

# КПО: Отчёт по домашнему заданию № 2

Page • Tag

Автор: Лефанов Никита Юрьевич

## 1. Реализованный функционал (Use Cases)



	Use case	Эндпоинт (HTTP)	Основные классы/ модули
1	Добавить / удалить животное	POST /animals , DELETE /animals/{id}	<i>Presentation:</i> AnimalController → <i>Application:</i> AnimalService → <i>Domain:</i> Animal → <i>Infrastructure:</i> InMemoryAnimalRepository
2	Добавить / удалить вольер	POST /enclosures , DELETE /enclosures/{id} }	EnclosureController → EnclosureService → Enclosure → InMemoryEnclosureRepository
3	Переместить животное между вольерами	POST /animals/{id}/transfer? toEnclosureId=...	AnimalController → AnimalService → EnclosureService.transferAnimal()
<i>Domain-событие:</i> AnimalMovedEvent			
4	Просмотреть расписание кормлений	GET /feedings	FeedingScheduleController → FeedingService.list()
5	Добавить новое кормление	POST /feedings	FeedingScheduleController → FeedingService.create() → FeedingSchedule
6	Отметить выполнение кормления	POST /feedings/{id}/execute	FeedingService.markExecuted() →

			FeedingTimeEvent
7	Просмотреть статистику зоопарка	GET /statistics	ZooStatisticsController → ZooStatisticsService
8	Кормление и лечение животного	POST /animals/{id}/feed, POST /animals/{id}/heal	AnimalService.feed()/heal() → методы Animal.feed() / Animal.treat()
9	Уборка вольера	POST /enclosures/{id}/clean	EnclosureService.clean() → Enclosure.clean()

## 2. Применённые концепции Domain-Driven Design

DDD-концепция	Классы / пакеты
Entities	Animal, Enclosure, FeedingSchedule (пакет domain)
Value Objects	Species, Gender, FoodType, Size (domain.vo) — неизменяемые record / enum
Domain Events	AnimalMovedEvent, FeedingTimeEvent (domain.events)
Aggregates	Enclosure агрегирует список animalIds, инкапсулируя правило вместимости
Инкапсуляция бизнес-правил	<ul style="list-style-type: none"> <li>Enclosure.addAnimal() — проверка лимита</li> <li>Animal.treat()/feed()/move()</li> <li>FeedingSchedule.markFeedingAsExecuted()</li> </ul>

### 3. Соблюдение принципов Clean Architecture

Принцип	Как реализовано
Слои зависят только внутри	<code>domain</code> ни от кого не зависит 2) <code>application</code> зависит только на <code>domain</code> и собственные порты ( <code>application.port.out.*</code> ) 3) <code>presentation</code> и <code>infrastructure</code> зависят на <code>application</code> , но не наоборот
Зависимости через интерфейсы	Порты ( <code>AnimalRepository</code> , <code>EnclosureRepository</code> , <code>FeedingScheduleRepository</code> ) объявлены в <code>application.port.out</code> ; конкретная in-memory реализация — в <code>infrastructure.repository.*</code>
Изоляция бизнес-логики	Вся логика находится в <code>domain</code> и service-классах <code>application.service.*</code> ; контроллеры лишь принимают/отдают DTO, инфраструктура — только хранит данные
Разделение слоёв в пакеты	<code>domain</code> , <code>application</code> (sub-packages <code>port.out</code> , <code>service</code> ), <code>infrastructure.repository</code> , <code>presentation</code>
<code>GlobalExceptionHandler</code>	<code>exception.GlobalExceptionHandler</code> — сквозная обработка ошибок без «протечки» деталей слоёв наружу

### 4. Хранилище данных

Инфраструктурный слой содержит In-Memory-репозитории, реализующие требуемые интерфейсы портов. Переход к любому другому типу БД — только через добавление новой реализации в Infrastructure, не затрагивая остальные слои.