Geekbrains

**Разработка CRM системы для сервисной компании на языке Python и фреймворках Django и Django Rest Framework**

IT-специалист:

Программист Python Цифровые профессии Черепанов Е.А.

Москва

2024

Содержание

Оглавление

[Введение 4](#_Toc159074056)

[Назначение и цели создания CRM системы. 7](#_Toc159074057)

[Структура приложения и требования к функциональности 7](#_Toc159074058)

[Формирование базы данных приложения 10](#_Toc159074059)

[Установка и настройка Django Rest Framework. 11](#_Toc159074060)

[Что такое Django Rest Framework 11](#_Toc159074061)

[Установка и настройка 14](#_Toc159074062)

[Настройка административной панели 16](#_Toc159074063)

[Кастомизация панели администрирования. 17](#_Toc159074064)

[Описание структуры и функций приложений проекта 18](#_Toc159074065)

[1. Приложение «user» 18](#_Toc159074066)

[Модели 18](#_Toc159074067)

[Сериализаторы 20](#_Toc159074068)

[Представления данных модели 25](#_Toc159074069)

[2. Приложение «staff» 27](#_Toc159074070)

[Модели 27](#_Toc159074071)

[Сериализаторы 29](#_Toc159074072)

[Представления данных модели 35](#_Toc159074073)

[3. Приложение «client-service» 39](#_Toc159074074)

[Модели 39](#_Toc159074075)

[Сериализаторы 40](#_Toc159074076)

[Представления 44](#_Toc159074077)

[4. Приложение «Common» 49](#_Toc159074078)

[Миксины для моделей 49](#_Toc159074079)

[Миксины для сериализаторов 51](#_Toc159074080)

[5. URLs 52](#_Toc159074081)

[6. Пагинация 53](#_Toc159074082)

[7. Итоговое представление REST API 55](#_Toc159074083)

[Дальнейшее развитие проекта 57](#_Toc159074084)

[Заключение 58](#_Toc159074085)

[Список литературы 59](#_Toc159074086)

# Введение

Что такое CRM — определение и расшифровка

**Customer Relationship Management (CRM)**— система, которая помогает выстроить отношения с клиентами и следить за совершаемыми сделками.

Программа собирает информацию о клиенте в электронную карточку, учитывая все действия:

момент заявки → консультацию со специалистом → визит в магазин → покупку → доставку

Работая в CRM, менеджер наглядно видит историю взаимодействия с каждым клиентом. В ней отображаются все сообщения и разговоры, ФИО, номера телефонов, email. Однако CRM выступает не только как хранилище данных. Она также подсказывает менеджеру или владельцу бизнеса, как лучше связываться с тем или иным клиентом, какова скорость ответа и качество сделки. Система собирает всю аналитику в отчеты, помогает работать с воронкой продаж, планировать дальнейшее взаимодействие с покупателями.

Используя CRM, вы больше не будете вручную составлять клиентские базы в электронных таблицах. Программа все сделает за вас, а также напомнит, когда клиенту пора отправлять оповещение о заказе или информацию о новинках.

[CRM](https://www.bitrix24.ru/features/crm/) — большой комплекс функций, включающий маркетинг, техподдержку, продажи, аналитику, коммуникацию с клиентами. Система автоматизирует основные процессы бизнеса.

Данная система будет строиться под нужды небольших сервисных компаний, занимающихся продажами оборудования, запчастей и поддержкой гарантийных обязательств.

Выбор СRM это ответственный шаг, определяясь с которым стоит учитывать не только текущие цели, но и то, что нам может потребоваться в перспективе. В этом плане фреймворк Django выступает универсальным инструментом для разработки веб-приложения. Вне зависимости от того, что вам необходимо в настоящее время, в будущем уже существующий ресурс, можно изменять, дополнять, видоизменять.

Определимся с понятием, что такое фреймворк? Академия Яндекса говорит нам, что это готовый набор инструментов, который помогает разработчику быстро создать продукт: сайт, приложение, интернет-магазин, CMS-систему. Звучит неплохо, а что там с Django. Django считается один из самых популярных.

Благодаря такой гибкости фреймворк завоевал сердца многих компаний. В настоящее время на данной платформе функционируют такие гиганты, как: YouTube, Google, Pinterest, Reddit и другие.

Почему компании выбирают Django в качестве фреймворка для написания больших и сложных проектов. Отметим основные преимущества Django:

1. Всё включено – Django со старта предлагает разработчику большое разнообразие уже установленных инструментов и библиотек.
2. Развитая система – структура Django позволяет нам собирать ресурс, как конструктор;
3. Долгая история сервиса – Django на рынке разработки функционирует уже более 18 лет, за которые сервис успел развиться в высокоуровневую платформу;
4. Расширяемость – большое количество библиотек и плагинов предоставляет даёт возможность быстро добавлять новые программные модули;
5. Библиотеки – с помощью библиотек мы получаем готовые решения реализации сложных задач. К популярным библиотека Django можно отнести: Django REST Framework для работы с API, Pillow – для работы с изображениями, Django-rest- swagger – для автоматической генерации документации.
6. Административная панель – ресурс автоматически генерируется при создании приложений. Отсутствие необходимости создавать вручную административную панель – весомый аргумент при выборе инструмента разработки;
7. SEO-дружественность – в Django мы легко можем реализовать требующиеся функции для настройки поисковой оптимизации.
8. ORM - в Django представлено объектно-реляционное отображение, благодаря которому происходит взаимодействие приложение с базами данных.

Django написан на языке программирования Python, что полностью отражается в его структуре. В основе лежит архитектура MVT – Model-ViewTemplate – модель-представление-шаблон.

В данной архитектуре в моделях мы описываем объекты, их свойства и функции. За счет моделей происходит создание и добавление объектов, их обновление и удаление.

В представлениях решаются задачи обработки HTTP-запросов, исполнения бизнес-логики с помощью прописанных методов и свойств, формирование ответов на запросы.

Если модели и представления – это бэкенд, то за фронтенд отвечают шаблоны. В Django продумана собственная система подключения HTMLшаблонов. Их главная функция в отличии от представлений не исполнение логики, а отображение данных. Стоит отметить, что шаблоны могут быть, как статическими, так и динамическими.

Возможность создавать программный продукт, написав всего несколько строк кода, очаровывает. Это обуславливает популярность Django в среде разработки и наш выбор в качестве дипломного проекта темы «Разработка CRM системы для сервисной компании на языке Python и фреймворках Django и Django Rest Framework».

В своей дипломной работе мы ставим перед собой следующие задачи:

1. Изучение фреймворков Django и Django Rest Framework и инструментов его работы
2. Разработка системы CRM
3. Выработка умений и навыков проектирования структуры базы данных

# Назначение и цели создания CRM системы.

За 10 лет своей трудовой деятельности я сменил несколько различных компаний. Все они занимались похожей деятельностью. Мне показалось интересным создать приложение, которое будет управлять процессами продаж оборудования и запчастей, а также взаимодействием с клиентами по вопросам ремонта или пуско-наладки оборудования. В данной дипломной работе я создал только лишь небольшой макет системы CRM. В дальнейшем планирую его развивать и добавлять функционал.

# Структура приложения и требования к функциональности

При запуске проекта на Django на старте автоматически создается директория с вложенными файлами. Структура проекта при первом запуске имеет следующий вид:

* myproject/
  + manage.py
  + myproject/
  + \_\_init\_\_.py
  + settings.py
  + urls.py
  + asgi.py
  + wsgi.py

С помощью файла manage.py мы будем осуществлять управление проектом – запускать сервер, создавать новые приложения, выполнять миграции информации в базу данных, создавать администратора, создавать и менять пароль и много другое.

Далее идет пакет Python одноименный с нашим проектов, в примере выше это пакет с названием «myproject». В моем же проекте он носит название «config». Как пакет его определяет наличие файла \_\_init\_\_.py. Данный пакет содержит стандартные файлы, которые используются для настройки проекта: • settings.py – файл, в котором прописываются настройки проекта. Включает в себя описание путей к приложениям, настройки статики, важные параметры, связанные с работой подключаемых модулей и настроек безопасности.

* urls.py – в данном файле прописываются маршруты приложения. По адресам, прописанным в данном файле, будет строиться навигация на нашем ресурсе. На смотря на то, что данный файл может содержать все ассоциации url адресов с представлениями, принято разделять его на части и в каждом приложении создавать свой файл urls.py.
* asgi.py – модуль, описывающий связь проекта с веб-сервером через интерфейс ASGI.
* wsgi.py - модуль, описывающий связь проекта с веб-сервером посредством интерфейса WSGI.

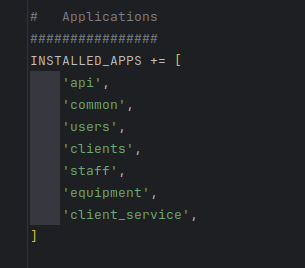
Если стартовый пакет содержит файлы, отвечающие за общие настройки проекта, то за функциональная логика прописывается в приложениях.

Приложение в Django представляет собой отдельный фрагмент функциональности сайта. Оно может, как реализовать работу всего сайта, так и отдельной функциональности (радела или подсистемы). При разработке структуры сайта рекомендуется выделять приложения по выполняемому спектру задач

В нашем проекте мы выделяем следующие приложения:

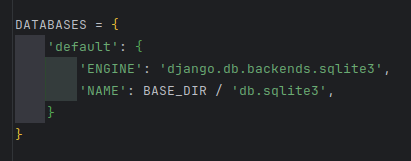
* Приложение «api» - данный пакет будет отвечать за представление всех запросов Django Rest Framework. Поможет в этом библиотека drf-spectacular. Также здесь мы создадим директорию spectacular, в которой будут хранится настройки этой библиотеки.
* Приложение «config» — это основное приложение проекта, в нем хранятся основные настройки.
* Приложение «common» - это приложение будет содержать в себе различный функционал, общий для всего проекта. Его можно применить в любом из приложений.
* Приложение «users» - этот пакет будет отвечать за создание новых пользователей, авторизацию и аутентификацию в системе.
* Приложение «staff» - пакет, отвечающий за создание сотрудников компании. А именно будут создаваться 2 типа сотрудников: менеджеры и сервис-работники. Он связан с таблицей базы данных «users».
* Приложение «clients» - пакет, в котором будет прописана логика работы c базой данных клиентов: добавление клиентов, сохранения информации о клиентах в базе данных.
* Приложение «equipment» - приложение, отвечающее за создание товаров в базе данных, таких как оборудование, зап.части и тд.
* Приложение «client-service» - будет отвечать за создание заказов на покупку или сервисное обслуживание.

Каждое приложение после создания необходимо зарегистрировать в основном файле settings.py:



# Формирование базы данных приложения

В данном приложении будет использоваться база данных SQLite.



В настройках прописано, что для нашего проекта мы будем использовать базу данных SQLite, что подходит для нас, так как не предполагает большое количество запросов. Данная настройка идёт «с коробки» и менять мы её не будем.

SQLite представляет собой быструю и легкую базу данных, которая предоставляет возможность хранить данные прямо внутри нашего приложения. Отметим ряд преимуществ использования SQLite: 1. Простота - SQLite не требует отдельного сервера или настройки. База данных работает, как часть приложения;

1. Надежность: Транзакции и ACID-свойства обеспечивают надежность и целостность данных;
2. Кроссплатформенность: SQLite поддерживается на множестве платформ, включая Windows, macOS и Linux;
3. Эффективность: SQLite требуются минимальные ресурсы системы, что отлично подходит, например, для мобильных устройств.

# Установка и настройка Django Rest Framework.

## Что такое Django Rest Framework

Сегодня сеть интернет построена по принципу Клиент-Серверного взаимодействия. Клиент посылает запрос — Сервер ему отвечает. В случае, когда между собой общаются два Сервера, мы условно называем Клиентом того, который отправил запрос и ожидает ответ, а Сервером будет тот, кто принимает запрос и отвечает не него. Взаимодействие браузеров и веб-сайтов (первые выступают в роли Клиента, а вторые в роли Сервера) традиционно делалось при помощи технологии html-рендеринга, именно так изначально это делал Django. Чтобы получить данные с веб-сайта, браузер отправляет запрос GET к Серверу. Сервер формирует ответ в виде html-страницы и передает ее браузеру. Так Сервер передает данные браузеру, но как браузер может передать данные Серверу? В этой самой html-странице Сервер заложил все необходимые веб-формы, заполнив которые, пользователь мог бы передать свои данные обратно на сервер. Когда вы ввели свои данные в форму на сайте, бразуер отправляет Серверу запрос POST, в котором содержатся ваши данные, а Сервер обрабатывает их и записывает в базу данных.

Все это отлично работало, но уже в середине нулевых такой подход перестал удовлетворять возрастающим требования в веб-разработке. Появлялись мобильные приложения, различные гаджеты с доступом в интернет, и для них уже не подходил стандартный способ html-рендеринга на сервере, ведь теперь каждому клиенту нужно было отрисовать данные по-своему. Постоянно увеличивалось взаимодействие серверов друг с другом, и html-формат уже не подходил. Для всех этих задач есть другой способ обмена данными — Web API. Смысл этого способа в том, что Сервер передает Клиенту не html-страницу, а непосредственно данные, никак не влияя на то, как эти данные будут в итоге представлены. Наиболее популярными форматами для передачи данных становятся XML и JSON. Таким образом Сервер полностью избавляется от задачи отрисовки данных. Какое-то время длился переходный период, когда разработчикам веб-приложений на Сервере приходилось поддерживать оба способа одновременно: html рендерился на Сервере для браузеров, а Web API использовался для мобильных приложений и интеграции с другими серверами. Понятно, что разработчикам приложений на Сервере приходилось делать двойную работу. Но в начале десятых ситуация стала меняться в пользу Web API. Этому способствовало молниеносное развитие инструментов на языке JavaScript, а также появление различных веб-фреймворков, одним из которых и является предмет данной статьи.

Браузерные приложения быстро научились отрисовывать веб-страницы самостоятельно, получая чистые данные с Сервера. Веб-приложения на сервере научились создавать API быстро и легко. Так сформировалась четкое разделение на Backend и Frontend разработку: тех, кто поддерживает приложение на Сервере, и тех, кто делает браузерные (клиентские) приложения. А Web API стал универсальным способом общения для Сервера и всех его клиентов (браузеров, мобильных приложений, других Серверов). Конечно, это не могло не привести к развитию стандартов в общении между системами. И Клиенту, и Серверу необходимо знать каким образом общаться с друг с другом, как передавать данные, как сообщать об ошибках. Разработчики всегда могли договориться о том, как взаимодействовать их приложениям, но наличие некоего стандарта в веб-разработке позволило бы эту задачу облегчить. И вот в начале десятых таким стандартом стала концепция REST.

REST

В 2000 году Рой Филдинг написал докторскую диссертацию, где изложил концепцию REST. Там были рекомендации о том, как спроектировать Сервер, чтобы ему было удобно общаться с Клиентами. Выделю два главных принципа создания приложений в стиле REST:

* Сервер не должен ничего знать о текущем состоянии Клиента. В запросе от Клиента должна быть вся необходимая информация для обработки этого запроса Сервером.
* Каждый ресурс на Сервере должен иметь определенный Id, а также уникальный URL, по которому осуществляется доступ к этому ресурсу.

На данный момент мы можем найти фреймворк для создания приложений в стиле REST практически для каждого языка программирования, используемого в веб-разработке. Логика построения Web API на Сервере в этих фреймворках реализована одинаково.

Действия для управления данными привязаны к определенным HTTP-методам. Существует несколько стандартных действий для работы с данными — это *Create, Read, Update, Delete*. Часто их обобщают как CRUD.

* Для создания объекта используется http-метод POST
* Для чтения — http-метод GET
* Для изменения — http-метод PUT
* Для удаления — http-метод DELETE

Django REST framework - это мощный и гибкий набор инструментов для создания Web API.

Некоторые причины, по которым вы можете захотеть использовать REST framework:

* Просматриваемый API - огромный выигрыш в удобстве использования для ваших разработчиков.
* Политики аутентификации, включая пакеты для OAuth1a и OAuth2.
* Сериализация, поддерживающая как ORM, так и non-ORM источники данных.
* Настраивается все - просто используйте обычные представления на основе функций, если вам не нужны более мощные возможности.
* Обширная документация и отличная поддержка сообщества.
* Используется и пользуется доверием всемирно известных компаний, включая Mozilla, Red Hat, Heroku и Eventbrite.

## Установка и настройка

pip install djangorestframework

pip install markdown # Markdown support for the browsable API.

pip install django-filter # Filtering support

После установки необходимо добавить этот фреймворк в установленные приложения и сконфигурировать его. В этом проекте это делано следующим образом.



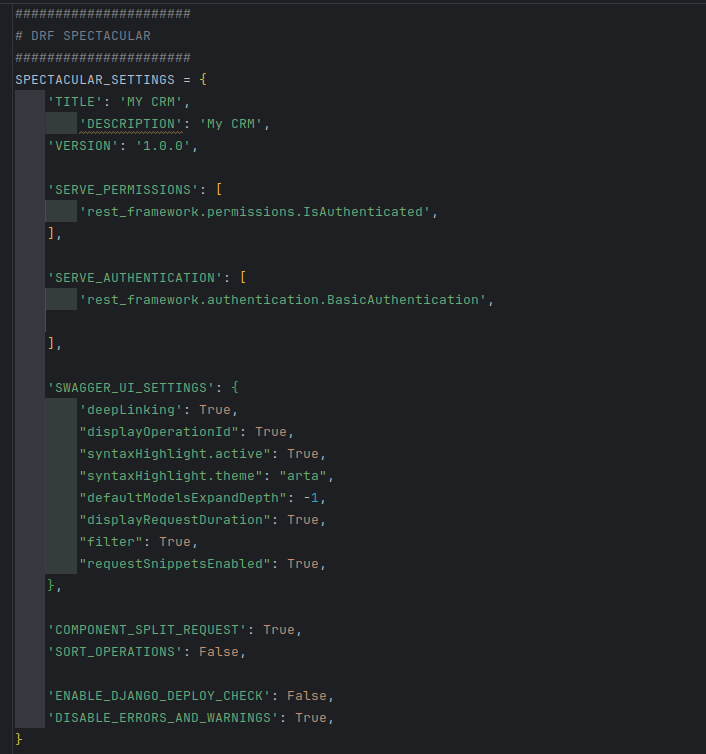
Так же в проекте я использую библиотеку генерации схем OpenAPI

Drf-spectacular.

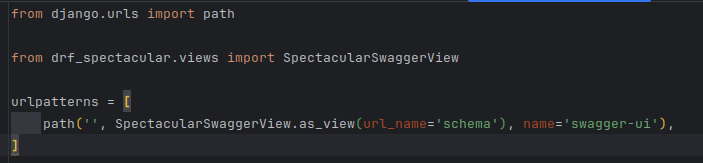
Drf-spectacular — это библиотека генерации схем OpenAPI 3 с акцентом на расширяемость, настраиваемость и генерацию клиентов.

Это рекомендуемый способ генерации и представления схем OpenAPI. Библиотека стремится извлечь как можно больше информации о схеме, предоставляя при этом декораторы и расширения для лёгкой настройки.

Ниже приведена настройка библиотеки в файле settings.py



Так же в приложении «api» в фале «spectacular/urls» необходимо определить следующие настройки:



# Настройка административной панели

Django дает разработчикам возможность использовать модели для автоматической генерации административной панели. Последняя предоставляет представляет собой полноценный интерфейс и даёт нам возможность управлять данными приложений: создавать, редактировать и удалять записи в базе данных, а также производить множество иных действий, управляющих работой приложений.

Административная панель даёт нам возможность быстро и удобно управлять данными и значительно упрощает жизнь разработчика в том числе и за счёт того, что является встроенным элементом и поставляет из «коробки».

С момента создания проекта административная панель уже доступна для использования по адресу [http://127.0.0.1:8000/admin/.](http://127.0.0.1:8000/admin/) Но для работы с ней необходимо выполнить ряд настроек.

Для входа нам необходимо будет создать суперпользователя. При запуске проекта в базе данных зарегистрированных пользователей нет. Суперпользователь создается из консоли с помощью команды:

• python manage.py createsuperuser Далее нам будет предложено системой ввести логин, email и пароль для нашего администратора. После это по введенным данным мы сможем зайти в административную панель.

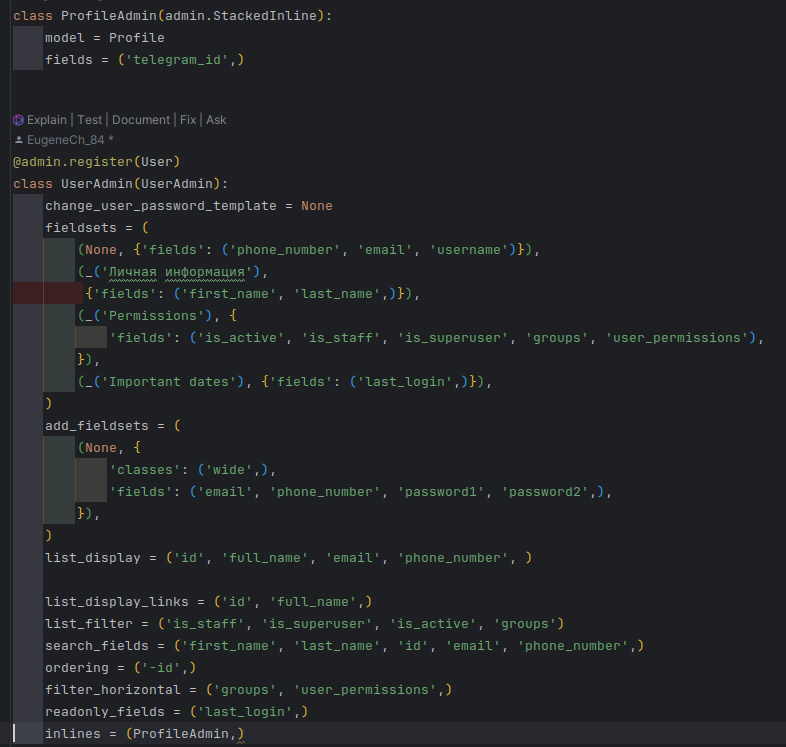
На старте админ-панель также будет пуста. Мы заполним её информацией в процессе разработки проекта.

Отметим, что разработчики фреймворка Django не просто с коробки добавили в него административную панель, но заложили возможность настраивать интерфейс под требования администратора сайта.

Кроме того, мы можем добавлять различные группы пользователей с разным уровнем доступа, в зависимости от стоящих перед ними задач. Например, менеджеру, обрабатывающему заказы, нет необходимости давать доступ к созданию и удалению пользователей, товаров или категорий. Мы можем открыть для него таблицу заказов на просмотр и редактирование. Это не только позволит избежать ошибок в работе сайта из-за неумелого обращения, но и облегчит процесс обучения персонала более ограниченному интерфейсу.

## Кастомизация панели администрирования.

В своем проекте я немного кастомизировал панель администрирования под свои нужды.



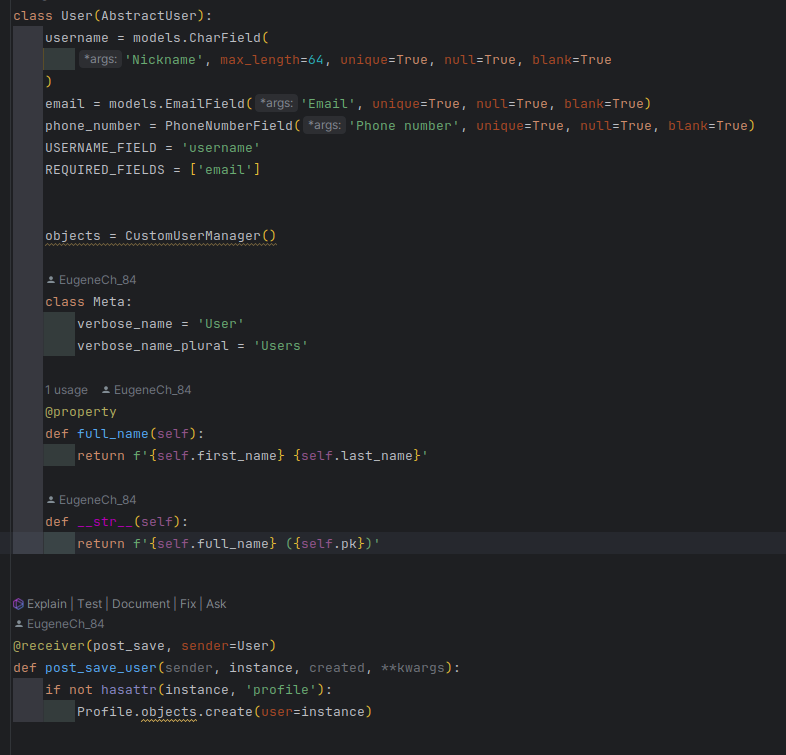
# Описание структуры и функций приложений проекта

## Приложение «user»

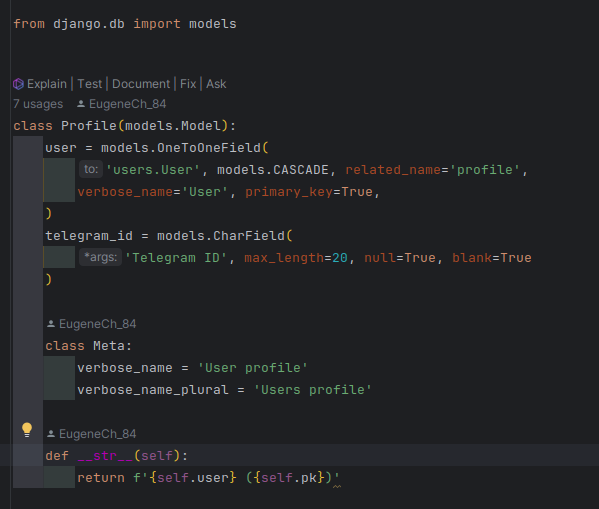
### Модели

Я создал 2 модели для пользователей:

-модель самого пользователя для регистрации, авторизации и аутентификации. Эта модель наследуется от модели Django AbstractUser.



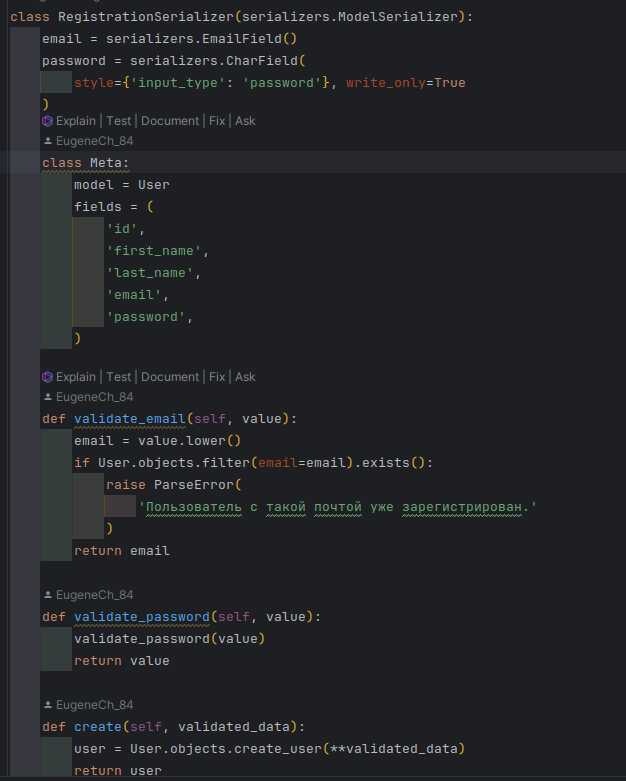
-модель профиля пользователя



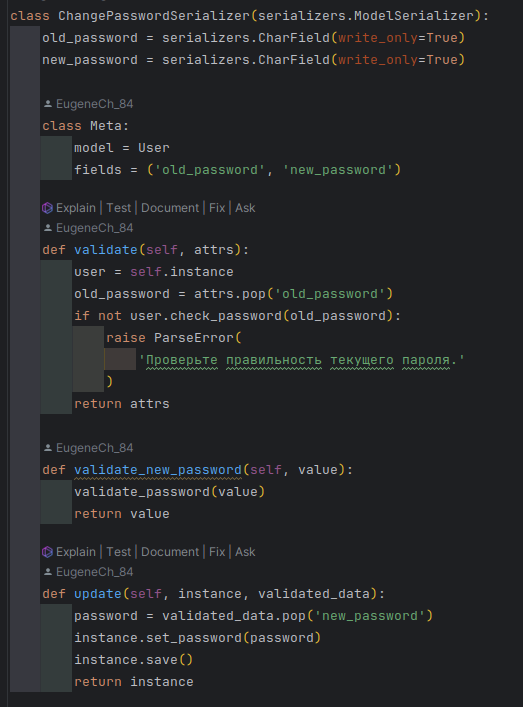
### Сериализаторы

Для каждой модели проекта я создавал сериализаторы.

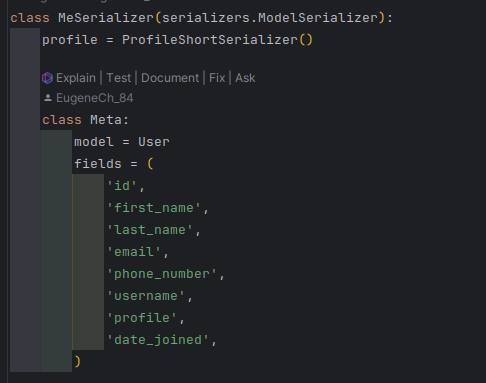
Сериализатор для регистрации пользователя



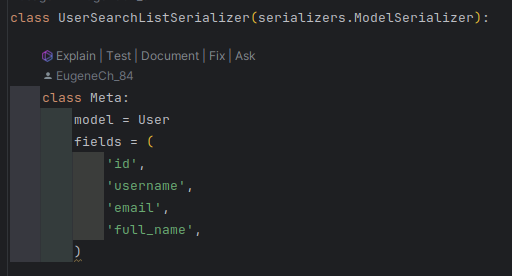
Сериализатор для смены пароля



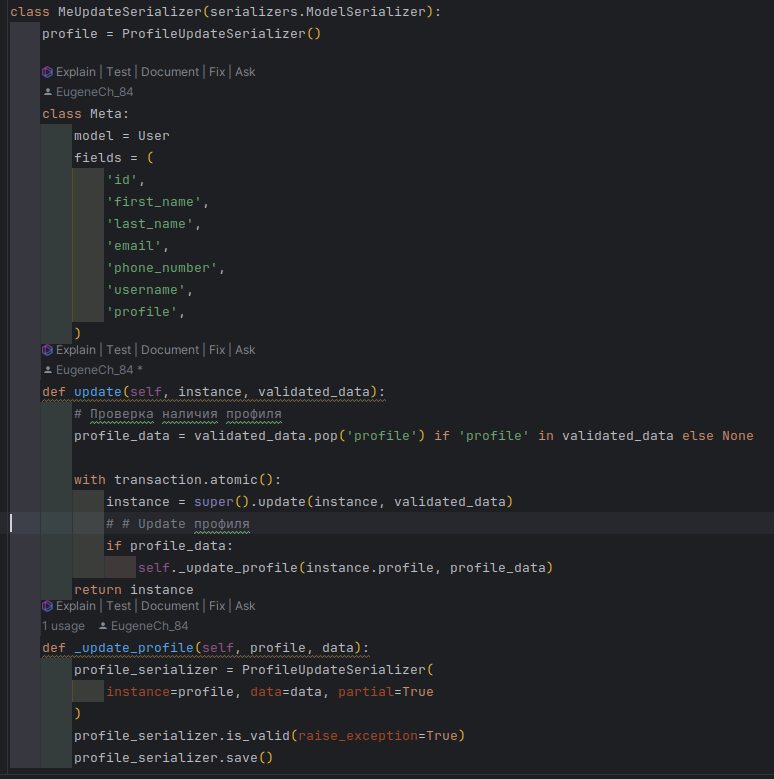
Сериализатор для представления данных пользователя.



Сериализатор для поиска пользователя



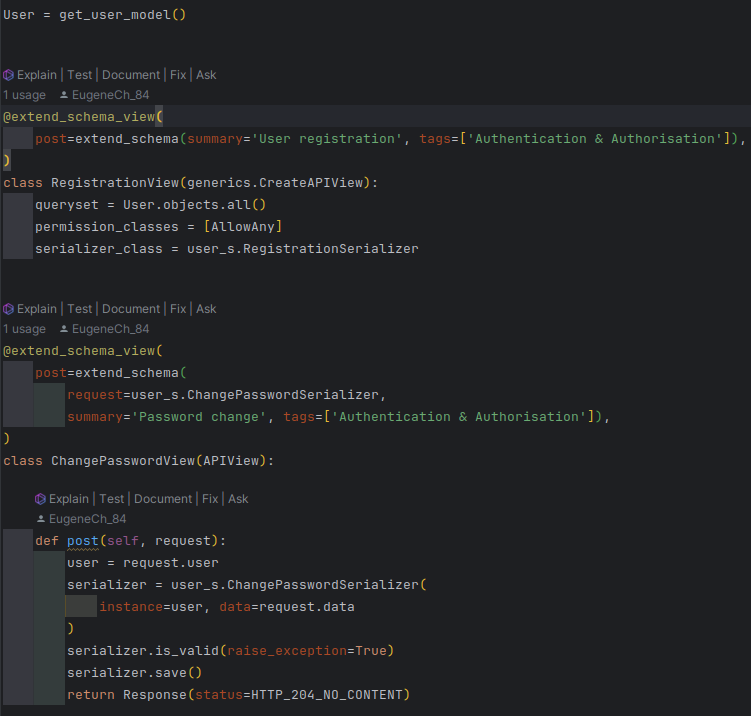
Сериализатор обновления данных пользователя.

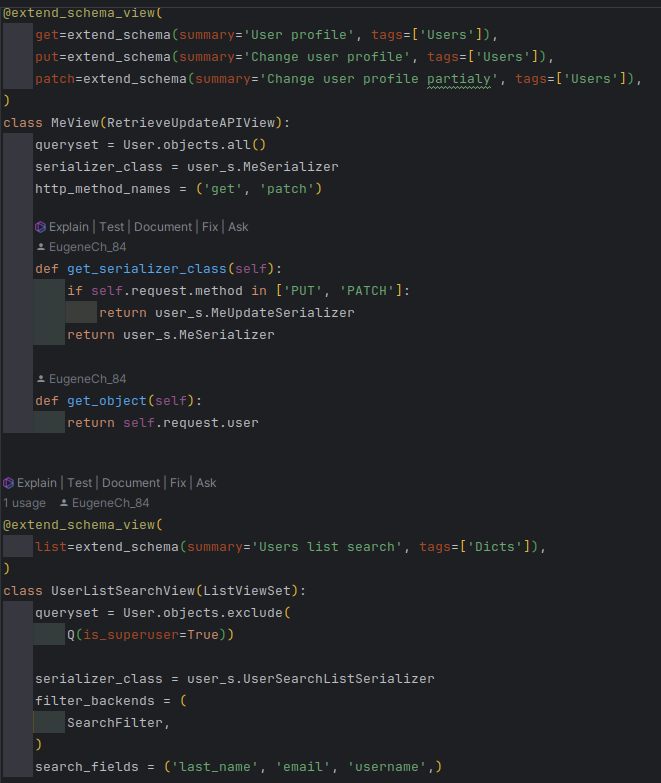


### Представления данных модели

Используя сериализаторы я создал несколько API представлений для пользователя.

Регистрация, изменение пароля и тд.





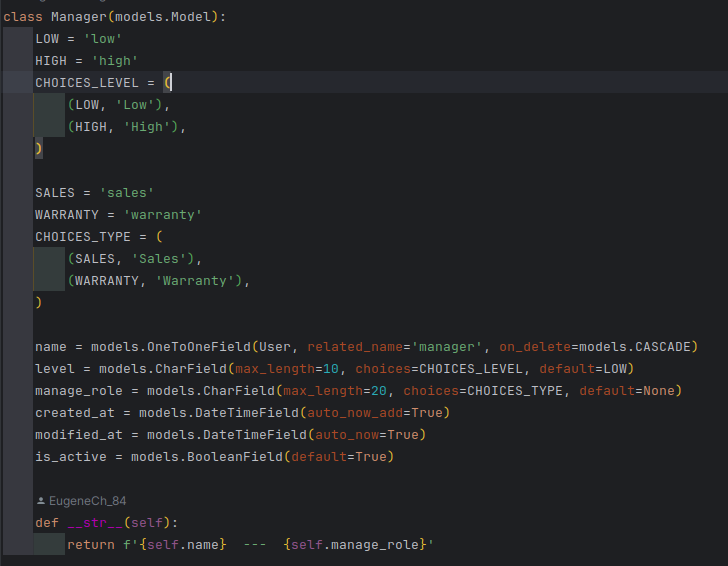
## Приложение «staff»

### Модели

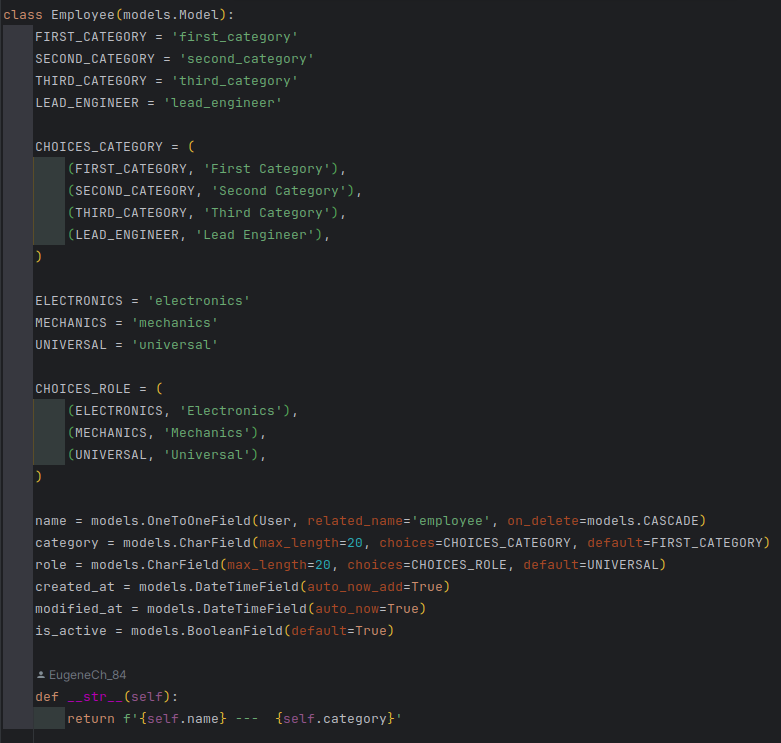
В данном приложении были созданы 2 модели данных:

- модель «Manager»

Она связана с моделью данных «User»



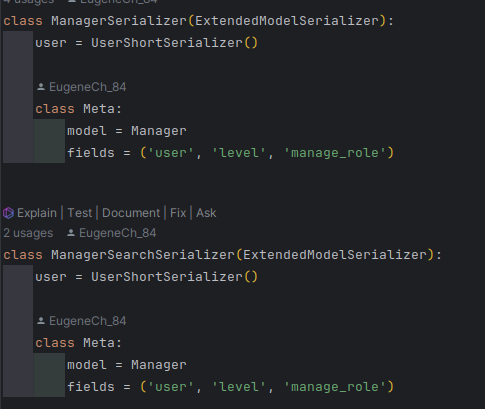
- модель данных «Employee», которая так же связана с «User»

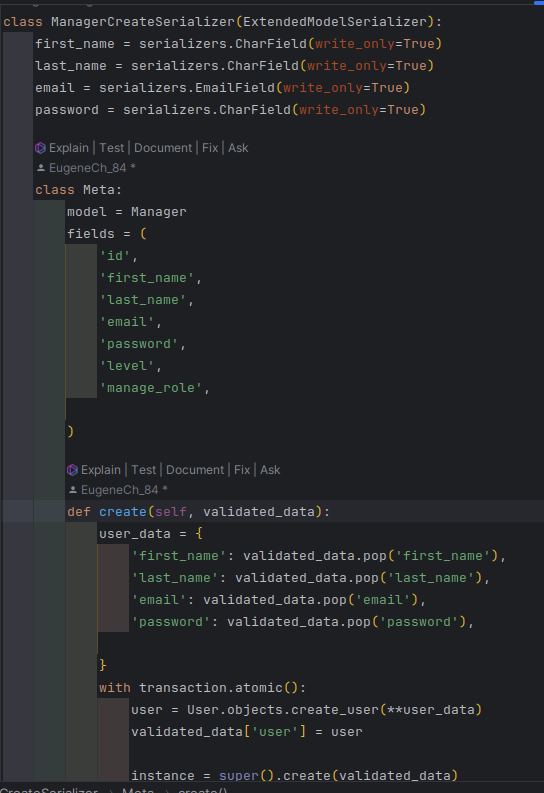


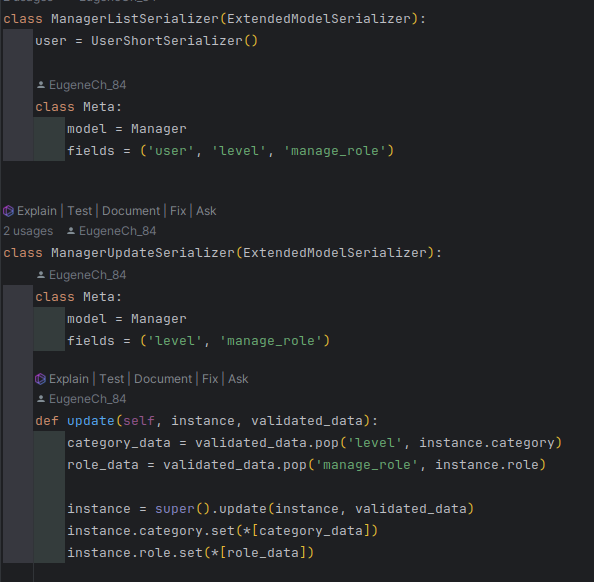
Обе модели содержат в себе данные на сотрудников. Часть является менеджерами, часть – сервисными работниками. Я использовал атрибут choices, для выбора типа, уровня и т.д. для сотрудников.

### Сериализаторы

- для модели «Manager»

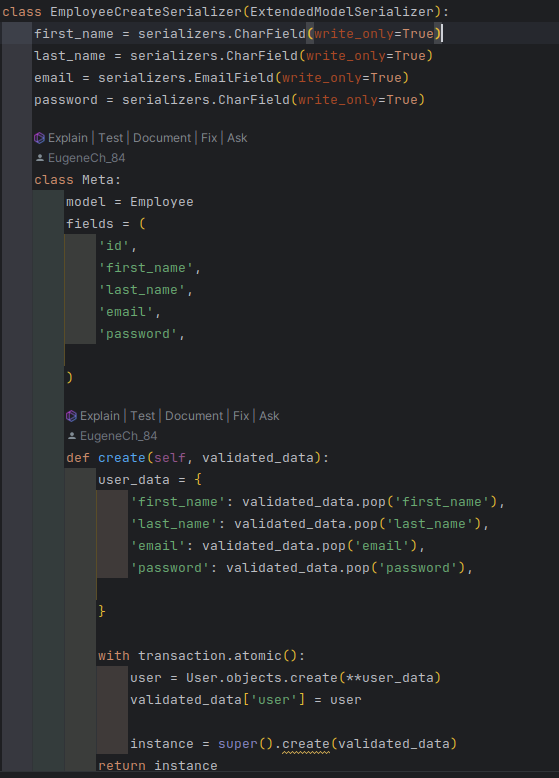


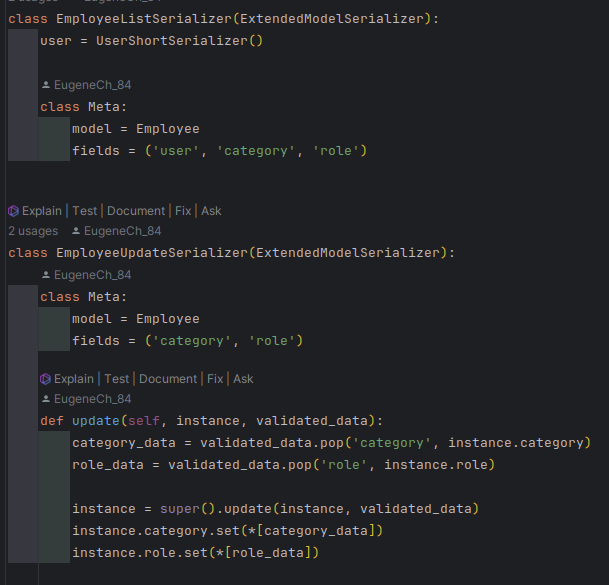




- для модели «Employee»

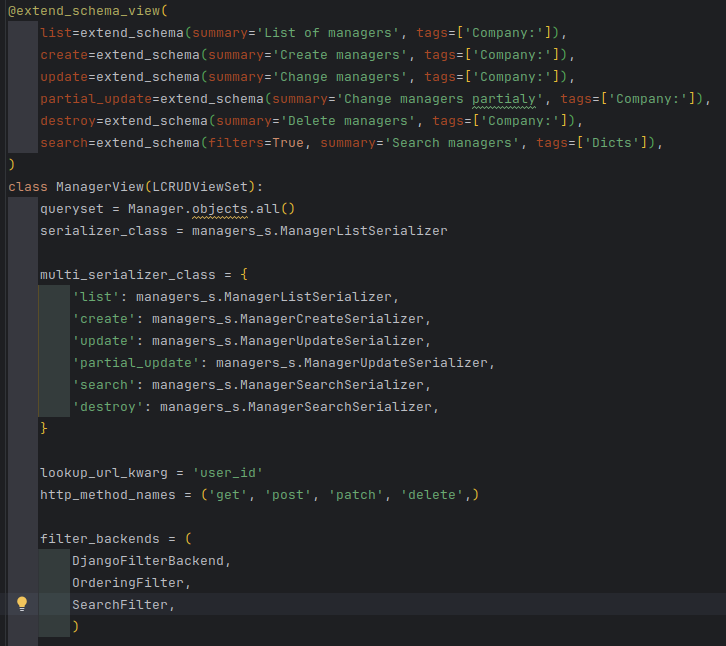


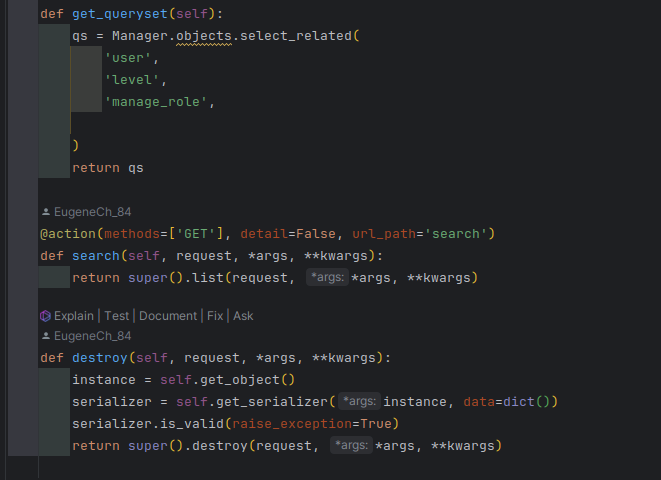




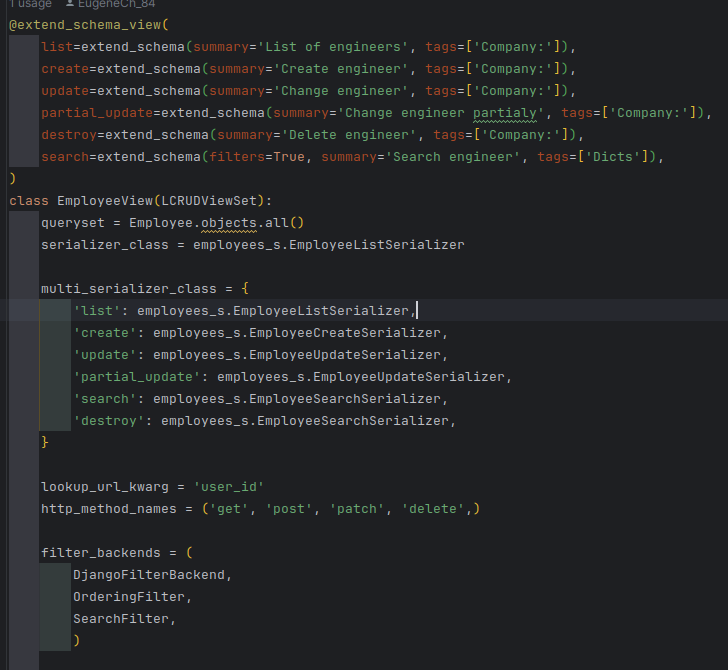
### Представления данных модели

- для данных модели «Manager»





- Для данных модели «Employee»





## Приложение «client-service»

В этом приложении модели имеют множественные связи с другими моделями. Они включают в себя менеджеров, клиентов, оборудование или работы, которые необходимо выполнить.

Это приложение формирует заказы на приобретение товаров или услуг.

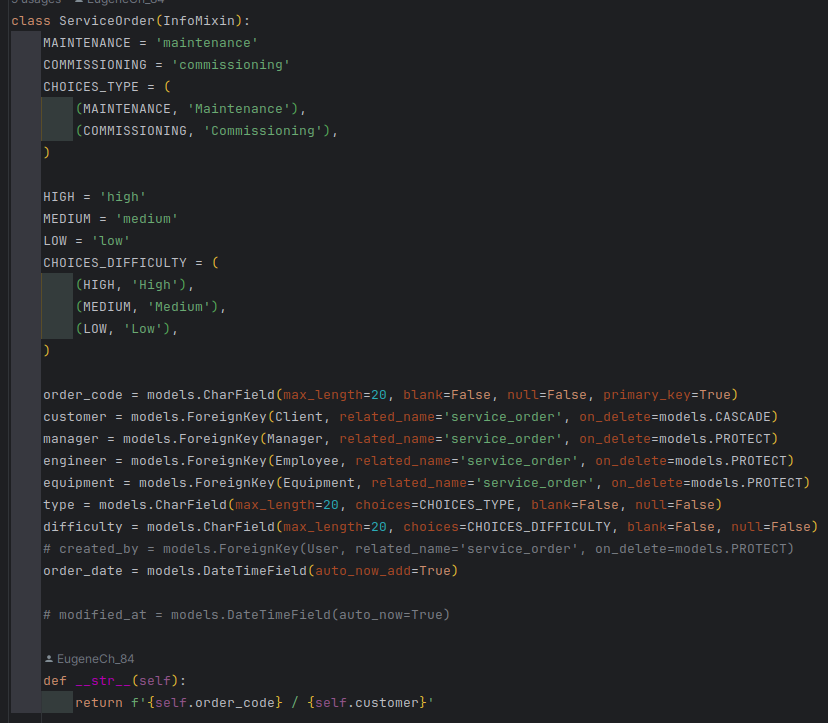
Рассмотрим модели данного приложения

### Модели

- модель данных «Sales\_order»

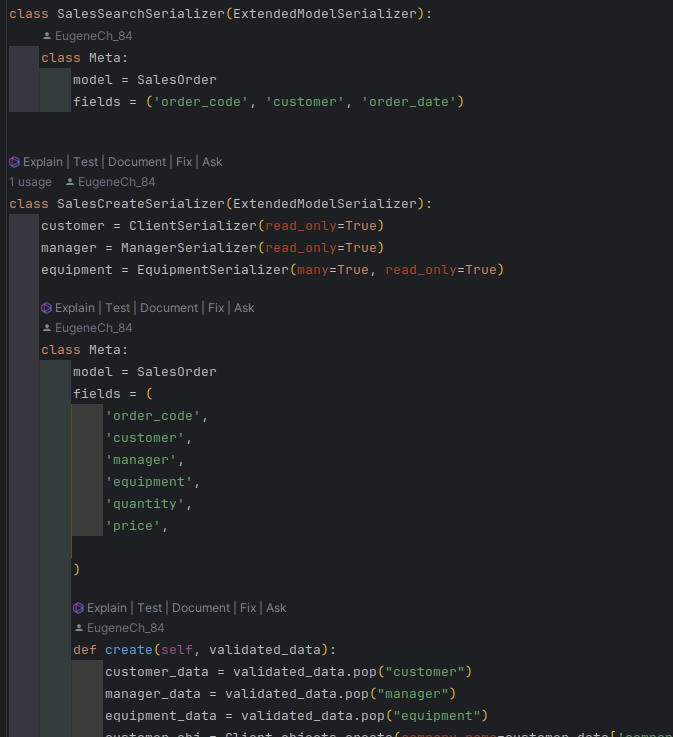


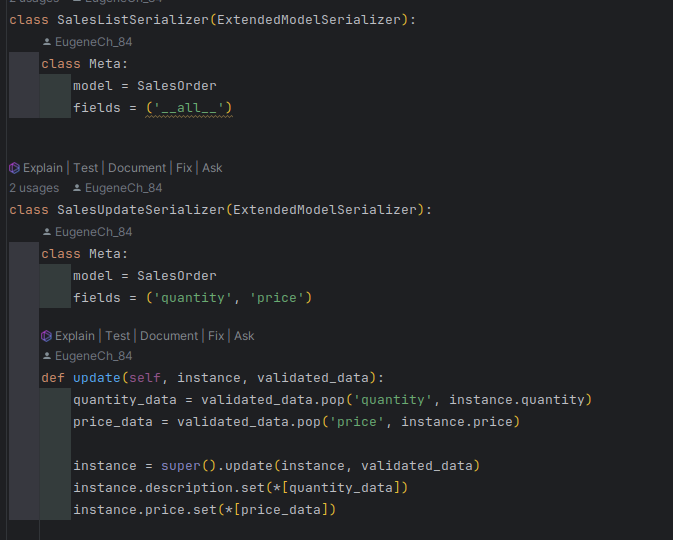
- Модель данных «Service\_order»



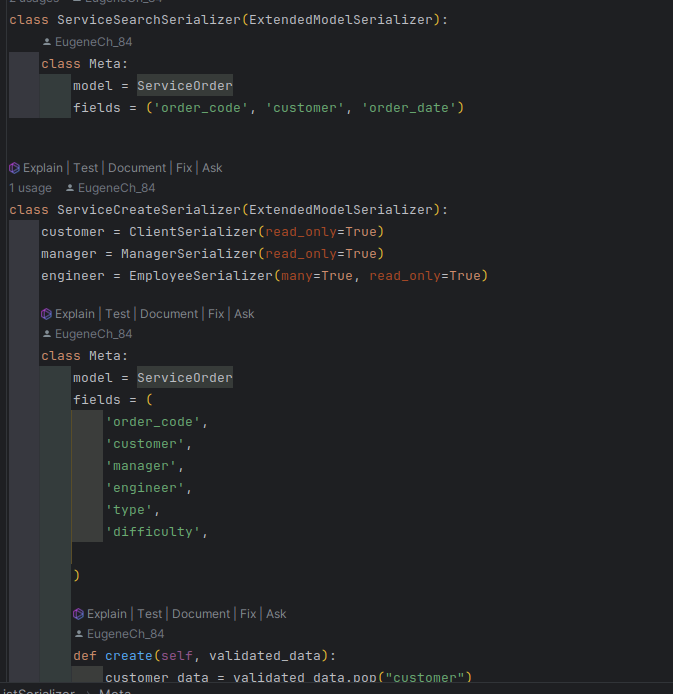
### Сериализаторы

- для данных модели «Sales\_order»





- для данных модели «Service\_order»

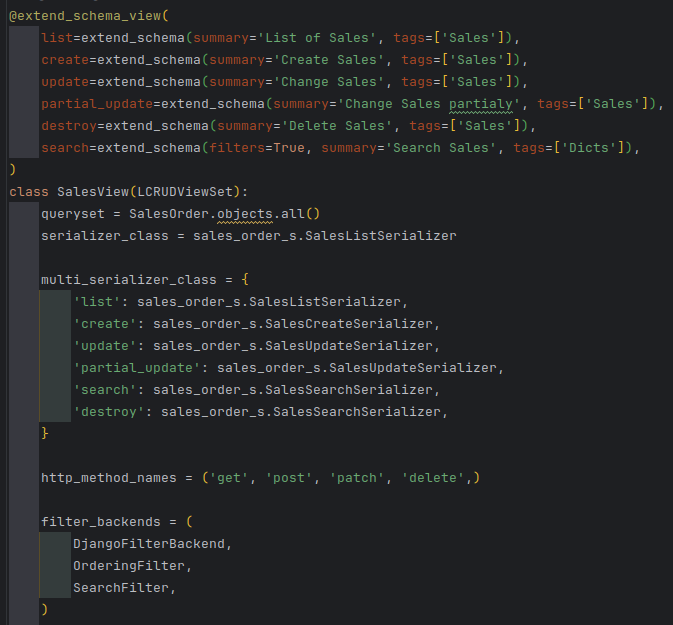




### Представления

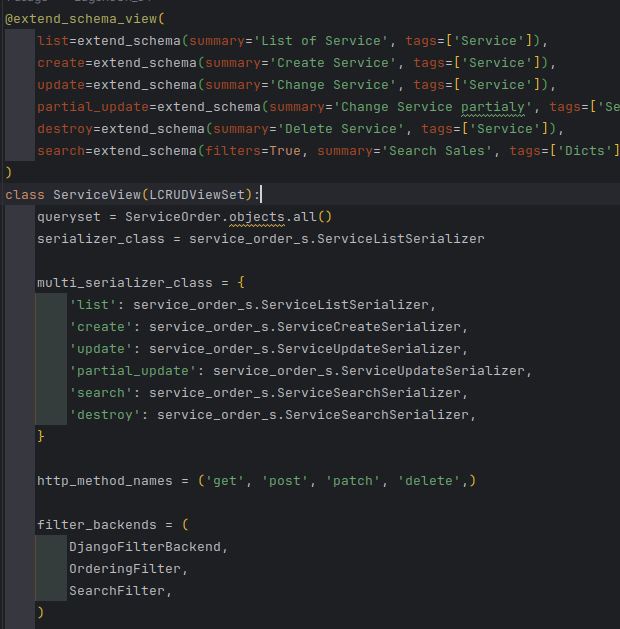
На базе моделей и сериализаторов были созданы представления этих данных.

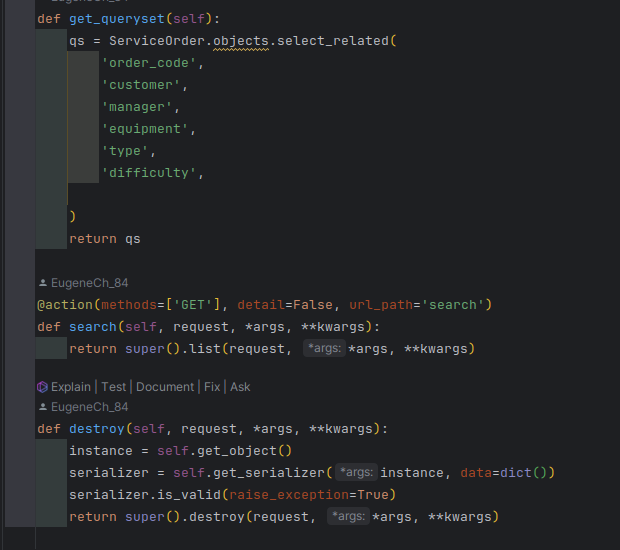
- для данных «Sales\_order»





- для данных «Service\_order»





## Приложение «Common»

### Миксины для моделей

В этом приложении собраны некоторые классы и функции которые используются в проекте в целом, вне зависимости от приложения.

Для упрощения создания моделей я использовал миксины.

В программировании можно столкнуться с ситуациями, когда нужно расширить функциональность классов без создания сложных иерархий наследования. Миксины в Python являются одним из способов решения этой проблемы.

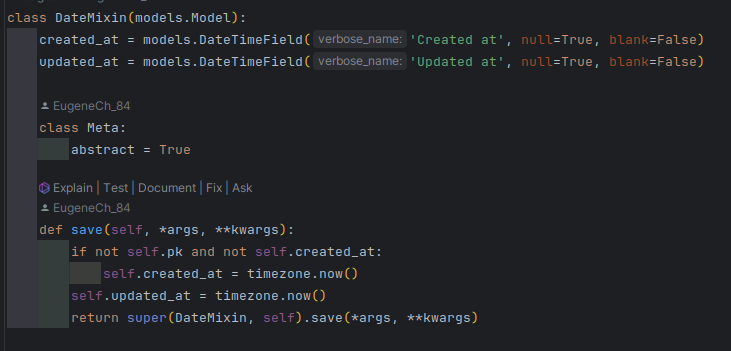
В этом уроке мы рассмотрим, что такое миксины, как они используются, и как можно с их помощью упростить структуру кода.

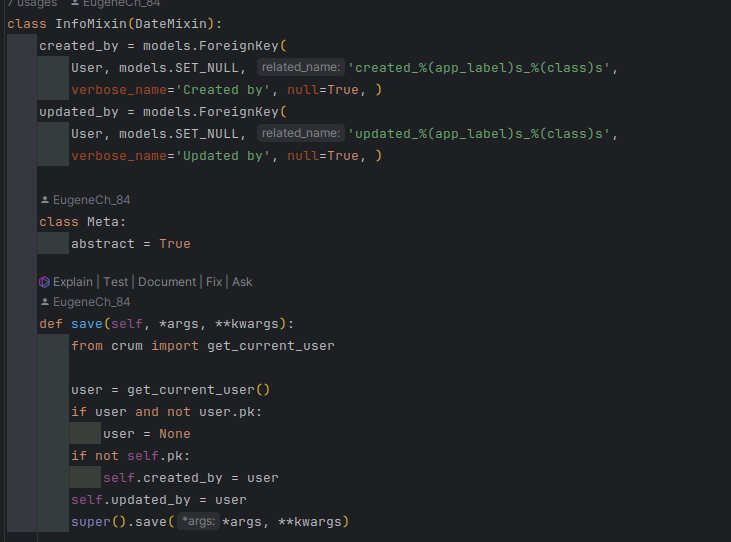
Понятие миксинов

**Миксины** или **Mixins** — это форма множественного наследования в Python и мощный инструмент, который позволяет преодолеть ограничения единственного наследования. Они представляют собой простые классы, которые включают набор методов, предназначенных для добавления к другому классу, и позволяют расширять функциональность классов без глубокой иерархии наследования.

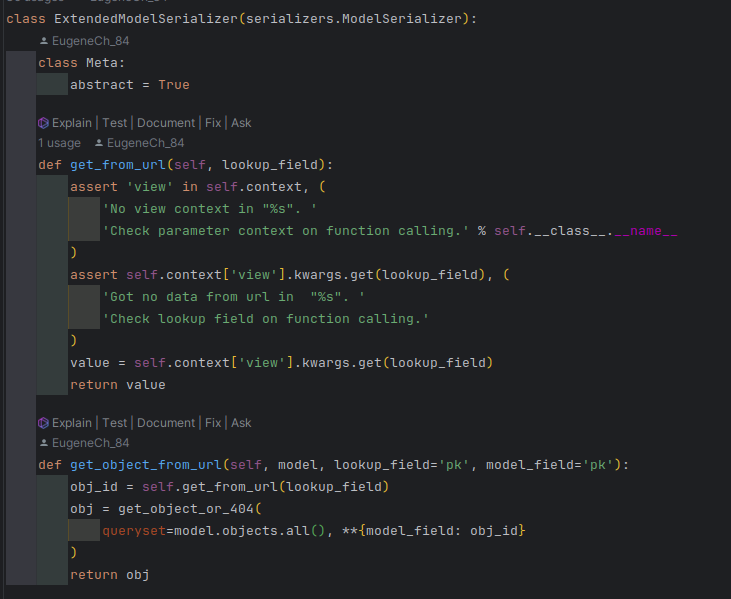
Это устраняет проблемы, связанные с множественным наследованием, и делает миксины гибким средством для улучшения и модификации структуры кода.

Миксины создаются для того, чтобы повторно использовать функции во множестве классов. Они не предполагают создание объектов, и не должны иметь своего состояния.





### Миксины для сериализаторов



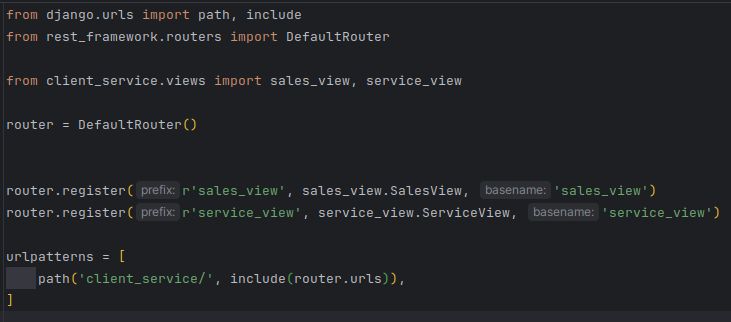
## 5. URLs

Файл urls.py в Django является основным механизмом для маршрутизации URL-адресов приложения. Он используется для соответствия URL-адресов определенным функциям (вызываемым view-функциям) в вашем приложении Django.

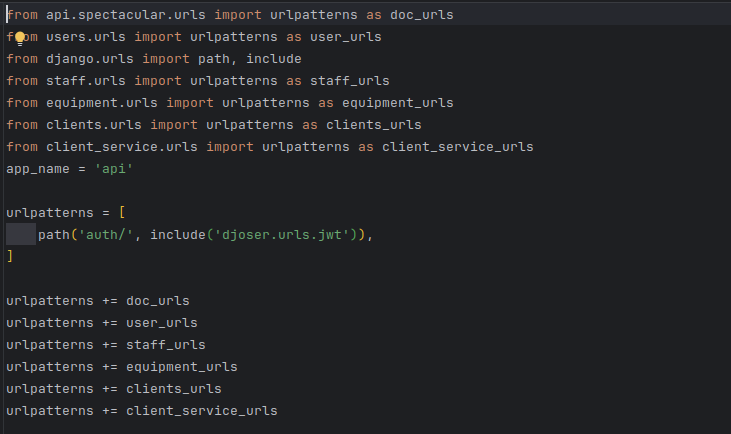
Обычно в Django проектах есть два вида файлов urls.py — один для основного проекта и один для каждого приложения в проекте. Файл urls.py приложения определяет URL-адреса, специфичные для данного приложения, а файл urls.py проекта используется для обработки URL-адресов, которые не определены в urls.py приложения.

urlpatterns — это список, содержащий все пути маршрутизации URL-адресов для приложения Django. Он обрабатывается в том порядке, в котором описаны URL-адреса, сверху вниз, до тех пор, пока не будет найден соответствующий маршрут для запроса.

Каждый элемент списка urlpatterns обычно содержит объект path() или re\_path(), который определяет маршрут URL-адреса, а также имя вызываемой view-функции. Также в этом списке можно использовать специальные функции, такие как include(), которая позволяет добавлять в urlpatterns другие списки urlpatterns из других модулей.



В данном приложении я использую списки urlpatterns и объединяю их в одном файле приложения «api».



## 6. Пагинация

DRF включает поддержку настраиваемых стилей пагинации. Это позволяет изменять, как большие наборы результатов разбиваются на отдельные страницы данных.

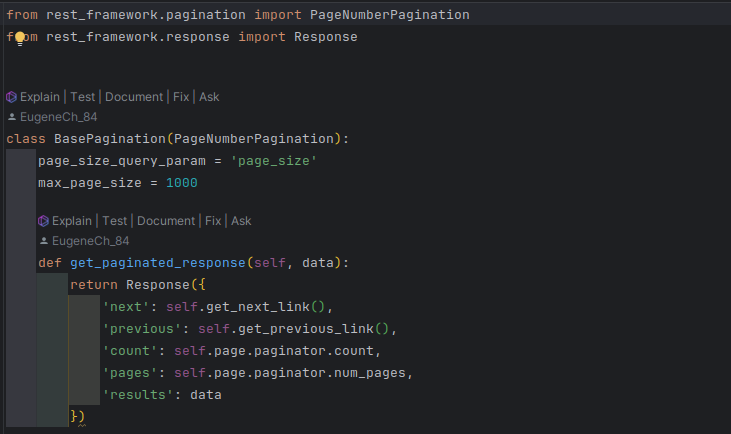
API пагинации может поддерживать любую из этих функций:

* Ссылки пагинации, которые предоставляются как часть содержимого ответа.
* Ссылки пагинации, включенные в заголовки ответа, такие как Content-Range или Link.

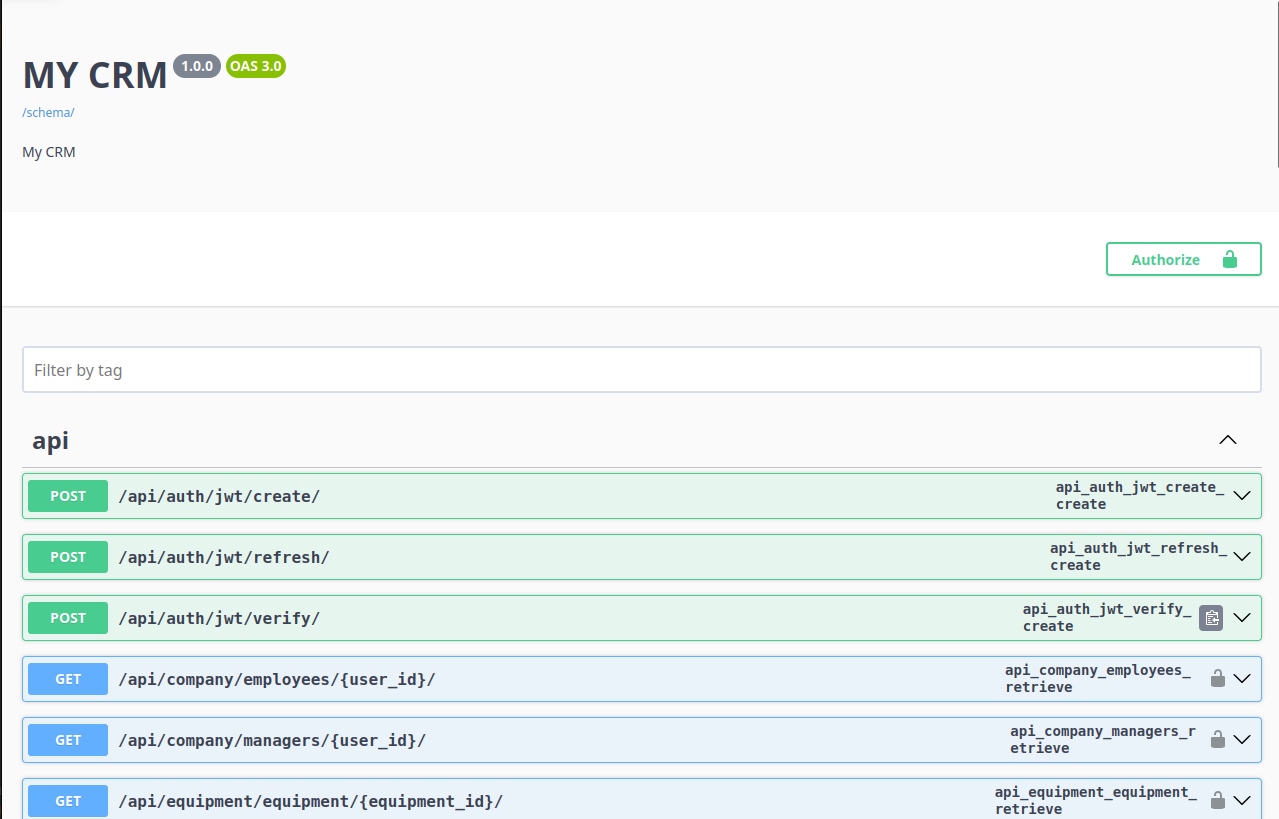
В настоящее время все встроенные стили используют ссылки, включенные как часть содержимого ответа. Этот стиль более доступен при использовании API с возможностью просмотра.

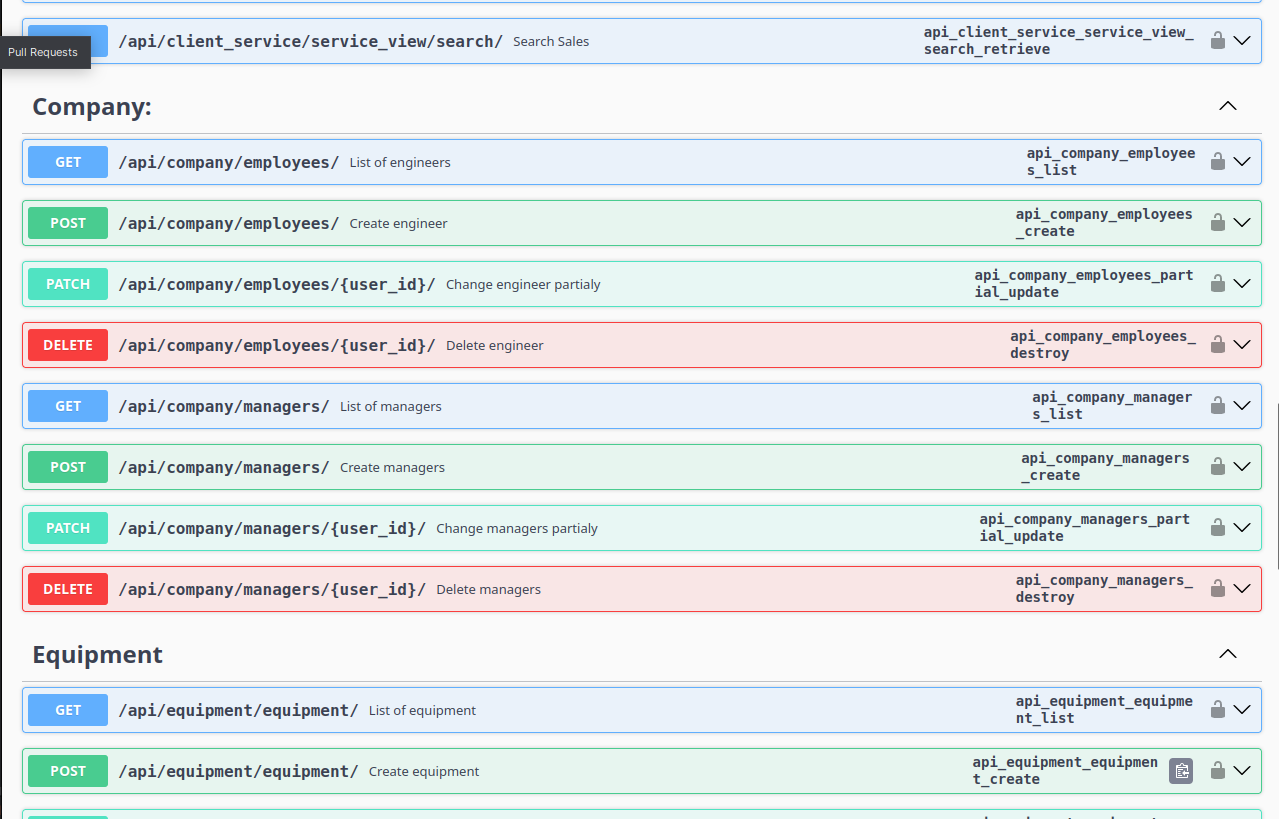
Пагинация выполняется автоматически, только если вы используете общие представления или наборы представлений. Если вы используете обычное APIView, вам нужно будет самостоятельно обратиться к API пагинации, чтобы убедиться, что вы возвращаете ответ с пагинацией. Пример смотрите в исходном коде классов mixins.ListModelMixin и generics.GenericAPIView.

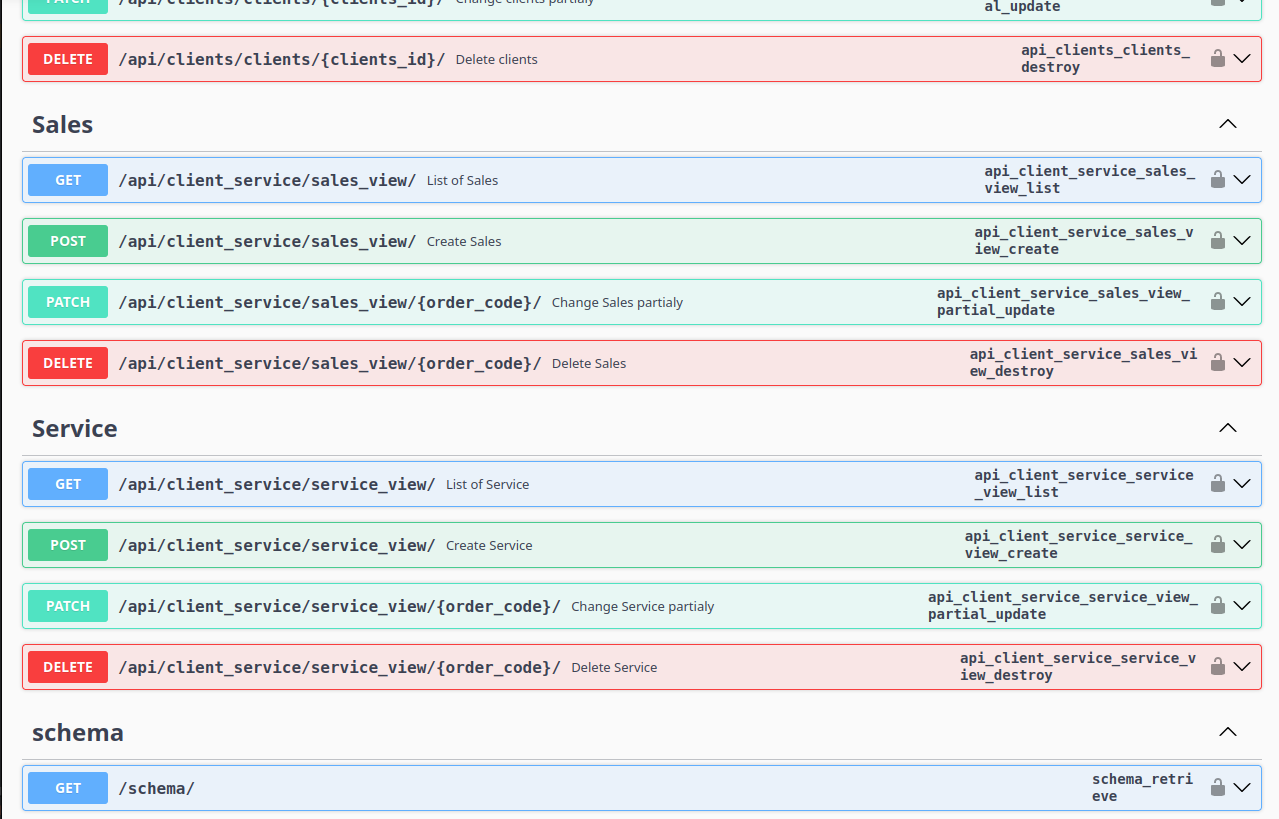
В приложении «common» я создал файл pagination.py со следующим содержимым:



## 7. Итоговое представление REST API







# Дальнейшее развитие проекта

Дальнейшее развития проекта я вижу в расширении функционала проекта.

* Создать front-end представление для данного проекта на VUE.js;
* Добавить к функционалу возможность расширенного менеджмента товарами и услугами.
* Заменить базу данных на Postgresql.
* Поработать с очередями задач используя celery
* Разработать систему логирования и сбора метрик.

Звучит просто, на деле, как всегда, потребует максимального погружения.

Django, как омут, но омут полезный, наполненный тоннами полезных фишек.

Подводя итоги данного раздела, отметим, что в какую бы сторону не стал развиваться наш проект, мы обязательно найдем решение для новой реализации.

# Заключение

Подводя итоги проведенной работы, можно смело сказать, что фреймворк Django справедливо считается одним из самых популярных в среде разработки.

Чтобы изучить все возможности Django мало одного проекта, с ним хочется работать и дальше, изучая другие библиотеки и средства реализации различных функциональностей. Фреймворк обладает развитой экосистемой. Все более чем 18 лет существования он изменялся и совершенствовался. Какая бы задача не стояла перед нами в веб-разработке с большой вероятностью мы сможем найти ответ в уже готовом решении. Последнему немало способствует большое комьюнити разработчиков, использующих данную платформу, а также большое количество поддерживаемых библиотек.

В результате своей работы мы полностью реализовали задачи, которые ставили перед собой на старте работы над проектом:

* Изучили структуру фреймворка Django;
* Познакомились и применили на практике инструменты работы с изображениями, сессиями, формами, моделями, базами данных;
* Получили по итогу полноценный REST API, готовый к работе с front-end.

Благодаря гибкости, которую нам даёт фреймворк проект может расти и развиваться в любую сторону. Мы можем добавить на наш сайт раздел с блогом, подключить регистрацию через социальные сети и много многое другое.

# Список литературы

1. [Введение в Django & DevOps](https://ivanovskiy999.gitbook.io/django/)
2. [Django учебник Часть 3: Использование моделей](https://developer.mozilla.org/ru/docs/Learn/Server-side/Django/Models)
3. [AndreiMalevanyi. Работа с SQLite в Python](https://habr.com/ru/articles/754400/)
4. [Начинаем работу с Django — подключение админки](https://itman.in/django-howto-2/)
5. [Django в примерах](https://pocoz.gitbooks.io/django-v-primerah/content/glava-7-sozdanie-internet-magazina/sozdanie-korzini/ispolzovanie-sessii-django.html)
6. [Python: Разработка на фреймворке Django](https://ru.hexlet.io/courses/python-django-basics/lessons/forms/theory_unit)
7. [Формы, связанные с моделями. Пользовательские валидаторы](https://proproprogs.ru/django/formy-svyazannye-s-modelyami-polzovatelskie-validatory)
8. [Пример Django Admin Stacked Inline: отношения многие-к-одному и многие-ко-многим](https://habr.com/ru/companies/otus/articles/690274/)
9. [Django template Using Message Message Framework](https://www.programmersought.com/article/88258121419/)
10. [Постраничная навигация (пагинация)](https://proproprogs.ru/django/postranichnaya-navigaciya-paginaciya)