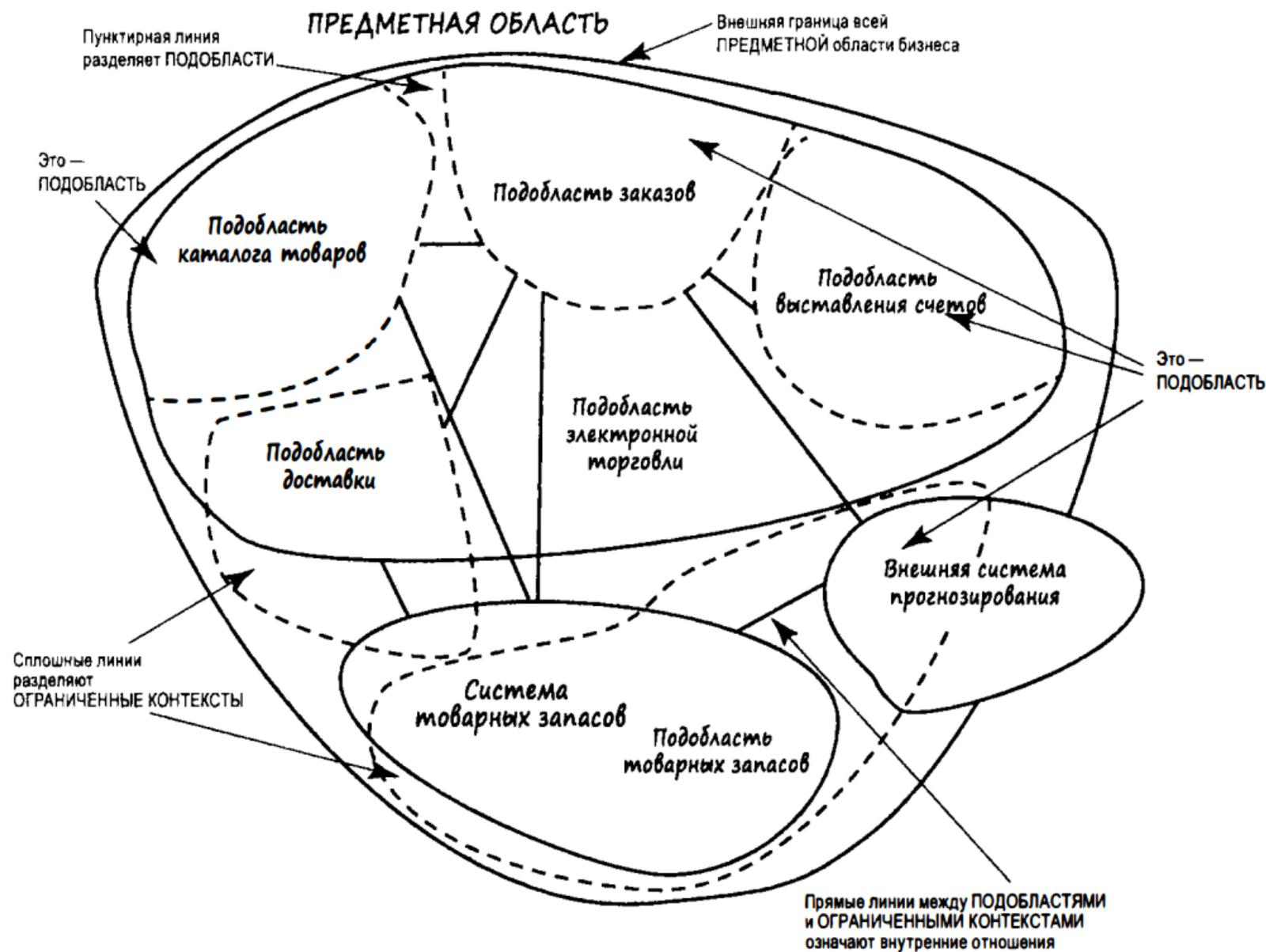
- Непонятно, как проектировать внутреннюю архитектуру сервиса
- Нет единого стандарта
- Разные архитектурные стили
- Где размещать бизнес-логику, валидацию, как работать с транзакциями, а также не зависеть от фреймворков и технологий
- Долго/дорого, надо много разбираться
- И т.д.

# С чего начать?

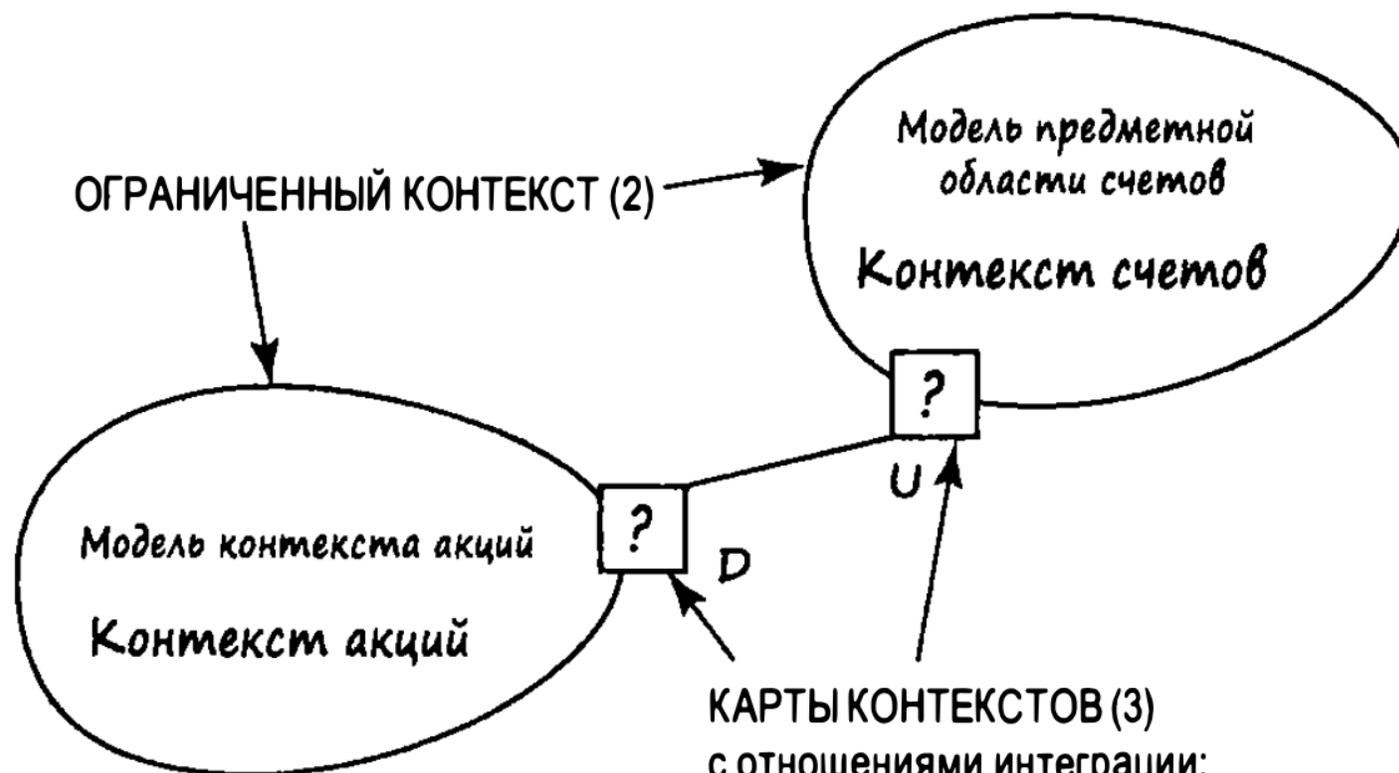
- Стратегическое проектирование
- Тактическое проектирование

# Стратегическое проектирование

- Проектирование карты контекстов
- Единый язык
- Архитектурный стиль: hexagonal, onion, clean architecture



# Единый язык и ограниченный контекст



КАРТЫ КОНТЕКСТОВ (3)  
с отношениями интеграции:

СЛУЖБЫ С ОТКРЫТЫМ ПРОТОКОЛОМ,  
ПРЕДОХРАНИТЕЛЬНЫЙ УРОВЕНЬ,  
ЗАКАЗЧИК-ПОСТАВЩИК,  
ОБЩЕЕ ЯДРО

# Тактическое проектирование

- Тактическое проектирование обычно сложнее стратегического
- Тактическое проектирование осуществляется внутри ограниченного контекста с использованием различных шаблонов проектирования

# Шаблоны

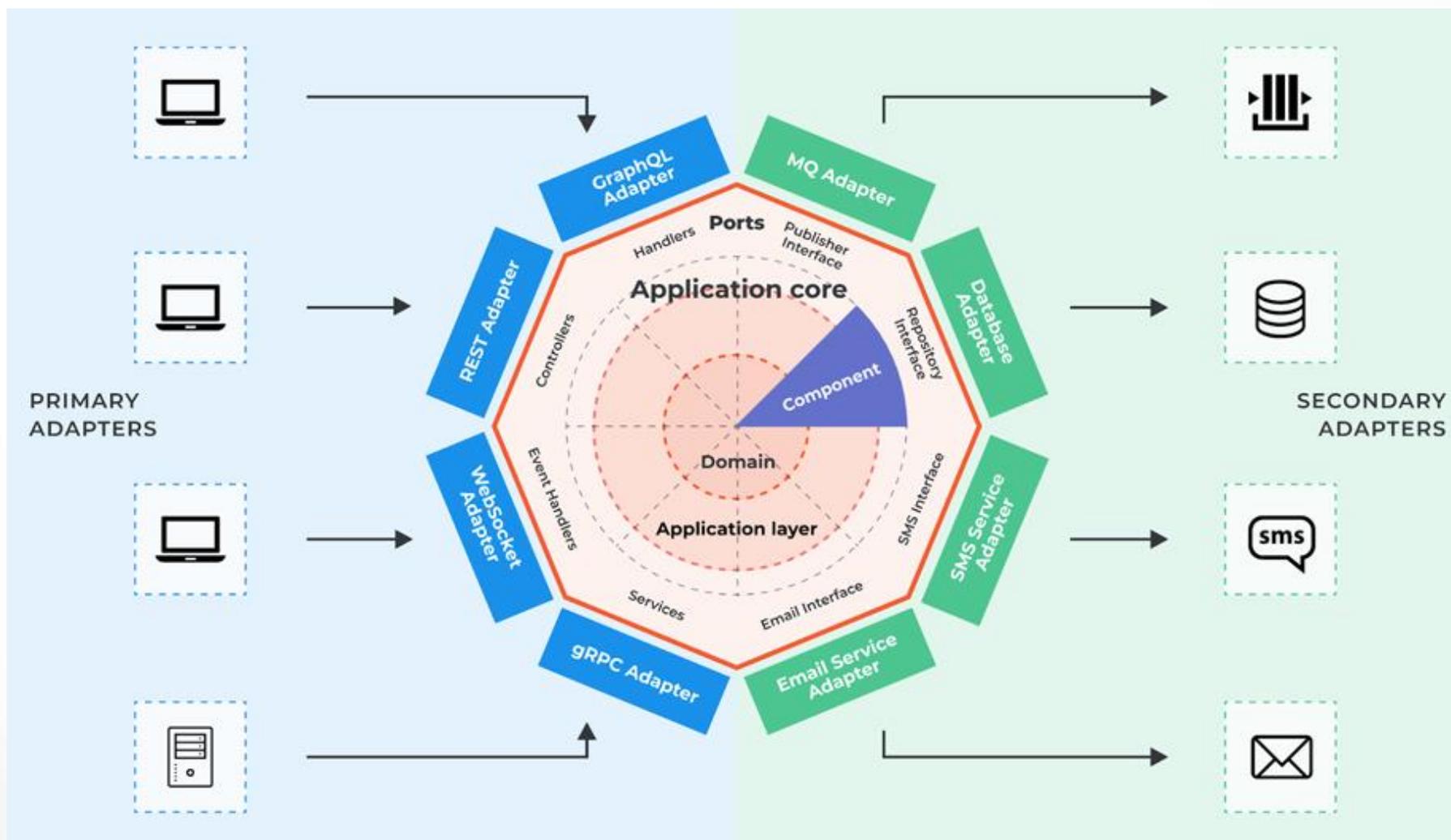
- Агрегат
- Сущности
- Объект-значение
- Хранилища
- Службы
- События предметной области
- Интеграция ограниченных контекстов

# Выбор архитектуры

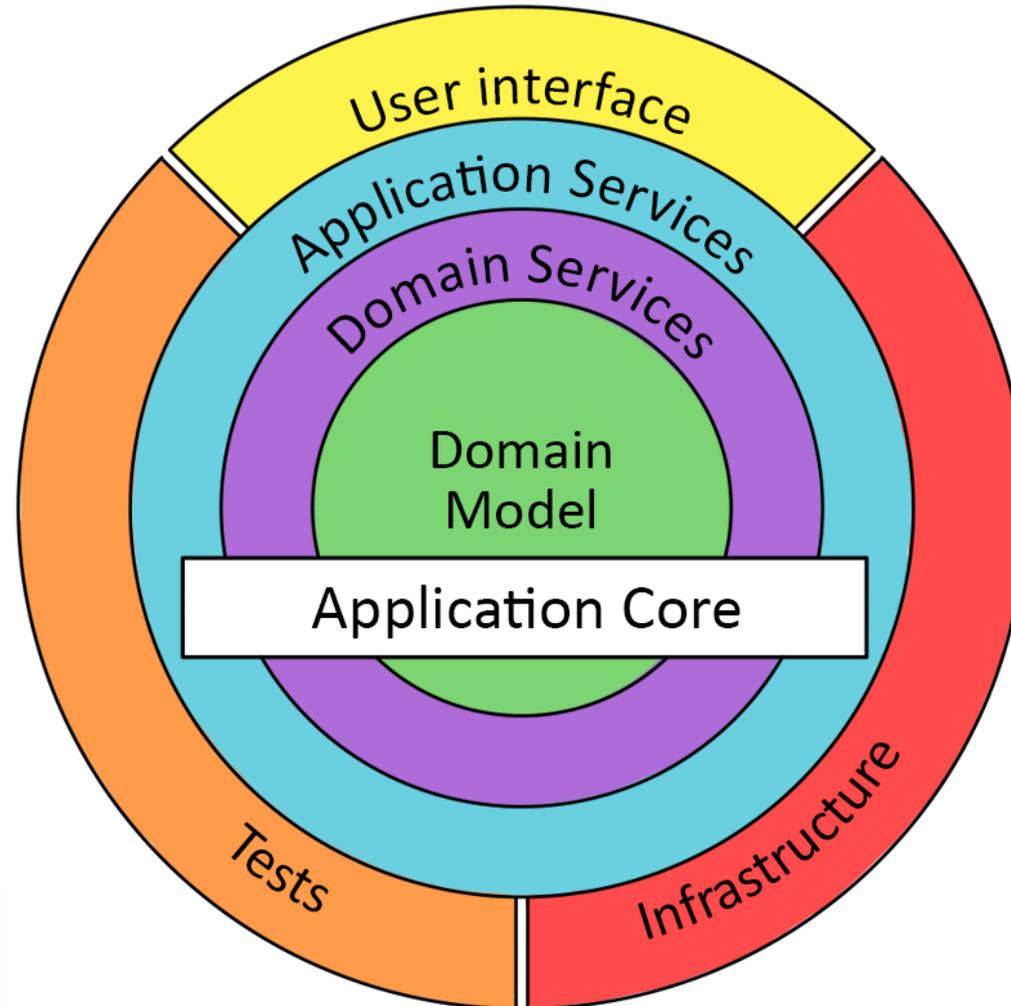
*Насколько большие различия?*

- Гексагональная архитектура
- Луковая архитектура
- Чистая архитектура

# Гексагональная архитектура

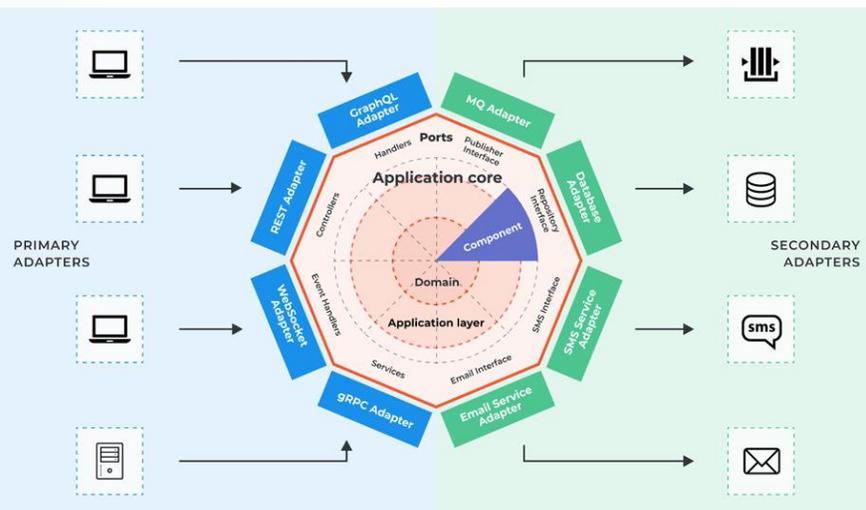


# Луковая архитектура

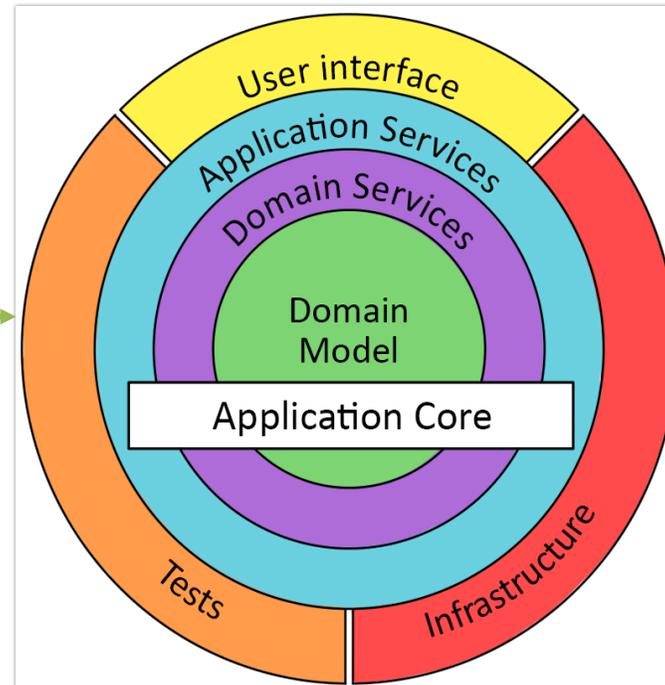


# Чистая архитектура

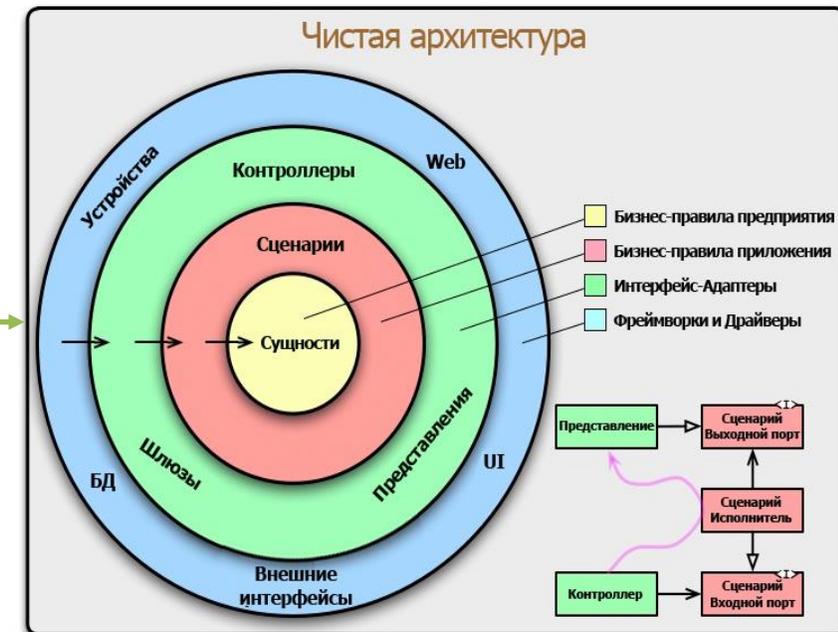




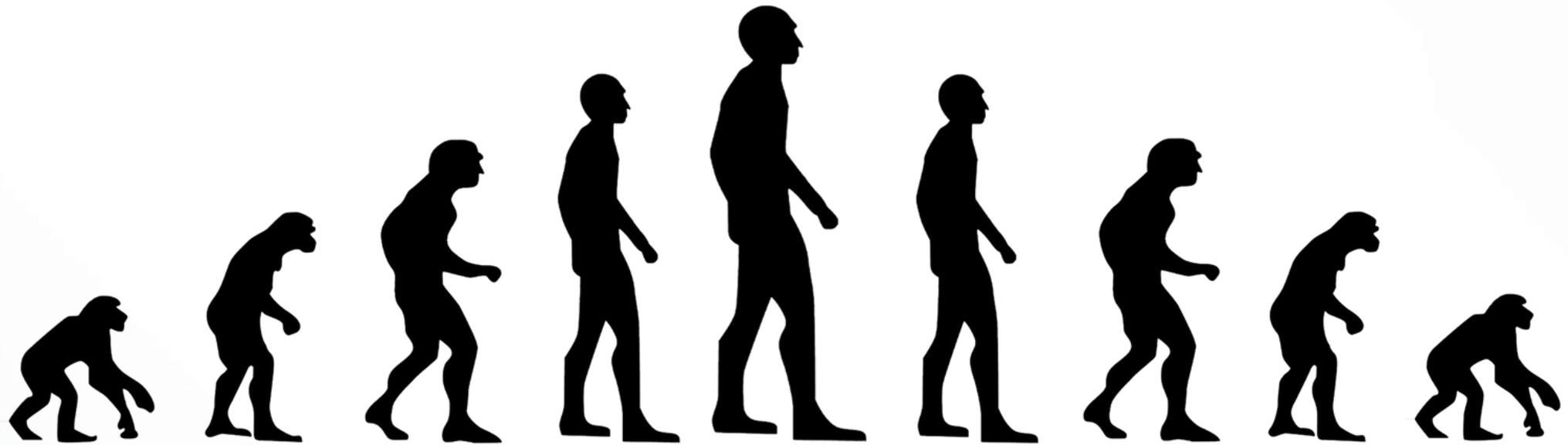
2005



2008



2012



# Чистая архитектура и её минусы

<ul style="list-style-type: none"> <li>&gt; docs</li> <li>▼ rentomatic             <ul style="list-style-type: none"> <li>▼ domain                 <ul style="list-style-type: none"> <li>stageroom.py</li> </ul> </li> <li>▼ repository                 <ul style="list-style-type: none"> <li>__init__.py</li> <li>memrepo.py</li> </ul> </li> <li>&gt; rest</li> <li>▼ serializers                 <ul style="list-style-type: none"> <li>stageroom_serializer.py</li> </ul> </li> <li>&gt; shared</li> <li>▼ use_cases                 <ul style="list-style-type: none"> <li>request_objects.py</li> <li>stageroom_use_cases.py</li> <li>__init__.py</li> <li>app.py</li> <li>settings.py</li> </ul> </li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>▼ app             <ul style="list-style-type: none"> <li>&gt; adapters</li> <li>▼ dtos                 <ul style="list-style-type: none"> <li>__init__.py</li> <li>user.py</li> </ul> </li> <li>▼ entities                 <ul style="list-style-type: none"> <li>__init__.py</li> <li>user.py</li> </ul> </li> <li>▼ repositories                 <ul style="list-style-type: none"> <li>__init__.py</li> <li>firestore.py</li> <li>memory.py</li> </ul> </li> <li>▼ routers/v1                 <ul style="list-style-type: none"> <li>__init__.py</li> <li>users.py</li> </ul> </li> <li>▼ use_cases                 <ul style="list-style-type: none"> <li>__init__.py</li> <li>users.py</li> </ul> </li> <li>__init__.py</li> <li>main.py</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>▼ auth             <ul style="list-style-type: none"> <li>▼ delivery/http                 <ul style="list-style-type: none"> <li>handler.go</li> <li>handler_test.go</li> <li>middleware.go</li> <li>middleware_test.go</li> <li>register.go</li> </ul> </li> <li>▼ repository                 <ul style="list-style-type: none"> <li>localstorage</li> <li>mock</li> <li>mongo</li> </ul> </li> <li>▼ usecase                 <ul style="list-style-type: none"> <li>mock.go</li> <li>usecase.go</li> <li>usecase_test.go</li> </ul> </li> <li>error.go</li> <li>repository.go</li> <li>usecase.go</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>▼ app             <ul style="list-style-type: none"> <li>▼ database                 <ul style="list-style-type: none"> <li>migrations</li> <li>seeds</li> </ul> </li> <li>▼ domain                 <ul style="list-style-type: none"> <li>post.go</li> <li>user.go</li> </ul> </li> <li>▼ infrastructure                 <ul style="list-style-type: none"> <li>env.go</li> <li>logger.go</li> <li>router.go</li> <li>sqlhandler.go</li> </ul> </li> <li>&gt; interfaces</li> <li>&gt; log</li> <li>&gt; usecases</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>▼ src             <ul style="list-style-type: none"> <li>▼ adapter                 <ul style="list-style-type: none"> <li>api</li> <li>spi</li> </ul> </li> <li>▼ application                 <ul style="list-style-type: none"> <li>__init__.py</li> <li>mappers</li> <li>repositories</li> <li>spi</li> <li>usecases</li> <li>utils</li> </ul> </li> <li>▼ domain                 <ul style="list-style-type: none"> <li>__init__.py</li> <li>api_exception.py</li> <li>base_entity.py</li> <li>cat_fact.py</li> <li>configuration_entity.py</li> <li>dog_fact.py</li> </ul> </li> <li>▼ infrastructure                 <ul style="list-style-type: none"> <li>__init__.py</li> <li>app.py</li> <li>config_mapper.py</li> <li>__init__.py</li> </ul> </li> </ul> </li> </ul>
1	2	3	4	5

# Зачем нужен единый подход?

- Единообразии
- Легкость внедрения новых разработчиков
- Практичность
- Удобство

# Структура проекта

```

playlist-manager ~/main/
├── .helm
├── assets
├── build
├── cmd
├── internal
├── application
├── common
├── domain
│   ├── consts
│   ├── entity
│   ├── interfaces
│   └── service
├── infrastructure
├── repository
├── service
├── presentation
│   ├── cron
│   └── http
├── server
├── pkg
├── test
└── vendor
    
```

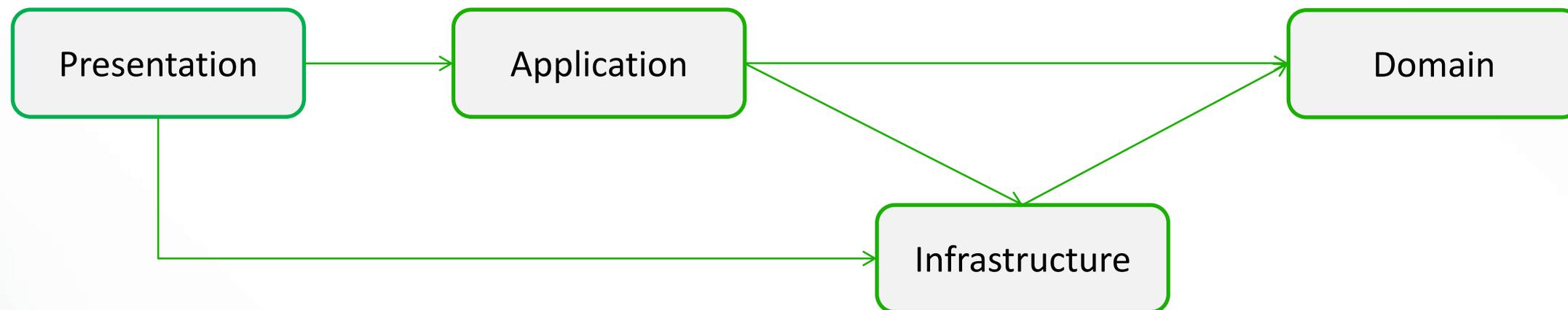
```

> .helm
> .mypy_cache
└── app
    ├── application
    ├── common
    ├── domain
    ├── infrastructure
    ├── presentation
    ├── __init__.py
    └── server.py
    ├── tests
    ├── .dockerignore
    ├── .gitignore
    ├── .gitlab-ci.yml
    ├── .python-version
    ├── CHANGELOG.md
    ├── Dockerfile
    ├── Makefile
    ├── poetry.lock
    ├── pyproject.toml
    ├── README.md
    └── setup.cfg
    
```

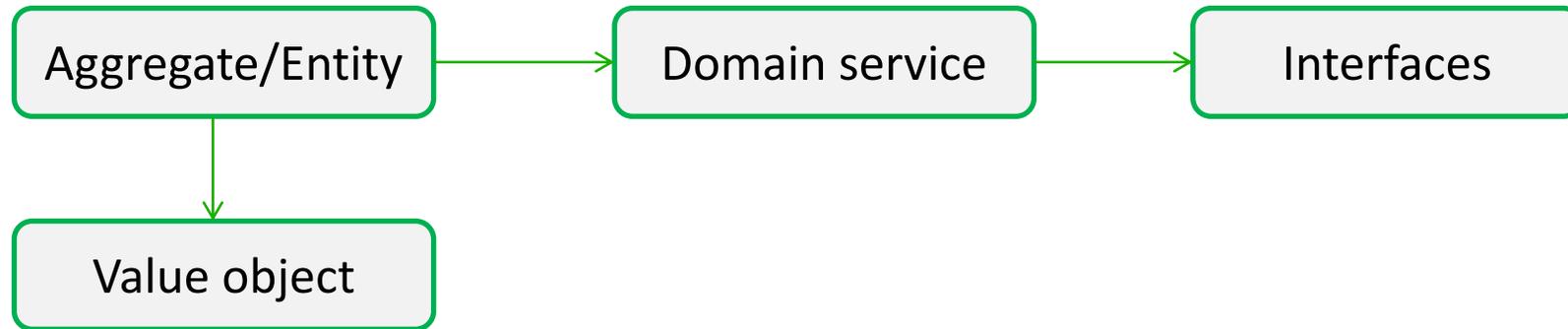
```

identityaccess
├── application
│   ├── command
│   └── representation
│       ├── AccessApplicationService.java
│       ├── ApplicationServiceRegistry.java
│       ├── IdentityAccessEventProcessor.java
│       ├── IdentityApplicationService.java
│       └── NotificationApplicationService.java
├── domain
├── model
│   └── access
│       ├── AuthorizationService.java
│       ├── GroupAssignedToRole.java
│       ├── GroupUnassignedFromRole.java
│       ├── Role.java
│       ├── RoleProvisioned.java
│       ├── RoleRepository.java
│       ├── UserAssignedToRole.java
│       └── UserUnassignedFromRole.java
│       └── identity
│           └── DomainRegistry.java
├── infrastructure
├── persistence
├── services
└── resource
    
```

# Упрощенная последовательность вызова



# Domain Layer



- ▼  domain
  - >  consts
  - ▼  entity
    -  cdn.go
    - >  hls.go
    -  playout.go
    -  profiles.go
    -  values.go
  - >  interfaces
  - >  service

# Entity/Aggregates

## Доменные объекты

```
playlist-manager ~/main/wc 71 // Chunk entity
> .helm 72 @ type Chunk struct { 24 usages anisov
> assets 73     Name string `json:"name"`
> build 74     MediaSequence int `json:"media_sequence"`
> cmd 75     Timestamp int `json:"timestamp"`
> internal 76     Tags *Tags `json:"tags"`
  application 77     OriginalUrl string `json:"original_url"`
  common 78     AdditionalSegmentData map[string]interface{} `json:"additional_segment_data"`
  domain 79     Discontinuity bool
    consts 80     Key string
    entity 81 }
  cdn.go 82
  hls.go 83 // HlsData entity for playlist and chunks
  playout.go 84 // easyjson:skip
  profiles.go 85 type HlsData struct { 10 usages anisov
  values.go 86     Chunks []*Chunk
  87     PlaylistData []byte
  88 }
  interfaces 89
  service 90 // MetaInfoStartOver meta info for startover
  infrastructure 91 @ type MetaInfoStartOver struct { 10 usages anisov
  presentation 92     PlaylistKey string
  server 93     StartTime int64
  pkg 94     EndTime int64
  test 95     LastSegmentTimestamp int
  vendor 96 }
```

# Anemic Domain Model

Не/Анемичная модель данных

```
type UserMediaData struct {
```

```
...
```

```
}
```

```
func (entity *UserMediaData) GenerateMinifyLink() error {
```

```
...
```

```
}
```

# Interfaces

Контракты, описывающие взаимодействие со сторонними системами/базами данных

```
playlist-manager ~/main/wc 12 // HlsRepository interface for getting data
> .helm 13 type HlsRepository interface { 5 usages 3 implementations anisov
> assets 14 GetHlsData(params *entity.PlaylistParams, getPlaylist bool) (*entity.HlsData, error) 2 implementations
> build 15 GetHlsUploadPlaylistContent(params *entity.PlaylistParams) ([]byte, error) 2 implementations
> cmd 16 UploadStartOverPlaylist(playlist string, key string, params *entity.PlaylistParams, lastSegmentTimestamp int) error
> internal 17 GetStartOverPlaylist(params *entity.PlaylistParams, key string) (*entity.StartOverPlaylistData, error) 2 implementat
  > application 18 BaseRepository
  > common 19 }
  > domain 20
    > consts 21 // ProfileCacheRepository interface for save/get profiles
    > entity 22 type ProfileCacheRepository interface { 6 usages 2 implementations anisov
      cdn.go 23 Get() (*entity.Profiles, error) 2 implementations
      hls.go 24 Save(profiles *entity.Profiles) error 2 implementations
      playout.go 25 }
      profiles.go 26
      values.go 27 // ProfileDownloadRepository interface for getting profiles from external system
    > interfaces 28 type ProfileDownloadRepository interface { 4 usages 5 implementations anisov
      repository.go 29 Get() (*entity.Profiles, error) 5 implementations
      service.go 30 }
  > infrastructure 31
  > presentation 32 // AccessRepository interface for getting a list of available regions and countries
  > server 33 type AccessRepository interface { 4 usages 2 implementations anisov
    34 Get(regions []*entity.Region, countries []*entity.Country, ip string) ([]*entity.Region, []*entity.Country, error)
    35 }
    36
```

# Domain Service

## Доменные сервисы

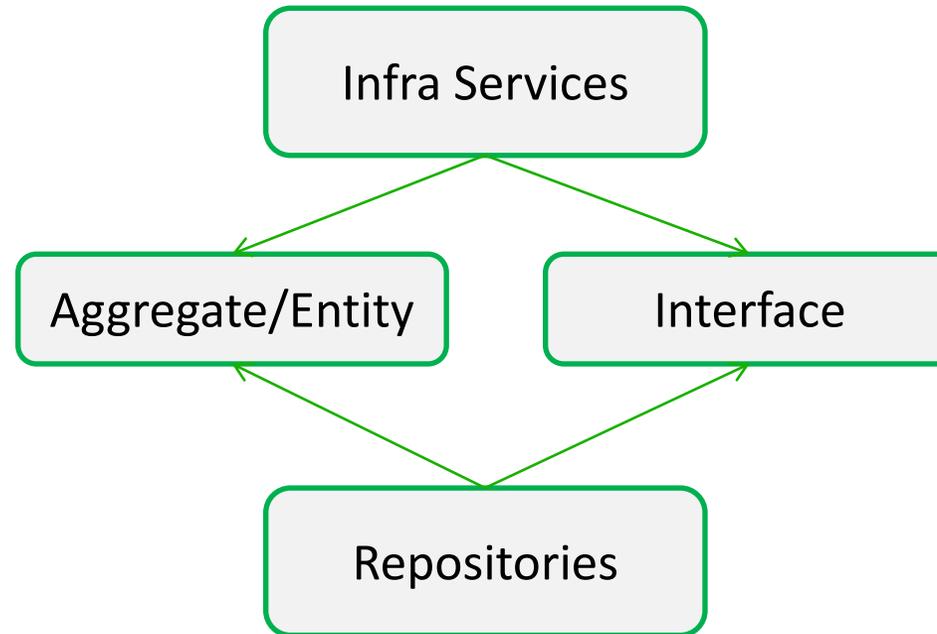
```
playlist-manager ~/main/wc 16 // ProfileService is a struct for domain service which prepare specific profile
> .helm 17 type ProfileService struct { 14 usages  2 anisov
> assets 18     repository      interfaces.ProfileCacheRepository
> build 19     accessService    *access.AccessService
> cmd 20     cdnMetricsService *metrics.CDNMetricsService
> internal 21 }
  > application 22
  > common 23 // ProfileUpdateService is a struct for domain service which update common profiles
  > domain 24 type ProfileUpdateService struct { 4 usages  2 anisov
    > consts 25     repository      interfaces.ProfileDownloadRepository
    > entity 26     cacheRepository interfaces.ProfileCacheRepository
    > interfaces 27 }
    > service 28
      > access 29 // NewProfilesService is a function for creating profile domain service
      > hls 30 func NewProfilesService( 4 usages  2 anisov
      > metrics 31     repository interfaces.ProfileCacheRepository,
      > playout 32     accessService *access.AccessService,
      > profiles 33     cdnMetricsService *metrics.CDNMetricsService,
        errors.go 34 ) *ProfileService {
        service.go 35     return &ProfileService{
          > infrastructure 36         repository:      repository,
          > presentation 37         accessService:    accessService,
          > server 38         cdnMetricsService: cdnMetricsService,
          39     }
          40 }
```

# Второй пример

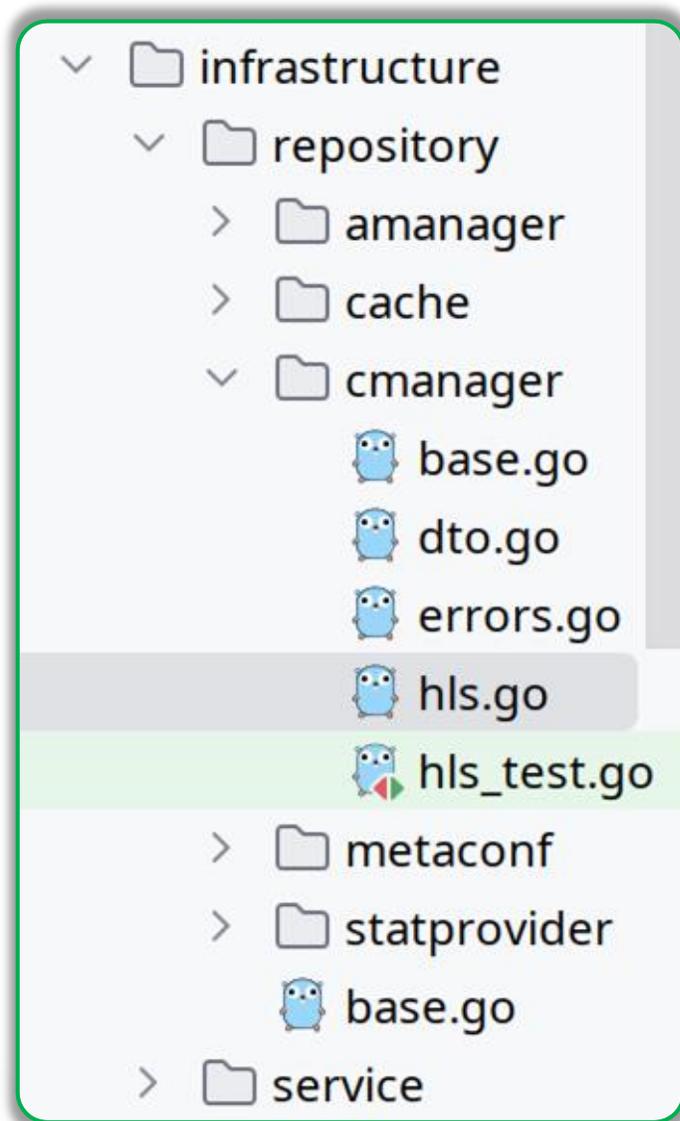
```
type VirtualMasterPlaylistService struct{} 6 usages  👤 anisov

func (service *VirtualMasterPlaylistService) CreateVirtualMasterPlaylist(
    playlist m3u8.Playlist,
    masterPlaylistURL string,
    userID string,
    operatorID string,
) (*m3u8.Playlist, []*entity.UserMediaData, error) {...}
```

## 2. Infrastructure Layer



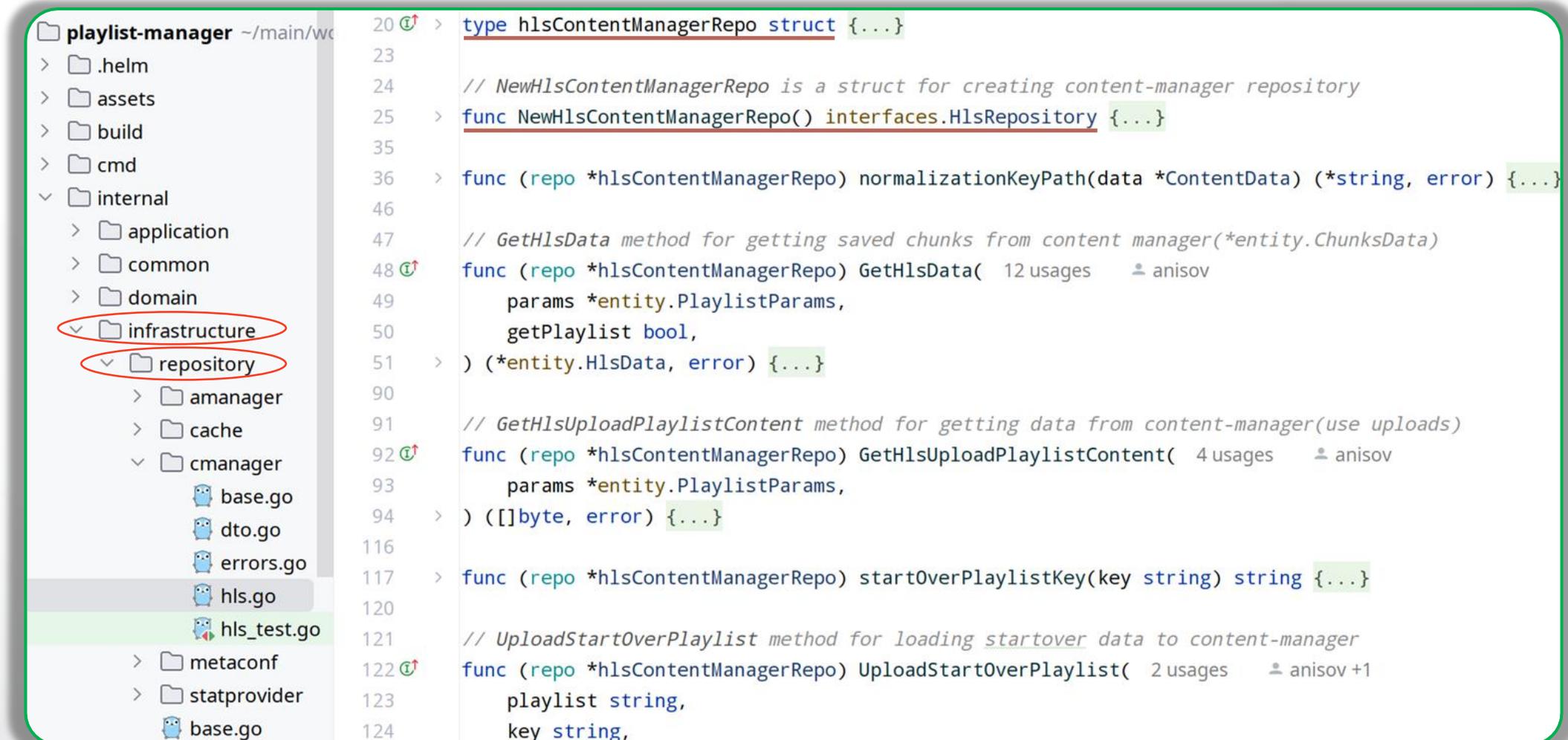
# Структура слоя



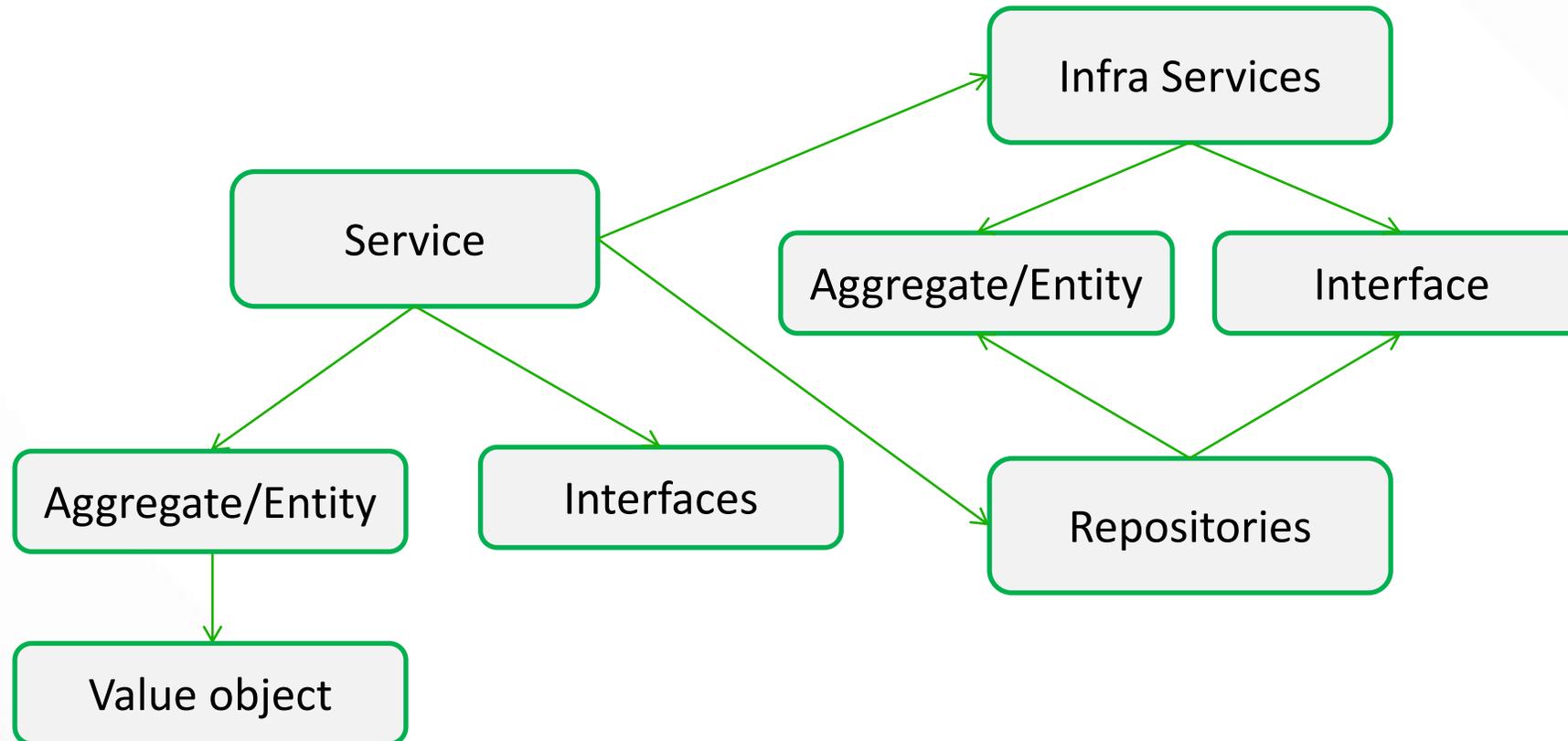
# Implementation

## Реализация интерфейсов

```
playlist-manager ~/main/wc 20 > type hlsContentManagerRepo struct {...}
23
24 // NewHlsContentManagerRepo is a struct for creating content-manager repository
25 > func NewHlsContentManagerRepo() interfaces.HlsRepository {...}
35
36 > func (repo *hlsContentManagerRepo) normalizationKeyPath(data *ContentData) (*string, error) {...}
46
47 // GetHlsData method for getting saved chunks from content manager(*entity.ChunksData)
48 > func (repo *hlsContentManagerRepo) GetHlsData( 12 usages  👤 anisov
49     params *entity.PlaylistParams,
50     getPlaylist bool,
51 > ) (*entity.HlsData, error) {...}
90
91 // GetHlsUploadPlaylistContent method for getting data from content-manager(use uploads)
92 > func (repo *hlsContentManagerRepo) GetHlsUploadPlaylistContent( 4 usages  👤 anisov
93     params *entity.PlaylistParams,
94 > ) ([]byte, error) {...}
116
117 > func (repo *hlsContentManagerRepo) startOverPlaylistKey(key string) string {...}
120
121 // UploadStartOverPlaylist method for loading startover data to content-manager
122 > func (repo *hlsContentManagerRepo) UploadStartOverPlaylist( 2 usages  👤 anisov +1
123     playlist string,
124     key string,
```



# 3. Application Layer

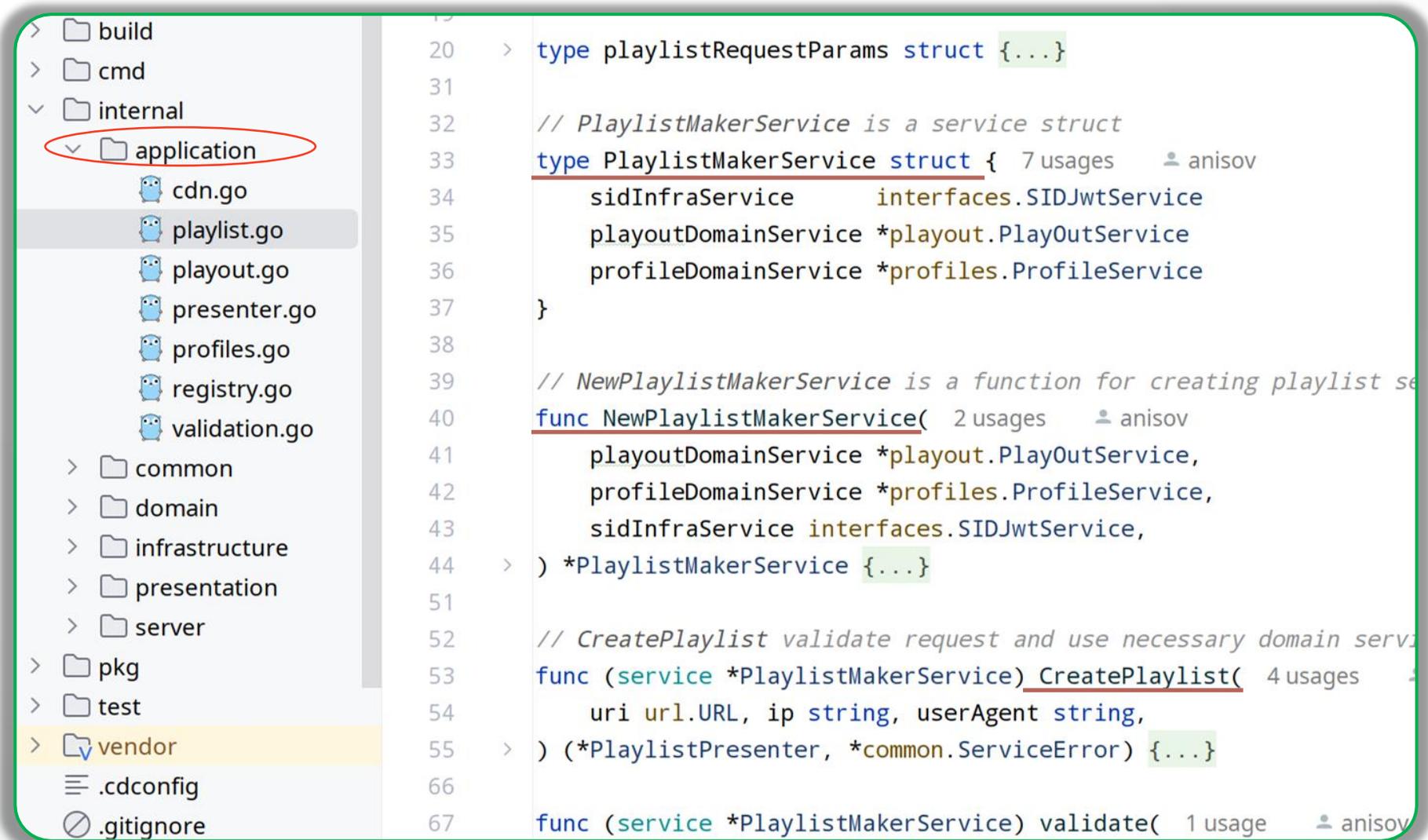


# Структура слоя

- internal
  - application
    - cdn.go
    - playlist.go
    - playout.go
    - presenter.go
    - profiles.go
    - registry.go
    - validation.go

# Application Service

Бизнес логика, интеграция с другими сервисами/репозиториями, публикация событий



```
20 > type playlistRequestParams struct {...}
31
32 // PlaylistMakerService is a service struct
33 type PlaylistMakerService struct { 7 usages  👤 anisov
34     sidInfraService      interfaces.SIDJwtService
35     playoutDomainService *playout.PlayOutService
36     profileDomainService *profiles.ProfileService
37 }
38
39 // NewPlaylistMakerService is a function for creating playlist se
40 func NewPlaylistMakerService( 2 usages  👤 anisov
41     playoutDomainService *playout.PlayOutService,
42     profileDomainService *profiles.ProfileService,
43     sidInfraService      interfaces.SIDJwtService,
44 > ) *PlaylistMakerService {...}
51
52 // CreatePlaylist validate request and use necessary domain serv
53 func (service *PlaylistMakerService) CreatePlaylist( 4 usages  👤
54     uri url.URL, ip string, userAgent string,
55 > ) (*PlaylistPresenter, *common.ServiceError) {...}
66
67 func (service *PlaylistMakerService) validate( 1 usage  👤 anisov
```

# Второй пример

```
// MasterVirtualPlaylistService is a service struct.
type MasterVirtualPlaylistService struct { 6 usages  👤 anisov *
    encryptService      interfaces.EncryptService
    cdnPlaylistService  interfaces.CDNPlaylistService
    cashRepository      interfaces.CashRepository
    operatorRepository  interfaces.OperatorSettingsRepository
    virtualPlaylistService service.VirtualMasterPlaylistService
}

// NewMasterVirtualPlaylistService is a function for creating master playlist service.
func NewMasterVirtualPlaylistService( 2 usages  👤 anisov *
    cdnPlaylistService interfaces.CDNPlaylistService,
    encryptService interfaces.EncryptService,
    cacheRepository interfaces.CashRepository,
    operatorRepository interfaces.OperatorSettingsRepository,
    virtualPlaylistService service.VirtualMasterPlaylistService,
) *MasterVirtualPlaylistService {...}

func (service *MasterVirtualPlaylistService) checkSignature( 1 usage  👤 anisov
    md5Cache string, userID string, operatorID string, expires string, secret string,
) bool {...}

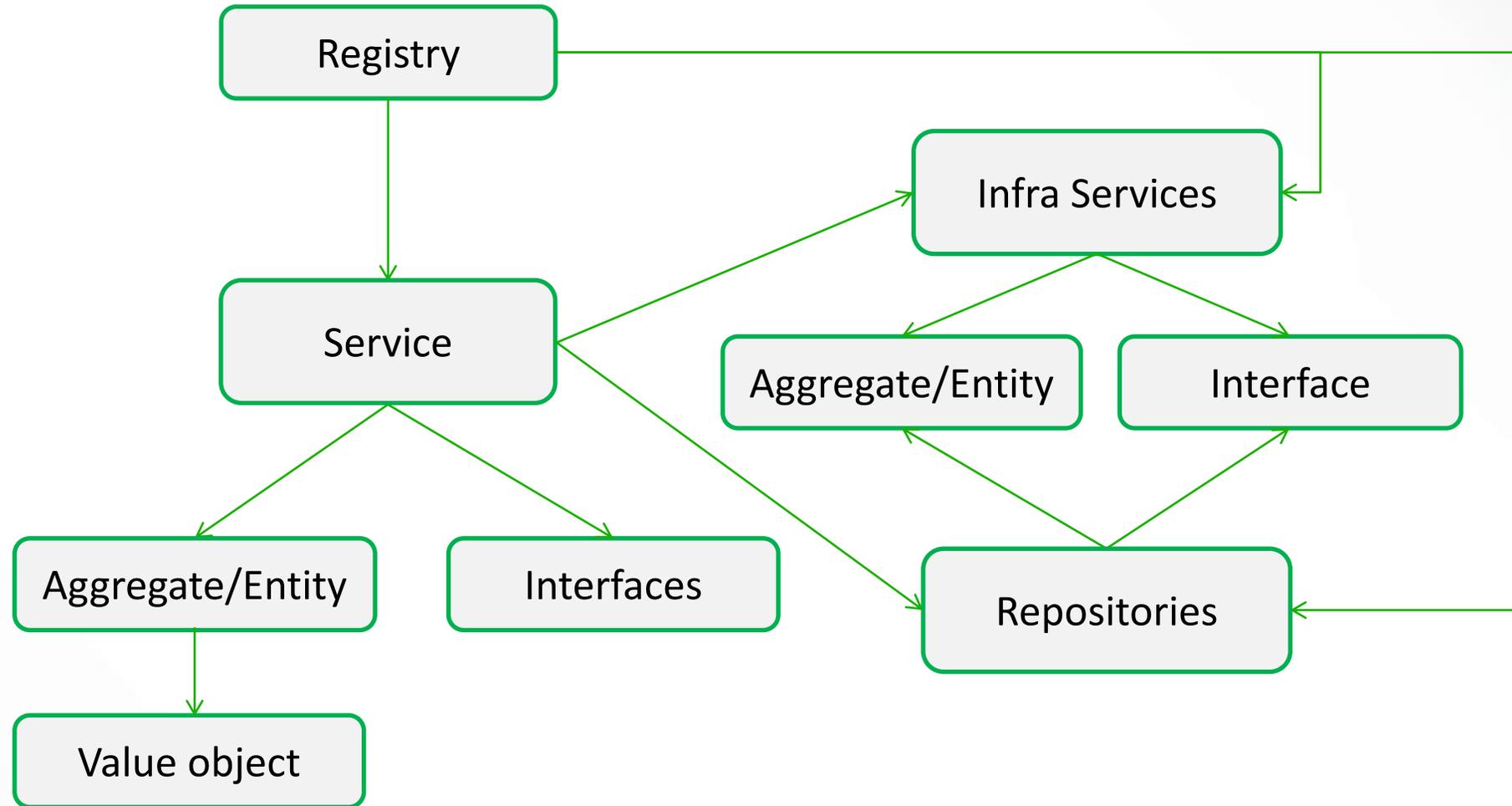
func (service *MasterVirtualPlaylistService) GetPlaylist( 1 usage  👤 anisov
    data PlaylistRequestParams,
    protectedLink bool,
    mediaFormat string,
) (*PlaylistPresenter, *common.ServiceError) {...}
```

# Domain Service/Application Service/Anemic Domain Model

- Не/Анемичная модель данных
- Domain Service
- Application Service

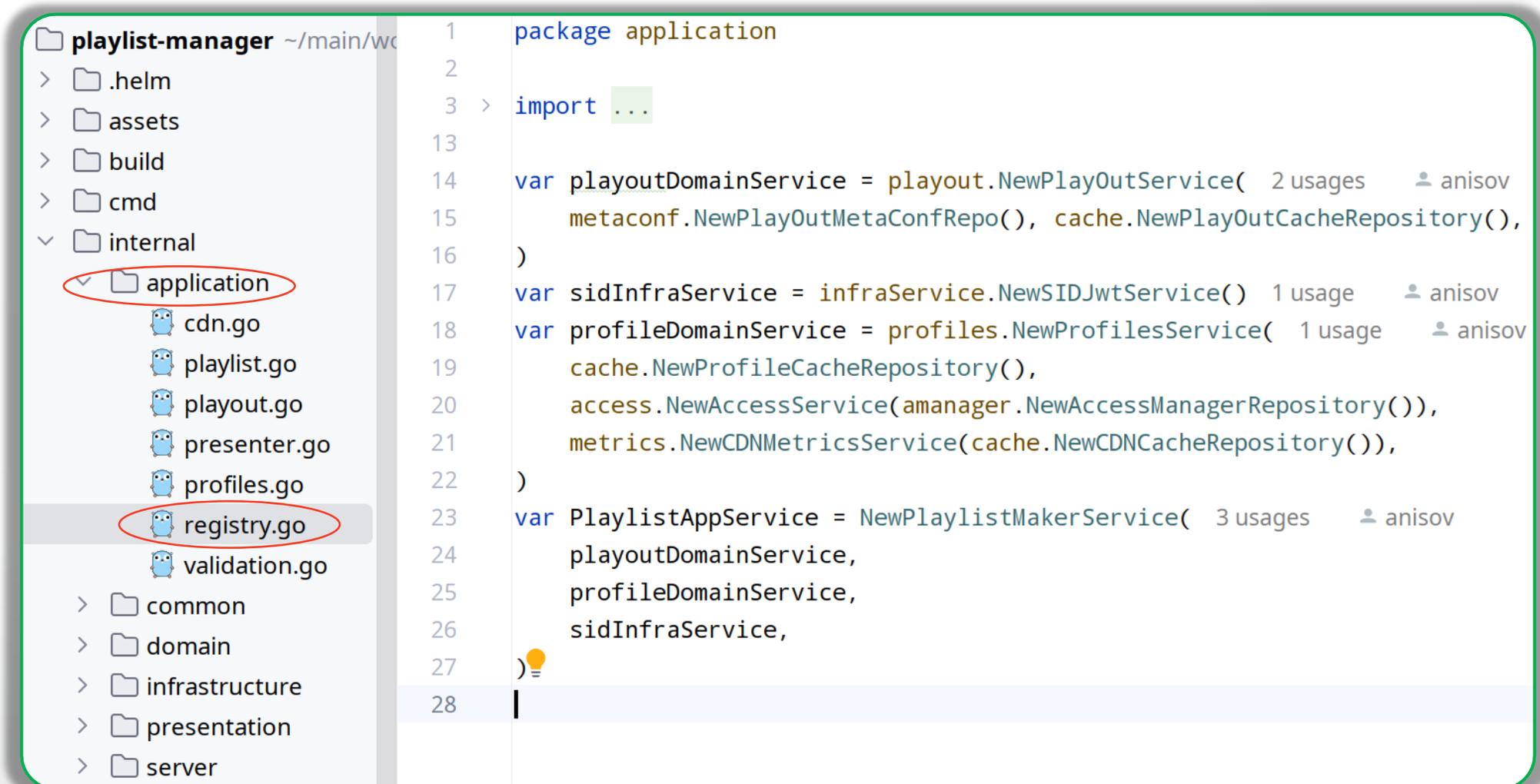


# Registry



# Registry

## Инициализация и внедрение зависимостей



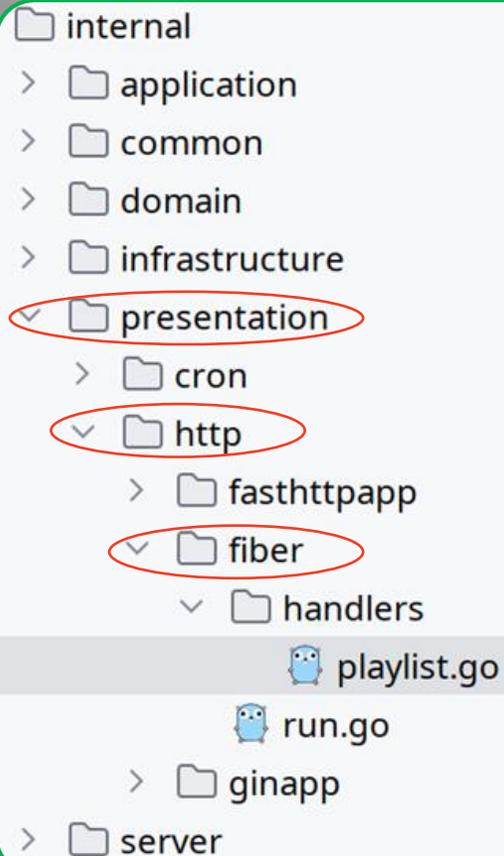
```
1 package application
2
3 > import ...
13
14 var playoutDomainService = playout.NewPlayOutService( 2 usages  👤 anisov
15     metaconf.NewPlayOutMetaConfRepo(), cache.NewPlayOutCacheRepository(),
16 )
17 var sidInfraService = infraService.NewSIDJwtService() 1 usage  👤 anisov
18 var profileDomainService = profiles.NewProfilesService( 1 usage  👤 anisov
19     cache.NewProfileCacheRepository(),
20     access.NewAccessService(amanager.NewAccessManagerRepository()),
21     metrics.NewCDNMetricsService(cache.NewCDNCacheRepository()),
22 )
23 var PlaylistAppService = NewPlaylistMakerService( 3 usages  👤 anisov
24     playoutDomainService,
25     profileDomainService,
26     sidInfraService,
27 )
28
```

# Второй пример

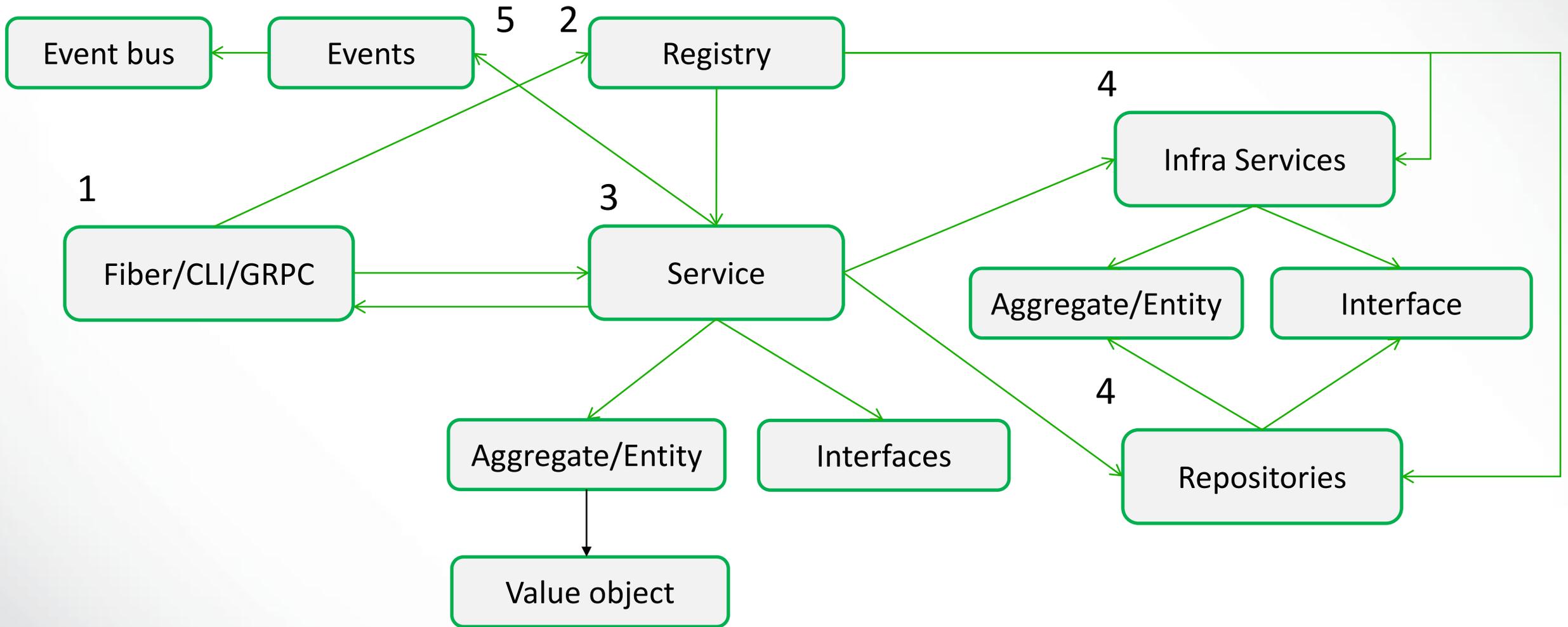
```
var (  
    MediaPlayerApplicationService *VariantVirtualPlaylistService 2 usages ⓘ an  
    MasterPlaylistApplicationService *MasterVirtualPlaylistService 2 usages ⓘ an  
)  
  
func init() { ⓘ anisov *  
    var (  
        encryptInfraService          = infraService.NewEncryptService()  
        cDNPlaylistInfraService      = infraService.NewCDNPlaylistService()  
        cacheRepository              = repository.NewCacheRepository()  
        vMasterPlaylistService       = domainService.VirtualMasterPlaylistService{}  
        vMediaPlayerService          = domainService.VirtualMediaPlayerService{}  
        operatorInfoInfraRepository = repository.NewOperatorSettingsService()  
        sequenceRepository            = repository.NewSequenceRepository()  
    )  
    if common.Settings.RandomSequence {  
        sequenceRepository = repository.NewRandomSequenceRepository()  
    }  
  
    MasterPlaylistApplicationService = NewMasterVirtualPlaylistService(  
        cDNPlaylistInfraService,  
        encryptInfraService,  
        cacheRepository,  
        operatorInfoInfraRepository,  
        vMasterPlaylistService,  
    )  
    MediaPlayerApplicationService = NewVariantVirtualPlaylistService(  
        MasterPlaylistApplicationService,  
        vMediaPlayerService,  
    )  
}
```

# 4. Presentation

## Слой представления



```
1 package handlers
2
3 > import ...
13
14 // CreateIndividualPlaylist is a handler which initializes the corresponding se
15 func CreateIndividualPlaylist(c *fiber.Ctx) error { 2 usages anisov
16 >     uri := url.URL{...}
20     clientIP := string(c.Context().Request.Header.Peek(common.CustomIpHeader))
21 >     if clientIP == "" {...}
24     playlistPresenter, err := application.PlaylistAppService.CreatePlaylist(
25         uri, clientIP, string(c.Context().UserAgent()),
26     )
27 >     if err != nil {...}
35     c.Set(key: "Content-type", playlistPresenter.ContentType)
36     return c.Status(fiber.StatusOK).SendString(playlistPresenter.Data)
37 }
```



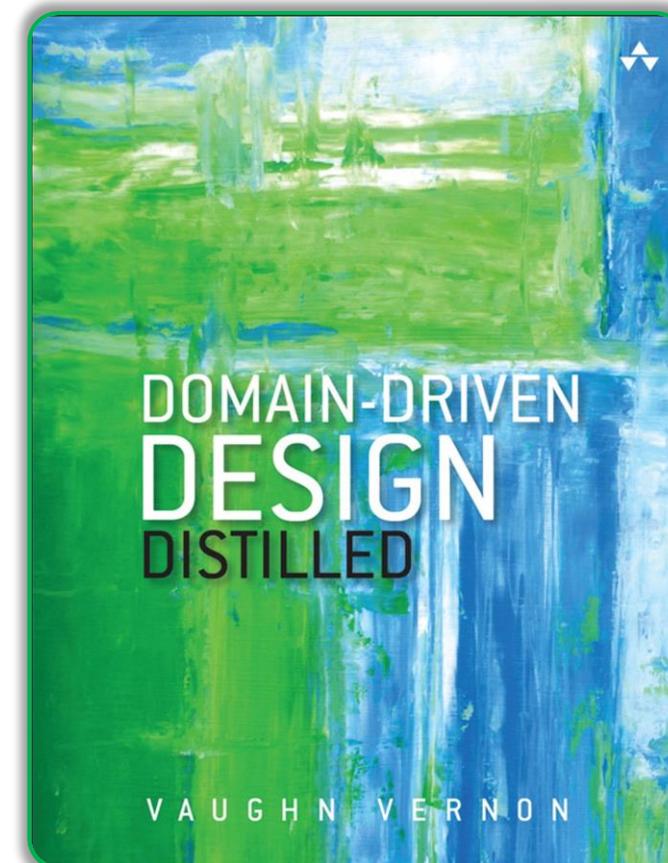
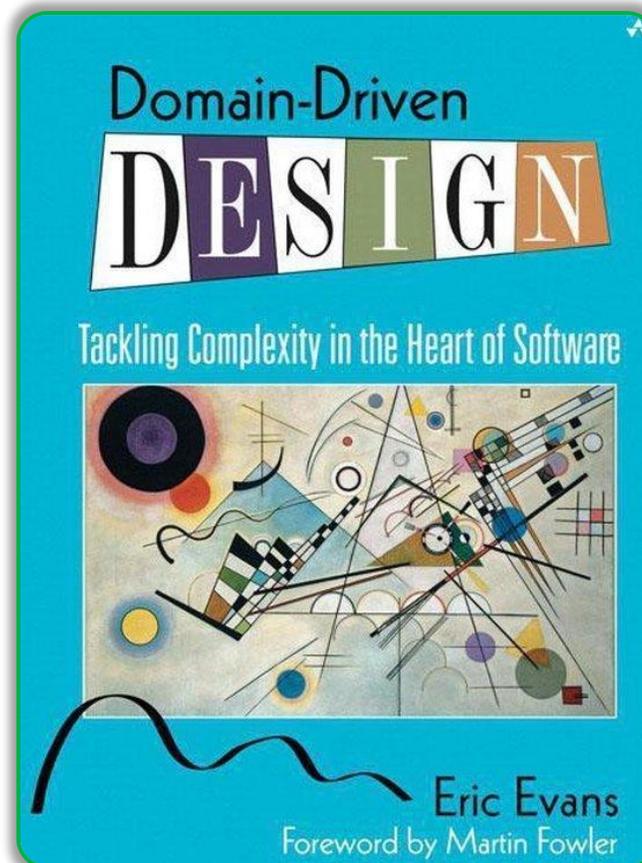
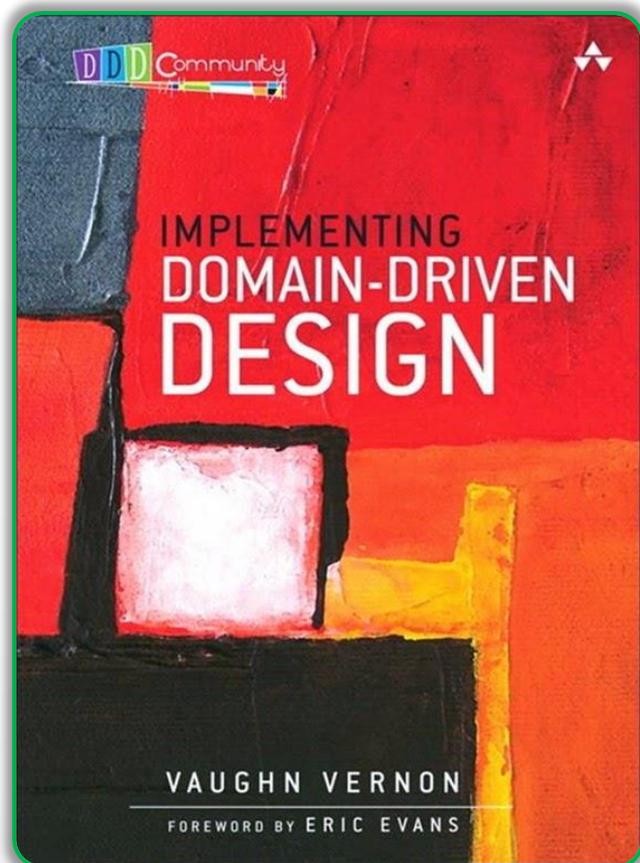
# Как работать с транзакциями?

- Транзакция должна быть в рамках одного Агрегата
- Если один Агрегат ссылается на другой Агрегат, они не должны изменяться в рамках одной и той же транзакции
- В реальной жизни достичь этого возможно далеко не всегда, по этому можно использовать передачу через контекст.
  - Причины нарушения правил:
    - Удобство пользовательского интерфейса
    - Отсутствие технических механизмов
    - Глобальные транзакции
    - Производительность запросов
- Принцип итоговой согласованности

# Производительность

- Производительность ниже (т.к. много слоев, а соответственно много вызовов)
- Используется большее количество объектов (Entity, DTO и т.д.), а соответственно различных конвертаций тоже больше
- Технические вещи сложнее реализуемы, т.к. мы оперируем не техническими деталями в DDD, а бизнес-сценариями

# Что почитать ?



# Итог

- Разобрали кратко зачем нужен DDD, какие плюсы/минусы есть у данного подхода, зачем нужен
- С чего начать проектирование
- Реализация тактического проектирования на конкретном примере
- Как структурировать код
- Стоит ли делать анимичную модель данных, почему не стоит использовать структурные теги
- Когда нужны доменные сервисы
- Как не зависеть от фреймворка и от протокола
- Где инициализировать application service
- Работа с транзакциями

# Спасибо за внимание!

Санкт-Петербург,  
Гельсингфорсская ул., 4  
+7 (812) 332 21 86  
[www.gs-labs.ru](http://www.gs-labs.ru)

## Контактная информация:

Telegram: @anisovd

VK: <https://vk.com/anisovd>

Mail: [dimaanisov24@gmail.com](mailto:dimaanisov24@gmail.com)