Как готовить Domain Driven Design

Анисов Дмитрий

- Ведущий Backend-разработчик в компании GS Labs
- Основные языки разработки - Python/Go
- Люблю и занимаюсь Devops и администрированием

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

Что это такое?

- Это подход к разработке программного обеспечения
- Данный подход объединяет экспертов в предметной области и разработчиков
- Позволяет писать правильно спроектированное программное обеспечение

Когда стоит использовать

- DDD используется в наиболее важных областях бизнеса
- Для упрощения работы с предметной областью
- При необходимости повышение качества разрабатываемого ПО, которое сэкономит время и облегчит поддержку данного проекта в будущем

О чём пойдёт речь. Цель доклада

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

Проблема. Боль DDD

- Не понятно как проектировать внутреннюю архитектуру сервиса
- Нет единого стандарта
- Разные архитектурные стили
- Где размещать бизнес логику, валидацию, как работать с транзакциями, а так же не зависеть от фреймвроков и технологий
- Долго/дорого, надо много разбираться
- И много чего ещё

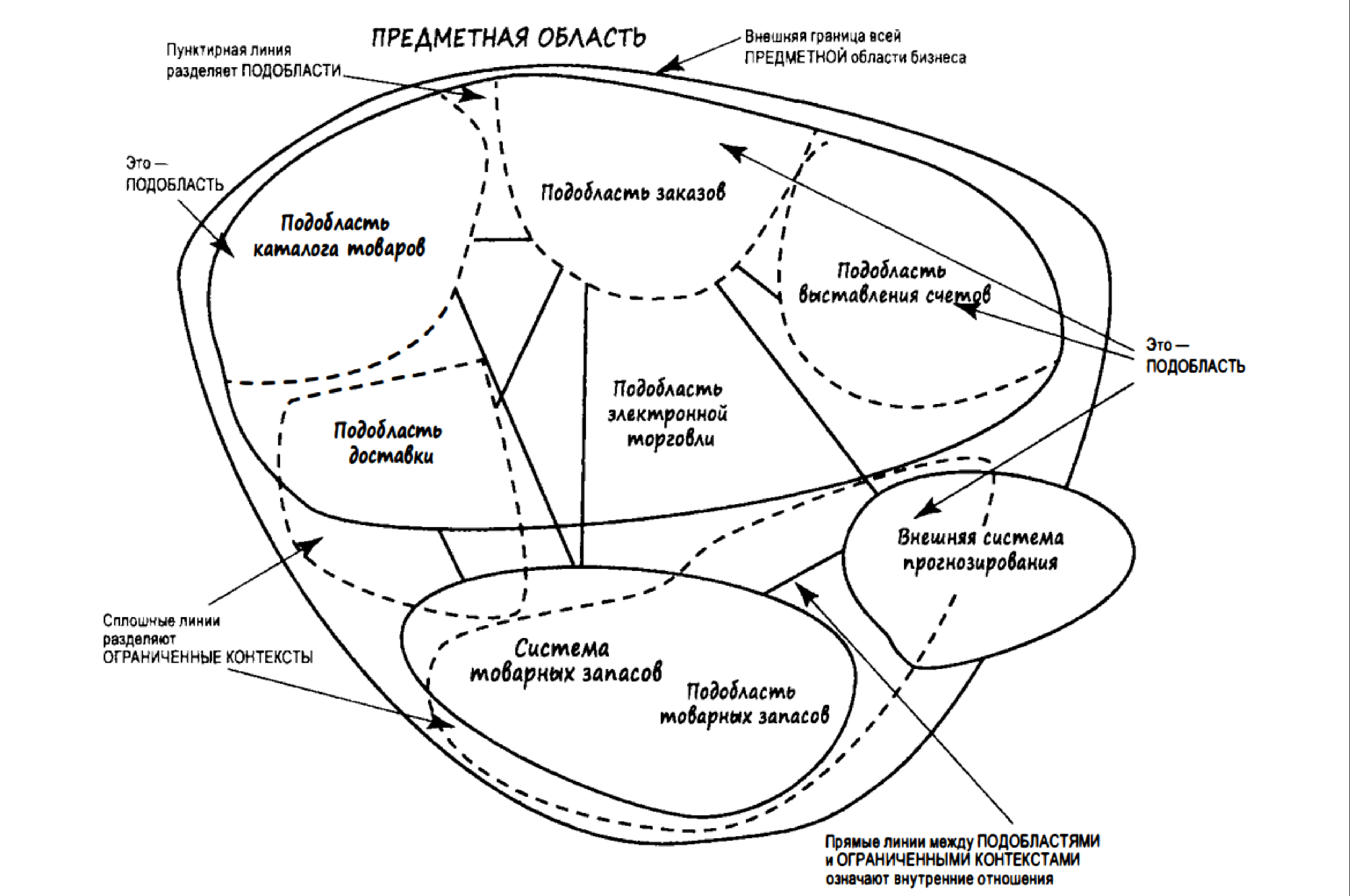
С чего начать?

- Стратегическое проектирование
- Тактическое проектирование

Стратегическое проектирование

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

GS LABS



Единый язык и Ограниченный контекст

ОГРАНИЧЕННЫЙ КОНТЕКСТ (2)

Модель предметной области счетов

Контекст счетов

?

U

Модель контекста акций

Контекст акций

?

D

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 и на сколько большие различия

- Гексагональная архитектура
- Луковая архитектура
- Чистая архитектура

Чистая архитектура

Чистая архитектура

Устройства
Web
Контроллеры
Сценарии
Сущности
Шлюзы
Представления
БД
UI
Внешние интерфейсы

Бизнес-правила предприятия
Бизнес-правила приложения
Интерфейс-Адаптеры
Фреймворки и Драйверы

Представление
Сценарий Выходной порт
Сценарий Исполнитель
Контроллер
Сценарий Входной порт

Гексагональная архитектура

PRIMARY ADAPTERS

GraphQL Adapter

REST Adapter

WebSocket Adapter

gRPC Adapter

Ports

Handlers

Publisher Interface

Application core

Controllers

Repository Interface

Component

Domain

Event Handlers

Application layer

SMS Interface

Services

Email Interface

MQ Adapter

Database Adapter

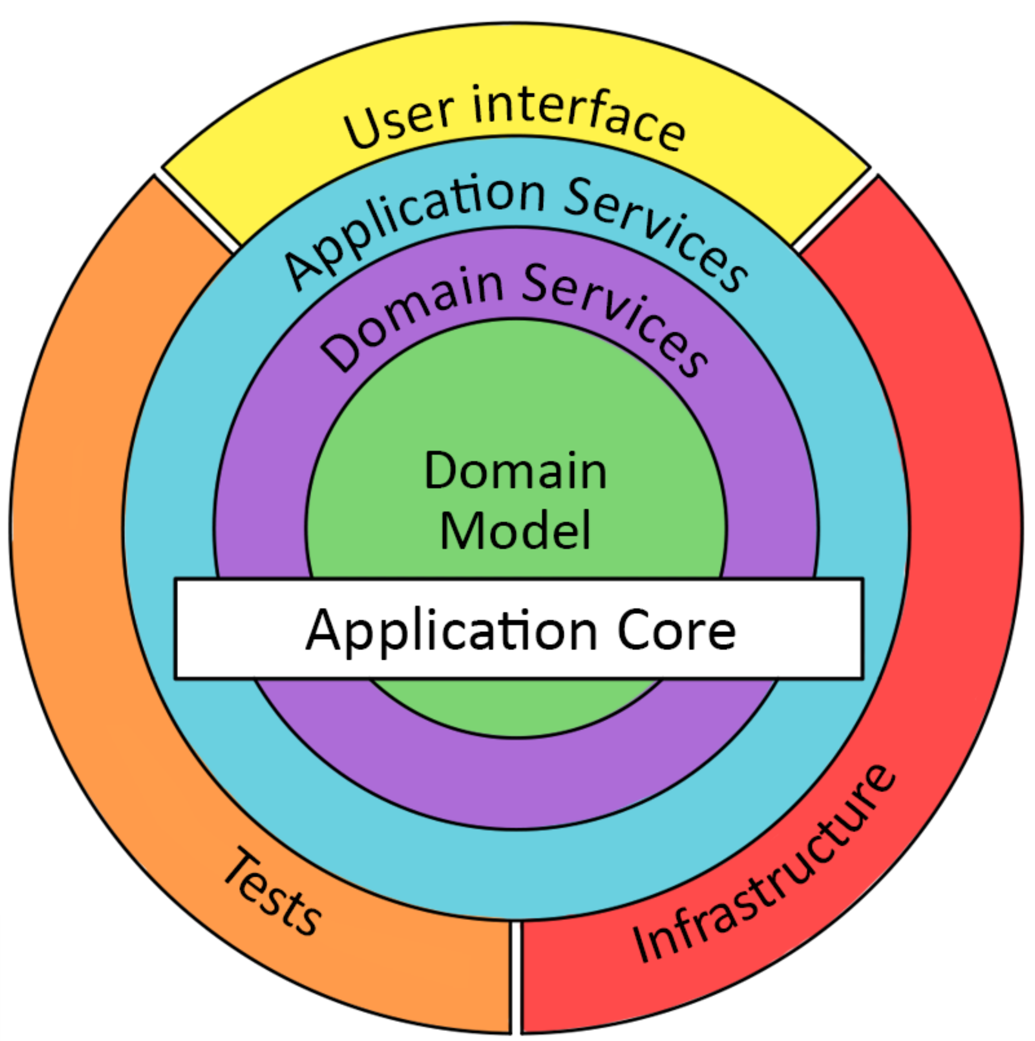
SMS Service Adapter

Email Service Adapter

SECONDARY ADAPTERS

sms

Луковая Архитектура



PRIMARY ADAPTERS

GraphQL Adapter
REST Adapter
WebSocket Adapter
gRPC Adapter
MQ Adapter
Database Adapter
SMS Service Adapter
Email Service Adapter
Ports
Handlers
Publisher Interface
Application core
Controllers
Repository Interface
Component
Domain
Event Handlers
SMS Interface
Application layer
Services
Email Interface
sms

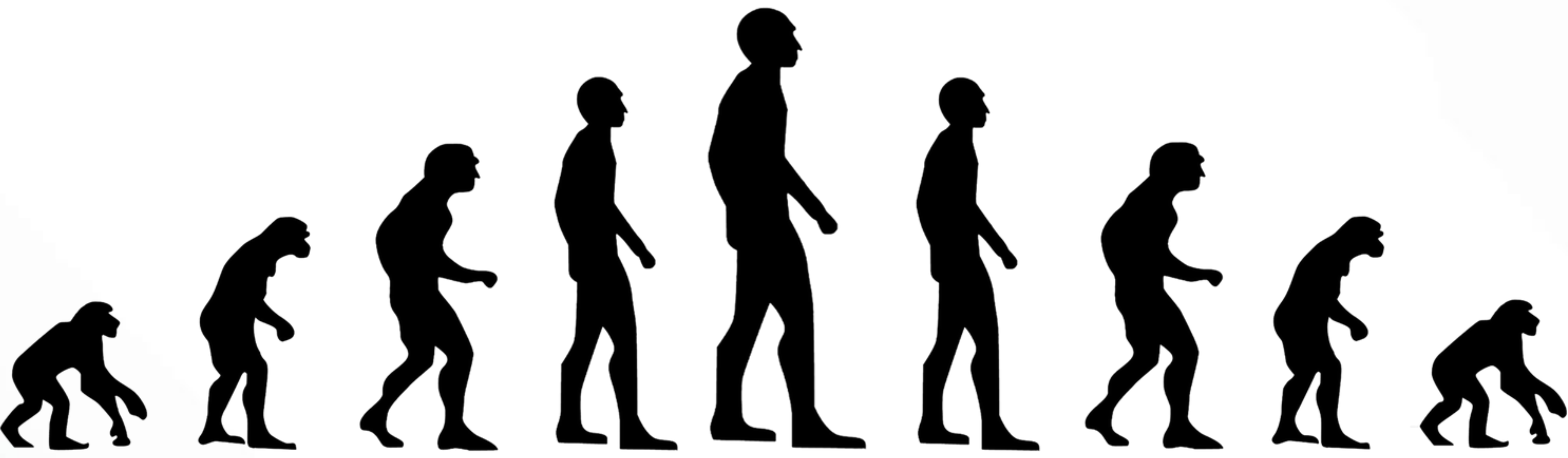
SECONDARY ADAPTERS

User interface
Application Services
Domain Services
Domain Model
Application Core
Tests
Infrastructure

Чистая архитектура

Устройства
Web
Контроллеры
Сценарии
Сущности
Шлюзы
Представления
БД
UI
Внешние интерфейсы
Бизнес-правила предприятия
Бизнес-правила приложения
Интерфейс-Адаптеры
Фреймворки и Драйверы
Представление
Сценарий Выходной порт
Сценарий Исполнитель
Контроллер
Сценарий Входной порт

GS LABS



Чистая архитектура и её минусы

1

- docs
- rentomatic
  - domain
    - storageroom.py
  - repository
    - __init__.py
    - memrepo.py
  - rest
  - serializers
    - storageroom_serializer.py
  - shared
  - use_cases
    - request_objects.py
    - storageroom_use_cases.py
  - __init__.py
  - app.py
  - settings.py

2

- app
  - adapters
  - dtos
    - __init__.py
    - user.py
  - entities
    - __init__.py
    - user.py
  - repositories
    - __init__.py
    - firestore.py
    - memory.py
  - routers/v1
    - __init__.py
    - users.py
  - use_cases
    - __init__.py
    - users.py
  - __init__.py
  - main.py

3

- auth
  - delivery/http
    - handler.go
    - handler_test.go
    - middleware.go
    - middleware_test.go
    - register.go
  - repository
    - localstorage
    - mock
    - mongo
  - usecase
    - mock.go
    - usecase.go
    - usecase_test.go
  - error.go
  - repository.go
  - usecase.go

4

- app
  - database
    - migrations
    - seeds
  - domain
    - post.go
    - user.go
  - infrastructure
    - env.go
    - logger.go
    - router.go
    - sqlhandler.go
  - interfaces
  - log
  - usecases

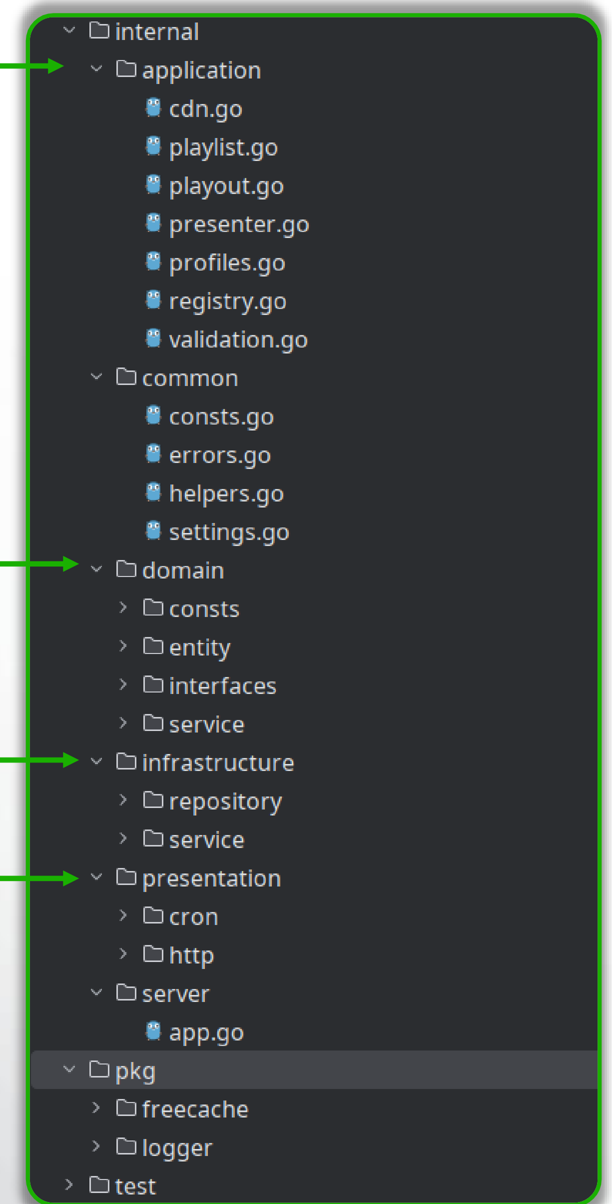
5

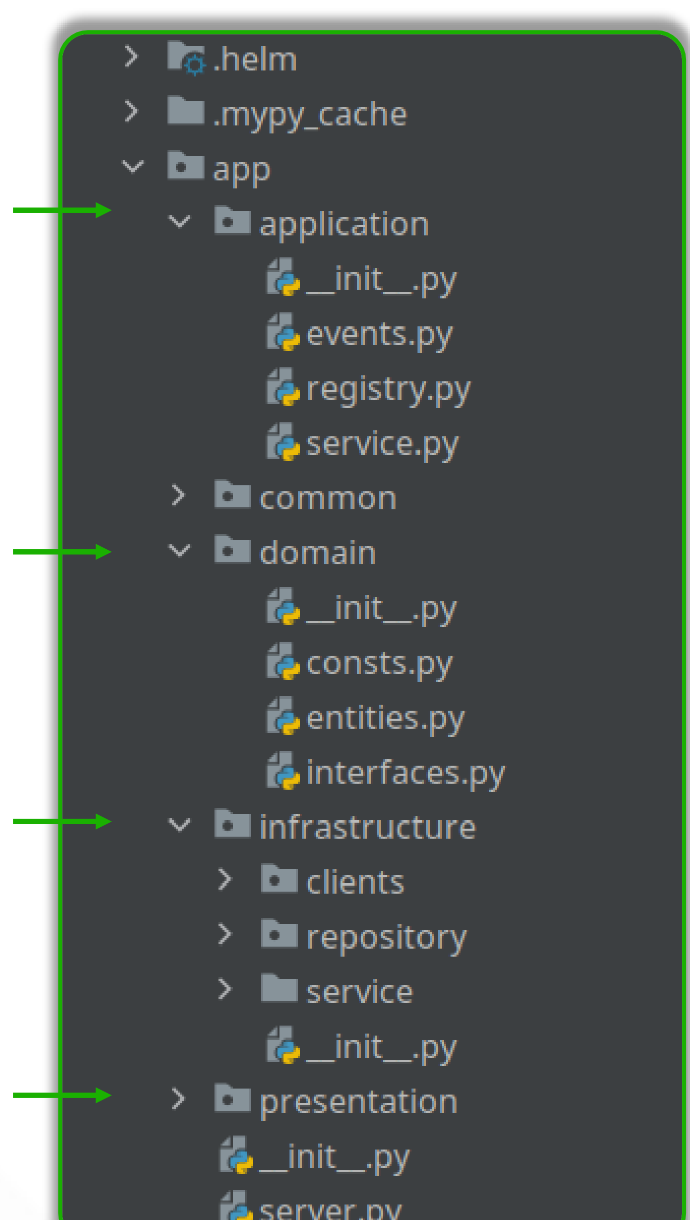
- src
  - adapter
    - api
    - spi
    - __init__.py
  - application
    - mappers
    - repositories
    - spi
    - usecases
    - utils
    - __init__.py
  - domain
    - __init__.py
    - api_exception.py
    - base_entity.py
    - cat_fact.py
    - configuration_entity.py
    - dog_fact.py
  - infrastructure
    - __init__.py
    - app.py
    - config_mapper.py
  - __init__.py

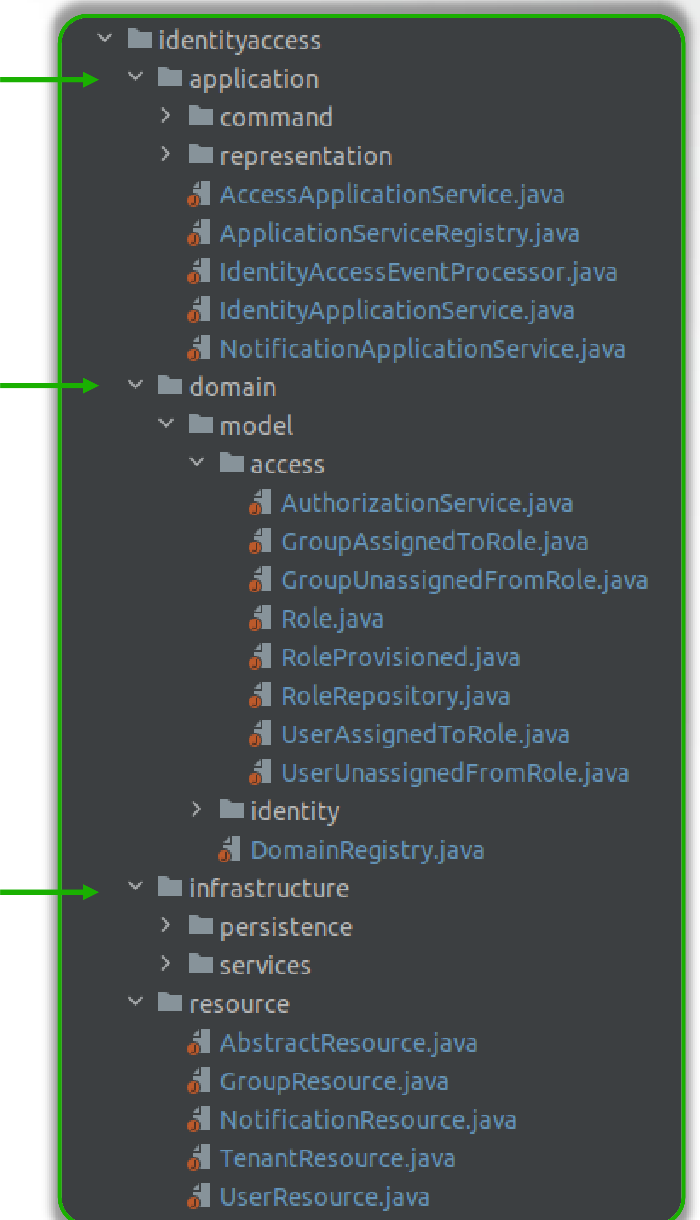
Зачем нужно приходить к единому стилю?

- Единообразие
- Легкость внедрения новых разработчиков
- Практичность
- Удобство

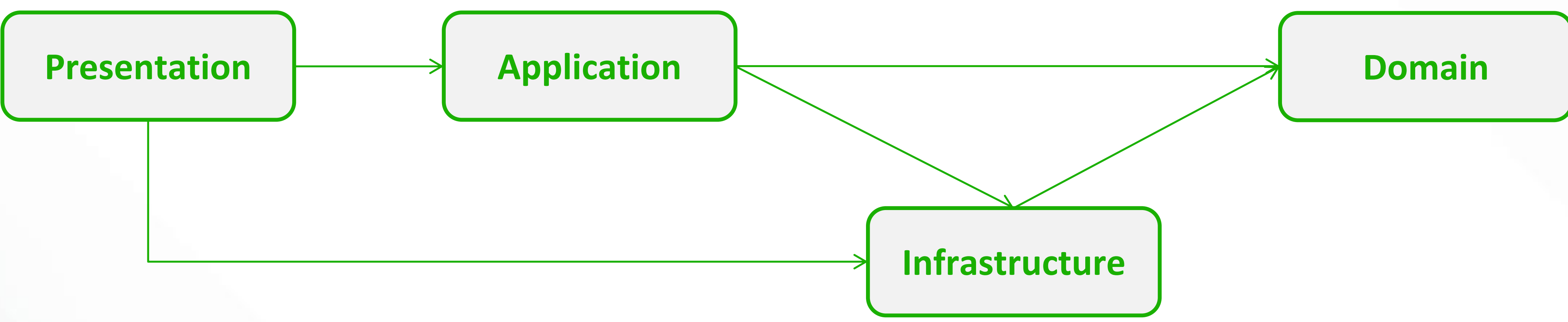
Структура проекта



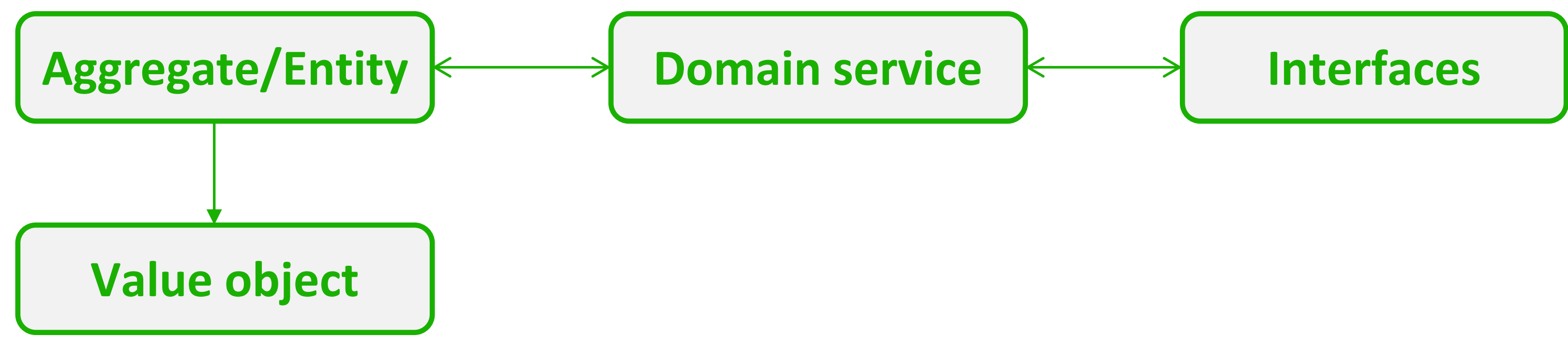




Упрощенная последовательность вызова



1. Domain Layer



- application
- common
- domain
  - __init__.py
  - consts.py
  - entities.py
  - interfaces.py
- infrastructure
- presentation

Entity/Aggregates

Доменные объекты

```
> .helm
> .mypy_cache
v app
  > application
  > common
  v domain
      __init__.py
      consts.py
      entities.py
      interfaces.py
  > infrastructure
  > presentation
    __init__.py
    server.py
> tests
  .dockerignore
  .gitignore
  .gitlab-ci.yml
  .python-version
  CHANGELOG.md
  Dockerfile
  Makefile
```

```python
38  class Genre(Base):
39      name: str
40
41
    anisov
42  class ChannelCategory(Base):
43      name: str
44
45
    anisov
46  class Channel(Base):
47      name: str
48      categories: List[ChannelCategory]
49      monetization: Optional[str]
50      timezones: List[str]
51      status: str
52      quality_code: str = Field(..., alias="quality-code")
53      content_id: Optional[str] = Field(None, alias="content-id")
54      service_id: Optional[int] = Field(None, alias="service-id")
55      original_network_id: Optional[int] = Field(
56          None, alias="original-network-id"
57      )
58      transport_id: Optional[int] = Field(None, alias="transport-id")
59      description: Optional[str]
```

Aggregate/Entity/Value object

Aggregate Type 1
Aggregate Root 1
Entity
Value Object

Aggregate Type 2
Aggregate Root 2
Entity
Entity
Value Object

Нужны ли агрегаты?

- “…Отсюда следует лишь, что большую часть АГРЕГАТОВ можно моделировать в виде простой СУЩНОСТИ, а не КОРНЯ.” Vaughn Vernon

Anemic Domain Model

- Не/Анемичная модель данных

Interfaces

Интерфейсы через которые происходит взаимодействие со сторонними системами/базами данных

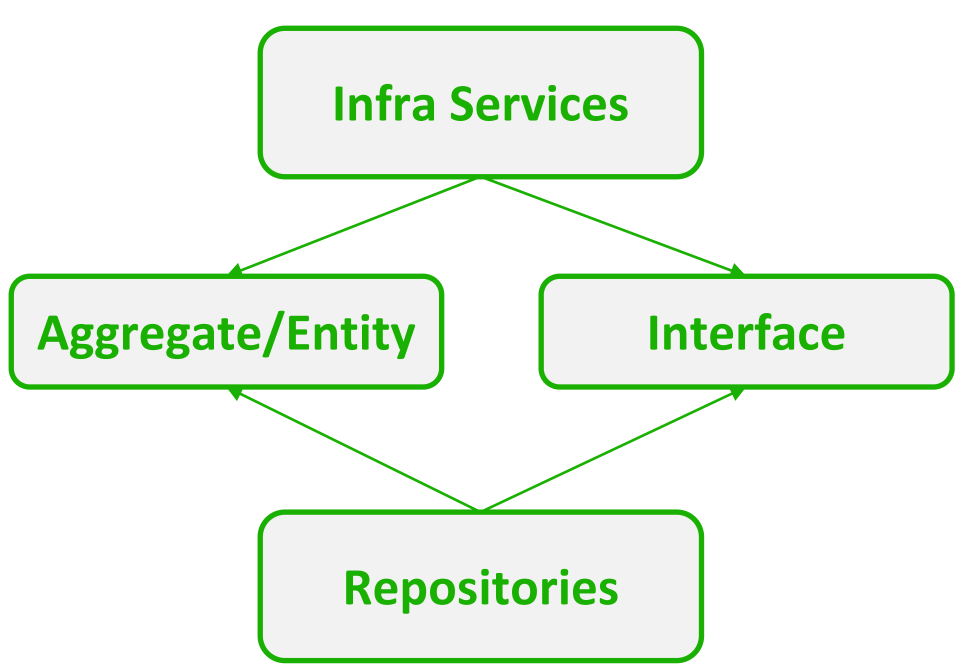
```
> .helm
> .mypy_cache
v app
  > application
  > common
  v domain
      __init__.py
      consts.py
      entities.py
      interfaces.py
  > infrastructure
  > presentation
    __init__.py
    server.py
> tests
  .dockerignore
  .gitignore
  .gitlab-ci.yml
  .python-version
```

```python
17  class EntityRepositoryInterface(Generic[DomainEntity], ABC):
18      @abstractmethod
19      async def all(self, api: bool = True) -> List[DomainEntity]:...
21
22      @abstractmethod
23      async def get(self, id_: int, api: bool = True) -> Optional[DomainEntity]:.
25
26      @abstractmethod
27      async def save(self, objs: List[DomainEntity]) -> None:...
29
30      @abstractmethod
31      async def update(self, obj: DomainEntity) -> None:...
33
34      @abstractmethod
35      async def delete(self, id_: int) -> None:...
```

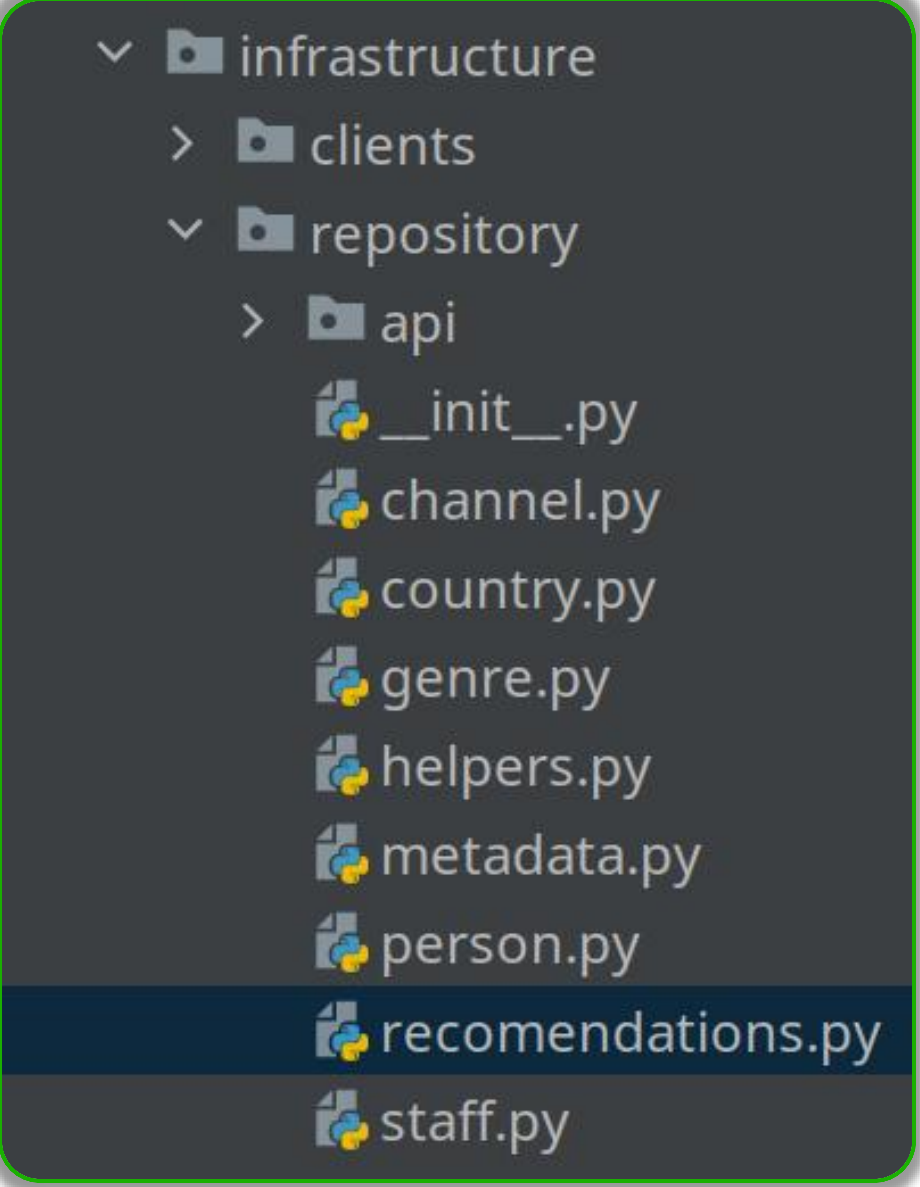
```
app
  application
  common
  domain
    __init__.py
    consts.py
    entities.py
    interfaces.py
  infrastructure
  presentation
  __init__.py
  server.py
tests
.dockerignore
.gitignore
.gitlab-ci.yml
.python-version
CHANGELOG.md
Dockerfile
```

```python
41
42
   5 usages   anisov
43 class RecommendationRepositoryInterface(ABC):
       anisov
44     @abstractmethod
45     async def get(
46         self, users_id: Optional[List[int]], version: int
47     ) -> Recommendations:...
49
       2 usages (1 dynamic)   anisov
50     @abstractmethod
51     async def upload(self, data: bytes) -> None:...
53
       anisov
54     @abstractmethod
55     async def delete(self, uuid: str) -> None:...
57
       3 usages   anisov
58     @abstractmethod
59     async def change_status(self, status: str, version: int) -> None:...
```

2. Infrastructure Layer



GS LABS


infrastructure
clients
repository
api
__init__.py
channel.py
country.py
genre.py
helpers.py
metadata.py
person.py
recomendations.py
staff.py

Implementation

Реализация интерфейсов

```
> .helm
> .mypy_cache
v app
  > application
  > common
  > domain
  v infrastructure
    > clients
    v repository
      > api
        __init__.py
        channel.py
        country.py
        genre.py
        helpers.py
        metadata.py
        person.py
        recomendations.py
        staff.py
    > service
      __init__.py
  > presentation
```

```
        4 usages   anisov
35      class MoviesRecommendationRepository(
36          ArchivariusAPI, RecommendationRepositoryInterface
37      ):
38          _mds_endpoint = "/api/manager/user-recommendations/upload-file/"
39          _mds_host = settings.MDS_HOST
40
            2 usages   anisov
41          async def _get_httpx_client(self) -> AsyncClient:
42              return AsyncClient(timeout=self._timeout, follow_redirects=True)
43
            anisov
44          async def get(
45              self, users_id: Optional[List[int]], version: int
46          ) -> Recommendations:...
89
            1 usage (1 dynamic)   anisov
90          async def upload(self, data: bytes) -> None:...
114
            anisov
115         async def delete(self, uuid: str) -> None:...
127
            anisov
128         async def change_status(self, status: str, version: int) -> None:...
```

3. Application Layer

- application
  - __init__.py
  - events.py
  - registry.py
  - service.py
- common
- domain
- infrastructure
- presentation
- __init__.py
- server.py

Application Service

Слой отвечающий за бизнес логику, а так же интеграцию с другими сервисами/репозиториями, работу с событиями

```
.helm
.mypy_cache
app
    application
        __init__.py
        events.py
        registry.py
        service.py
    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
```

```python
199  class ChannelService:
         anisov
200      def __init__(
201          self,
202          channel_repository: EntityRepositoryInterface[Channel],
203          category_repository: EntityRepositoryInterface[ChannelCategory],
204      ):
205          self.channel_repository: ... = channel_repository
208          self.category_repository: EntityRepositoryInterface[
209              ChannelCategory
210          ] = category_repository
211
         2 usages (2 dynamic)   anisov
212      @backoff.on_exception(
213          backoff.expo, exception=Exception, max_tries=settings.INIT_JOB_RETRIES
214      )
215      @error_logging
216      async def init_channels(self) -> None:...
223
         1 usage (1 dynamic)   anisov
224      @backoff.on_exception(
225          backoff.expo, exception=Exception, max_tries=settings.INIT_JOB_RETRIES
226      )
227      @error_logging
228      async def full_update_channels(self) -> None:...
235
         1 usage (1 dynamic)   anisov
236      async def get_categories(self) -> List[ChannelCategory]:...
```

```python
class RecommendationService:
    anisov
    def __init__(
        self,
        repository: RecommendationRepositoryInterface,
    ):
        self.repo: RecommendationRepositoryInterface = repository

    1 usage    anisov
    @backoff.on_exception(
        partial(backoff.expo, max_value=60),
        exception=Exception,
        max_tries=settings.UPDATE_JOB_RETRIES,
    )
    @error_logging
    async def update_recommendations(self, version: int) -> None:...
```

Domain Service/Application Service

- Domain Service
- Application Service

Registry

Registry

Инициализация и совмещение в единый компонент application service и infrastructure

```
app
  application
    __init__.py
    events.py
    registry.py
    service.py
  common
  domain
  infrastructure
  presentation
  __init__.py
  server.py
tests
.dockerignore
.gitignore
.gitlab-ci.yml
.python-version
CHANGELOG.md
```

```python
anisov
38  class ApplicationServiceRegistry:
39      metadata_service = MetaDataService(
40          metadata_repository=MetaDataRepository(),
41          country_repository=CountryRepository(),
42          genre_repository=GenreRepository(),
43          person_repository=PersonRepository(),
44          duration_repository=MetadataDurationRepository(),
45      )
46      channel_service = ChannelService(
47          channel_repository=ChannelRepository(),
48          category_repository=ChannelCategoryRepository(),
49      )
50      movies_recommendation_service = RecommendationService(
51          repository=(
52              MoviesRecommendationRepository()
53              if not settings.FTP_RECOMMENDATIONS
54              else FTPMoviesRecommendationRepository()
55          ),
56      )
```

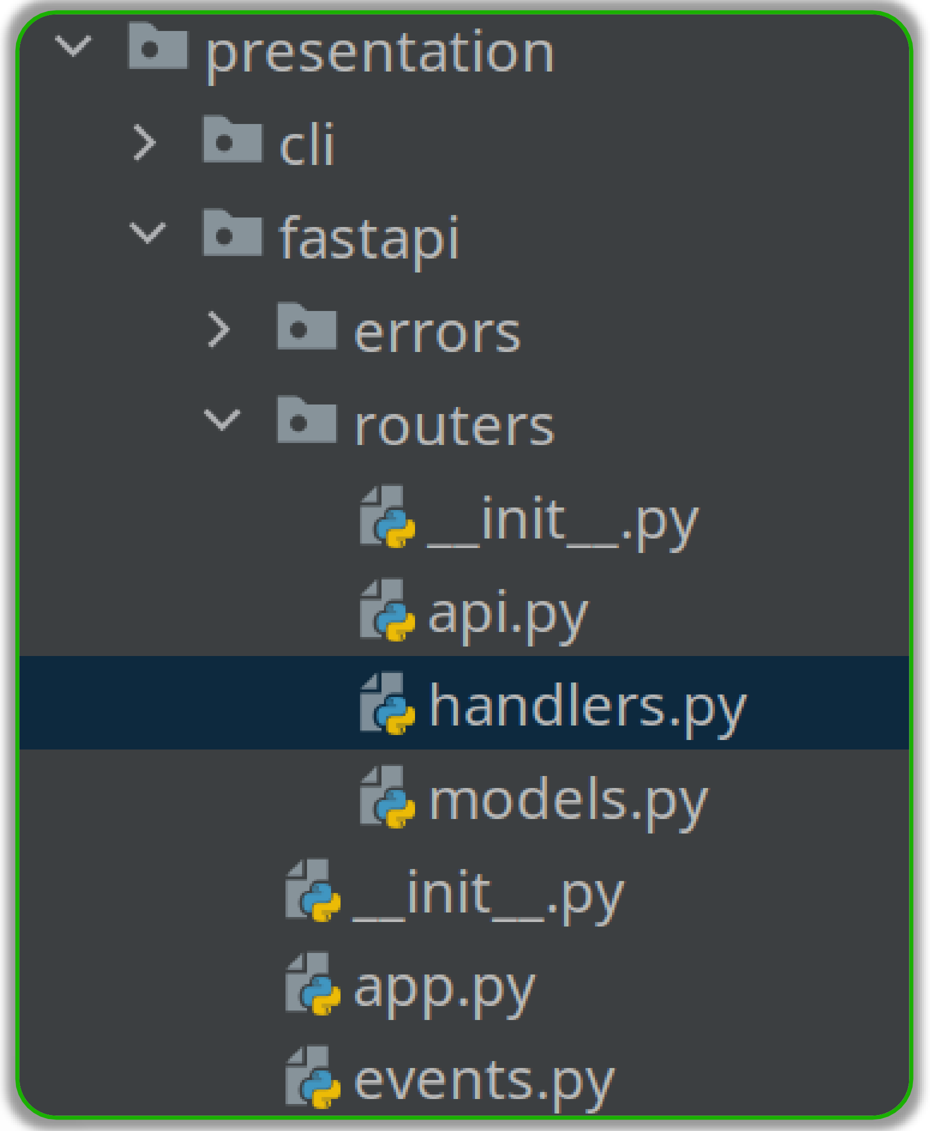
4. Presentation

Слой представления

```
presentation
  cli
  fastapi
    errors
    routers
      __init__.py
      api.py
      handlers.py
      models.py
    __init__.py
    app.py
    events.py
  pulsar
  __init__.py
__init__.py
server.py
```

```python
121
    anisov
122 @router.post(
123         path: "/update_recommendations/{recommendation_type}",
124         response_model=str,
125         name="update_recommendations",
126         response_class=UJSONResponse,
127 )
128 async def update_recommendations(
129         recommendation_type: str, data: UpdateRecommendationsModel
130 ) -> str:...
155
```

4. Presentation



Event bus

Events

5

2

Registry

4

Infra Services

1

3

FastApi/CLI/GRPC

Service

Aggregate/Entity

Interface

4

Aggregate/Entity

Interfaces

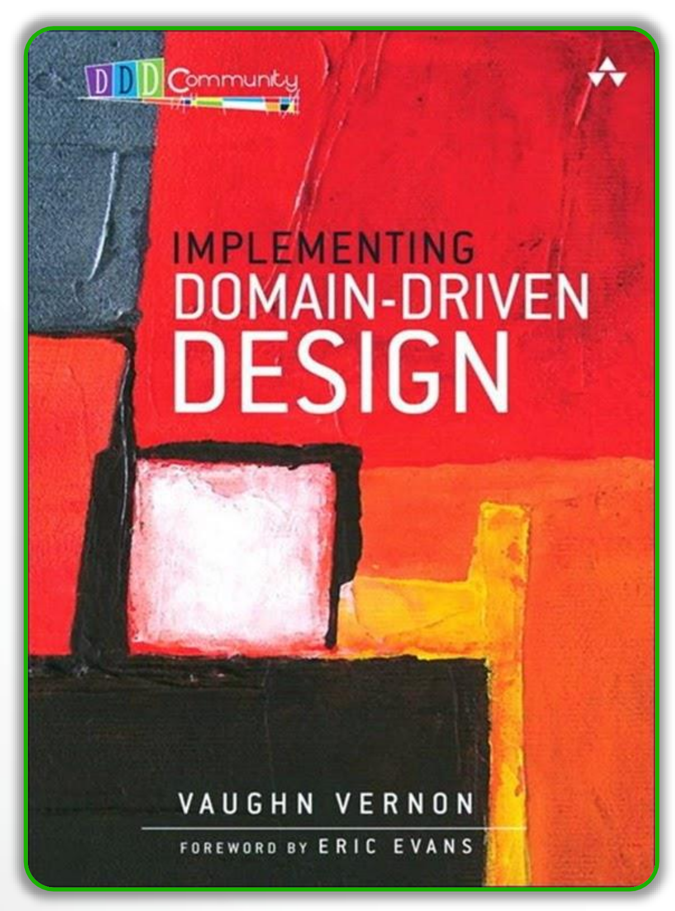
Repositories

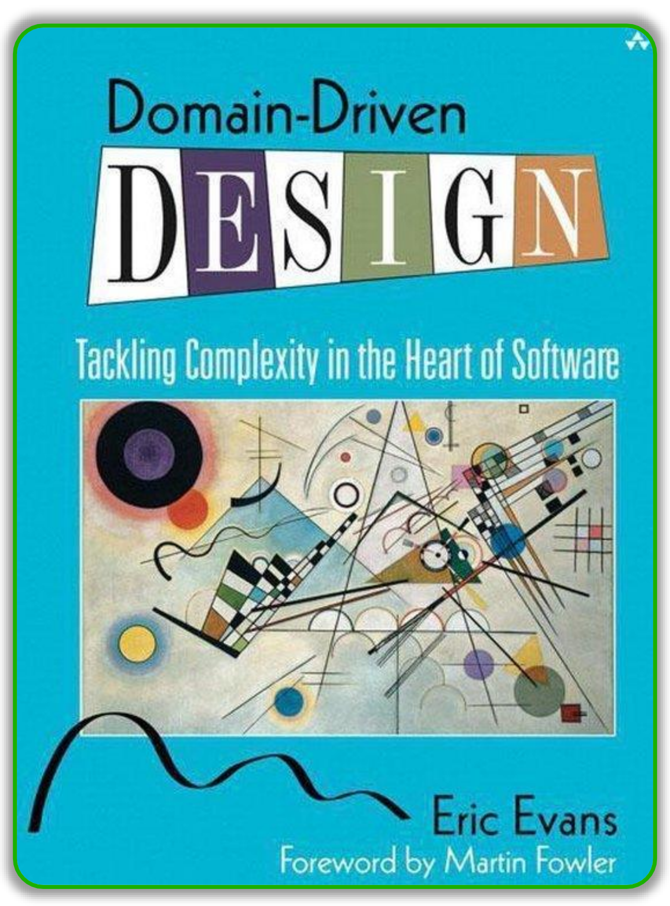
Value object

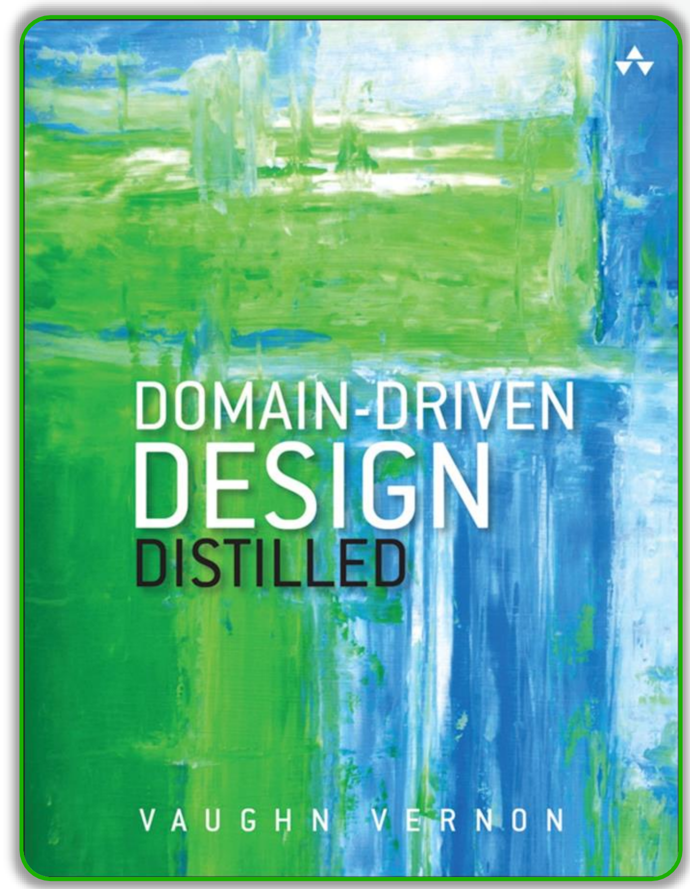
Как работать с транзакциями?

- 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
- Но в реальной жизни достичь этого возможно далеко не всегда, по этому можно использовать любой аналог декоратора transaction или схожий механизм.
  – Причины нарушения правил:
    - Удобство пользовательского интерфейса
    - Отсутствие технических механизмов
    - Глобальные транзакции
    - Производительность запросов
- Принцип итоговой согласованности

Что почитать ?







Итог

- Разобрали кратко зачем нужен DDD, какие плюсы/минусы есть у данного подхода, кому нужен
- Основные понятия
- С чего начать проектирование
- Реализация тактического проектирования на конкретном примере
- Как структурировать код
- Стоит ли делать анемичную модель данных, почему не стоит добавлять `alias`
- 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

Контактная информация:

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