



ТИНЬКОФФ

Как DWH Core докатились до Clean Architecture

Software design

О чем будет доклад

- ✓ Эволюция дизайна сервисов на примере команды DWH Core
- ✓ **План:**
 1. Рассмотрим пример разработки функциональности сервиса
 2. Будем постепенно усложнять условия
 3. Рассмотрим три этапа эволюции:
 - плоская структура
 - структура со слоями
 - чистая архитектура

- ✓ **Ответим на вопросы:**
 1. С какими проблемами может столкнуться разработчик при росте кодовой базы
 2. Как решить эти проблемы с помощью дизайна

Примеры на Python, но принципы проектирования подойдут для любого ЯП

Для кого доклад

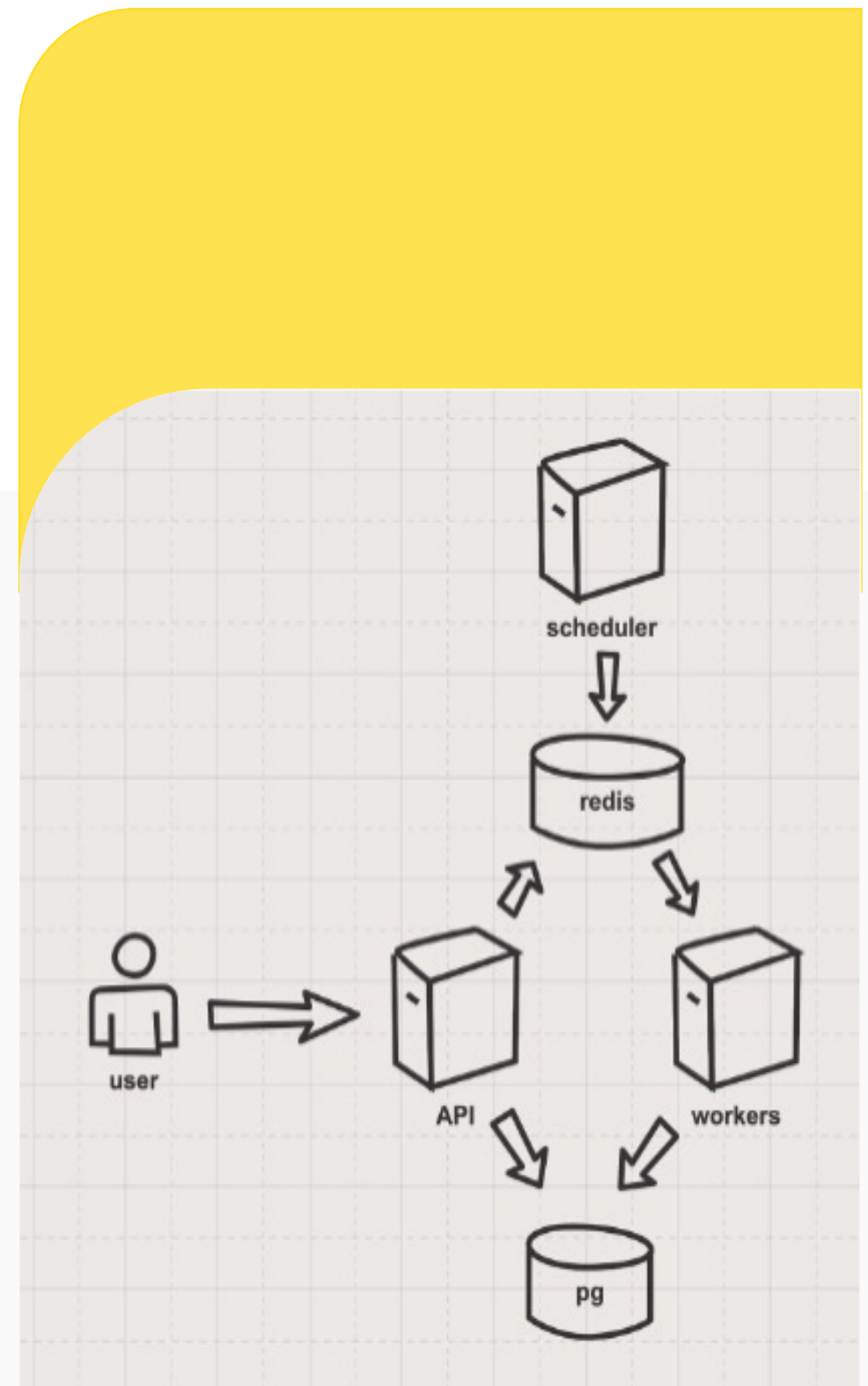
- ✓ Для тех кто работает с кодом сервисов
- ✓ Для тех кто пишет много кода и когда-либо задавался вопросом – как правильно структурировать написанный код
- Домохозяйки

Команда разработки DWH Core

- 6 человек
- основной продукт – Distributed ETL
- еще 3 небольших продукта на python
- 9 микросервисов на python + 2 на go

Характеристики микросервисов:

- 2000 – 20000 строк кода
- долгоживущие, развивающиеся и нуждающиеся в поддержке
- имеют API, имеют БД, и выполняют некоторую полезную работу – бизнес-логику



Фича-реквест к сервису

Есть сущность task.

По запросу GET tasks необходимо вернуть идентификаторы задач, подходящих под условия фильтрации.



Необходимо реализовать:

1. Метод API GET tasks
2. Запрос в БД, для получения задач, удовлетворяющих условию

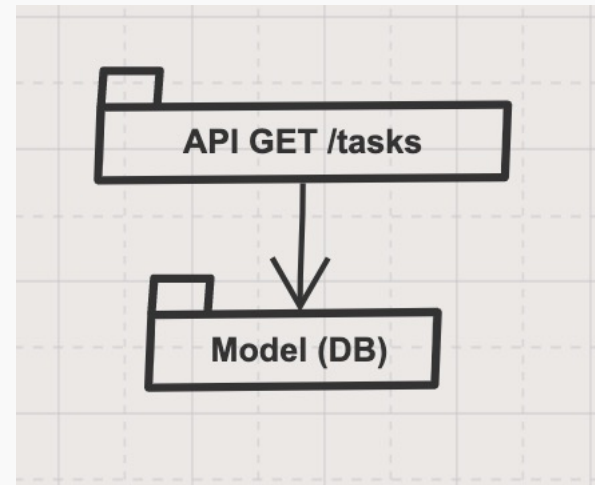
DB Model (ORM)

```
class TaskModel(Model):  
    __tablename__ = "tasks"  
  
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)  
    table = db.Column(db.String(64), nullable=False)  
    name = db.Column(db.String(64), nullable=False)
```

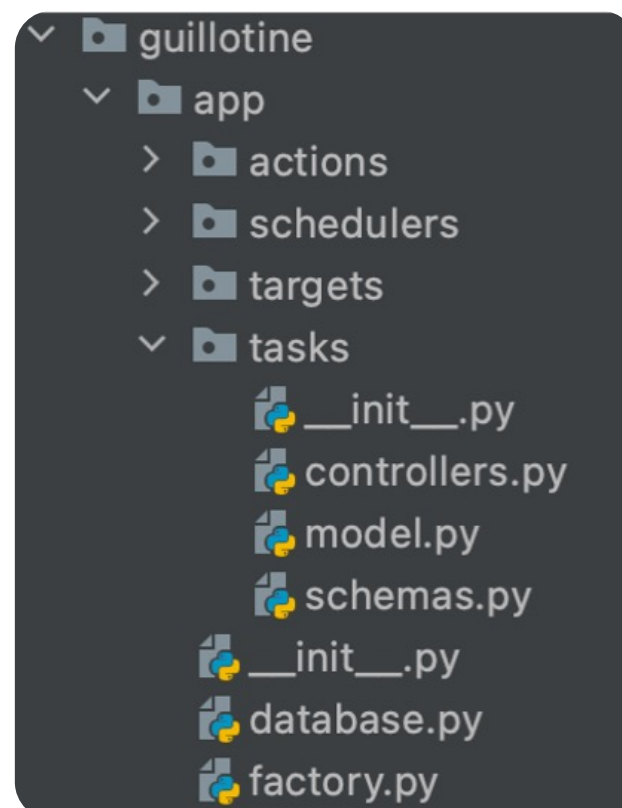
Controller

```
from app.models.task import TaskModel  
  
@TaskNs.ns.route("/tasks")  
class Tasks(Resource):  
    @TaskNs.ns.expect(TaskNs.filters_parser)  
    @TaskNs.ns.marshal_list_with(TaskNs.id, code=HTTPStatus.OK)  
    def get(self):  
        """Получить список заданий."""  
        args = TaskNs.filters_parser.parse_args()  
        return db.session.query(TaskModel.id).filter_by(**args), HTTPStatus.OK
```

Схема связей



«Плоская» структура



Правила проектирования плоской структуры:

1. Код организован на основании сущностей приложения
2. Пакет содержит все необходимое для работы с сущностью
 - Модель – ORM для работы с БД
 - Схема – структура данных, которые мы вернем в ответе
 - Контроллер – реализация реакции на действие пользователя
3. Никаких правил

Дополнительное требование

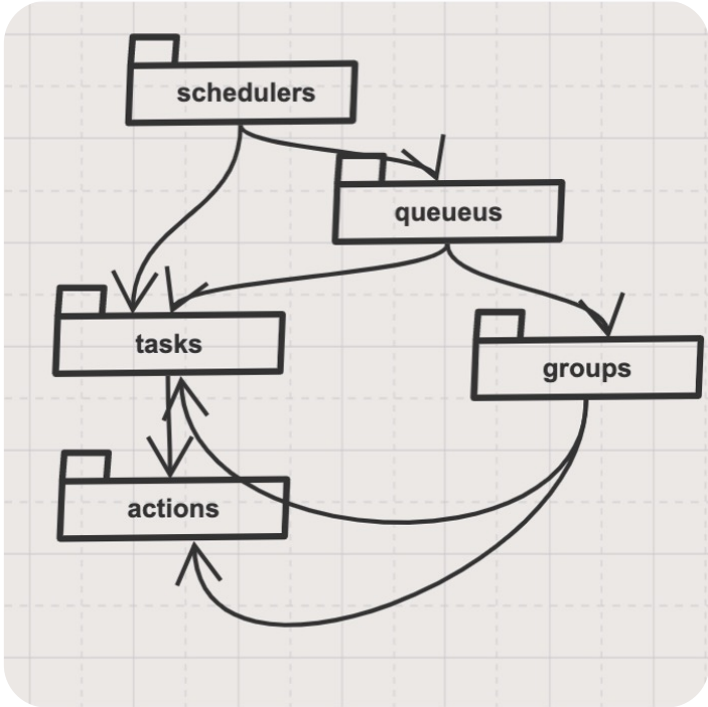
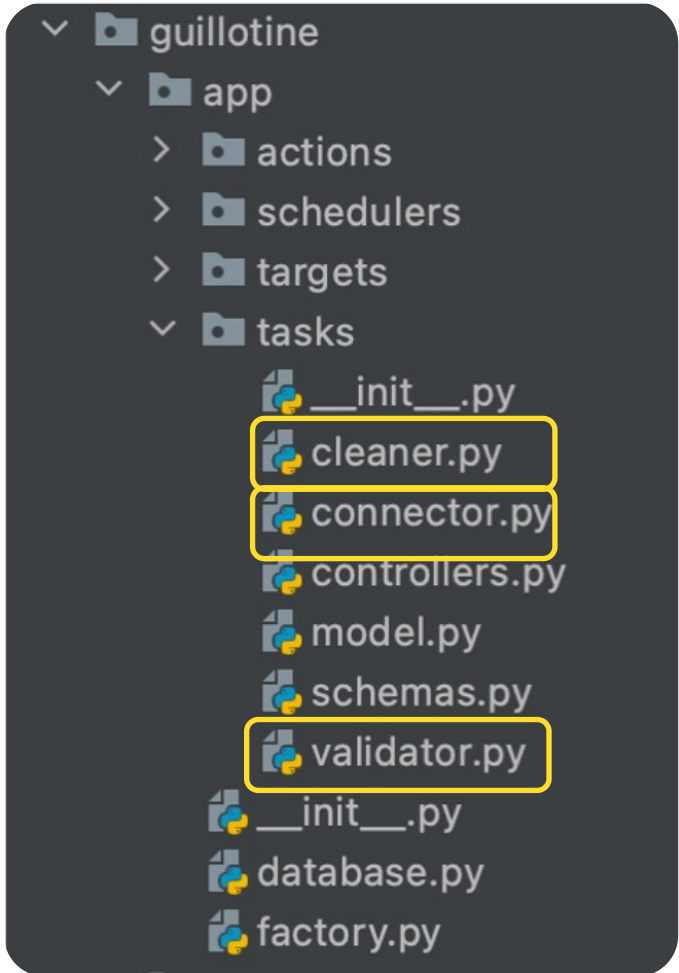
Необходимо проверить наличие таблицы в GP
Отдать в ответе идентификаторы таблиц, которые есть в GP

```
class TaskValidator:

    def validate(self, **args) -> List[int]:

        tasks = db.session.query(TaskModel.id, TaskModel.table).filter_by(**args)
        connector = GpConnector()

        return [id for id, table in tasks if connector.table_exists(table)]
```



СИМПТОМЫ



Пакет содержит множество модулей разного назначения



Расположение функциональности неочевидно



Связи между модулями запутаны










**Наличие циклических зависимостей
(взаимное использование пакетов)**



Код сильно связан

Проблемы

-  Сложная навигация по коду
-  Сложно проектировать
-  Сложный поиск багов
-  Внесение изменений затрагивают множество модулей
-  Увеличивается вероятность появления новых багов
-  Увеличивается время разработки
-  Повышается стоимость поддержки

Части приложения

1. Алгоритм бизнес логики:

- получить задачи из внутреннего хранилища
- проверить наличие таблиц в GP

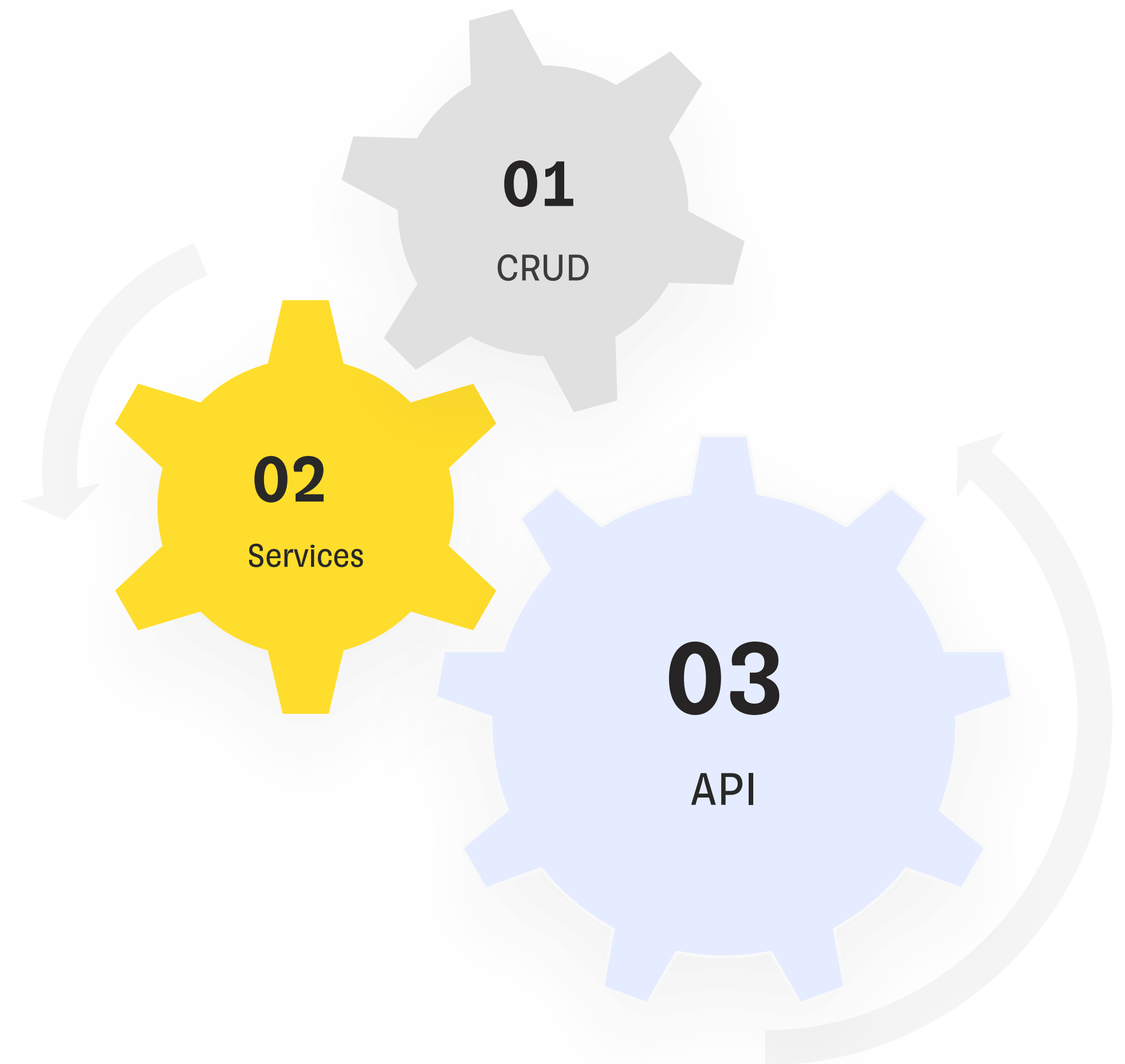
2. Пользовательский интерфейс:

- API

3. Взаимодействие с БД (внутренним хранилищем сервиса)

Выделим **слои**

- модели данных в БД
- операции по работе с БД сервиса - CRUD
- слой бизнес-логики – Services
- слой API



Адаптация кода

```
class TaskCRUD:

    def get(self, **args) -> List[Tuple[int, str]]:
        return db.session.query(TaskModel.id, TaskModel.table).filter_by(**args)
```

```
class TaskValidatorService:

    def validate(self, **args) -> List[int]:

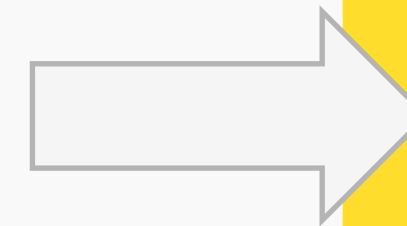
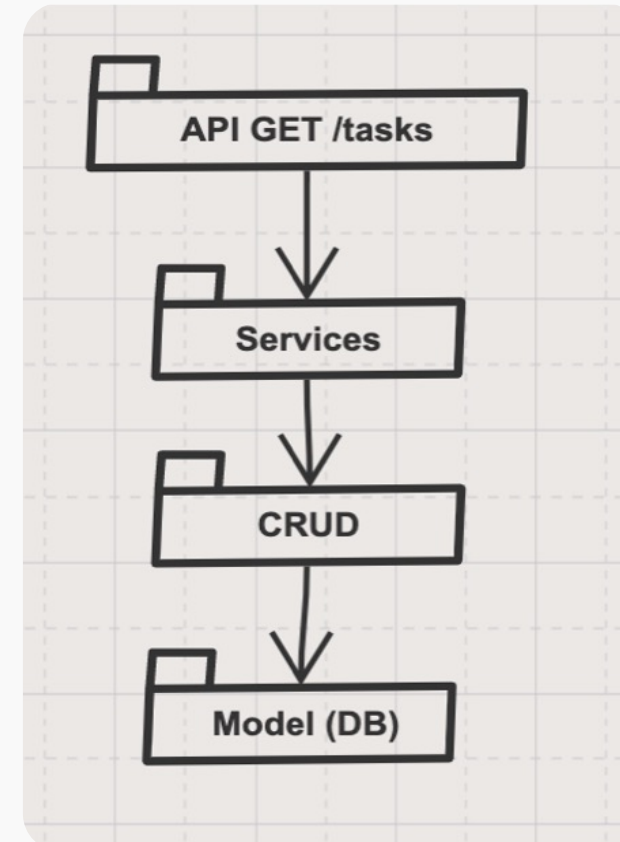
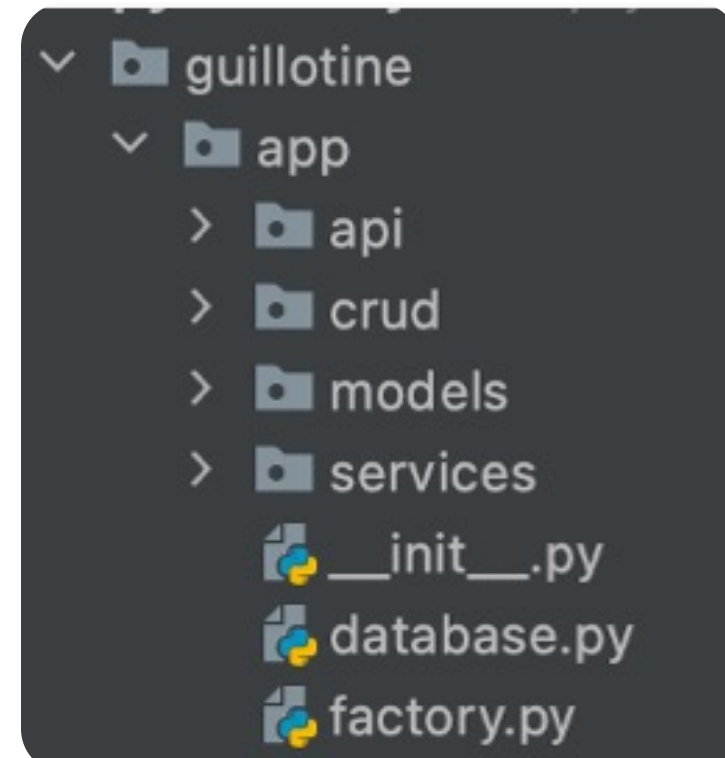
        tasks = TaskCRUD().get(**args)
        connector = GpConnector()

        return [id for id, table in tasks if connector.table_exists(table)]
```

```
@TaskNs.ns.route("/tasks")
class Tasks(Resource):
    @TaskNs.ns.expect(TaskNs.filters_parser)
    @TaskNs.ns.marshal_list_with(TaskNs.id, code=HTTPStatus.OK)
    def get(self):
        """Получить список заданий."""
        args = TaskNs.filters_parser.parse_args()
        return TaskValidatorService.validate(**args), HTTPStatus.OK
```

Правила проектирования:

- Функционал отправляется в определенный слой, в зависимости от зоны ответственности
- Бизнес-логика размещается в services
- Зависимости слоев упорядочены
- Слой зависит только от одного (нижнего) слоя



N-слойная структура

Application Layers

User Interface

Business Logic

Data Access

Health-check



Пакеты содержат модули одного назначения



Определены зоны ответственности слоев



Заданы четкие связи между слоями

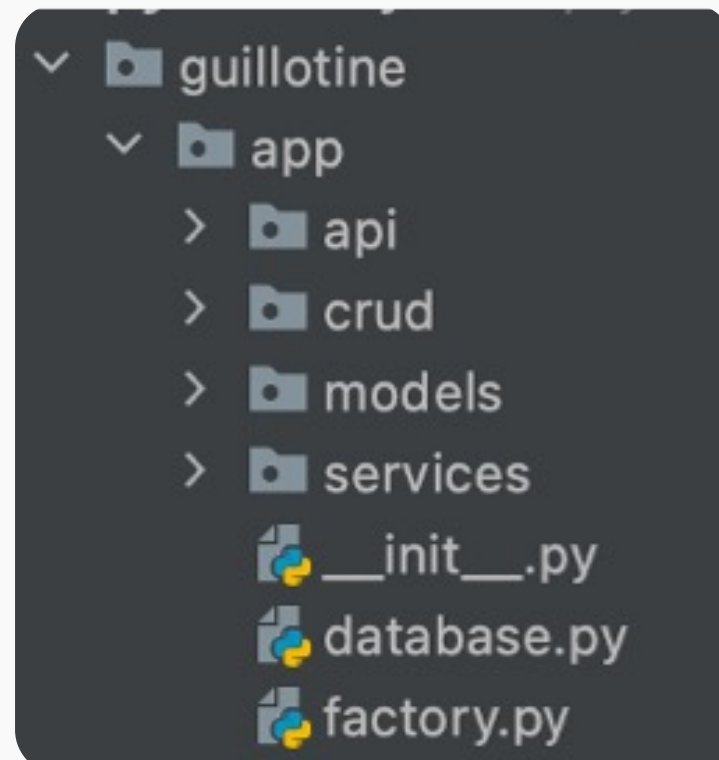


Отсутствуют циклические зависимости



Связанность кода значительно уменьшается

Добавляем требования



```
class TaskCRUD:

    def get(self, **args) -> List[Tuple[int, str]]:
        return db.session.query(TaskModel.id, TaskModel.table).filter_by(**args)
```

Предположим TaskCRUD должен вернуть схему данных, а не простые типы.

Варианты:

в слое CRUD

- Неочевидное размещение схемы данных
- Нарушений зоны ответственности слоя

в services

- Нарушается направление зависимостей
- Возникает циклическая зависимость

новый слой

- Увеличивается количество слоев
- Усложняется структура

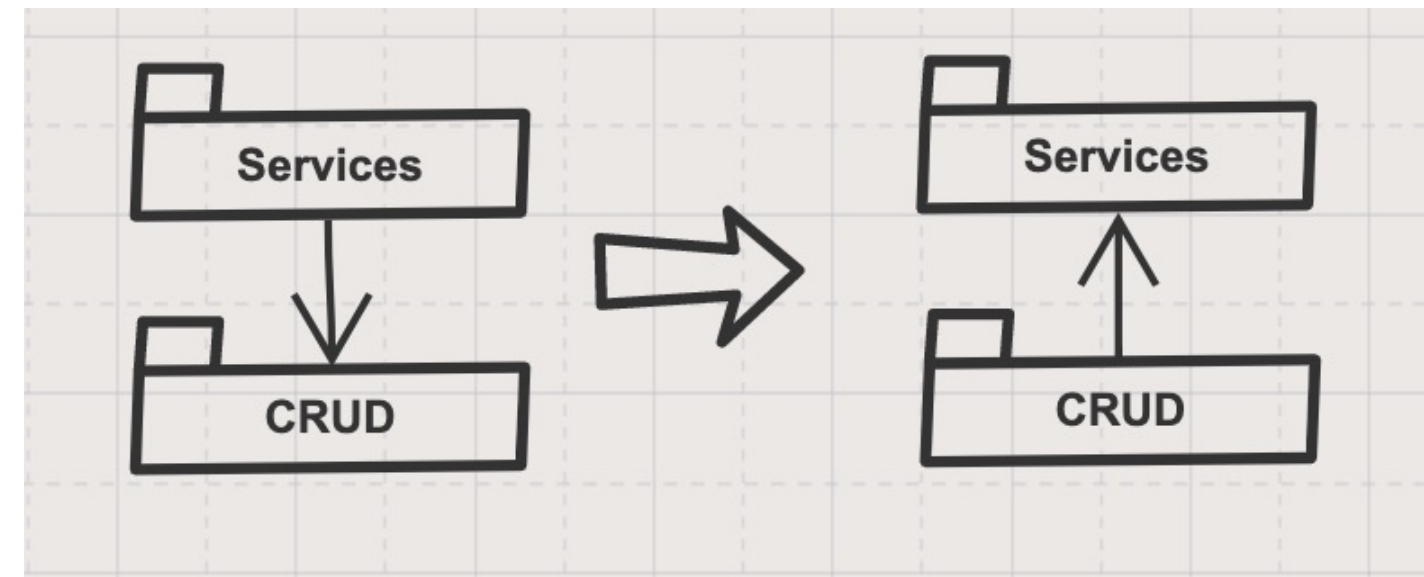
Проблемы

- Нарушается направление зависимостей
- Появляются циклические зависимости
- Остается высокая связанность CRUD->Services

Проблемы

- Нарушается направление зависимостей
- Появляются циклические зависимости
- Остается высокая связанность `CRUD->Services`

Решение



Инвертируем и внедряем зависимости

Внедрение зависимостей

Стиль настройки объекта, при котором поля объекта задаются внешней сущностью

Объекты настраиваются внешними объектами

Принцип инверсии зависимостей

Модули верхних уровней не должны импортировать сущности из модулей нижних уровней. Модули должны зависеть от абстракций

Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций

Адаптация кода

В слое services определим:

- Схему данных, которую должен вернуть CRUD
- Интерфейс для работы с БД

```
from pydantic import BaseModel

class TaskSchema(BaseModel):

    id: int
    table: str
```

```
class TaskStorageInterface(ABC):

    @abstractmethod
    def get(self, **args) -> List[TaskSchema]:
        pass
```

Реализуем интерфейс в слое CRUD

```
from app.services.interfaces.task_storage import TaskStorageInterface
from app.services.schemas.task import TaskSchema

@dataclass
class TaskCRUD(TaskStorageInterface):

    session: Session

    def get(self, **args) -> List[TaskSchema]:
        return self.session.query(TaskModel.id, TaskModel.table).filter_by(**args)
```

Инжектим реализацию в сервис

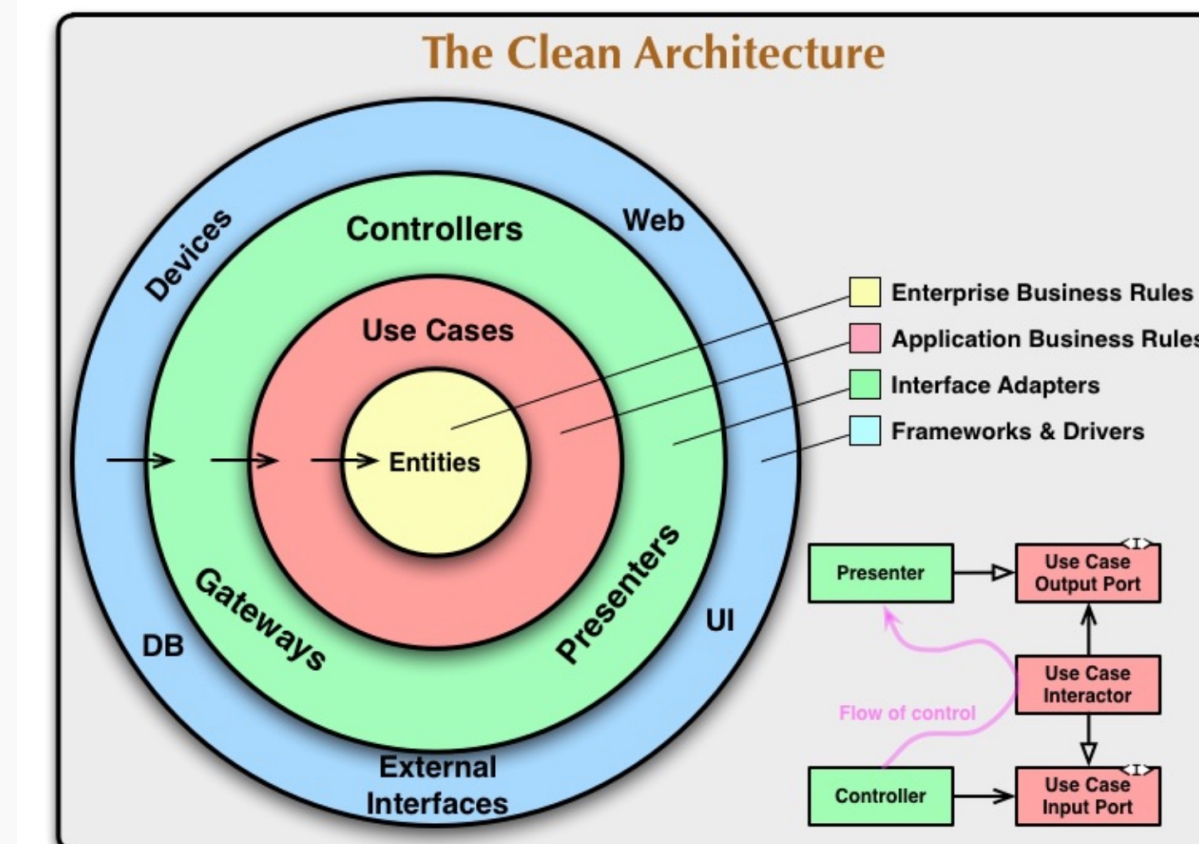
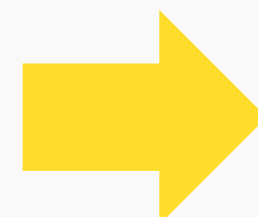
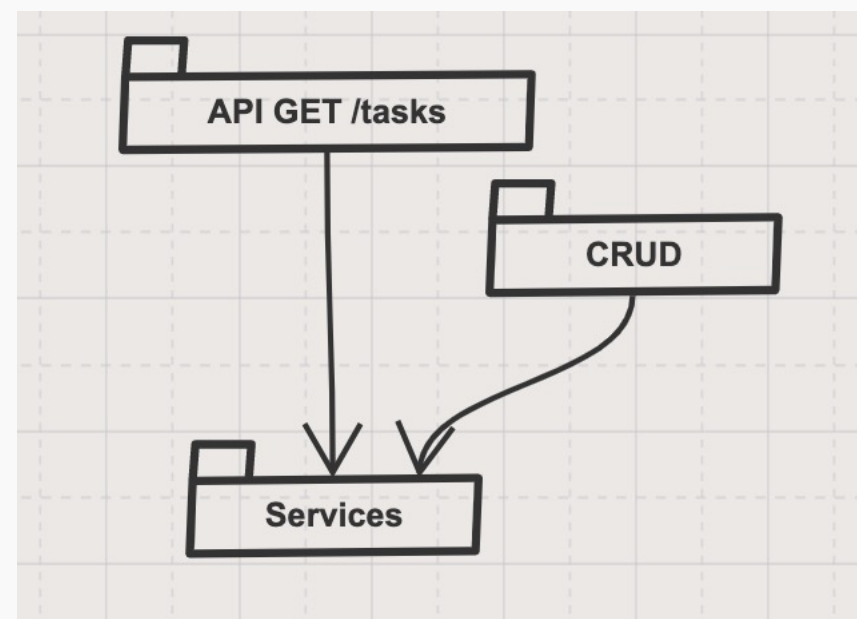
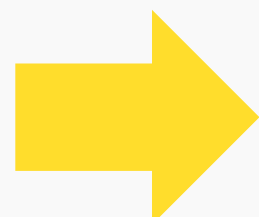
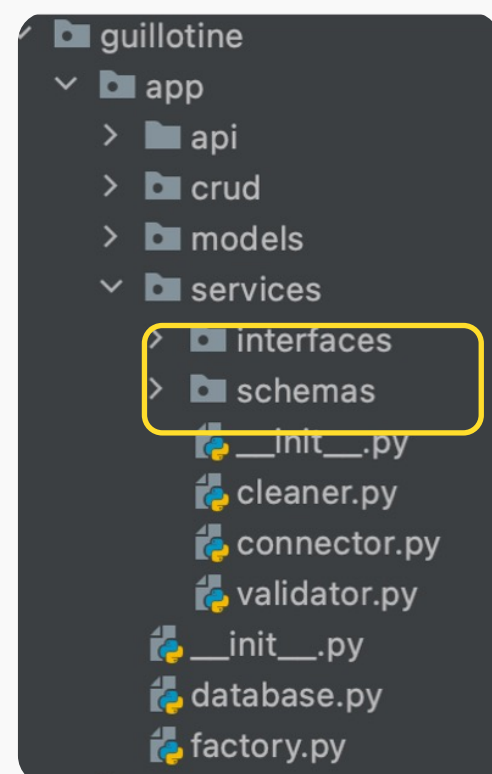
```
@dataclass
class TaskValidatorService:

    storage: TaskStorageInterface

    def get(self, **args) -> List[int]:

        tasks = self.storage.get(**args)
        connector = GpConnector()

        return [id for id, table in tasks if connector.table_exists(table)]
```



Services - ядро приложения

Содержит бизнес-модель, которая включает в себя сущности, службы и интерфейсы

CRUD - инфраструктура

Пакет инфраструктуры включает реализацию доступа к данным. Объекты инфраструктуры реализовывают интерфейсы, определенные в ядре приложения
Инфраструктура содержит ссылку на проект ядра приложения

API - слой пользовательского интерфейса

Точка входа для приложения
Пакет ссылается на слой ядра приложения

Основные правила чистой архитектуры

- Бизнес-логика важнее всего
- Ничто из внутреннего круга не может указывать на внешний круг

Бонусы



Правильный процесс работы над задачей



Удобство декомпозиции задач



Тестируемость



Схожая структура во всех сервисах



Независимость от фреймворка

Для внимательных

```
@dataclass
class TaskValidatorService:

    storage: TaskStorageInterface

    def get(self, **args) -> List[int]:

        tasks = self.storage.get(**args)
        connector = GpConnector()

        return [id for id, table in tasks if connector.table_exists(table)]
```

```
✓ adapters
  > crud
  > external
```




ТИНЬКОФФ

