САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Web-разработка

Отчет

Лабораторная работа №3

Выполнил:

Егоров Мичил

Группа

К33401

Проверил:
Говоров А. И.

Санкт-Петербург

2021 г.

Задача

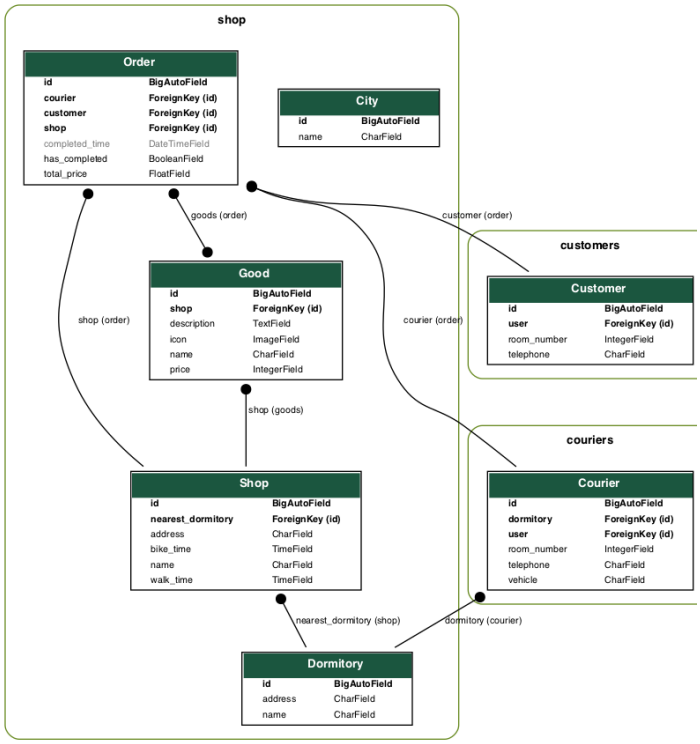
Реализация серверной части на django rest. Документирование API.

Проект: Delmitary (Сервис доставки/заказа товаров для жителей общежитий)

Клиенты смогут договориться с сожителями общежития привезти товар/еду из ближайших магазинов за символическую плату прямо до двери команты. Есть несколько ролей: курьер, заказчик, администратор сайта. Заказчики имеют доступ к списку магазинов и их товаров, могут добавлять их в корзину. После набора корзины клиент оставляет запрос. Курьерам предоствляется доступ к списку заказов с именами/телефонами/описаниями/корзинами, где они смогут решить какой заказ брать или не брать вообще Наполнением и актуализацией информации занимаются администраторы сервиса У всех ролей есть возможность редактировать личную информацию, помимо этого у курьеров есть график работы (смены) доставки товаров. Администратор может уточнять и редактировать информацию курьеров/клиентов.

Ход работы

Определим UML



Создадим виртуальное окружение *delmitary* и установим нужные библиотеки (список находится в файле bin/requirements.txt)

Создадим проект

```
$ django-admin startproject delmitary
```

А также приложения shop, couriers, customers, где будем разрабатывать соответствующие функциональности.

Routers

DRF пути будем прописывать в файле config/routers.py

Swagger

Добавим swagger к проекту используя библиотеку drf_yasg и определим его по адресу /api/swagger/

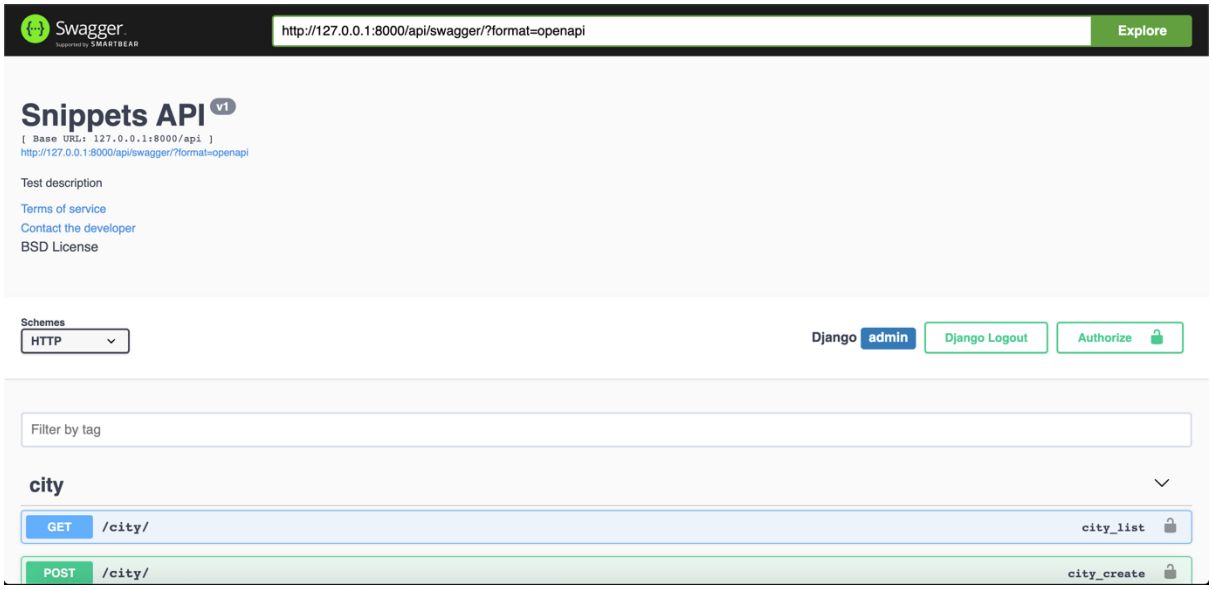


Рисунок 1. Swagger для ручек проекта.

Serializers

Для каждой сущности пропишем его сериализатор. Нужно переопределить класс Meta для наследников ModelSerializer, где указывается атрибут model и fields

Там, где нужно, можно определить нужные поля, например prices_from (минимальная цена среди товаров внутри магазина) для класса CompactShopSerializer

Filters

Для некоторых ручек list определим FilterSet, например CourierFilter для получения списка курьеров.

Можно сортировать по:

- Логину;
- Общежитию;
- Типу передвижения (пешком, на велосипеде, на машине);
- Есть ли рабочая смена в данный момент времени.

Так же фильтры определены для ручки заказов и магазинов

View

Для всех сущностей определим viewsets.ModelViewSet, для того, чтобы определились все нужные ручки

Определим класс GenericSerializerClass, чтобы можно было писать какие именно сериализаторы использовать при конкретных запросах. Например чтобы переопределить сериализатор для получения единичного объекта, нужно определить в классе атрибут retrieve_serializer_class (например это реализовано в классе CourierViewSet, чтобы попутно возвращать смены когда курьер может работать)

Для объектов, у которых нужно возвращать в ручках атрибуты связанные с помощью ManyToMany или ForeignKey определим prefetch_related и select_related. Это нужно для того, чтобы делать join внутри одного запроса.

Важные моменты:

1. При создании заказа дополнительно в базу сохраняется общая стоимость. Это сделано для того, чтобы каждый раз не обращаться за продуктами для высчитывания цены, например, для анализа данных

2. При добавлении новой смены дежурного нужно проверить, есть ли пересечение уже с существующим рабочим графиком дежурного.
3. При создании заказа можно указать только поля shop, customer и goods, так как все остальные пока не известны и будут подсчитаны потом.
4. Запрос для получения списка курьеров, которые могут работать в данный момент, происходит за один запрос за счет prefetch_related.

Вывод

Создали проект DjangoRESTFull, задокументировали с помощью swagger, написали фильтры, сериализаторы, валидаторы входных данных и ручки доступа.