Домашнее задание №2
Закрепление концепции Domain-Driven Design и принципов проектирования Clean Architecture

Студент

Апрель 2025

1. Введение

Данный отчет описывает разработанную систему управления зоопарком с использованием принципов Domain-Driven Design (DDD) и Clean Architecture. Система разработана для автоматизации следующих бизнес-процессов: управление животными, вольерами и расписанием кормлений.

Основное внимание уделено контролю правильного расселения животных по вольерам, чтобы предотвратить размещение травоядных в вольерах с хищниками.

2. Реализованная функциональность

В соответствии с требованиями заказчика была реализована следующая функциональность:

2.1. Управление животными

1. Добавление новых животных (реализовано в классе `AnimalController`, метод `Create`)
2. Удаление животных (реализовано в классе `AnimalController`, метод `Delete`)
3. Просмотр информации о животных (реализовано в классе `AnimalController`, методы `GetAll` и `GetById`)
4. Изменение статуса здоровья (реализовано в доменной модели `Animal`, методы `Treat` и `MarkAsSick`)

2.2. Управление вольерами

1. Добавление новых вольеров (реализовано в классе `EnclosureController`, метод `Create`)
2. Удаление вольеров (реализовано в классе `EnclosureController`, метод `Delete`)
3. Просмотр информации о вольерах (реализовано в классе `EnclosureController`, методы `GetAll` и `GetById`)

4. Просмотр списка животных в вольере (реализовано в классе `EnclosureController`, метод `GetAnimalsInEnclosure`)

2.3. Перемещение животных между вольерами

1. Перемещение животного в вольер (реализовано в классе `AnimalController`, метод `MoveToEnclosure`)
2. Удаление животного из вольера (реализовано в классе `AnimalController`, метод `RemoveFromEnclosure`)
3. Бизнес-логика проверки совместимости животных и вольеров (реализовано в классе `Enclosure`, метод `AddAnimal`)

2.4. Управление расписанием кормлений

1. Создание новых кормлений (реализовано в классе `FeedingScheduleController`, метод `Create`)
2. Просмотр всех запланированных кормлений (реализовано в классе `FeedingScheduleControl` метод `GetAll`)
3. Просмотр кормлений по дате (реализовано в классе `FeedingScheduleController`, метод `GetByDate`)
4. Просмотр кормлений для конкретного животного (реализовано в классе `FeedingScheduleCon` метод `GetByAnimalId`)
5. Отметка о выполнении кормления (реализовано в классе `FeedingScheduleController`, метод `MarkAsCompleted`)

2.5. Просмотр статистики зоопарка

1. Общее количество животных (реализовано в классе `ZooStatisticsController`, метод `GetAnimalCount`)
2. Статистика по видам животных (реализовано в классе `ZooStatisticsController`, метод `GetSpeciesStatistics`)
3. Статистика по состоянию здоровья животных (реализовано в классе `ZooStatisticsControll` метод `GetAnimalHealthStatistics`)
4. Информация о вольерах (реализовано в классе `ZooStatisticsController`, методы `GetEnclosureCount`, `GetEnclosureTypeStatistics`, `GetEnclosureOccupancyRate`
5. Общее резюме зоопарка (реализовано в классе `ZooStatisticsController`, метод `GetZooSummary`)

3. Применение Domain-Driven Design

В рамках проекта были применены следующие концепции Domain-Driven Design:

3.1. Использование Value Objects для примитивов

1. `FoodType` - созданный Value Object для представления типа пищи. Класс обеспечивает неизменяемость значения и корректное сравнение на равенство.

2. `EnclosureType` - Value Object для представления типа вольера (для хищников, травоядных, птиц, аквариум). Содержит предопределенные типы и обеспечивает неизменяемость.

3. `Gender` - перечисление, представляющее пол животного.

Пример реализации Value Object:

```
public class FoodType
{
    public string Name { get; private set; }

    private FoodType(string name)
    {
        if (string.IsNullOrWhiteSpace(name))
            throw new ArgumentException("Food name cannot be empty", nameof
    (name));

        Name = name;
    }

    public static FoodType Create(string name) => new FoodType(name);

    public override bool Equals(object obj)
    {
        if (obj is not FoodType other)
            return false;

        return string.Equals(Name, other.Name, StringComparison.
    OrdinalIgnoreCase);
    }

    public override int GetHashCode()
    {
        return Name.ToLowerInvariant().GetHashCode();
    }
}
```

3.2. Богатая доменная модель с инкапсуляцией бизнес-правил

1. `Animal` - инкапсулирует бизнес-правила для животных, включая логику проверки корректности создания (корректные имя, вид, дата рождения) и изменения состояния (здоровье, перемещение).

2. `Enclosure` - содержит бизнес-правила для размещения животных, проверяя совместимость хищников и травоядных, а также контроль максимальной вместимости.

3. `FeedingSchedule` - инкапсулирует логику планирования и выполнения кормлений.

Пример инкапсуляции бизнес-правил в доменной модели:

```
public bool AddAnimal(Animal animal)
{
    if (animal == null)
        throw new ArgumentNullException(nameof(animal));

    if (!CanAddAnimal())
        return false;

    // Domain logic to determine if the animal can be placed in this
    enclosure
    // For example, don't put predators with herbivores
    if (Type == EnclosureType.Predator && !IsPredator(animal.Species))
        return false;

    if (Type == EnclosureType.Herbivore && IsPredator(animal.Species))
        return false;

    _animals.Add(animal);
    animal.MoveToEnclosure(Id);

    return true;
}
```

3.3. Доменные события

Реализовано два основных доменных события для обеспечения интеграции между различными частями системы:

1. `AnimalMovedEvent` - возникает при перемещении животного между вольерами.
2. `FeedingTimeEvent` - возникает при наступлении времени кормления.

Реализованы базовые классы для работы с событиями:

1. `DomainEvent` - базовый класс для всех доменных событий.
2. `DomainEvents` - статический класс для управления событиями.
3. `IDomainEventHandler` - интерфейс для обработчиков событий.

4. Реализация Clean Architecture

Проект структурирован согласно принципам Clean Architecture:

4.1. Domain Layer (ядро)

Содержит основные сущности, Value Objects и бизнес-правила. Этот слой полностью независим от внешних слоев и не содержит никаких зависимостей от внешних фреймворков или библиотек.

1. `Domain/Entities` - содержит основные сущности (Animal, Enclosure, FeedingSchedule)
2. `Domain/ValueObjects` - содержит Value Objects (FoodType, EnclosureType, Gender)

3. `Domain/Events` - содержит доменные события и их обработчики
4. `Domain/Interfaces` - содержит интерфейсы репозиториев, используемые для взаимодействия с хранилищами данных

4.2. Application Layer

Содержит бизнес-логику приложения, использует сущности из доменного слоя и определяет интерфейсы для внешних сервисов:

1. `Application/Services` - содержит сервисы, реализующие бизнес-логику (AnimalTransferServ FeedingOrganizationService, ZooStatisticsService)
2. `Application/Interfaces` - содержит интерфейсы сервисов
3. `Application/DTOs` - содержит объекты для передачи данных между слоями

Этот слой зависит только от Domain Layer и не имеет зависимостей от внешних фреймворков или библиотек.

4.3. Infrastructure Layer

Реализует интерфейсы, определенные в доменном и прикладном слоях, для взаимодействия с внешними системами и ресурсами:

1. `Infrastructure/Data` - содержит реализацию хранилища данных (InMemoryContext)
2. `Infrastructure/Repositories` - содержит реализацию репозиториев

4.4. Presentation Layer

Содержит компоненты пользовательского интерфейса и API:

1. `Presentation/Controllers` - содержит веб-контроллеры для обработки HTTP-запросов

5. Принципы Clean Architecture в проекте

5.1. Зависимости направлены только внутрь

Внутренние слои не зависят от внешних:

1. Domain Layer не имеет никаких зависимостей от других слоев.
2. Application Layer зависит только от Domain Layer.
3. Infrastructure Layer зависит от Domain и Application Layers.
4. Presentation Layer зависит от Domain, Application, и Infrastructure Layers.

5.2. Зависимости между слоями через интерфейсы

Все взаимодействия между слоями осуществляются через интерфейсы:

1. В Domain Layer определены интерфейсы репозиториев (`IAnimalRepository`, `IEnclosureRepository`, `IFeedingScheduleRepository`).
2. В Application Layer определены интерфейсы сервисов (`IAnimalService`, `IEnclosureServic` `IFeedingScheduleService`).
3. Infrastructure Layer реализует интерфейсы из Domain Layer.
4. Presentation Layer использует интерфейсы сервисов из Application Layer.

5.3. Бизнес-логика изолирована в Domain и Application слоях

1. В Domain Layer содержатся основные бизнес-правила, связанные с сущностями (проверка здоровья животных, совместимость животных и вольеров).
2. В Application Layer содержится бизнес-логика приложения (перемещение животных между вольерами, организация кормления, сбор статистики).
3. Infrastructure Layer отвечает только за доступ к данным.
4. Presentation Layer отвечает только за взаимодействие с пользователем.

6. Заключение

Система управления зоопарком полностью реализована с использованием принципов Domain-Driven Design и Clean Architecture. Применение этих подходов обеспечивает:

1. Чистую и понятную структуру проекта
2. Изоляцию бизнес-логики от деталей реализации инфраструктуры и представления
3. Легкое тестирование компонентов
4. Гибкость и расширяемость системы

Все требуемые бизнес-процессы автоматизированы, и система позволяет эффективно управлять животными, вольерами и кормлениями, предотвращая размещение несовместимых животных в одном вольере.