Структура проекта

/

/app

/core – (Users, Auth, Groups)

....

/config

/settings

__init__.py (local -> common\settings)

common.py (settings.py)

local.dist.py

local.py – (in .gitignore)

/templates

- Не создавать отдельно management/commands, создавать только в конкретном приложении.
- Все шаблоны хранить в одной папке.

Структура common.py (settings.py)

INSTALLED_APPS = [

… Все что идет из коробки …,

… Сторонние приложения …,

… Внутренние приложения …,

]

- Порядок по алфавиту (В рамках группы приложений)
- Разделять на отдельные переменные не надо

- Авторизационные\Секретные\Любые другие Ключи, которые могут давать доступ к сторонним приложениям, использоваться для шифрования\кодировки не храним в common.py (Допускается хранения в local.py)

- Прописать *AUTH_USER_MODEL* прописывать на модельку юзера.
- Импортировать через *get_user_model()*

- Если есть необходимость расширения пользователя, то наследуемся от *AbstractBaseUser*.

Импорты

- Полный абсолютный путь
- Прим *from app.core.models import SomeModel*
- Звездочки не используем

Именование

- Константы – Капсом (прим. *TIMEOUT = 30*)
- Модели – В единственном числе, без лишних слов (прим. *Book*).
- Вьюхи – Если вьюха наследуется от *DRF* вьюх то добавляется *ViewSet* (прим. *UserViewSet*), если наследуется от вьюх из джанги, то просто *View* (Прим. *LoginView*)
- Сериализаторы – Использовать в именовании уровень масштаба(?)
    - UserBaseSerializer – id, name; (id + поле для отображения)
    - UserSerializer (UserBaseSerializer) – id, name, email, username;
    - UserDetailedSerializer(UserSerializer) – id, name, email, username, phone, address, last_login;

Реиспользуемые функции

- Придерживаться принципа *DRY (Don't repeat yourself)*
- Общие вещи в рамках app складывать в *utils.py* данного app, если функция\класс используется в разных *app*, складывать в *app.core.utils*.

- В случае если *CHOICES* один и тот же набор используется за пределами одной сущности, скажем модели, имеет смысл все чойсы складывать в *app.core.choices*
- Набор методов используемых для схожих вещей оформлять в класс

Миграции

- Не удаляем никогда миграции
- Миграции делаются в главной ветке(develop), одним человеком.

По возможности, любые новые классы\функции добавлять в конец файла.

.gitignore

- local.py
- .idea/
- __pycache__/
- db.qslite3
- static/
- media/

Админка

- Для регистрации админки использовать декоратор *admin.register*

Условные операторы

- вместо конструкции if elif по возможности использовать dict;
- избегать множественных *return* в функций (Использовать тернарный или логические операторы).

Строки

Одинарные кавычки для всех случаев, кроме конкретной работы со строкой.

- values.get(‘abcd’);
- values = {‘a’: 1};
- a = ‘qweqweqwe’;
- get_something(‘tralala’).

Двойные кавычки используются для конкретной работы со строками как с текстом.

- f”Tralalala at {time}”;
- email_text = “ You fired ”.

Именование перменных

- Придерживаться PEP8.

Пример перечисления полей в сериализаторе, вьюх и т.д.

```
fields = [
    ‘id’,
]
```

GIT

- Создается ветка develop, только для готовых изменений
- На каждую таску отдельна ветка

Обсуждение

Регистр методов декоратора *action (DRF, ViewSet)*

Мерж реквест на 2-ух человек