Django Rest API

API - *Application programming interface,* контракт, который предоставляет программа, согласно которому ваша программа обязуется делать определенные действия, в случае совершения определенных запросов (действий)

Необходимость

Наличие API у вашего ресурса, повышает количество клиентов, а так же лояльность к вашему ресурсу

Способы реализации Rest API

1. Используя только Django — не самый удобный и гибкий вариант на данный момент, так как придется производить сериализацию данных самостоятельно

2. Используя дополнение — самый удобный и гибкий вариант на данный момент, самое популярное дополнение — **djangorestframework**

Установка DjangoRestFramework

pip install djangorestframework

Регистрация

Необходимо добавить **rest_framework** в список **INSTALLED_APPS**

Проектирования API

Хорошей практикой считается, если ваш URL запрос содержит только сущности из вашей БД, никаких строк, указывающих на действие, которое необходимо произвести с сущностью не должно быть, действие определяется на основании HTTP запроса

GET https://some-site.com/categories — получение категорий (GOOD JOB)

GET https://some-site.com/get_categories — BAD!

POST https://some-site.com/categories — добавление новой категории (GOOD JOB)

POST https://some-site.com/add_category — BAD!

Так же URL запрос не должен содержать **query string**, это считается плохой практикой

GET https://some-site.com/categories/1/ - получение категории с ID = 1 (GOOD JOB)

GET https://some-site.com/categories?category_id=1 — BAD!

Сущности указанные в начале запроса, называются корневыми (независимыми)

Сущности, указанные после другой сущности в URL называются зависимыми (от корневой) сущностями

GET https://some-site.com/categories/1/ — корневая сущность (данные о 1 категории)

GET https://some-site.com/categories/1/products/ - получение товаров 1ой категории (связанная сущность)

Нумерация версий API

API необходимо нумеровать, это позволит работать клиенту стабильно и не бояться изменений, если вы выпустили версию API, то в данной версии никаких изменений не должно уже производится, потому, что клиент уже написал код под ваше API, и если вы его изменили, ему придется править свой код, что снижает лояльность к вашему сервису.

Поэтому API принято нумеровать, хорошей практикой является нумерация API прямо в URL запросу

https://some-site.com/api/v1/…

https://some-site.com/api/v2/…

Good Experience

Список хороших решений в проектировании API

1. SSL везде, без нее не возможна авторизация и аутентификация

2. Версионность и документация со старта разработки

3. GET и POST должны возвращать обратно объект, который изменили или создали — снижает количество обращений к сервису в двое

4. Использование стандартных HTTP кодов запроса

Сериализация моделей

Все сериализаторы принято помещать в файл **serializers.py**

Для удобно сериализации данных связанных с моделью, удобным решением является использование класса **ModelSerializer**, он автоматически генерирует поля на основании модели, а так же автоматически генерирует валидаторы и простые реализации методов **create/update**

Описание сериализатора на основании модели очень похоже на описание формы связанной с моделью

```python
from rest_framework import serializers

from .models import Category


class CategorySerializer(serializers.ModelSerializer):
    class Meta:
        model = Category
        fields = ('id', 'title', 'descr')
```

Класс **Meta** так же может принимать атрибут **exclude** для указания атрибутов, которые не будут сериализованы, а так же можно указать значение **«__all__»** в качестве значения атрибута **fields**, если необходимо сериализовать все атрибуты модели

ModelViewSet

```python
class CategoryViewSet(ModelViewSet):
    queryset = Category.objects.all()
    serializer_class = CategorySerializer
```

Класс отвечающий за представление сериализованных данных на основании модели

Диспетчеризация URL

```python
    path('api/categories/', CategoryViewSet.as_view({'get': 'list'})),
```

Представления сериализатора, работает аналогично как и любой класс представления, за исключением того, что необходимо передать действие на вход метода **as_view** но данный способ не очень удобен, так как один

и тот же класс представления будет фигурировать сразу в нескольких URL запросах, для более удобной работы, существует **routers**

```python
from rest_framework import routers

from .views import CategoryViewSet

router = routers.SimpleRouter()
router.register(r'categories', CategoryViewSet)


urlpatterns = [
    path('api/', include(router.urls)),
```

Стоит обратить внимание, что данный подход к реализации REST API с использованием дополнительной библиотеки, сразу реализует нам набор API запрос, в том числе на добавление/изменение/удаление записей, а так же на получение конкретной записи по ID