GS GROUP

DDD на практике

Анисов Дмитрий

GS LABS

- 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

GS LABS

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

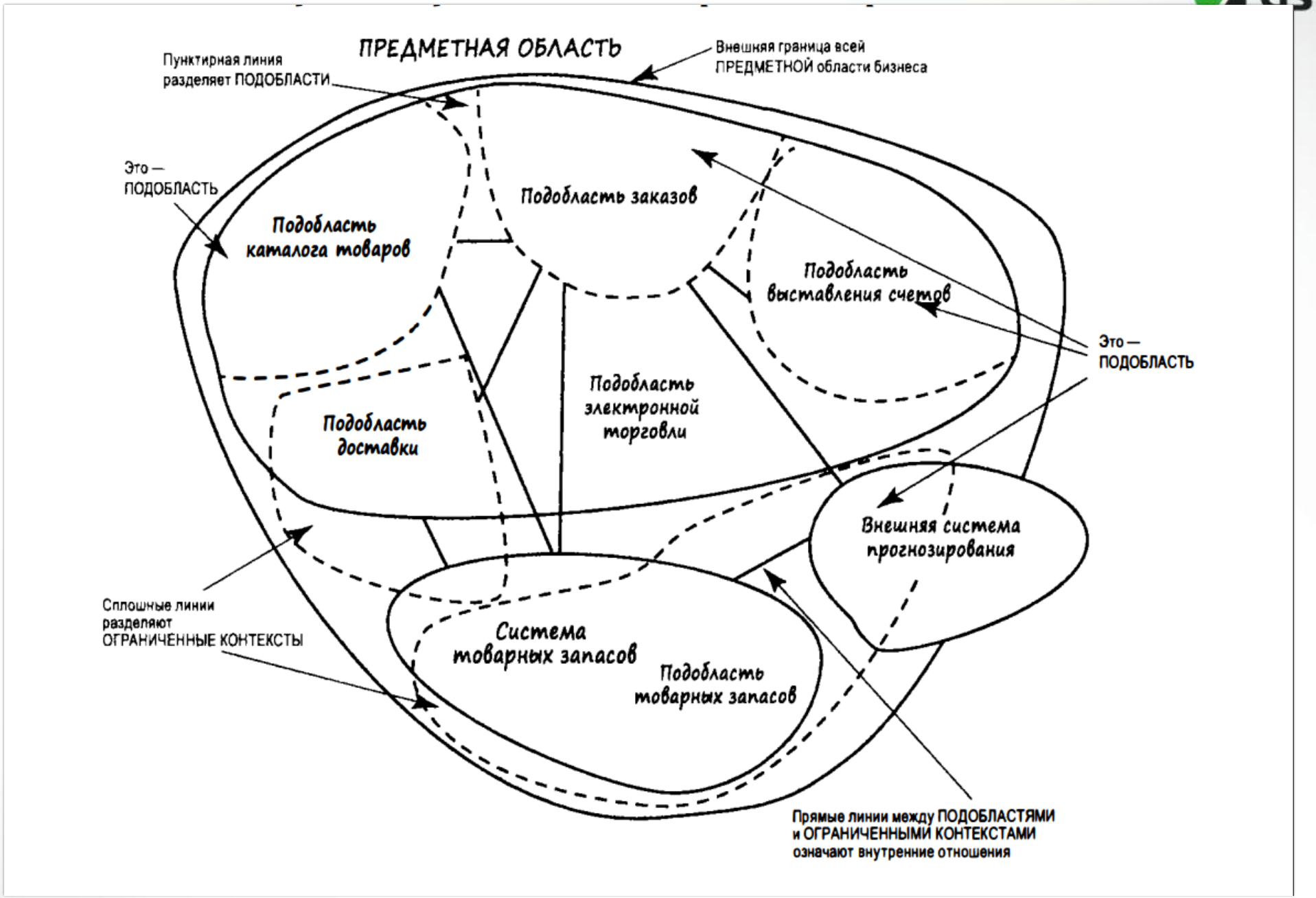
- Непонятно, как проектировать внутреннюю архитектуру сервиса
- Нет единого стандарта
- Разные архитектурные стили
- Где размещать бизнес-логику, валидацию, как работать с транзакциями, а также не зависеть от фреймворков и технологий
- Долго/дорого, надо много разбираться
- И т.д.

С чего начать?

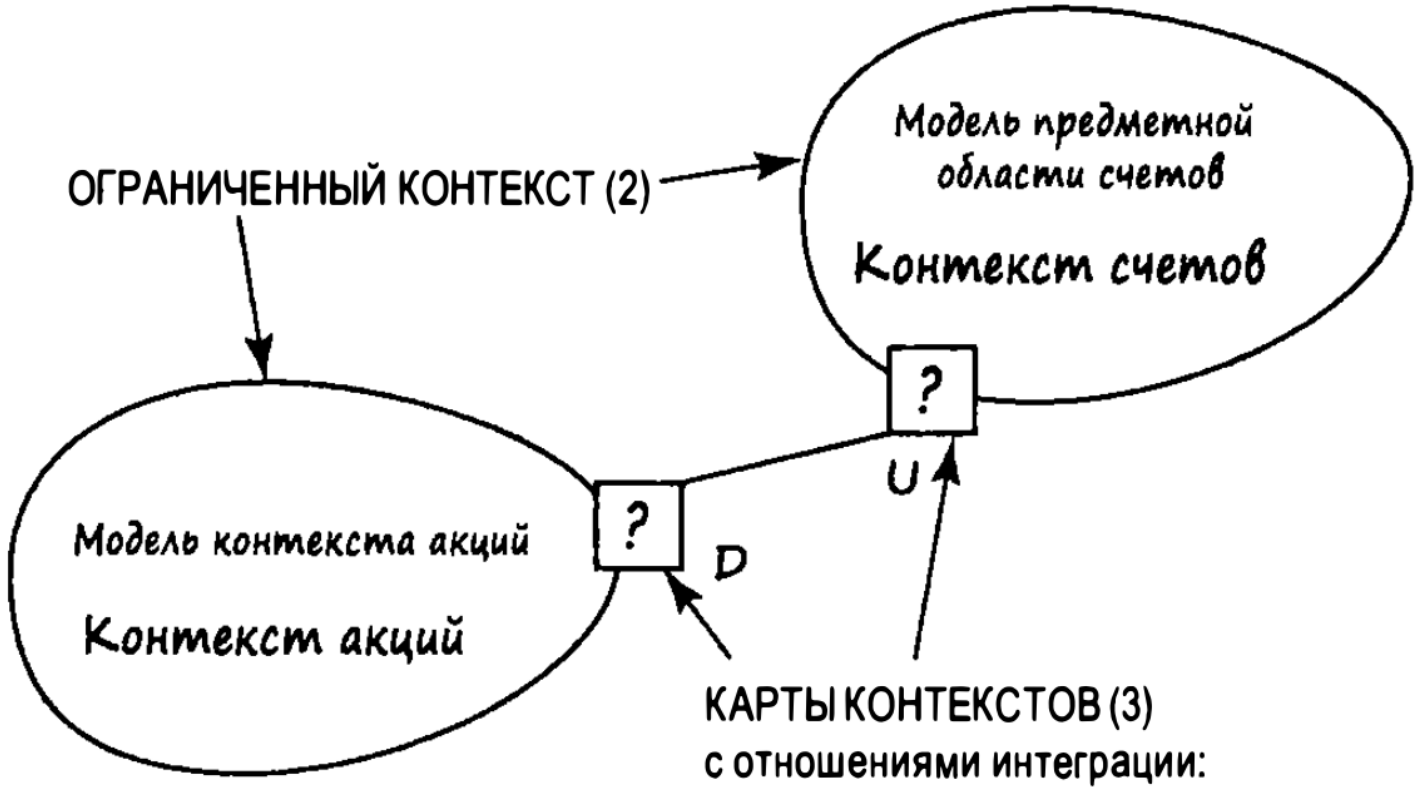
- Стратегическое проектирование
- Тактическое проектирование

Стратегическое проектирование

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


ПРЕДМЕТНАЯ ОБЛАСТЬ
Пунктирная линия разделяет ПОДОБЛАСТИ
Внешняя граница всей ПРЕДМЕТНОЙ области бизнеса
Это — ПОДОБЛАСТЬ
Подобласть каталога товаров
Подобласть заказов
Подобласть выставления счетов
Это — ПОДОБЛАСТЬ
Подобласть доставки
Подобласть электронной торговли
Внешняя система прогнозирования
Сплошные линии разделяют ОГРАНИЧЕННЫЕ КОНТЕКСТЫ
Система товарных запасов
Подобласть товарных запасов
Прямые линии между ПОДОБЛАСТЯМИ и ОГРАНИЧЕННЫМИ КОНТЕКСТАМИ означают внутренние отношения

Единый язык и ограниченный контекст



Тактическое проектирование

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

Шаблоны

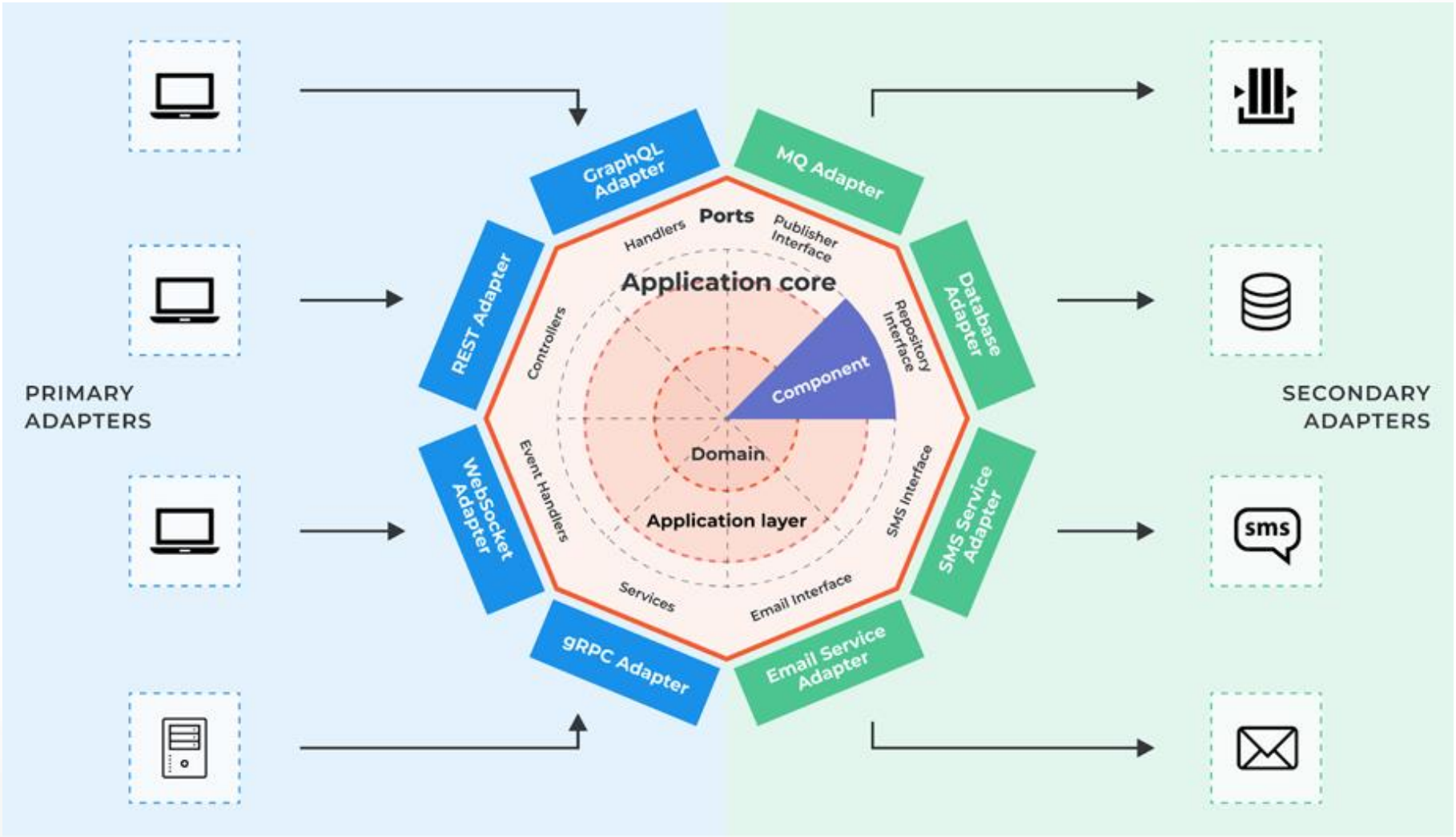
- Агрегат
- Сущности
- Объект-значение
- Хранилища
- Службы
- События предметной области
- Интеграция ограниченных контекстов

Выбор архитектуры

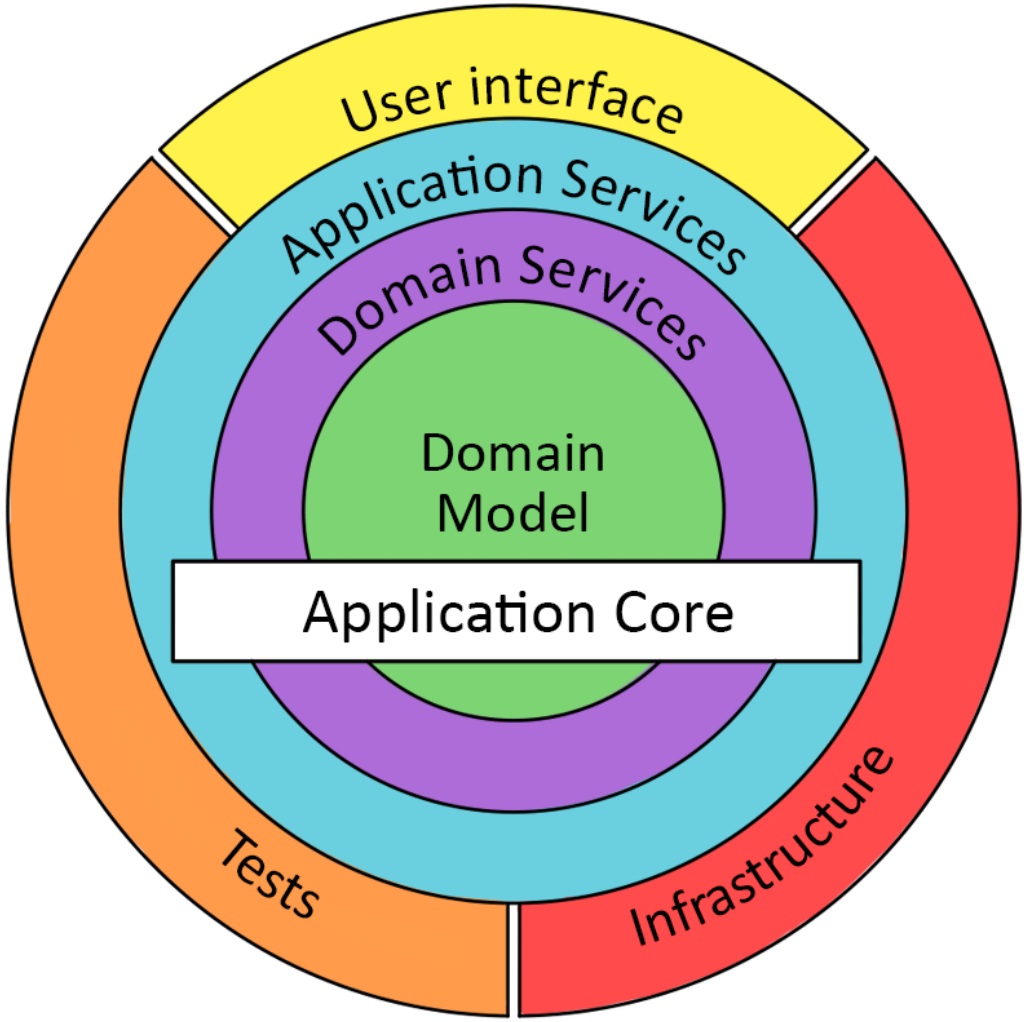
Насколько большие различия?

- Гексагональная архитектура
- Луковая архитектура
- Чистая архитектура

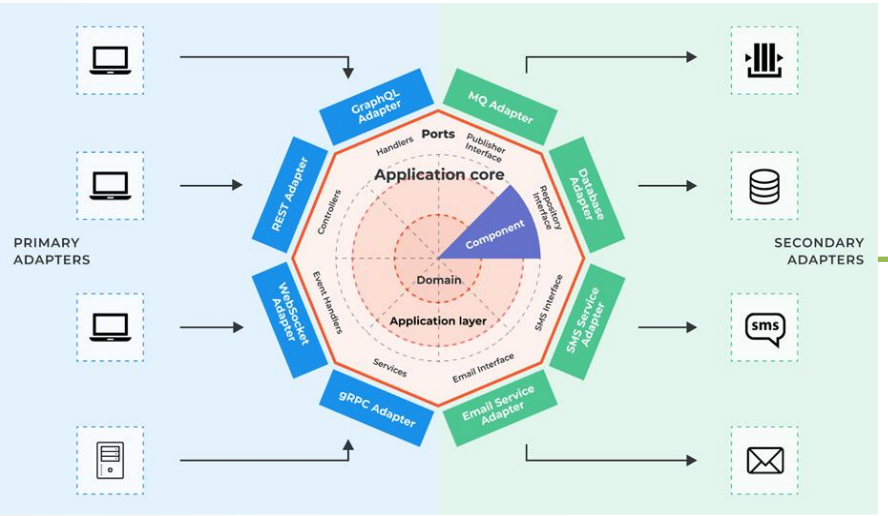
Гексагональная архитектура

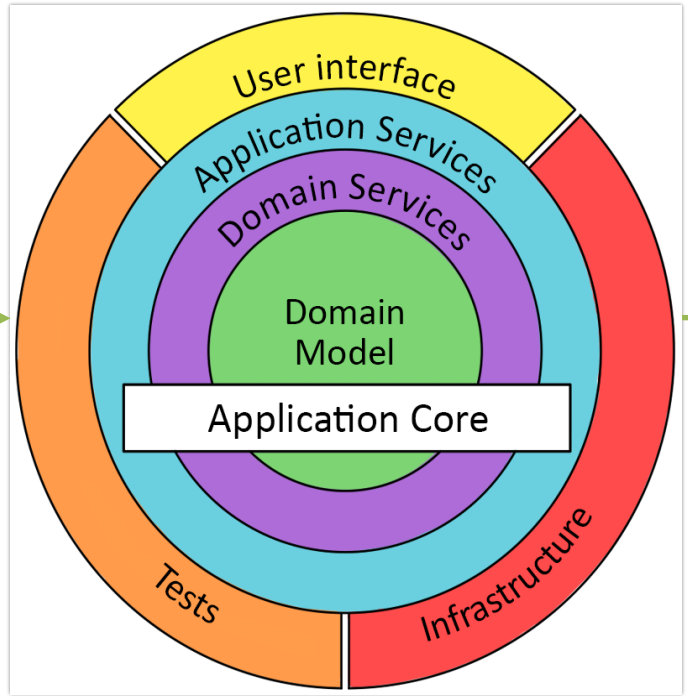


Луковая архитектура

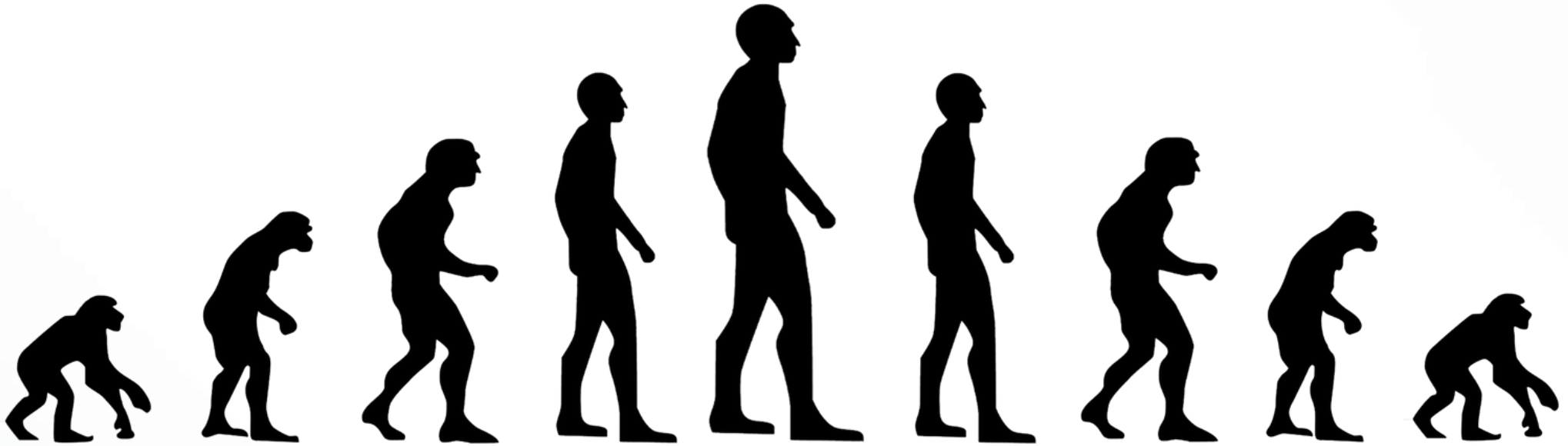


Чистая архитектура

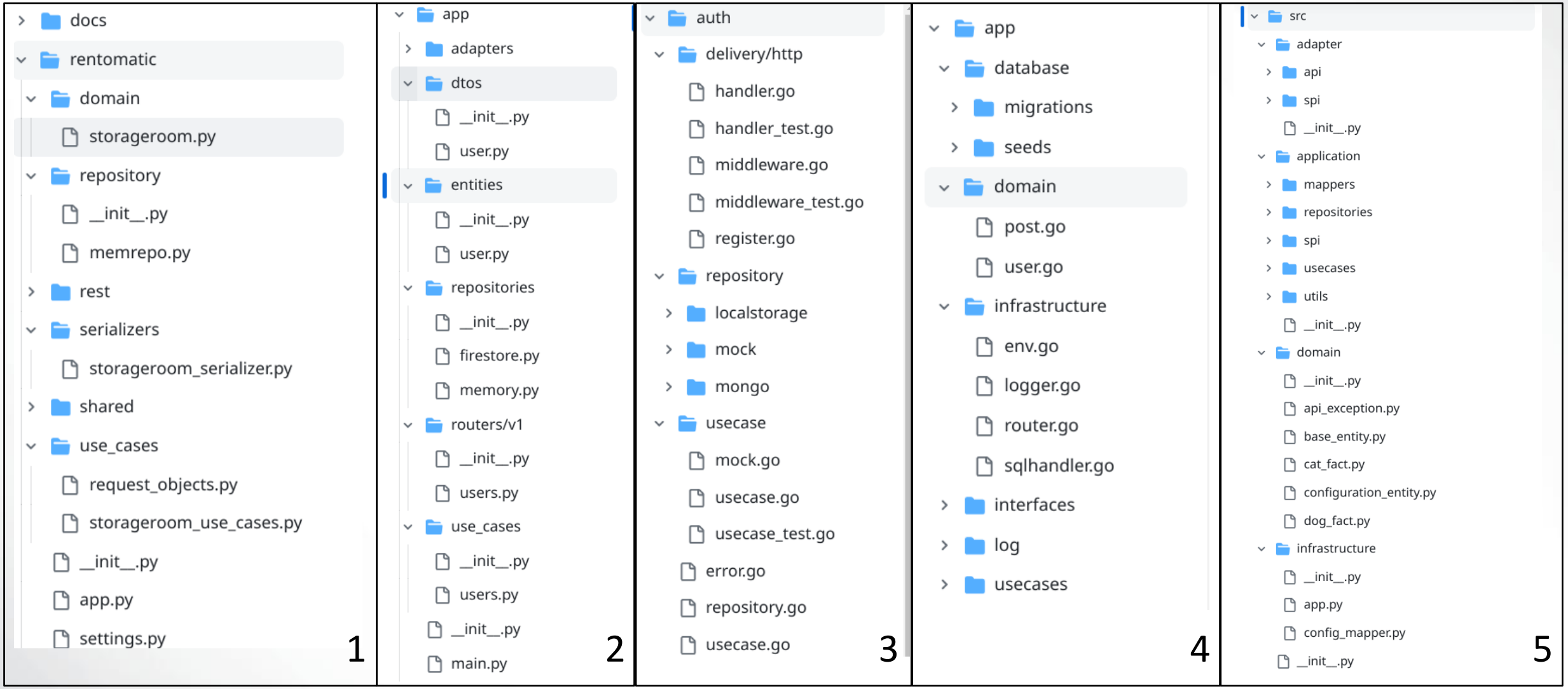




2005 2008 2012



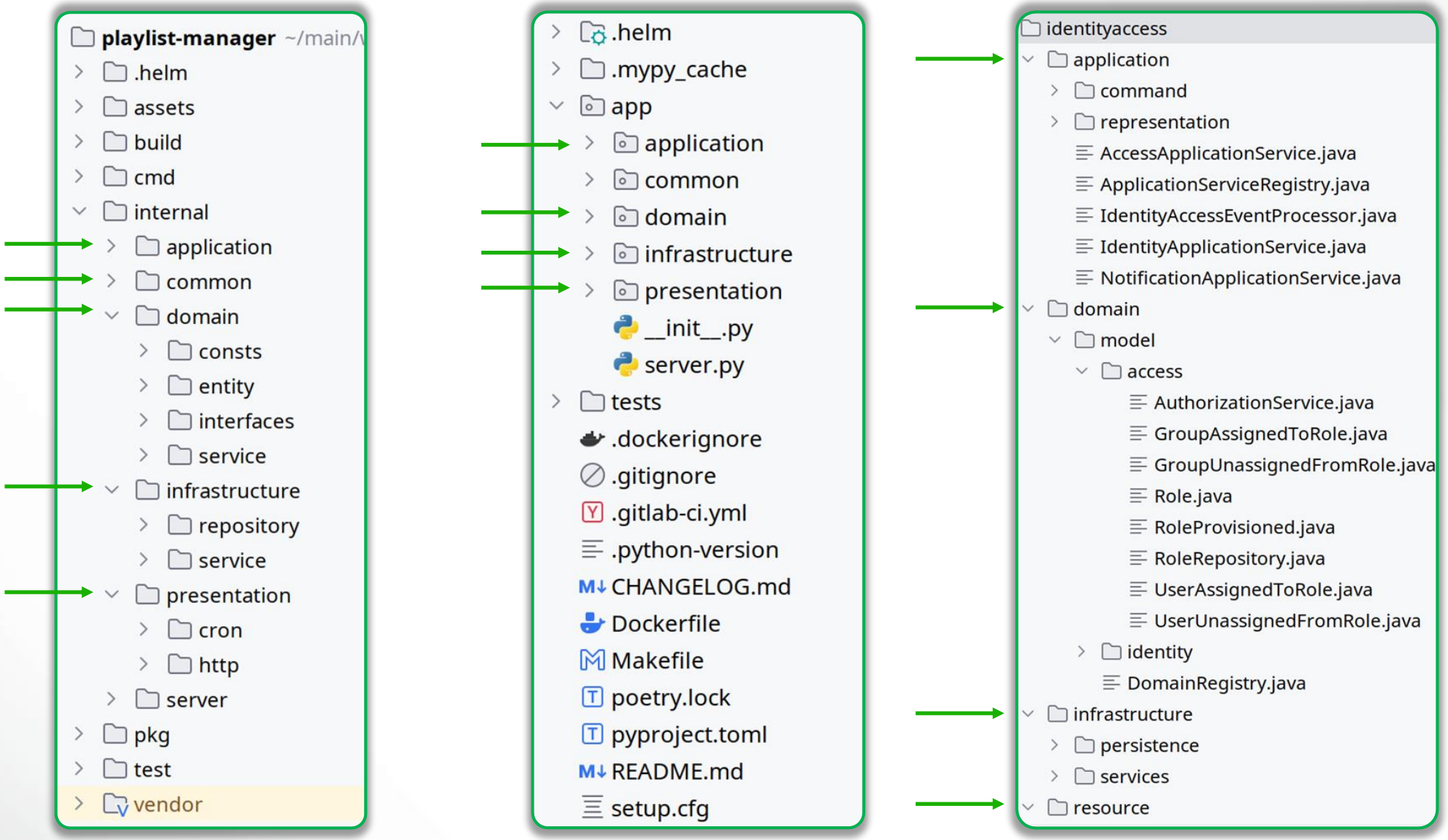
Чистая архитектура и её минусы



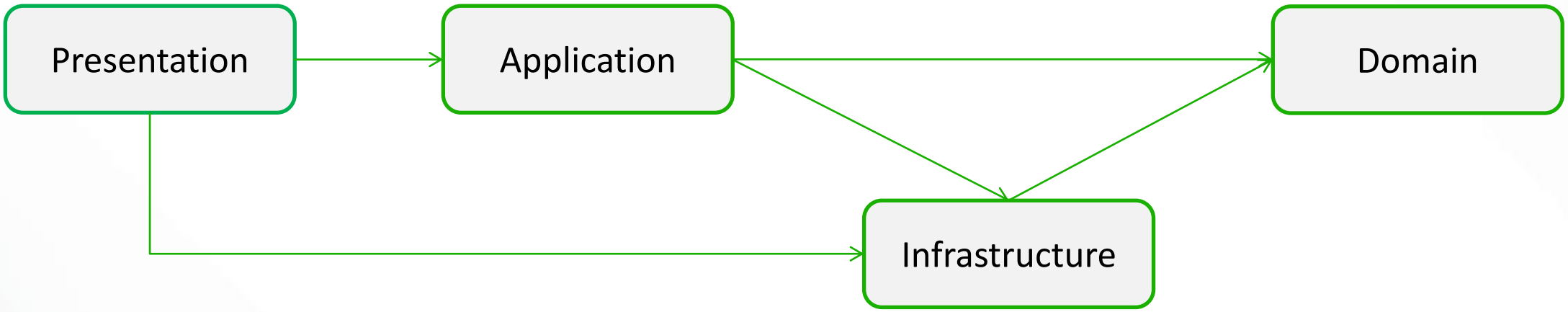
Зачем нужен единый подход?

- Единообразие
- Легкость внедрения новых разработчиков
- Практичность
- Удобство

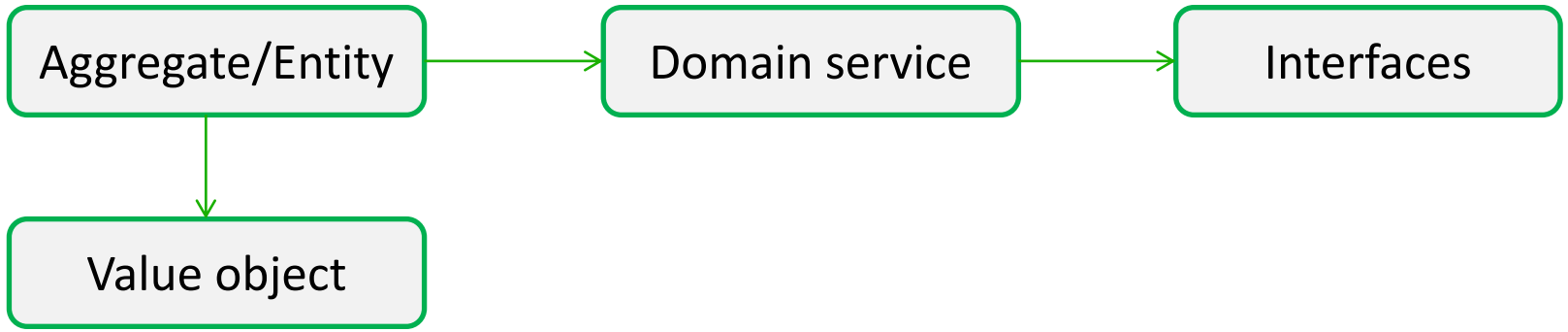
Структура проекта

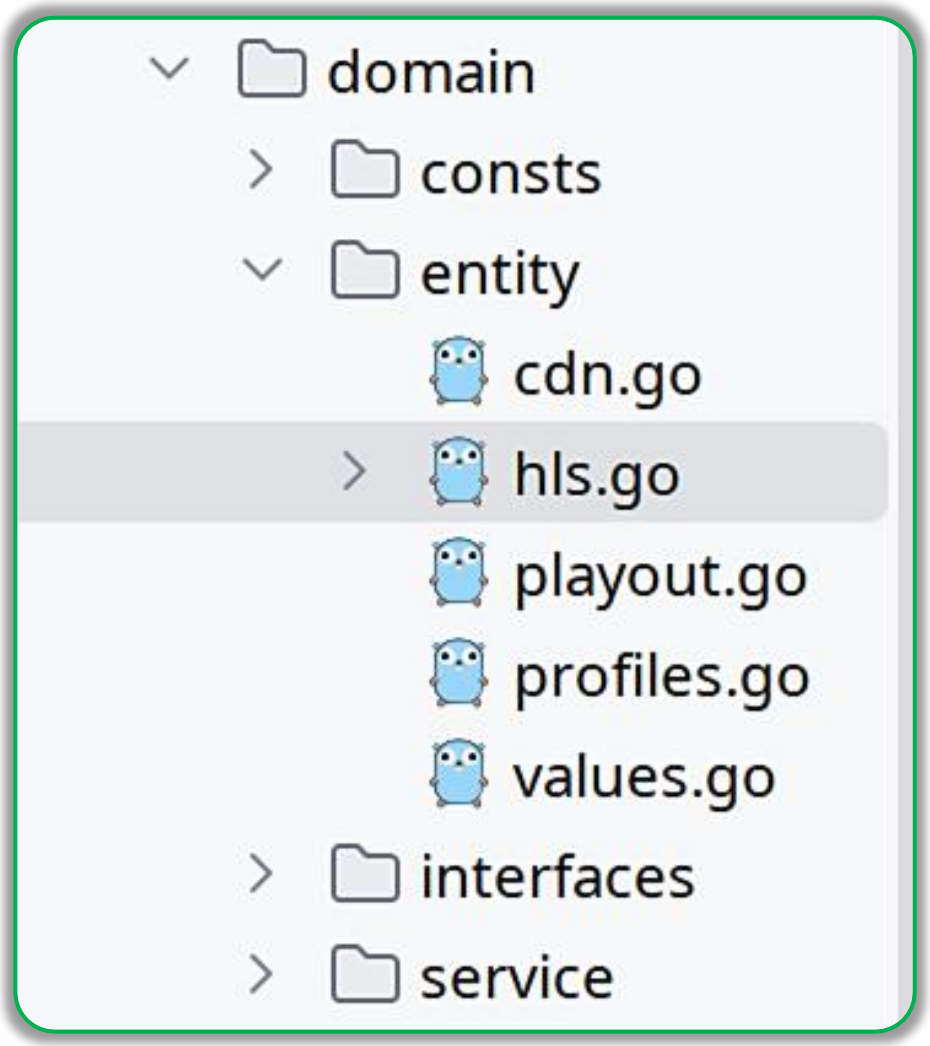


Упрощенная последовательность вызова



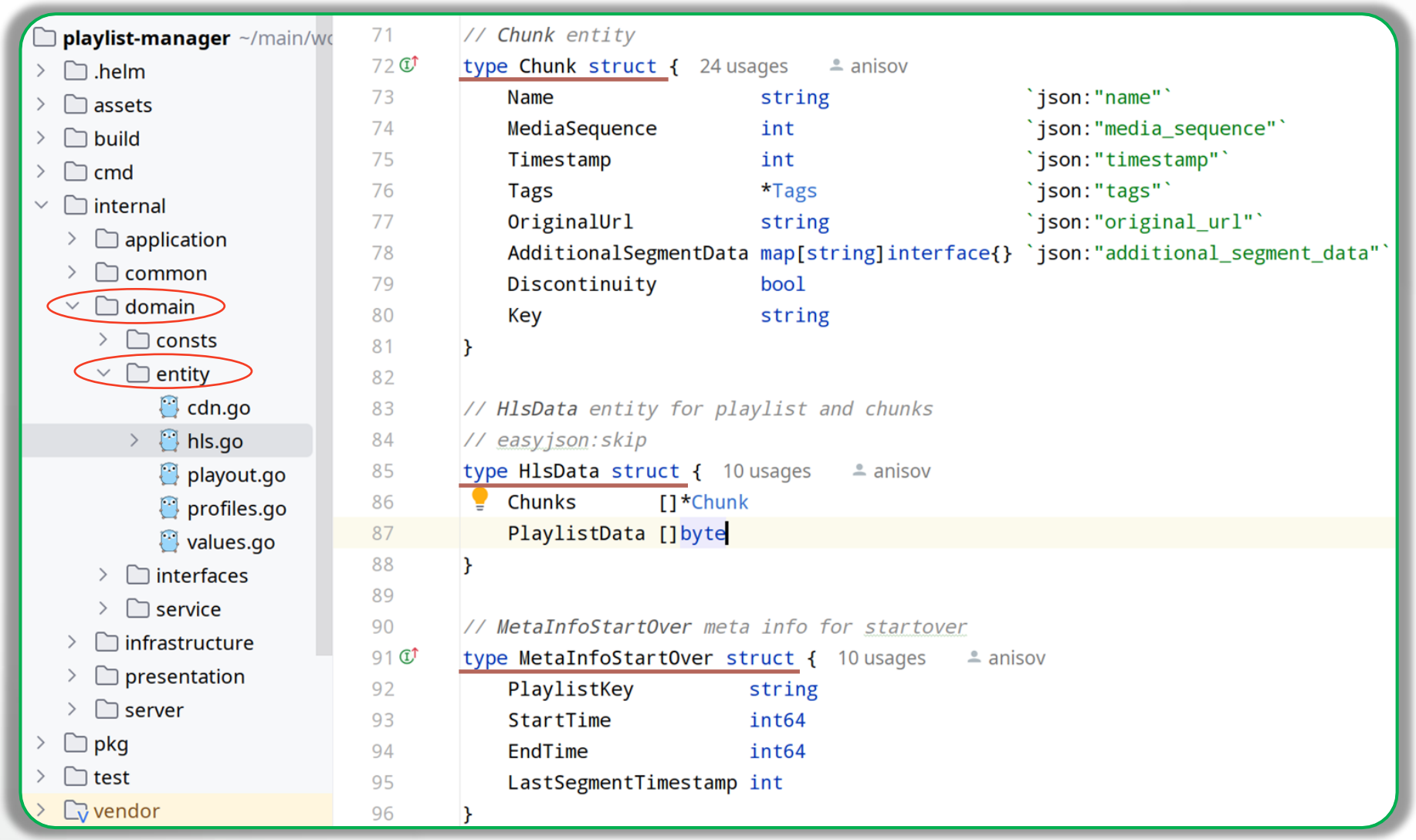
Domain Layer




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

Entity/Aggregates

Доменные объекты



Anemic Domain Model

Не/Анемичная модель данных

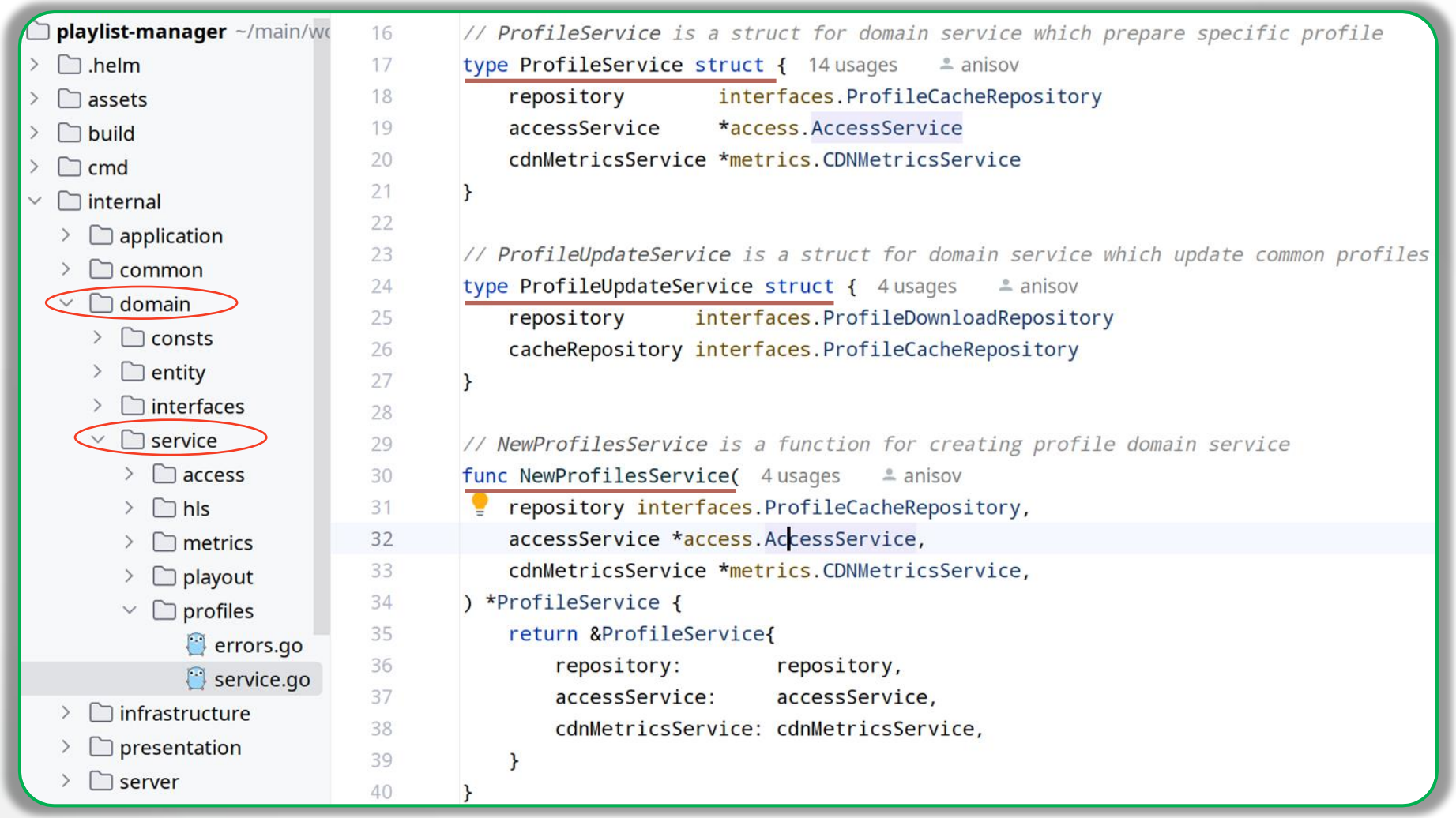
```
type UserMediaData struct {
...
}

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

Interfaces

Контракты, описывающие взаимодействие со сторонними системами/базами данных

Domain Service

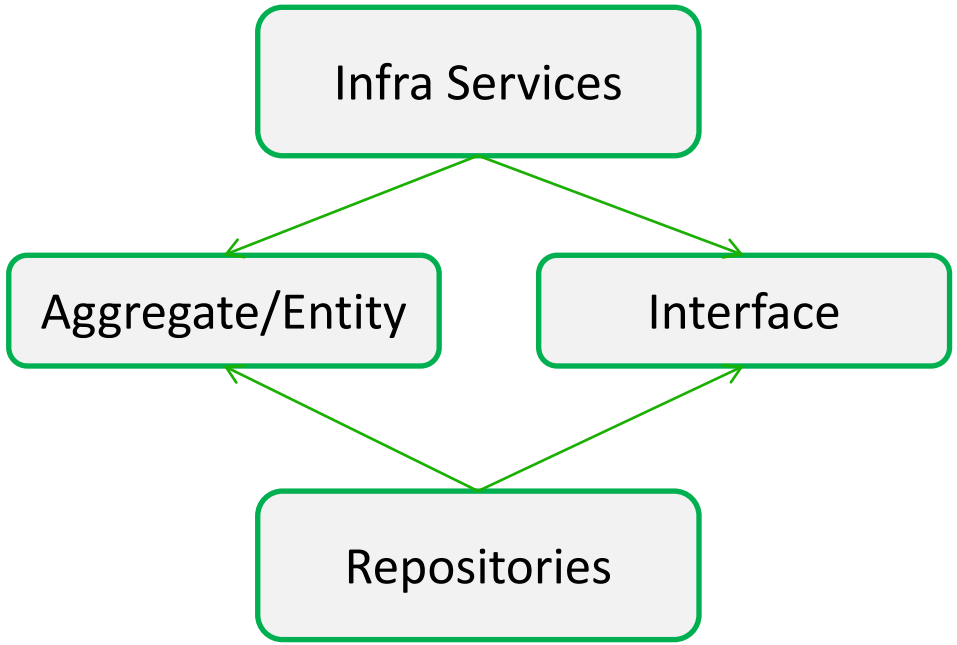
Доменные сервисы



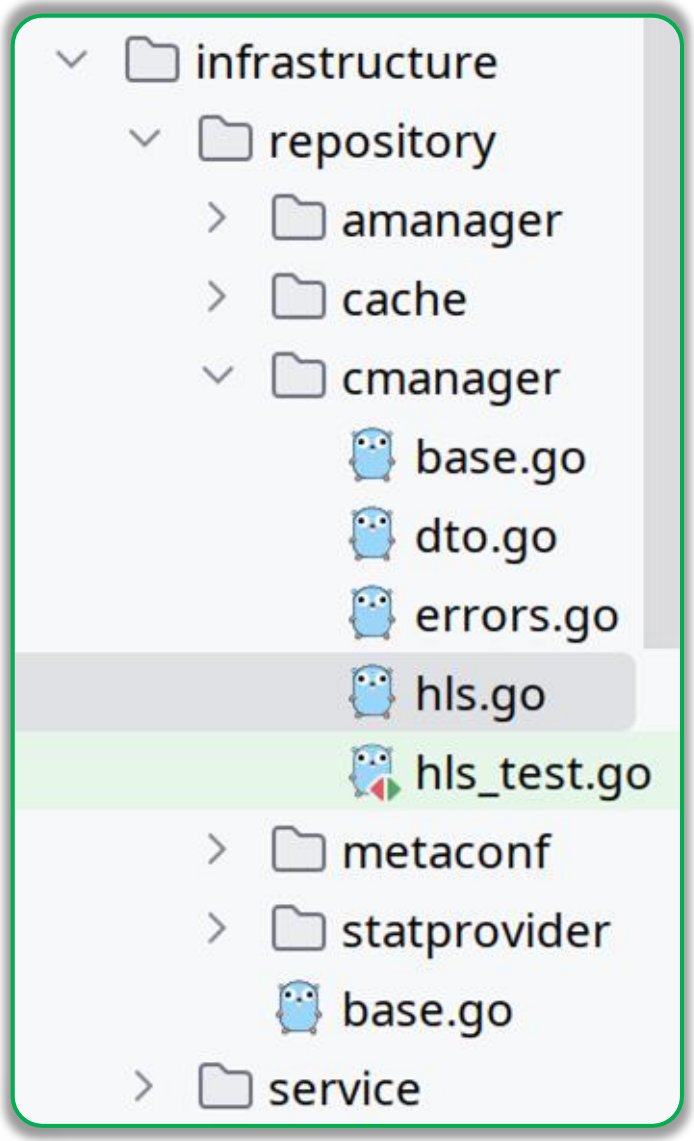
Второй пример

```
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

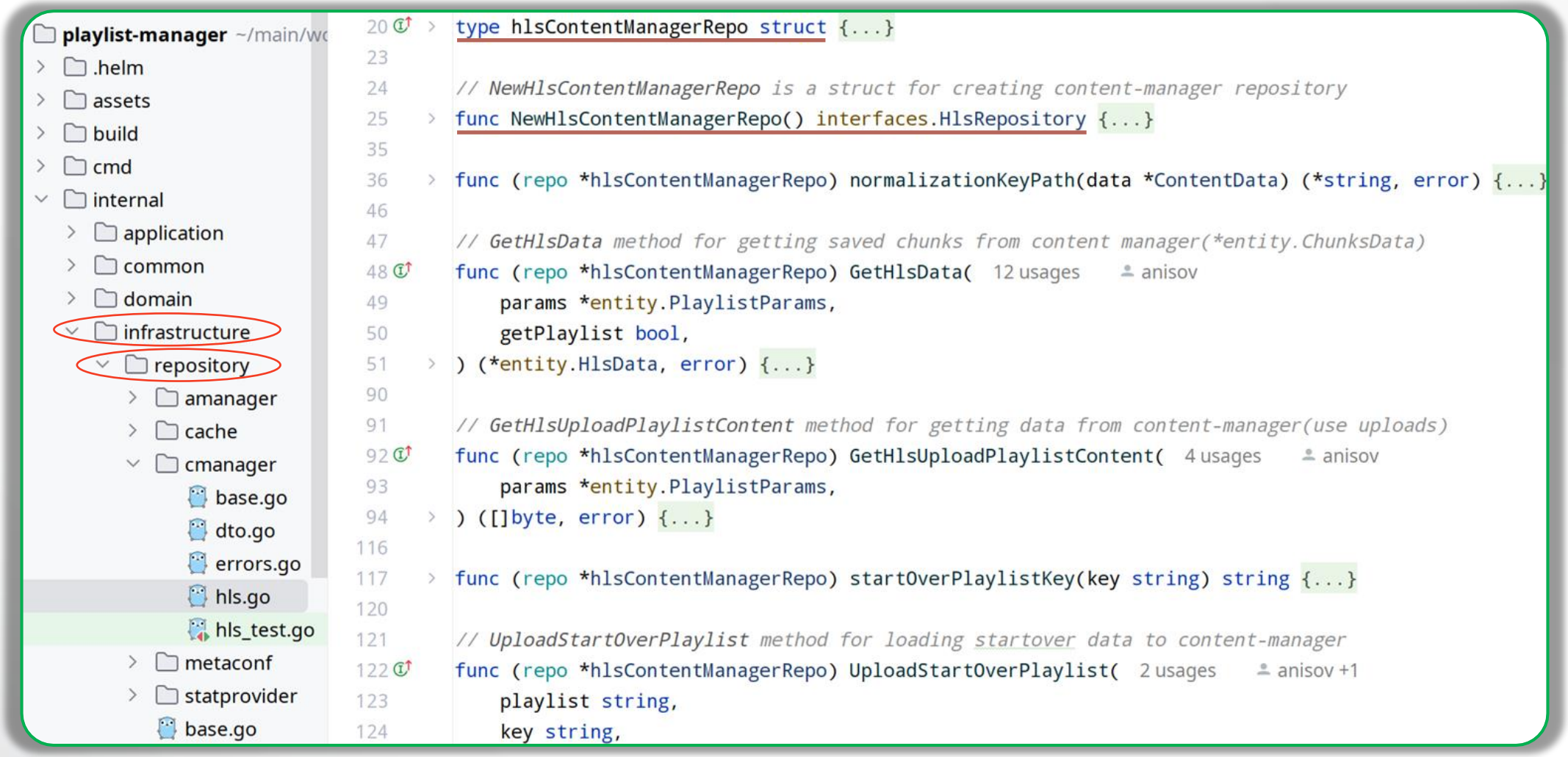


Структура слоя

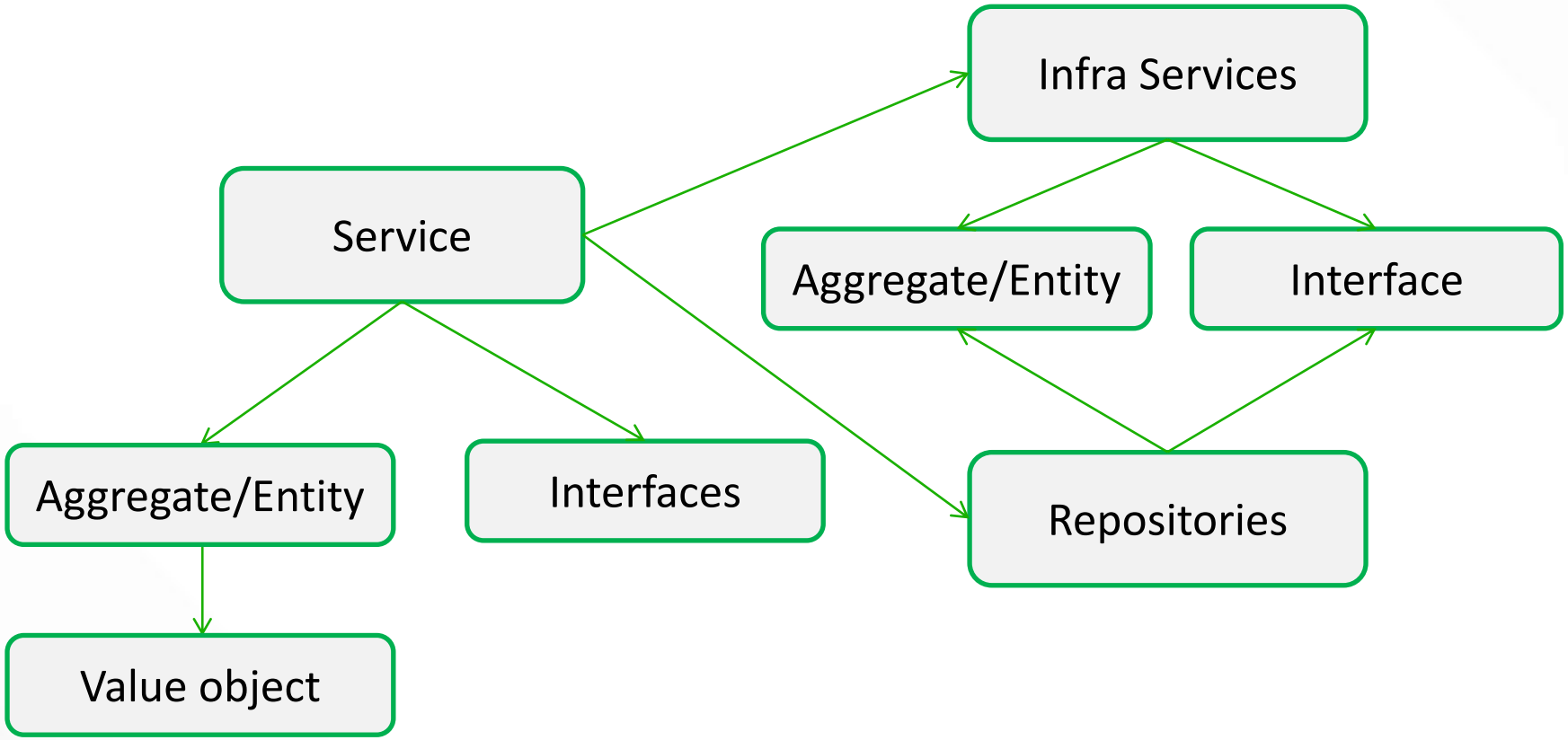


Implementation

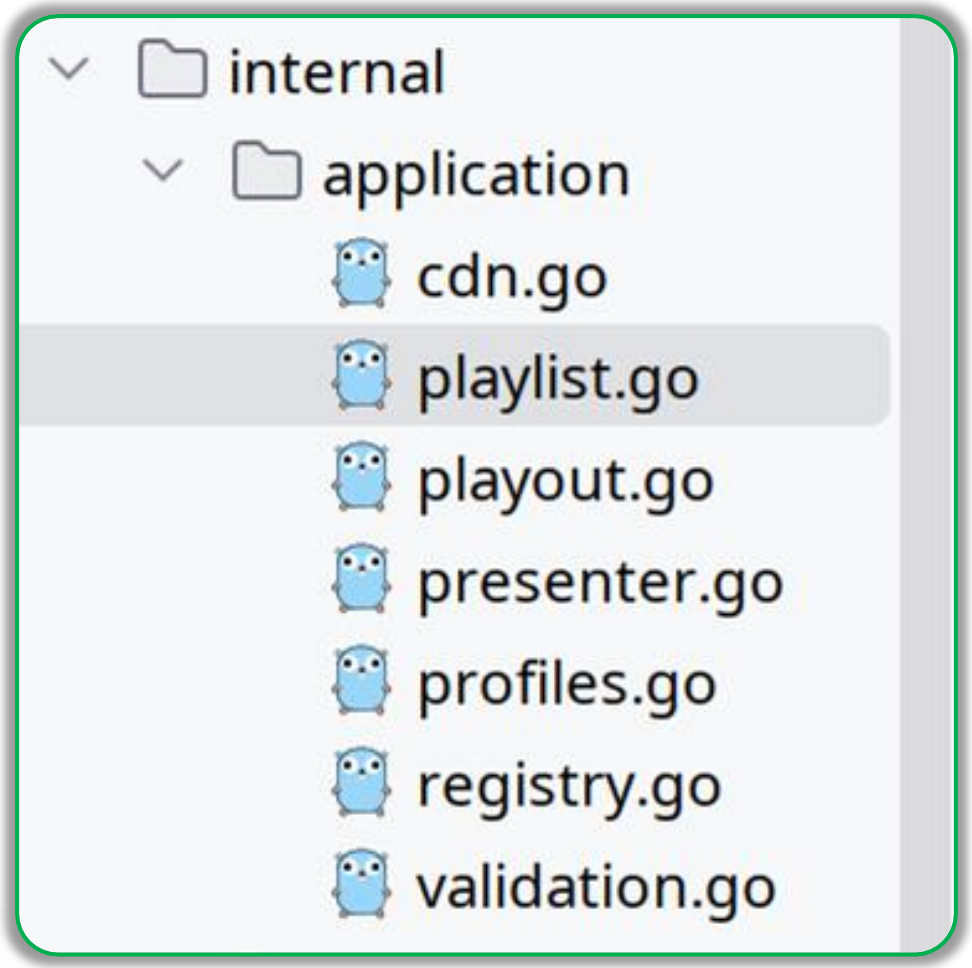
Реализация интерфейсов



3. Application Layer



Структура слоя



Application Service

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

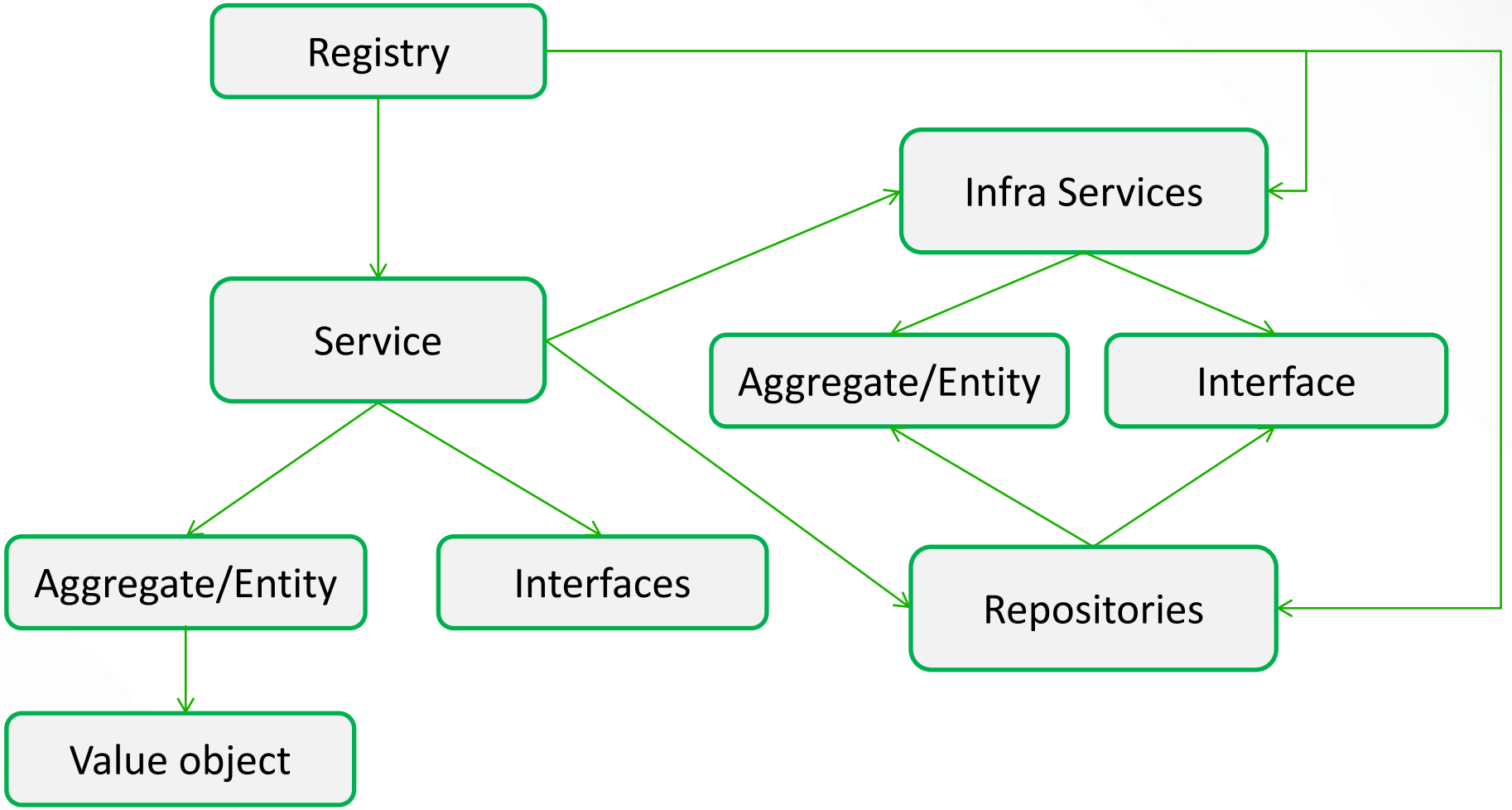
Второй пример

```
// 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
Infra Services
Service
Aggregate/Entity
Interface
Aggregate/Entity
Interfaces
Repositories
Value object

Registry

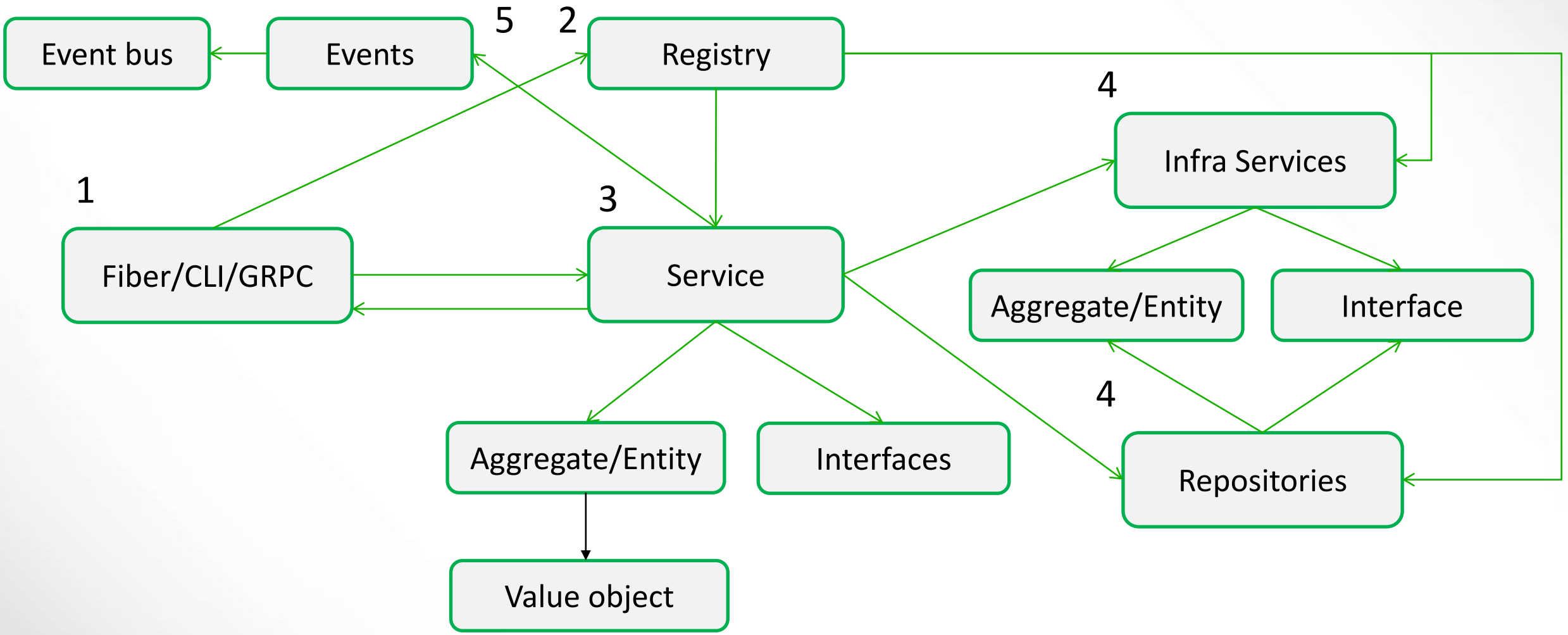
Инициализация и внедрение зависимостей

Второй пример

```
var (
    MediaPlaylistApplicationService  *VariantVirtualPlaylistService  2 usages
    MasterPlaylistApplicationService *MasterVirtualPlaylistService  2 usages
)

func init() {  anisov *
    var (
        encryptInfraService         = infraService.NewEncryptService()
        cDNPlaylistInfraService     = infraService.NewCDNPlaylistService()
        cacheRepository             = repository.NewCacheRepository()
        vMasterPlaylistService      = domainService.VirtualMasterPlaylistService{}
        vMediaPlaylistService       = domainService.VirtualMediaPlaylistService{}
        operatorInfoInfraRepository = repository.NewOperatorSettingsService()
        sequenceRepository          = repository.NewSequenceRepository()
    )
    if common.Settings.RandomSequence {
        sequenceRepository = repository.NewRandomSequenceRepository()
    }

    MasterPlaylistApplicationService = NewMasterVirtualPlaylistService(
        cDNPlaylistInfraService,
        encryptInfraService,
        cacheRepository,
        operatorInfoInfraRepository,
        vMasterPlaylistService,
    )
    MediaPlaylistApplicationService = NewVariantVirtualPlaylistService(
```

4. Presentation

Слой представления


Event bus
Events
5
2
Registry
4
Infra Services
1
3
Fiber/CLI/GRPC
Service
Aggregate/Entity
Interface
4
Aggregate/Entity
Interfaces
Repositories
Value object

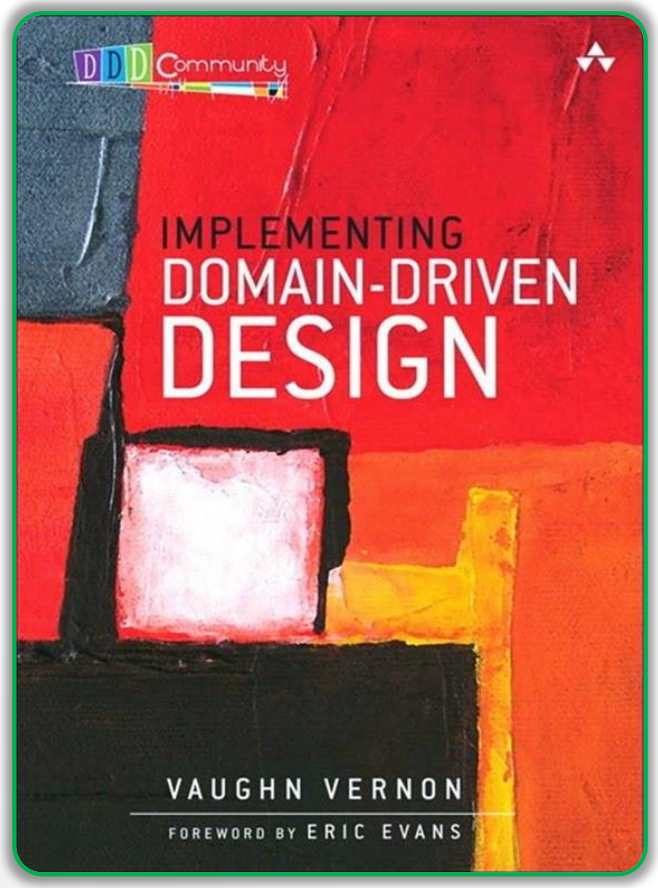
Как работать с транзакциями?

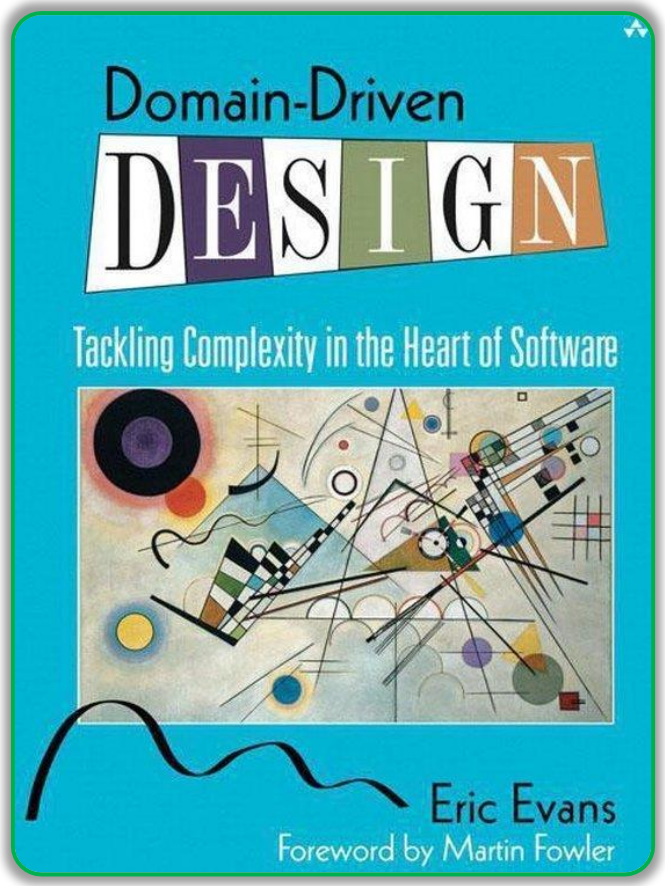
- Транзакция должна быть в рамках одного Агрегата
- Если один Агрегат ссылается на другой Агрегат, они не должны изменяться в рамках одной и той же транзакции
- В реальной жизни достичь этого возможно далеко не всегда, по этому можно использовать передачу через контекст.
  - Причины нарушения правил: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

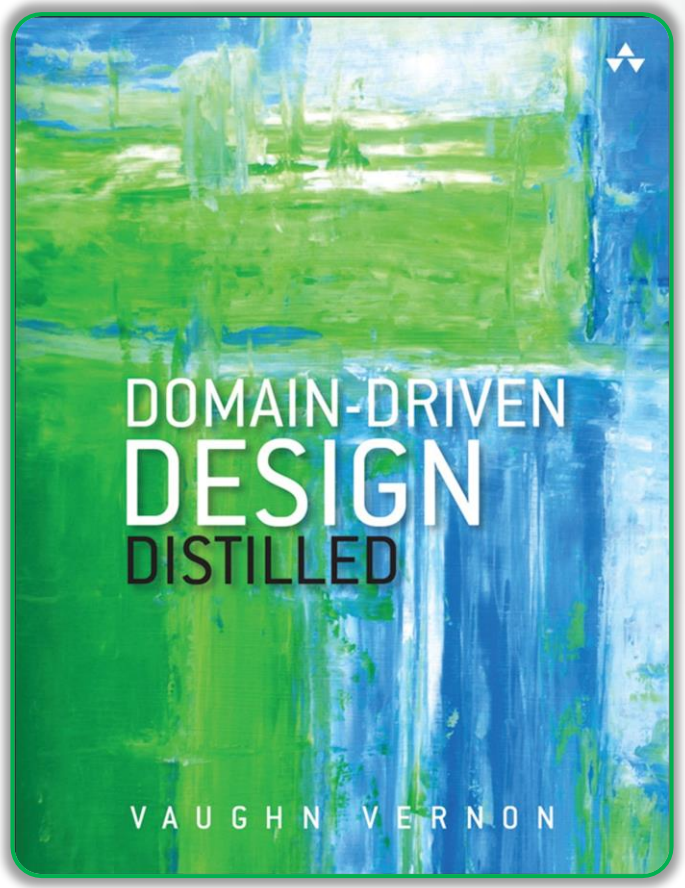
Производительность

- 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

GS LABS

Спасибо за внимание!

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

Контактная информация:

Telegram: @anisovd
VK: https://vk.com/anisovd
Mail: dimaanisov24@gmail.com