Teach Me Skills

Занятие 29.
Тестирование Django

1

Основы тестирования веб-приложений

Что важно учесть

- **Покрытие тестами:** Убедитесь, что вы тестируете все ключевые компоненты вашего приложения, включая представления (views), модели, URL-адреса и формы. Цель состоит в том, чтобы достичь максимального покрытия кода тестами, чтобы убедиться, что все функциональные части вашего приложения работают должным образом.

- **Изоляция тестов:** При тестировании каждый тест должен быть независимым и изолированным от других тестов. Это означает, что каждый тест должен создавать свои собственные данные, модели и среду выполнения, чтобы не возникало конфликтов между тестами.

Что важно учесть

- **Настройка тестовой базы данных**: Django предоставляет возможность использования отдельной базы данных для тестирования. Это позволяет вам создать и использовать отдельную базу данных только для целей тестирования, чтобы не затрагивать реальные данные в процессе тестирования. Убедитесь, что вы настроили тестовую базу данных и используете ее при запуске тестов.

- **Использование фикстур**: Фикстуры в Django позволяют создавать начальные данные для тестов. Используйте фикстуры, чтобы создать необходимые объекты моделей или загрузить начальные данные для вашего тестового окружения. Это поможет сделать ваши тесты более надежными и предсказуемыми.

Что важно учесть

- **Обработка ошибок и исключений:** Убедитесь, что ваши тесты проверяют корректное обработку ошибок и исключений в вашем приложении. Проверьте, что ваше приложение правильно возвращает соответствующие статусы HTTP, обрабатывает ошибки форм и валидацию данных.

- **Тестирование асинхронных задач:** Если ваше приложение использует асинхронные задачи с помощью Django Celery или аналогичных инструментов, убедитесь, что вы тестируете их правильность выполнения и результатов. Используйте мок-объекты или фикстуры для имитации выполнения асинхронных задач в тестовой среде.

Что важно учесть

- **Постоянное обновление тестов:** Помните, что ваше приложение может меняться со временем, и ваши тесты должны соответствовать последним изменениям. Регулярно обновляйте и поддерживайте ваши тесты, чтобы они отражали текущее состояние вашего приложения.

- **Автоматическое выполнение тестов:** Настройте систему автоматического выполнения тестов, чтобы она запускала тесты при каждом коммите или развертывании вашего приложения. Это позволит обнаруживать проблемы и ошибки в ранней стадии разработки и избежать непредвиденных проблем в продакшене.

2

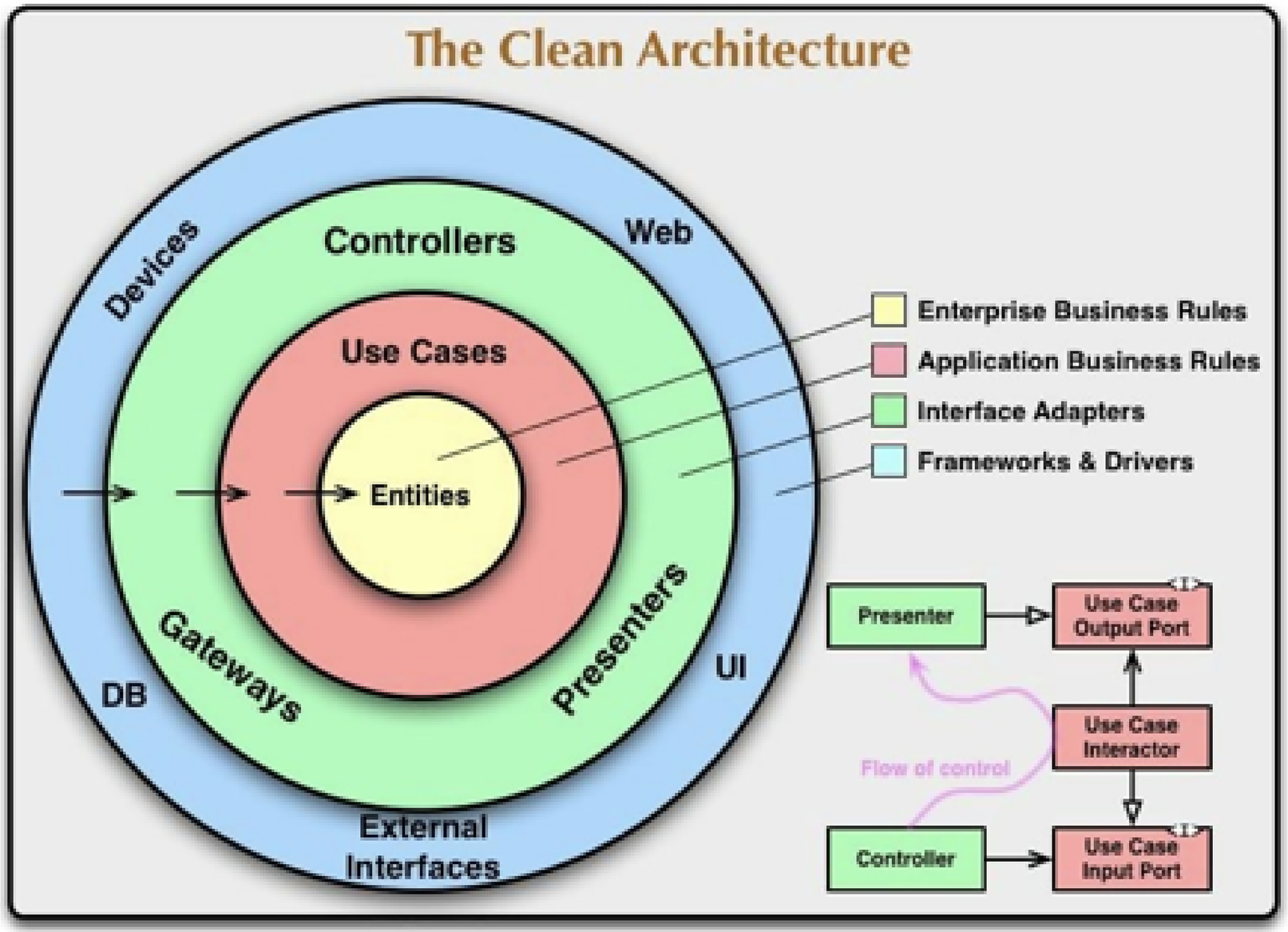
Clean Architecture

Clean Architecture

- **Чистая архитектура (Clean Architecture) -** это архитектурный подход к разработке программного обеспечения, который нацелен на создание модульных, гибких и независимых систем. Этот подход был предложен Робертом Мартином (также известным как Uncle Bob) и основывается на принципах **SOLID.**

- Главная идея чистой архитектуры заключается в том, чтобы создать приложение, которое может быть легко изменено и тестируемо без необходимости изменения его внутренних компонентов или зависимостей. Вместо того, чтобы привязывать код к конкретным фреймворкам или библиотекам, чистая архитектура ставит акцент на разделение кода на различные уровни, каждый из которых имеет свою ответственность и зависимости.

Схема


The Clean Architecture
Devices
Web
Controllers
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers
Presenter
Use Case
Output Port
Use Case
Interactor
Flow of control
Controller
Use Case
Input Port

Принципы чистой архитектуры

- **Независимость от фреймворков:** Чистая архитектура стремится к созданию кода, который не зависит от конкретных фреймворков или библиотек. Внешние фреймворки должны быть адаптированы к приложению, а не наоборот. Это позволяет легко заменять или обновлять внешние компоненты без влияния на основную бизнес-логику.

- **Разделение на уровни:** Чистая архитектура предлагает разделение кода на различные уровни (слои) с ясно определенными ответственностями. Обычно используются следующие уровни: внешний (пользовательский) уровень, уровень бизнес-логики, уровень доступа к данным и внутренний уровень (ядро приложения). Каждый уровень имеет свою модель данных, логику и интерфейсы.

Принципы чистой архитектуры

- **Зависимость от абстракций:** Вместо того, чтобы код зависел от конкретных реализаций или деталей, чистая архитектура нацелена на зависимость от абстракций. Интерфейсы и абстрактные классы используются для определения контрактов между различными компонентами. Это позволяет легко заменять или расширять конкретные реализации без необходимости изменения других частей системы.

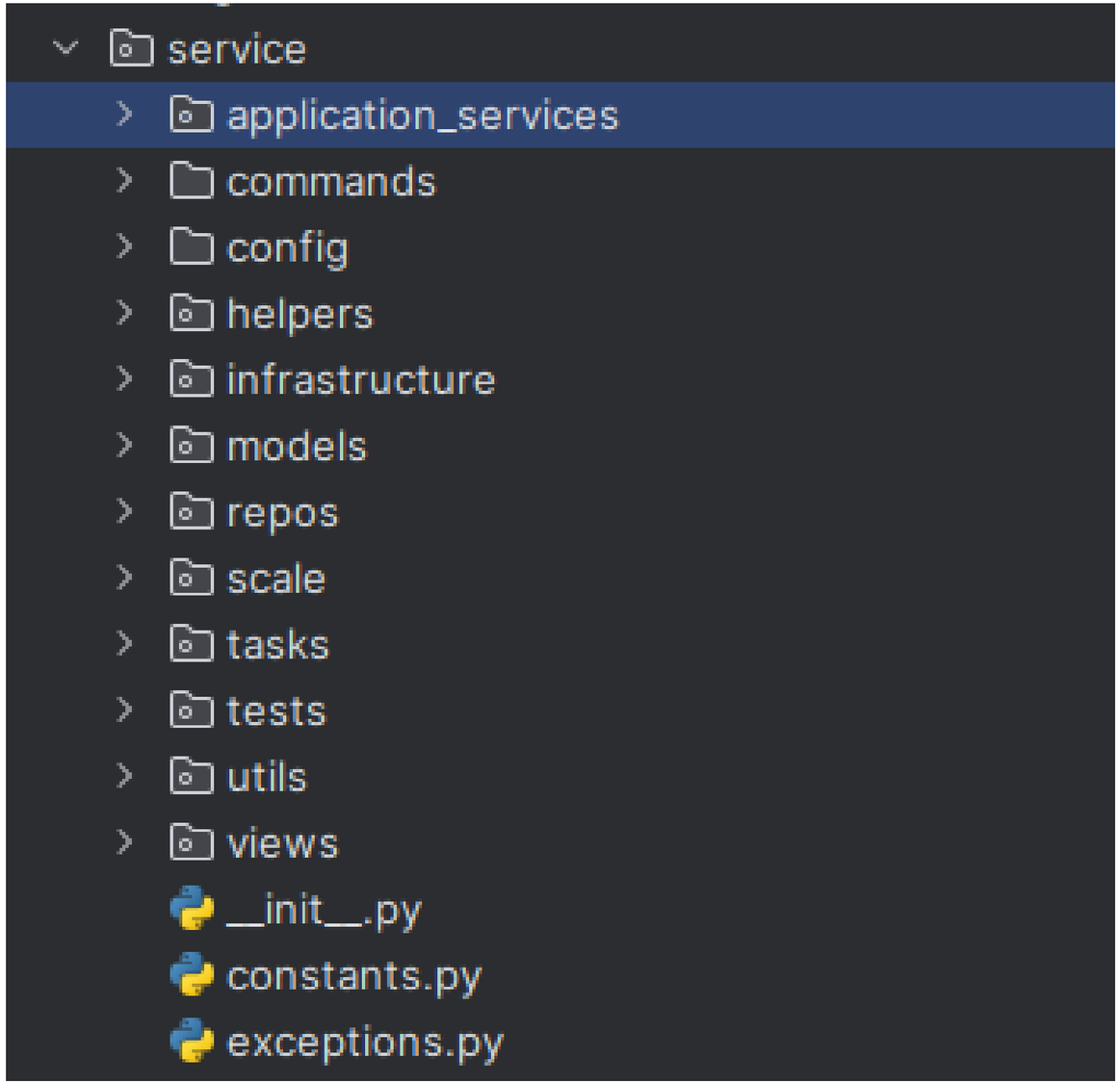
- **Однонаправленный поток данных:** Чистая архитектура настаивает на однонаправленном потоке данных, где данные движутся от внешнего уровня к внутреннему, а не наоборот. Внешние компоненты не должны напрямую зависеть от внутренних деталей. Вместо этого используются интерфейсы или службы для взаимодействия между уровнями.

Принципы чистой архитектуры

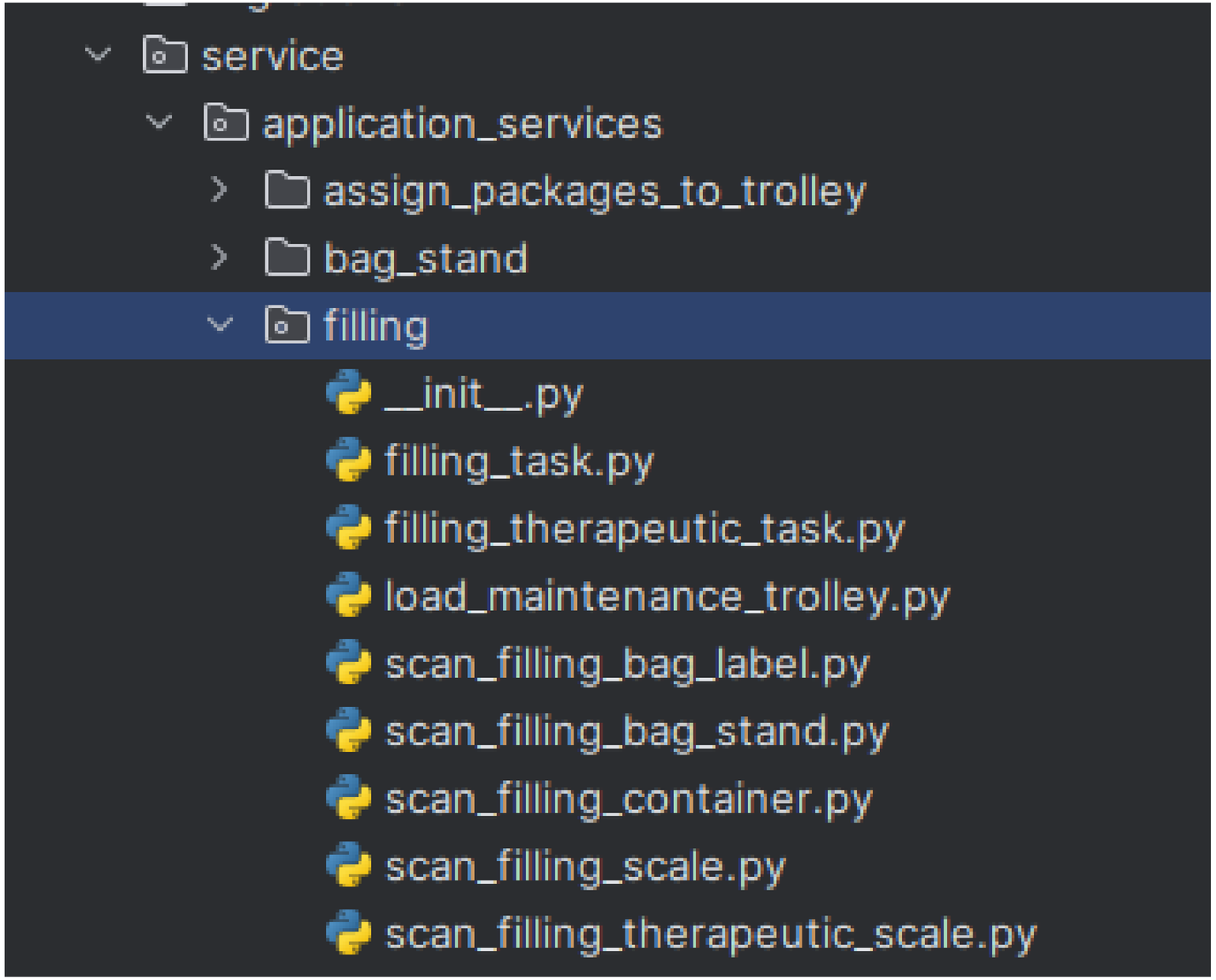
- **Тестирование и обслуживание:** Чистая архитектура облегчает тестирование кода, поскольку каждый уровень может быть тестирован отдельно. Также упрощается поддержка и развитие системы, так как изменения в одном уровне не должны затрагивать другие уровни.

Чистая архитектура не является конкретным шаблоном или реализацией, а скорее набором принципов и рекомендаций для построения гибких и легко поддерживаемых систем. Она может быть применена в различных языках программирования и фреймворках, включая Django, для создания хорошо структурированных и расширяемых приложений.

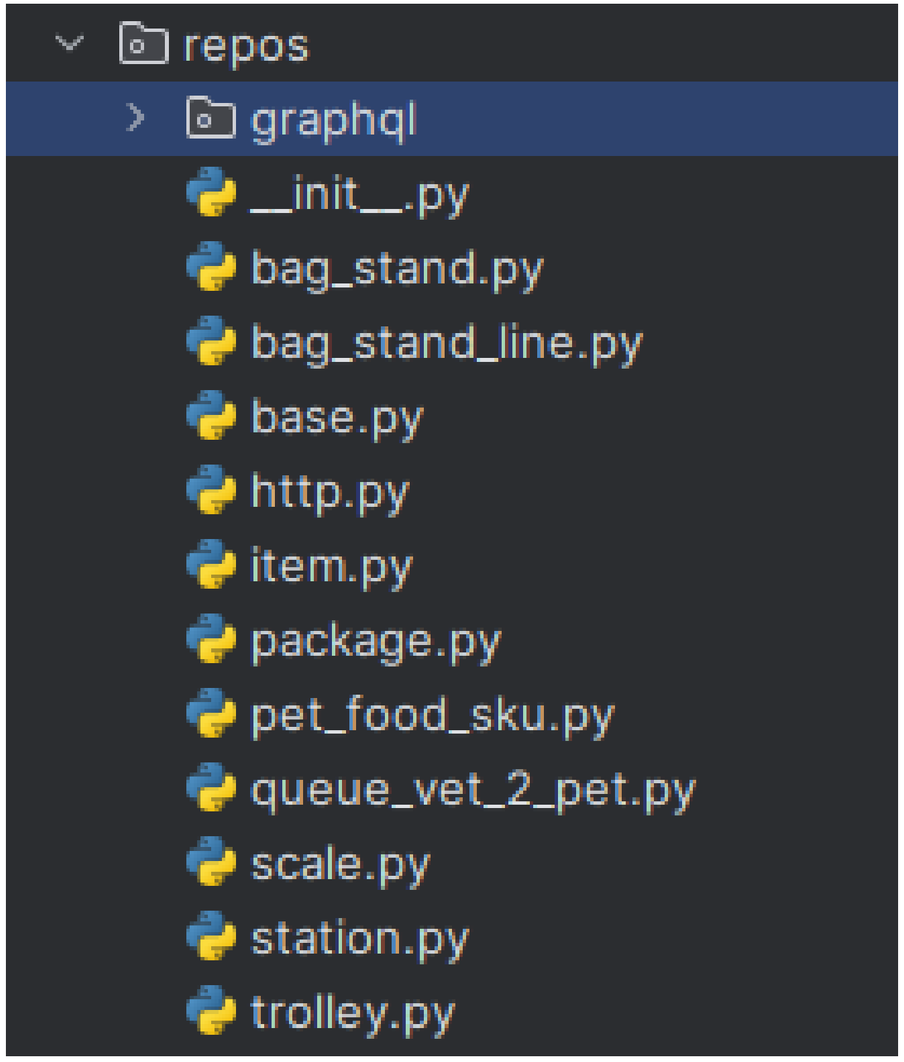
Пример

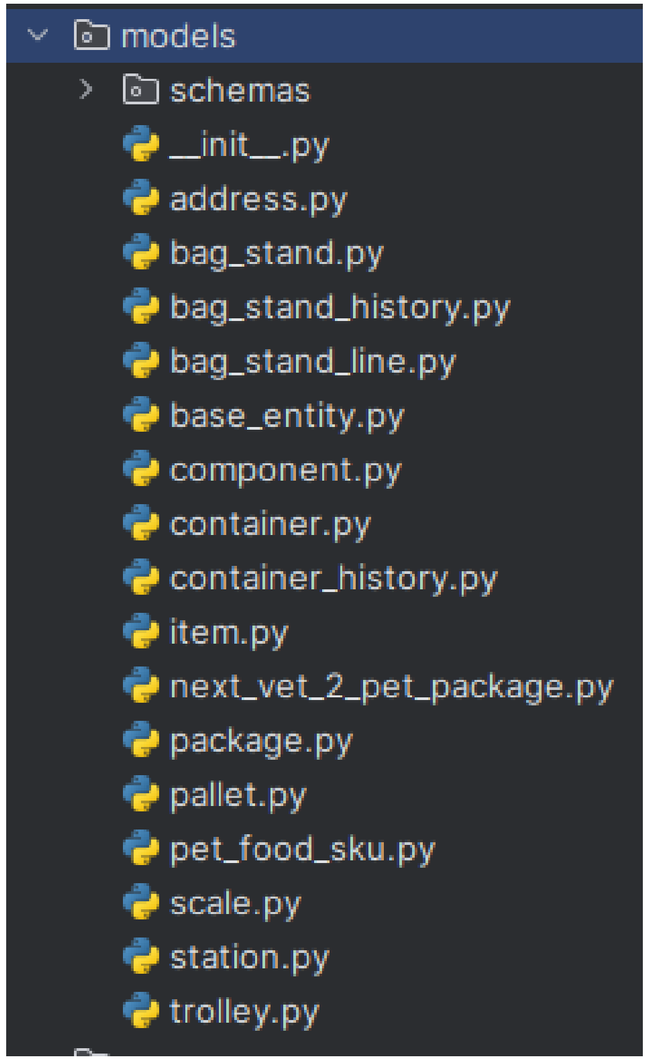

service
application_services
commands
config
helpers
infrastructure
models
repos
scale
tasks
tests
utils
views
__init__.py
constants.py
exceptions.py

Пример


service
application_services
assign_packages_to_trolley
bag_stand
filling
__init__.py
filling_task.py
filling_therapeutic_task.py
load_maintenance_trolley.py
scan_filling_bag_label.py
scan_filling_bag_stand.py
scan_filling_container.py
scan_filling_scale.py
scan_filling_therapeutic_scale.py

Пример






```python
@dataclass
class Item:
    """Class to represent an Item."""

    STATUS_FILLED = 'filled'
    STATUS_FILLING = 'filling'
    STATUS_PENDING = 'pending'
    STATUS_PACKED = 'packed'
    STATUS_PRINTED = 'printed'

    TYPE_KIBBLE = 'kibble'

    item_id: str
    package_id: str
    type: Optional[str] = None
    status: Optional[str] = None
    barcode: Optional[str] = None
    name: Optional[str] = None
    weight: Optional[float] = None
    components: List[Component] = field(default_factory=lis

    @property
    def is_pending(self):
        """Is the item in a pending state."""
        return self.status == self.STATUS_PENDING
```

Пример

```
myapp/
├── core/
|   ├── entities/
|   |   ├── __init__.py
|   |   └── models.py
|   ├── repositories/
|   |   ├── __init__.py
|   |   └── repository.py
|   ├── services/
|   |   ├── __init__.py
|   |   └── service.py
|   └── usecases/
|       ├── __init__.py
|       └── usecase.py
├── interfaces/
|   ├── controllers/
|   |   ├── __init__.py
|   |   └── controller.py
|   └── gateways/
|       ├── __init__.py
|       └── gateway.py
└── web/
    ├── views.py
    ├── serializers.py
    └── urls.py
```

3

Домашнее задание

Домашнее задание

- Добавить тесты для api эндпоинтов

Спасибо за внимание!