Соглашения, архитектура, тесты

Solid, REST, MVC и прочие, MVT, dry, kiss, CAP-теорема. Тесты на версии, fixture, coverage.

Клоков Стас 13.04.2023

SOLID

Single Responsibility

Принцип единства ответственности

- Класс решает только одну задачу
- Для его изменения есть только одна причина
- Не храним то, что к нам не относится
- Проще версионируемся - меньше конфликтом
- Если не соблюдается единство задачи - декомпозируем

Open-closed

Принцип открытости-закрытости

- Открытость для расширения
- Закрытость для модификации
- Не дает рушить логику
- Позволяет реализовывать совместимость

Liskow Substitution

Принцип подстановки Барбары Лисков

- Подклассы должны уметь передаваться туда же, производными кого они являются

Interface Segregation

Принцип разделения интерфейсов

- Много интерфейсов вместо одного - не плохо
- Один общий на все и сразу плохо
- Основные плюсы - гибкость и расширяемость
- Легко находить зоны ответственности

Dependency inversion

Принцип инверсии зависимостей

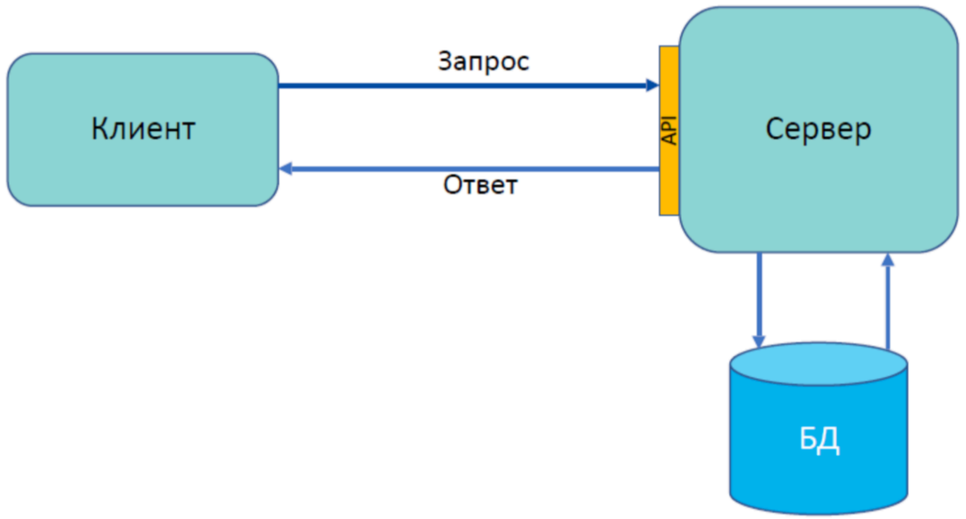
- Классы должны зависеть от интерфейсов и абстрактных классов
- Классы не должны зависеть от конкретных классов и функций

REST

Клиент-сервер

Идея о разделении зон ответственности

- Клиент - исполнение логики взаимодействующего
- Сервер - исполнение логики, связанной с самой системой



Stateless

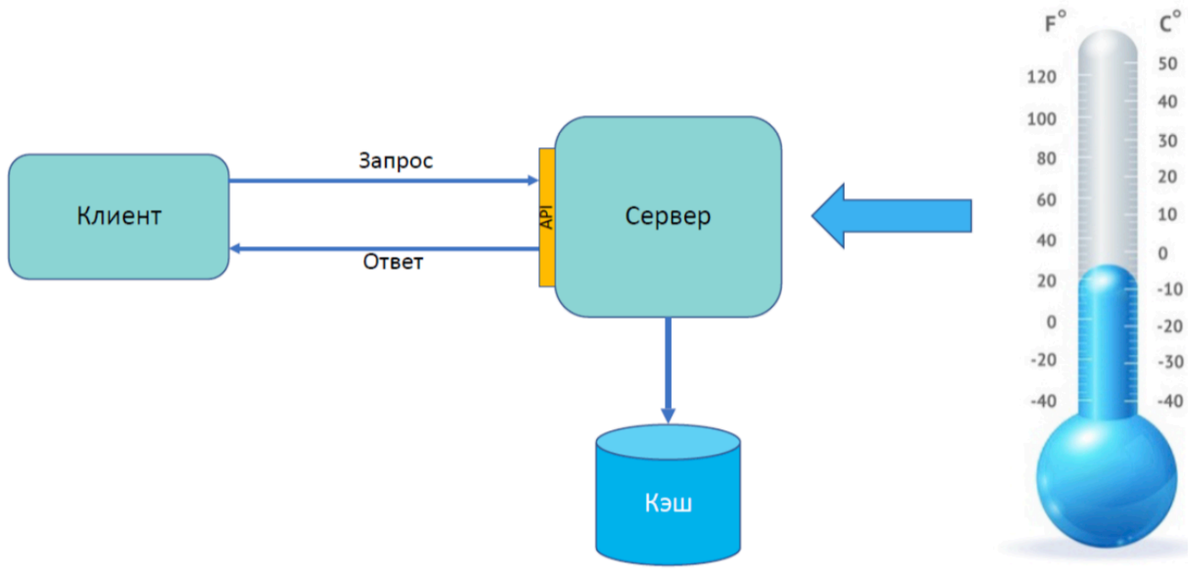
Сервер не должен хранить информацию о сессии с клиентом.

- Для этого клиент бросает сессионные куки/токены
- Сервер при каждом запросе не должен завязываться на состояние клиента, а должен получать всю информацию
- Запрос по своей сути полноценный набор данных о происходящем событии

Кэширование

Каждый ответ сервера должен сказать можно ли его кэшировать

- Мы не должны кэшировать то, что нельзя, чтобы не нарушить целостность и достоверность



Единообразие интерфейсов

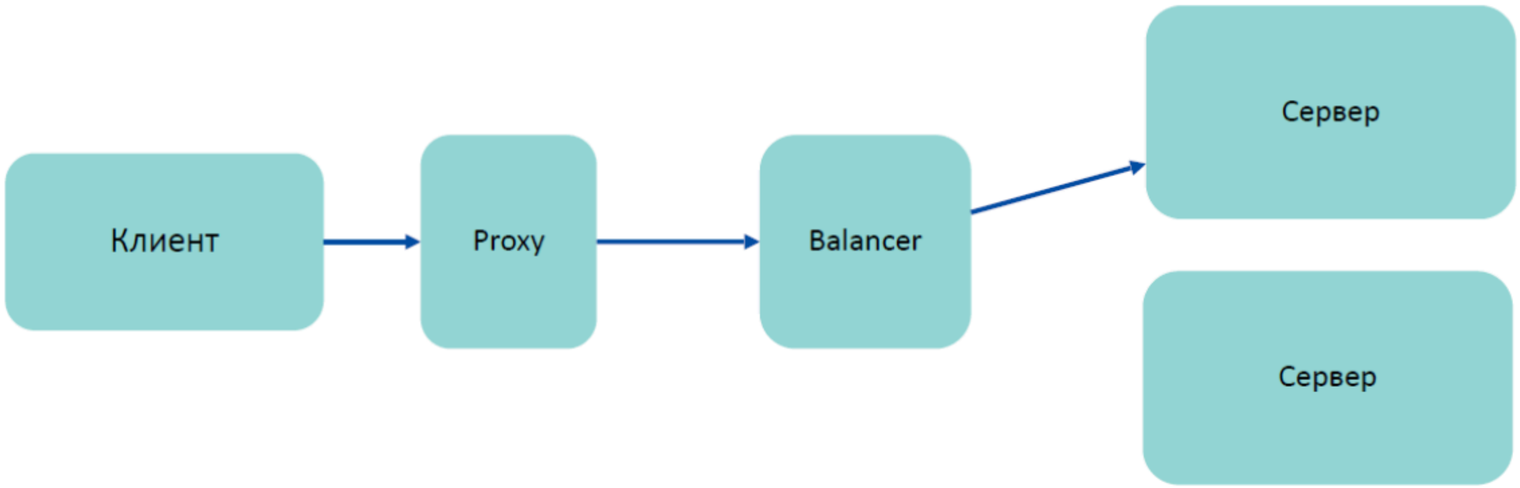
Мысль в том, чтобы были стандарты для проекта

- Клиент всегда понимает что он может делать
- Клиент видит связи по возможности
- Работать с изменениями в системе легко

Слоистая архитектура

Клиент и сервер не знают о цепочке ничего дальше соседей

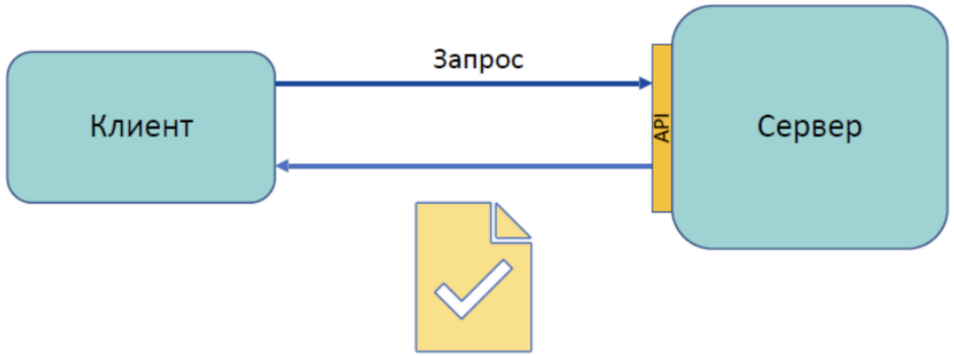
- Посредники(proxy) не должны влиять на общие представления не взаимодействующих с ними частей.



Код по требованию

Меняем если надо на сервере и отдаем клиенту

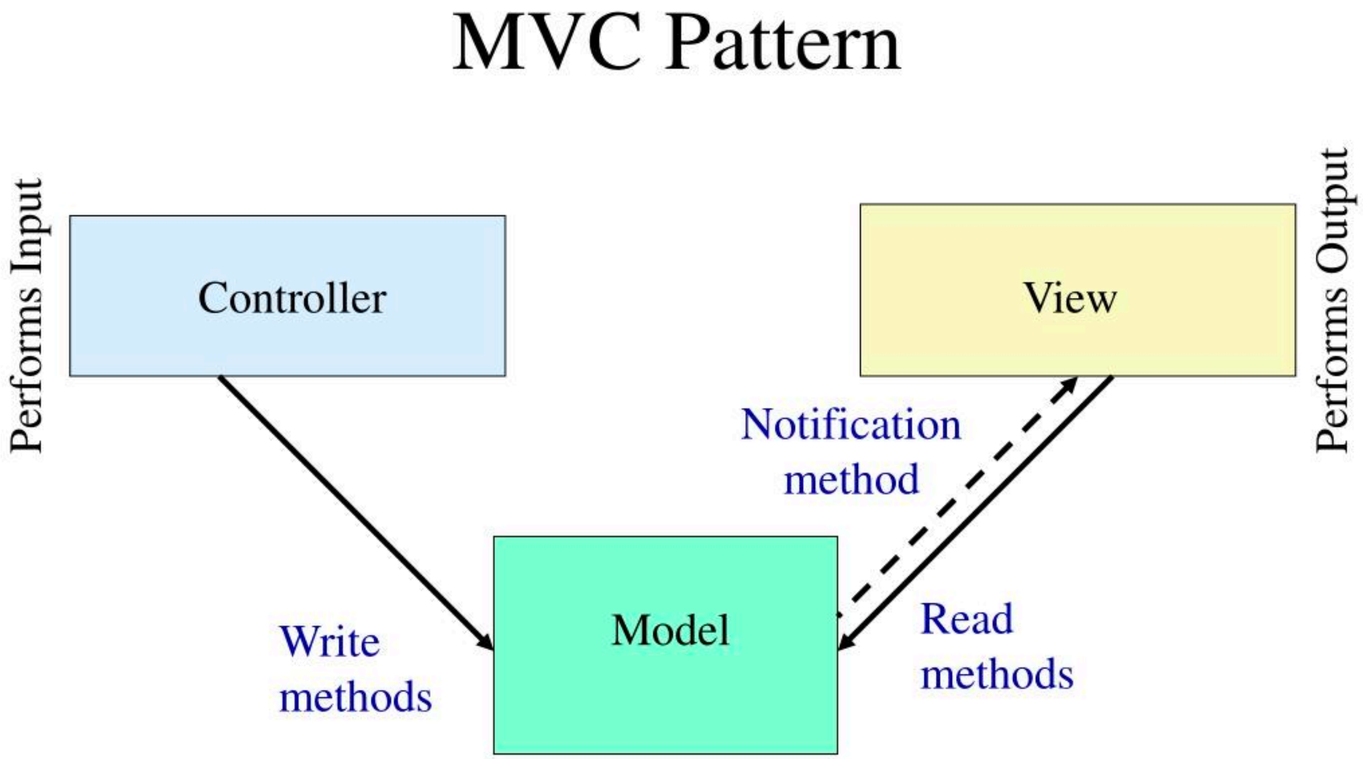
- Пример - работа с интерфейсами из бадлов js.



MVC и прочие

MVC

Model-view-controller



MVVM

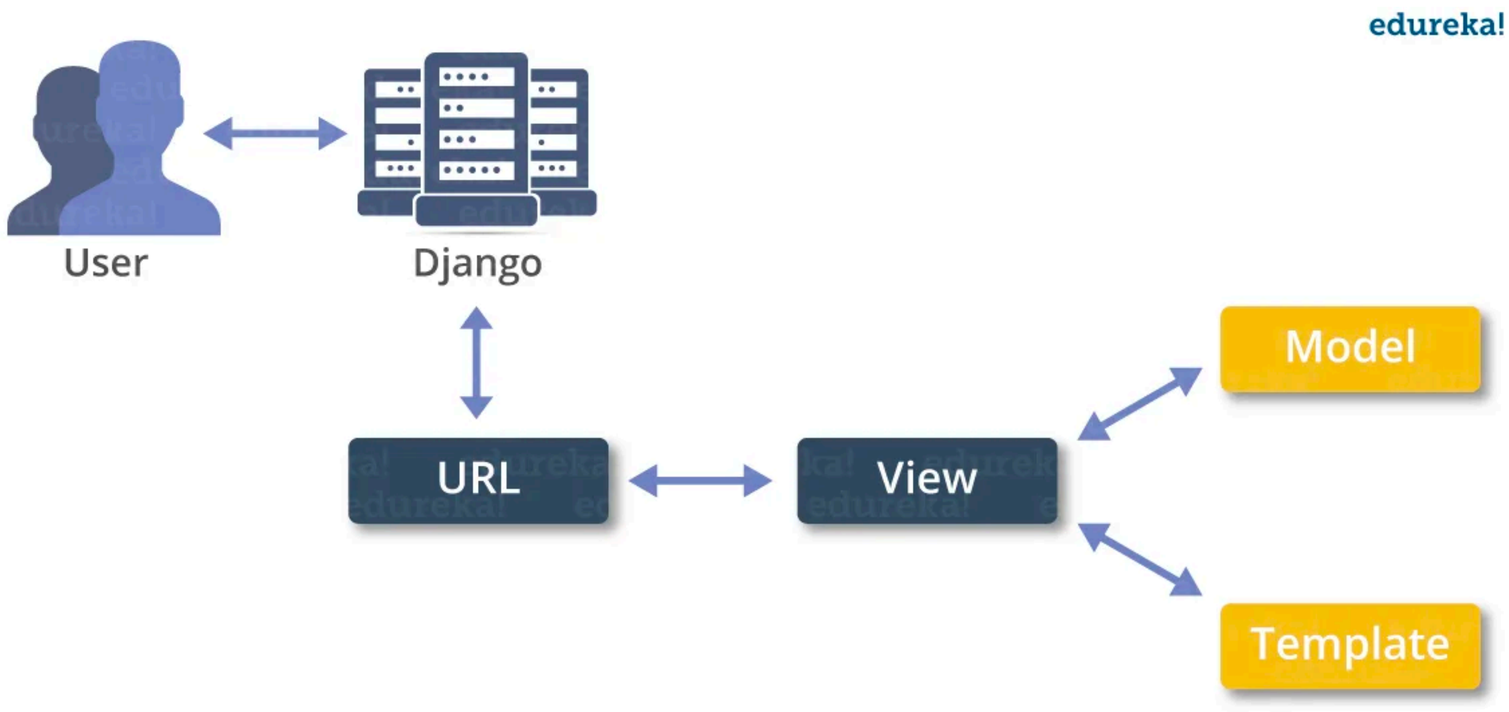
Model-view-viewmodel

MVVM

MODEL VIEW VIEWMODEL

MVT

Model-view-template



-

Dry

DRY

Don’t repeat yourself

- Дублирование кода - трата ресурса и размазывание ответсвенности.
- Наша задача избегать подобного и не повторять уже готовые решения.
- Самая частая ошибка - плохое знание системы.
- Чтобы знать систему хорошо - пишите изначально по DRY))

Kiss

KISS

Keep it simple, stupid

- Самое хорошее решение - самое тупое по-возможности.
- Здесь важно помнить, что тащим пакеты только ради необходимости/удобства, но не ради хайпа.
- Не усложняем решение скоростными подходами если оценка сложности на наших данных не так важна (ожидаем, что размеры данных гарантированно мелкие - зачем мучаться).
- Сюда же идея, что время разработчика дороже времени машины. Не пишите жесть, которую надо разбирать пол дня.

CAP-теорема

CAP

Теорема

- Теорема гласит, что в распределенной системе можно выбрать только 2 из трех свойств и гарантировать их выполнение.
- О гарантии говорим условно, так как например доступность у нас не измеряется никаким параметром и частичную доступность мы не рассматриваем, но можем думать про нее в каких то ситуациях.

Cap

Consistency

- Согласованность данных.
- Это свойство гарантирует, что каждое чтение дает последнюю запись.

cAp

Availability

- Каждый живой узел всегда успешно выполняет запросы на чтение и запись.
- То есть каждый не упавший узел системы обязан быть доступен.

caP

Partition tolerance

- Это свойство означает устойчивость к распределению.
- Здесь мы говорим, что все узлы распределенной системы работают независимо вне контекста существования связи между ними.

Примеры

О применении теоремы и частных случаях

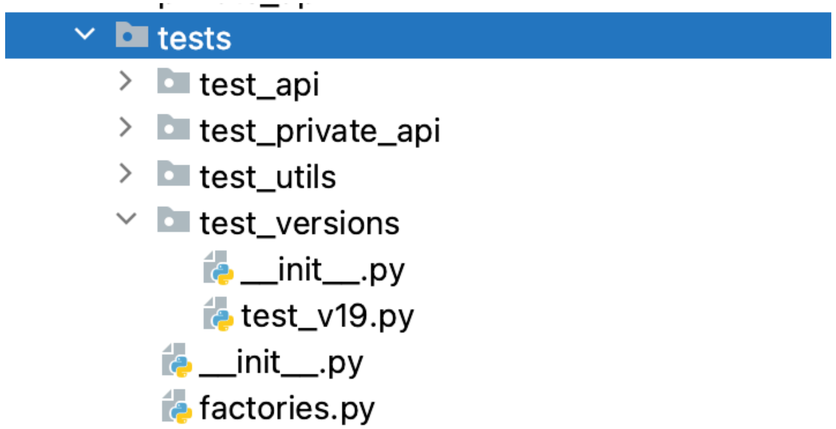
- Postgresql - CA
- MongoDB - CP
- Большинство систем - P
- CA рассматриваем редко в рамках разработки, потому что почти везде есть асинхрон. Между CA и CP скорее выберут CP.

Тестирование

Тестирование версий

Как мы это делаем и что оно дает

- Мы формируем тесты на апи, содержащие актуальный набор тестов для последней версии.
- Далее мы версионируем этот файл под управлением VCS проекта, но при этом старые кейсы выносим в директорию с тестами на версию конкретных кейсов.



Fixture

Фикстуры чуть ближе - как выглядят

- Фикстура - сущность, нужная для эмуляции работы нужных нам для тестирования объектов.

```python
@fixture(autouse=True)
def mocked_opis_upload(mocker):
    mocked_upload = mocker.Mock(return_value=DEFAULT_OPIS_API_RESPONSE)
    return mocker.patch.object(opis_api, 'upload', mocked_upload)
```

Coverage

Как думать о покрытии и не пропустить кейсы

- 1) Покрываем кейсы, где функция не должна запускать алгоритм
- 2) Покрываем кейсы, зависящие от окружения
- 3) Покрываем кейсы, которые имеют тривиальное разрешение
- 4) Покрываем кейсы, которые описывают ключевые случаи
- 5) Покрываем кейсы, которые описывают логически важные ветвления
- 6) Покрываем переборы кейсы

Методы покрытия

О том, что из теории может нам помочь

- Попарное покрытие - покрытие тестами таким образом, что тестовые кейсы на каждый аргумент перебирают все возможные пары.

```python
x_cases = [1, 2, 4]
y_cases = [(1, 2), (), (4), (1, 2, 4), (6, 7)]



@pytest.mark.parametrize("x", x_cases)
@pytest.mark.parametrize("y", y_cases)
def test_solver(x: int, y: tuple) -> None:
    received = solver(x, y)
    assert x in y == received
```