

МИНОБРНАУКИ РОССИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет прикладной математики, информатики и механики

Кафедра математического и прикладного анализа

Направление 01.03.02 – Прикладная математика и информатика

Отчет по учебной практике  
по получению первичных профессиональных умений и навыков проектной и  
производственно-технологической деятельности  
студента 2 курса  
факультета Прикладной математики, информатики и механики  
Марных Никиты Владимировича

Сроки прохождения практики 06.07.2023 – 19.07.2023

Выполнила Ивановская Л.М.

Руководитель от кафедры д.ф.-м.н., проф. Половинкин И.П.

Воронеж 2024

Введение .....	3
1. Описание теоретических и практических аспектов выполненной работы.....	4
1.1. Примечания .....	4
2. Установка PyCharm и необходимых библиотек для создания проекта.....	7
3. Создание проекта в Django и реализация приложений для проверки критериев.....	10
3.1 Создание моделей.....	11
3.2 Создание Сериализаторов. ....	14
3.3 Создание представлений. ....	16
3.4 Добавление URL адресов приложений.....	19
3.5 Создание разрешений.....	21
3.7 Добавление авторизации.....	23
3.7.1 Авторизация по сессиям.....	24
3.7.2 Авторизация по токенам. ....	24
3.7.3 Авторизация по JWT токенам. ....	25
3.7.8 Работа с PostgreSQL .....	25
4. Исследование библиотек автодокументаций.....	27
4.1 Библиотека drf-yasg.....	27
4.1.1 Скорость разработки.....	27
4.1.2 Безопасность.....	28
4.1.3 Удобство интерфейса.....	29
4.1.4 Репозиторий.....	32
4.2 Библиотека drf-spectacular.....	34
4.2.1 Скорость разработки.....	34
4.2.2 Безопасность.....	35
4.2.3 Удобство интерфейса.....	35
4.2.4 Репозиторий.....	39
4.3 Библиотека drf-swagger.....	40
4.4 Библиотека drf-docs.....	41
5. Заключение.....	41
6. Список литературы.....	42
7. Ссылка на GitHub с исследованием.....	42

# Введение

Цель проведения учебной практики заключается в исследовании

Python-библиотек, которые обеспечивают автодокументирование API фреймворка Django, с учетом следующих критериев:

1. Скорость разработки
2. Безопасность
3. Удобство интерфейса
4. Репозиторий

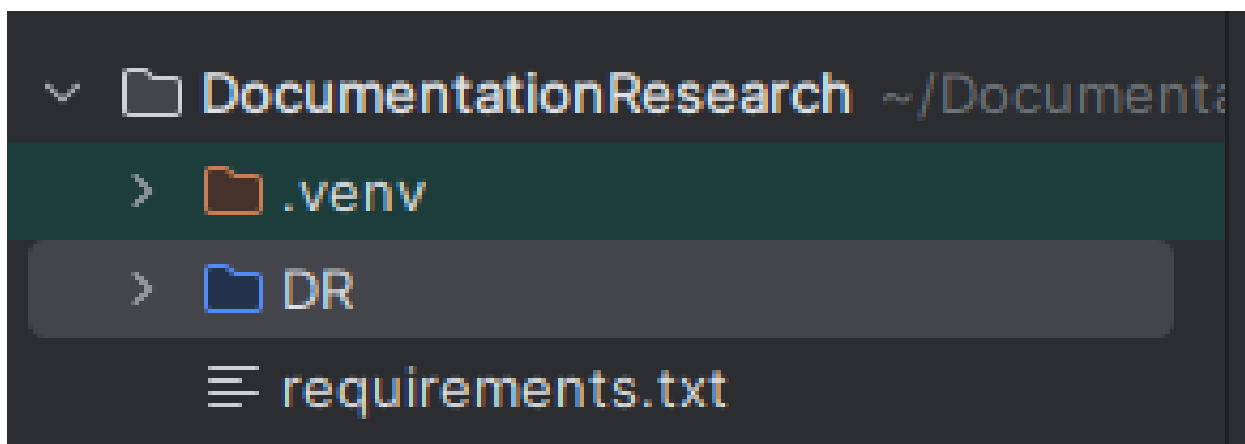
Для достижения данной цели потребовалось решить следующие задачи:

1. Изучение теории для работы с Python-библиотеками и базами данных PostgreSQL, Django, Django Rest Framework, Requests, Djoser, Django Filter.
2. Выполнить следующие практические задания:
  - 2.1 Установить Pycharm и библиотеки Django, Django Rest Framework, Django Filter, Djoser, Requests, Psycopg2.
  - 2.2 Запустить Django. Создать проект.
  - 2.3 Разработать приложения для оценки библиотеки автодокументации по критериям.
  - 2.4 Необходимо провести анализ существующих библиотек автодокументирования и оценить их эффективность на приложениях по заданным критериям.

# 1. Описание теоретических и практических аспектов выполненной работы.

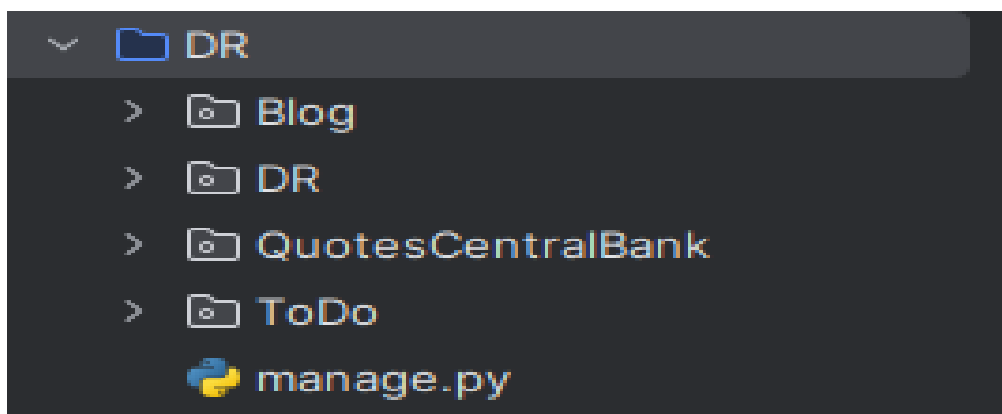
## 1.1. Примечания

Внешний вид директории DocumentationResearch в Pycharm:



(Рис. 1)

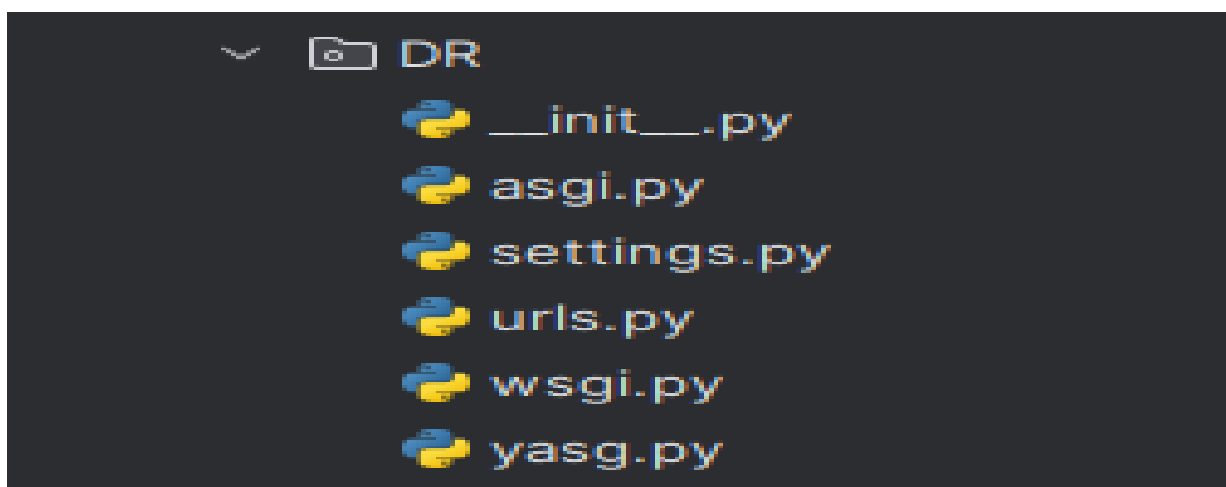
Внешний вид проекта DR (research on the autodocumentation library) в PyCharm:



(Рис. 2)

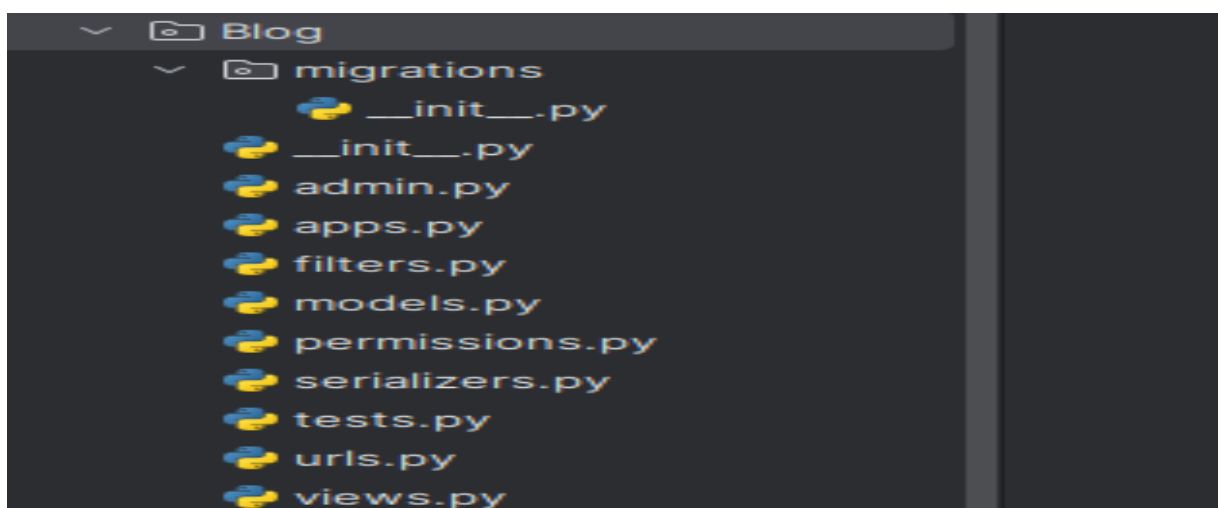
Внешний вид основного приложения DR (research on the autodocumentation

library) в PyCharm:



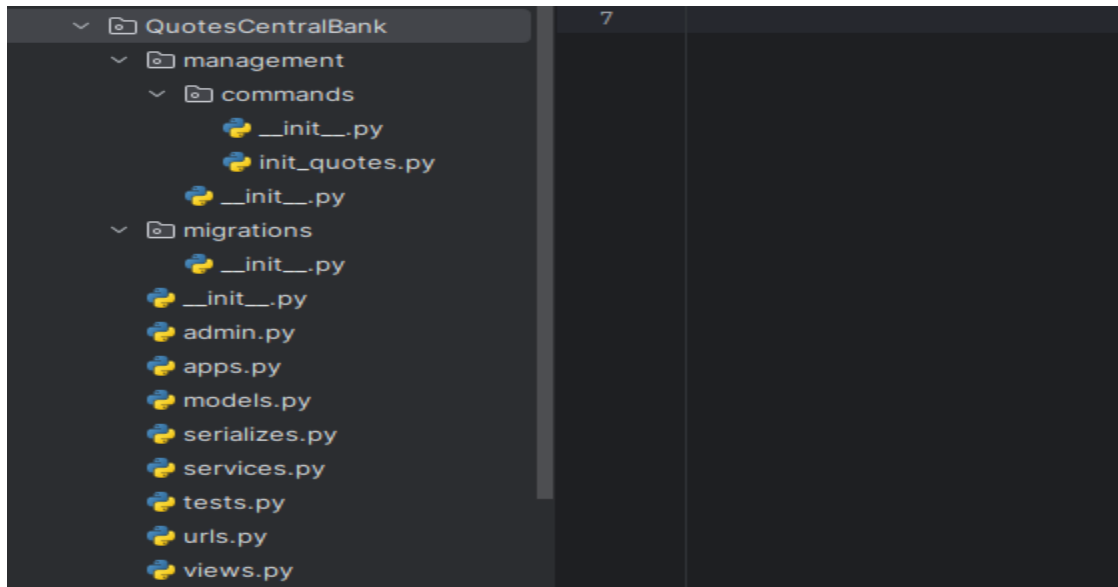
(Рис. 3)

Внешний вид приложения Blog в PyCharm:



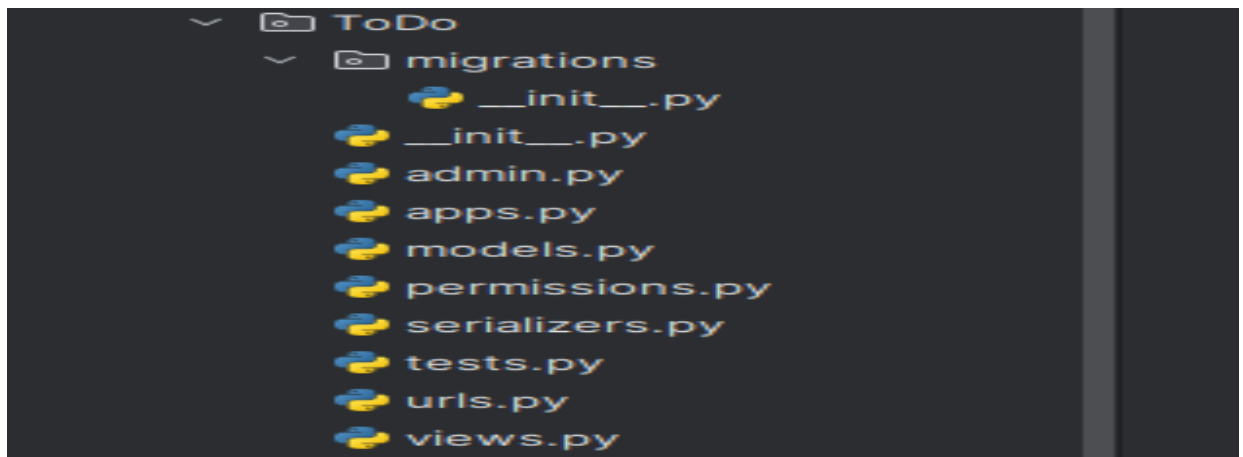
(Рис. 4)

Внешний вид приложения QuotesCentralBank в PyCharm:



(Рис. 5)

Внешний вид приложения ToDo в PyCharm:



(Рис. 6)

**pip install -r requirements.txt** установит все необходимые для запуска проекта библиотеки. (Django, Django Rest Framework, Django Filter, Djoser, Requests, Psycopg2)

**python manage.py makemigrations** – создаёт миграции БД.

**python manage.py migrate** - запускает миграции в БД.

**python manage.py runserver** - запускает локальный сервер.

**python manage.py createsuperuser** - создание суперпользователя в БД.

**django-admin startproject** <Имя проекта> - создание проекта.

**python manage.py startapp** <Имя приложения> - создание приложения.

**pip install** <Название библиотеки> - установка дополнительных пакетов.

В случае ошибки установки пакетов из requirements.txt:

**pip install django** - установка фреймворка Django

**pip install djangorestframework** - установка Django Rest Framework

**pip install requests** - установка Requests

**pip install django\_filter** - установка Django Filter

**pip install djoser** - установка Djoser

**pip install psycpg2** - установка Psycpg2

**pip install drf-spectacular[sidecar]** - установка drf-spectacular

**pip install drf-yasg** - установка drf-yasg

**python manage.py init\_quotes** - добавления котировок в базу данных

## **2. Установка PyCharm и необходимых библиотек для создания проекта.**

Необходимо загрузить установочный пакет PyCharm с официального сайта JetBrains и установить его.

**pip install -r requirements.txt** используется для установки всех необходимых библиотек (если скачали репозиторий).

**pip install requests** используется для установки библиотеки Requests.

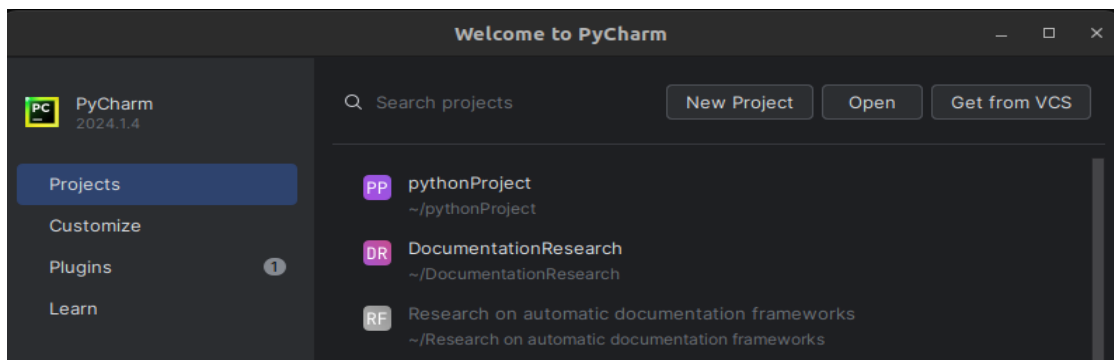
**pip install django** используется для установки библиотеки Django.

**pip install djangorestframework** используется для установки библиотеки Django Rest Framework.

**pip install djoser** используется для установки библиотеки Djoser.

**pip install psycpg2** используется для установки библиотеки Psycpg2.

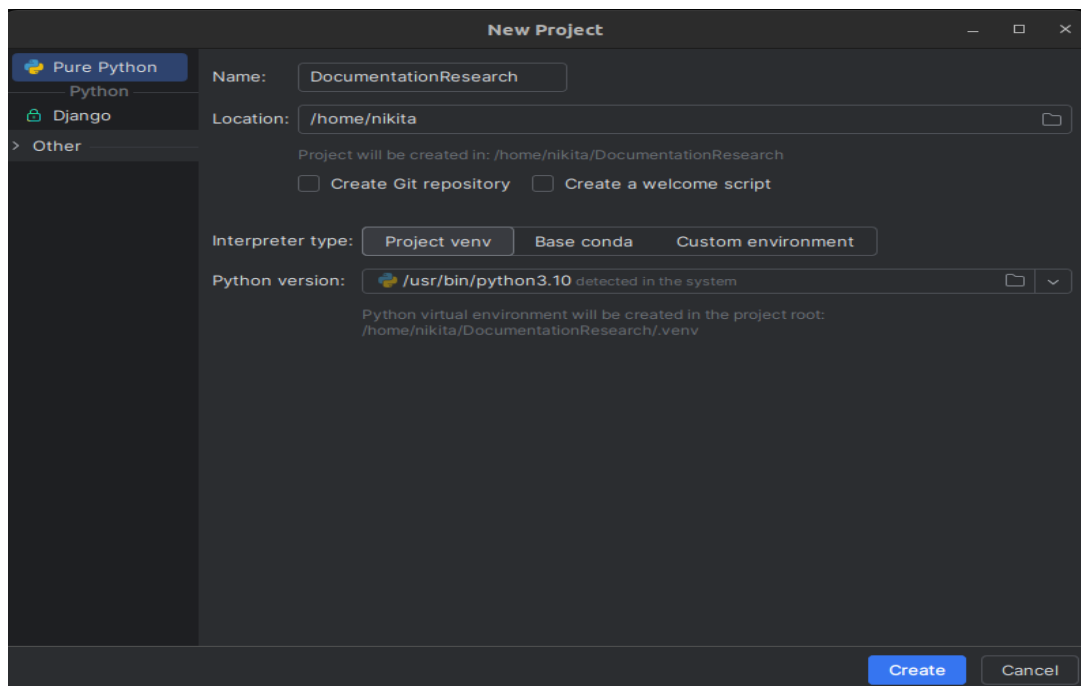
Открываем PyCharm.



(Рис. 7)

Нажимаем на “New Project” и вводим в поле “name” название директории DocumentationsResearch, нажимаем на “create”.





(Рис. 8)

В созданном проекте открываем консоль и прописываем команды:

**pip install django**

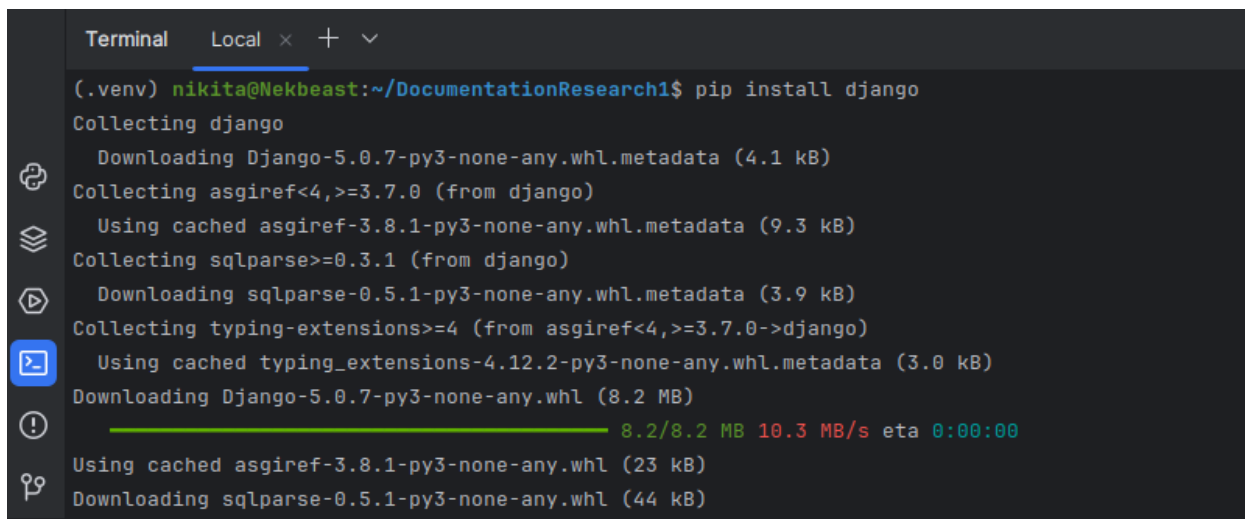
**pip install djangorestframework**

**pip install request**

**pip install djoser**

**pip install django\_filter**

**pip install psycopg2**



```
Terminal Local x + v
(.venv) nikita@Nekbeast:~/DocumentationResearch1$ pip install django
Collecting django
  Downloading Django-5.0.7-py3-none-any.whl.metadata (4.1 kB)
Collecting asgiref<4,>=3.7.0 (from django)
  Using cached asgiref-3.8.1-py3-none-any.whl.metadata (9.3 kB)
Collecting sqlparse>=0.3.1 (from django)
  Downloading sqlparse-0.5.1-py3-none-any.whl.metadata (3.9 kB)
Collecting typing-extensions>=4 (from asgiref<4,>=3.7.0->django)
  Using cached typing_extensions-4.12.2-py3-none-any.whl.metadata (3.0 kB)
Downloading Django-5.0.7-py3-none-any.whl (8.2 MB)
 8.2/8.2 MB 10.3 MB/s eta 0:00:00
Using cached asgiref-3.8.1-py3-none-any.whl (23 kB)
Downloading sqlparse-0.5.1-py3-none-any.whl (44 kB)
```

(Рис. 9)

### 3. Создание проекта в Django и реализация приложений для проверки критериев

Теперь в консоли прописываем:

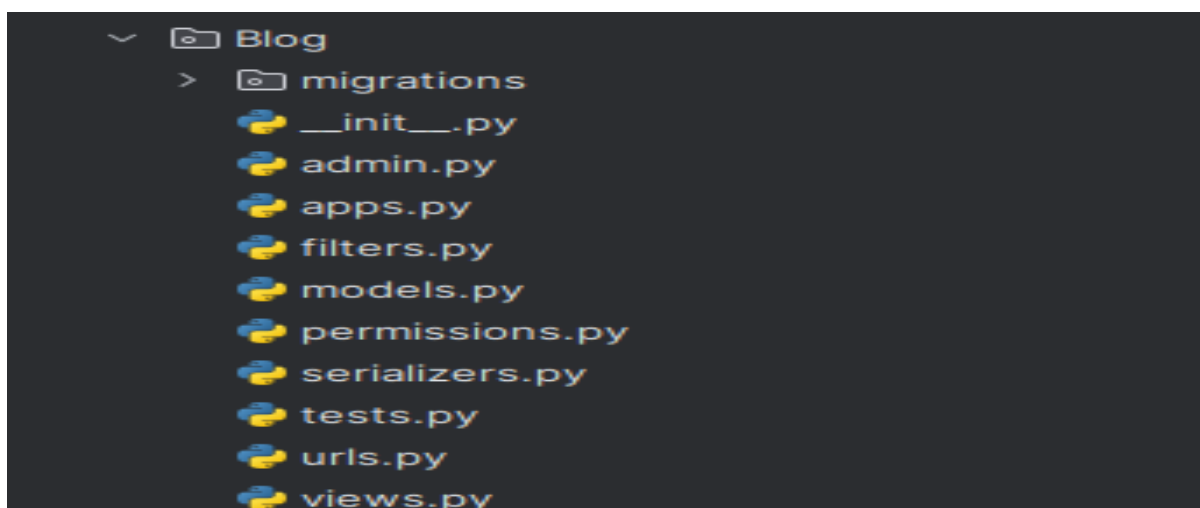
**django-admin startproject D**

**python manage.py startapp Blog**

**python manage.py startapp QuotesCentralBank**

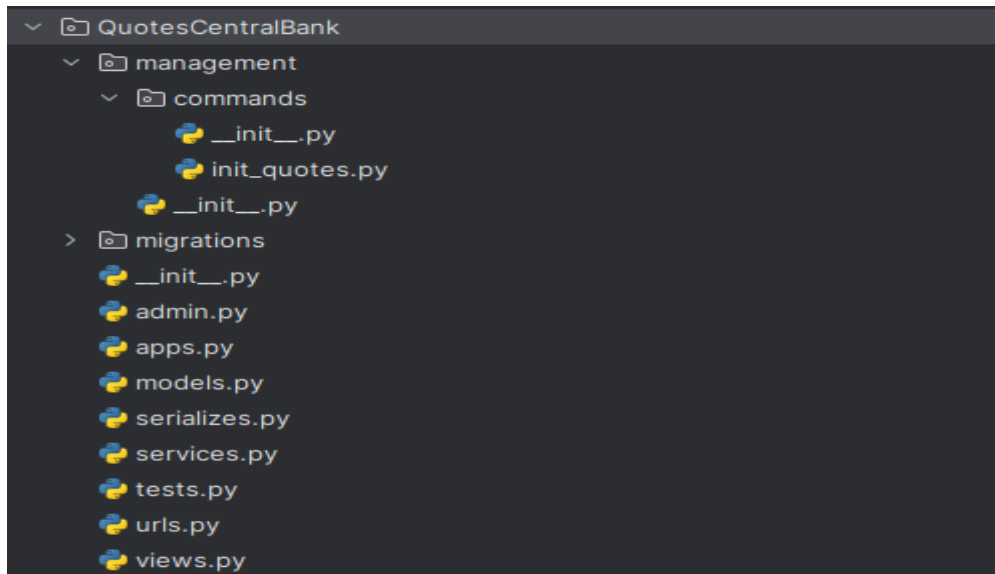
**python manage.py startapp ToDo**

В приложении Blog создаём файлы `filter.py`, `permissions.py`, `serializers.py`, `urls.py`.



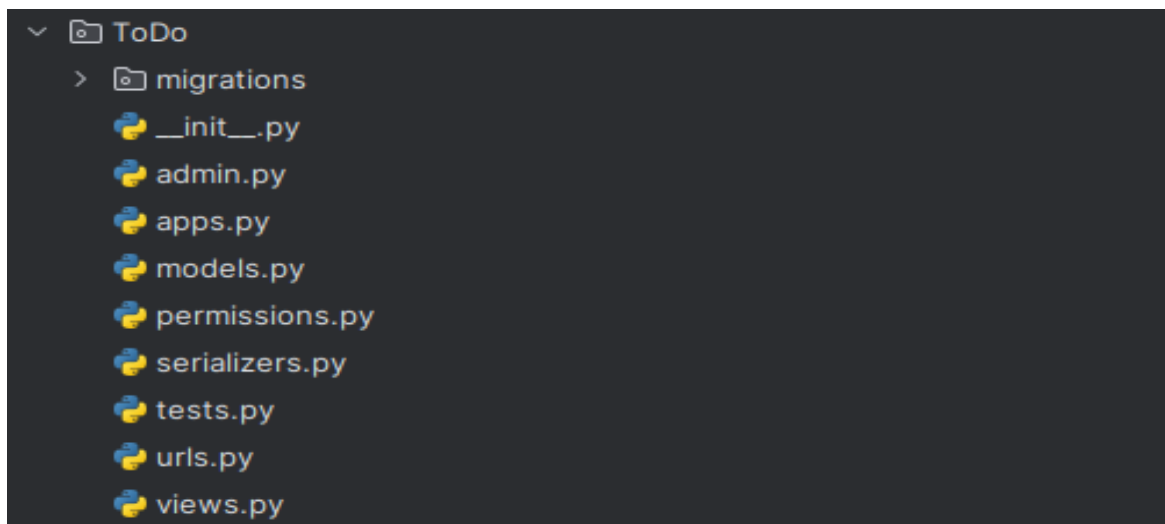
(Рис. 10)

В Приложении QuotesCentralBank файлы `serializes.py`, `service.py`, `urls.py`, а также создаём папку `management` и в ней же папку `commands`. В папке `commands` создаём файл `init_quotes`.



(Рис. 11)

В приложении ToDo создаём файлы `permissions.py`, `serializers.py`, `urls.py`.



(Рис. 12)

### 3.1 Создание моделей.

**Модели** в Django представляют собой классы Python, которые описывают структуру, свойства и взаимосвязи таблиц в базе данных. **Модели** не только

играют ключевую роль в создании и управлении схемой базы данных, но также обеспечивают удобный интерфейс для работы с данными.

**Модели** содержат поля, которые представляют столбцы в таблице БД. Эти поля определяют тип данных, валидацию и другие свойства для хранения, извлечения и обновления данных.

Когда приложения созданы их необходимо реализовать.

Начнём с создания моделей.

В приложении Blog создаём модели постов, комментариев и категорий в файле `models.py`:

```
from django.db import models

# Модель постов
8 usages
class Post(models.Model):
    created = models.DateTimeField(null=False, blank=False, auto_now_add=True)
    title: str = models.CharField(null=False, max_length=100, blank=True, default='')
    body: str = models.TextField(null=False, blank=True, default='', max_length=30000)
    owner = models.ForeignKey(to='auth.User', related_name='posts', on_delete=models.CASCADE, null=False, blank=False)

    class Meta:
        ordering = ['created']

    def __str__(self):
        return self.title

    def __repr__(self):
        return self.title, self.body, self.owner, self.created
```

(Рис. 13)

```
# Модель комментариев
6 usages
class Comment(models.Model):
    created = models.DateTimeField(null=False, blank=False, auto_now_add=True)
    body: str = models.TextField(null=False, blank=True, default='', max_length=2000)
    owner = models.ForeignKey(to='auth.User', related_name='comments', on_delete=models.CASCADE)
    post = models.ForeignKey(to='Post', related_name='comments', on_delete=models.CASCADE)

    class Meta:
        ordering = ['created']

    def __str__(self):
        return self.owner

    def __repr__(self):
        return self.body, self.owner, self.post, self.created
```

(Рис. 14)

```
# Модель категорий
6 usages
class Category(models.Model):
    name: str = models.CharField(max_length=100, blank=False, null=False, default='')
    owner = models.ForeignKey(to: 'auth.User', related_name='categories', on_delete=models.CASCADE)
    posts = models.ManyToManyField(to: 'Post', related_name='categories', blank=True)

    class Meta:
        verbose_name_plural = 'categories'

    def __str__(self):
        return self.name

    def __repr__(self):
        return self.name, self.owner, self.posts
```

(Рис. 15)

В приложении QuotesCentralBank создаём модель котировок в файле models.py:

```
from django.db import models

# Create your models here.

# Модель котировок
9 usages
class Quote(models.Model):
    name: str = models.CharField(null=False, blank=False, max_length=50)
    nominal: int = models.IntegerField(null=False, blank=False)
    num_code: int = models.CharField(null=False, blank=False, max_length=5)
    char_code: int = models.CharField(null=False, blank=False, max_length=5)
    vunit_rate: float = models.FloatField(null=False, blank=False)

    def __str__(self):
        return self.name

    def __repr__(self):
        return self.name, self.nominal, self.num_code, self.char_code, self.vunit_rate
```

(Рис. 16)

В приложении ToDo создаём модель списка задач в файле models.py:

```

from django.db import models

# Модель списка задач
6 usages
class TaskList(models.Model):
    title: str = models.CharField(max_length=50, null=False, blank=False)
    content: str = models.CharField(max_length=5000, null=True, blank=True)
    owner = models.ForeignKey(to='auth.User', on_delete=models.CASCADE)

    def __str__(self):
        return self.title

    def __repr__(self):
        return self.title, self.content, self.owner

```

(Рис. 17)

## 3.2 Создание Сериализаторов.

Сериализаторы позволяют преобразовывать сложные данные, такие как наборы запросов и экземпляры моделей, в собственные типы данных Python, которые затем можно легко визуализировать в JSON, XML или другие типы контента. Сериализаторы также обеспечивают десериализацию, позволяя преобразовывать проанализированные данные обратно в сложные типы после предварительной проверки входящих данных.

В приложении Blog в файле `serializers.py` создаём сериализаторы постов, категорий, комментариев:

```

from rest_framework import serializers
from .models import Post, Comment, Category

# Сериализатор категорий
1 usage
class CategorySerializer(serializers.ModelSerializer):
    owner = serializers.ReadOnlyField(source='owner.username')
    posts = serializers.PrimaryKeyRelatedField(many=True, read_only=True)

    class Meta:
        model = Category
        fields = ('id', 'name', 'owner', 'posts')

# Сериализатор постов
2 usages
class PostSerializer(serializers.ModelSerializer):
    owner = serializers.ReadOnlyField(source='owner.username')
    comments = serializers.PrimaryKeyRelatedField(many=True, read_only=True)

    class Meta:
        model = Post
        fields = ('id', 'title', 'body', 'owner', 'comments', 'categories')

# Сериализатор комментариев
1 usage
class CommentSerializer(serializers.ModelSerializer):
    owner = serializers.ReadOnlyField(source='owner.username')

    class Meta:
        model = Comment
        fields = ('id', 'body', 'owner', 'post')

```

(Рис. 18)

В приложении QuotesCentralBank в файле serializers.py создаём сериализатор котировок:

```

from rest_framework import serializers
from .models import Quote

# Сериализация моделей для API

# Сериализатор котировок
2 usages
class QuoteSerializer(serializers.ModelSerializer):
    class Meta:
        model = Quote
        fields = '__all__'

```

(Рис. 19)

В приложении ToDo в файле `serializers.py` создаём сериализатор списка задач:

```
from rest_framework import serializers

from ToDo.models import TaskList

# Сериализатор списка задач
2 usages
class TaskListSerializer(serializers.ModelSerializer):
    owner = serializers.ReadOnlyField(source='owner.username')

    class Meta:
        model = TaskList
        fields = '__all__'
```

(Рис. 20)

### 3.3 Создание представлений.

Представления в Django — это компоненты, которые обрабатывают запросы от клиента и возвращают соответствующие ответы. Они являются связующим звеном между моделями данных и шаблонами пользовательского интерфейса.

Основная цель представлений — принимать данные из моделей или других источников, обрабатывать их и передавать результаты в шаблоны для отображения пользователю.

В приложении Blog в файле `views.py` создадим представления постов, комментариев и категорий на основе класса:



```

from rest_framework import mixins
from rest_framework.viewsets import GenericViewSet
from django_filters.rest_framework import DjangoFilterBackend

from .models import Post, Comment, Category
from .serializers import PostSerializer
from .permissions import IsOwnerOrAdminOrAuthenticatedReadAndCreate
from .filters import PostFilter
from . import serializers

# Представление API постов с возможностью фильтровать по названию и создателю,
# отдельный JSON каждого поста с возможностью удаления, обновления,
# создания для зарегистрированных пользователей'''
! usage
class PostViewSet(mixins.ListModelMixin,
                  mixins.RetrieveModelMixin,
                  mixins.CreateModelMixin,
                  mixins.UpdateModelMixin,
                  mixins.DestroyModelMixin,
                  GenericViewSet):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
    filter_backends = (DjangoFilterBackend,)
    filter_set_class = PostFilter
    permission_classes = (IsOwnerOrAdminOrAuthenticatedReadAndCreate, )

    def perform_create(self, serializer):
        serializer.save(owner=self.request.user)

```

(Рис. 21)

```

# Представление API комментариев с возможностью фильтровать по названию и создателю,
# отдельный JSON каждого поста с возможностью удаления, обновления,
# создания для зарегистрированных пользователей'''
! usage
class CommentViewSet(mixins.ListModelMixin,
                     mixins.RetrieveModelMixin,
                     mixins.CreateModelMixin,
                     mixins.UpdateModelMixin,
                     mixins.DestroyModelMixin,
                     GenericViewSet):
    queryset = Comment.objects.all()
    serializer_class = serializers.CommentSerializer
    permission_classes = (IsOwnerOrAdminOrAuthenticatedReadAndCreate, )

    def perform_create(self, serializer):
        serializer.save(owner=self.request.user)

```

(Рис. 22)

```

✓ # Представление API категорий с возможностью фильтровать по названию и создателю,
# отдельный JSON каждого поста с возможностью удаления, обновления,
# создания для зарегистрированных пользователей'''
1 usage
class CategoryViewSet(mixins.ListModelMixin,
                      mixins.RetrieveModelMixin,
                      mixins.CreateModelMixin,
                      mixins.UpdateModelMixin,
                      mixins.DestroyModelMixin,
✓                      GenericViewSet):
    queryset = Category.objects.all()
    serializer_class = serializers.CategorySerializer
    permission_classes = (IsOwnerOrAdminOrAuthenticatedReadAndCreate, )

✓ def perform_create(self, serializer):
    serializer.save(owner=self.request.user)

```

(Рис. 23)

В приложении QuotesCentralBank в файле views.py создаём представление котировок:

```

1  from rest_framework import permissions
2
3  from rest_framework.viewsets import mixins, GenericViewSet
4
5  from .models import Quote
6  from .serializers import QuoteSerializer
7
8
9  # Представление API всех котировок или одной необходимой
2 usages
10 class QuotesViewSet(mixins.ListModelMixin, mixins.RetrieveModelMixin, GenericViewSet):
11  @+   queryset = Quote.objects.all()
12  @+   ⚡ serializer_class = QuoteSerializer
13  @+   permission_classes = (permissions.IsAuthenticated, )

```

(Рис. 24)

В приложении ToDo в файле views.py создаём представление списка задач:

```
1 from rest_framework import mixins
2 from rest_framework.viewsets import GenericViewSet
3
4 from ToDo.models import TaskList
5 from ToDo.serializers import TaskListSerializer
6 from ToDo.permissions import IsOwnerOrAdminOrAuthenticatedReadAndCreate
7
8
9 # Представление API списка задач с возможностью фильтровать по названию и создателю,
10 # отдельный JSON каждого списка с возможностью удаления, обновления,
11 # создания для зарегистрированных пользователей'''
12 usage
13 class TaskListViewSet(mixins.ListModelMixin,
14                      mixins.RetrieveModelMixin,
15                      mixins.CreateModelMixin,
16                      mixins.UpdateModelMixin,
17                      mixins.DestroyModelMixin,
18                      GenericViewSet):
19
20     queryset = TaskList.objects.all()
21     serializer_class = TaskListSerializer
22     permission_classes = (IsOwnerOrAdminOrAuthenticatedReadAndCreate, )
23
24     def perform_create(self, serializer):
25         serializer.save(owner=self.request.user)
26
27 # Каждому пользователю показывает только его список задач, администратору все списки
28 def get_queryset(self):
29     if self.request.user.is_staff:
30         return super().get_queryset().all()
31     return super().get_queryset().filter(owner=self.request.user)
```

(Рис. 25)

### 3.4 Добавление URL адресов приложений.

URL - адресация в Django является важной частью разработки веб-приложений, поскольку она определяет, как приложение отвечает на запросы пользователей. В Django URL - адресация осуществляется с помощью механизма маршрутизации, который связывает конкретные URL - адреса с функциями представления, обрабатывающими эти запросы.

В приложениях Blog, ToDo, QuotesCentralBank в файлах urls.py создаём роутеры для маршрутов и регистрируем в них классы представлений:

```

from django.urls import path, include
from . import views
from rest_framework import routers

router = routers.SimpleRouter()

# Подключаем к роутеру посты
router.register(prefix: 'posts', views.PostViewSet)

# Подключаем к роутеру комментарии
router.register(prefix: 'comments', views.CommentViewSet)

# Подключаем к роутеру категории
router.register(prefix: 'categories', views.CategoryViewSet)

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

```

(Рис. 26)

```

from django.urls import path, include
from rest_framework import routers

from .views import QuotesViewSet

router = routers.SimpleRouter()

# подключаем к роутеру котировки
router.register(prefix: '', QuotesViewSet)

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

```

(Рис. 27)

```

from django.urls import include, path
from rest_framework import routers

from ToDo import views

router = routers.SimpleRouter()
router.register(prefix: '', views.TaskListViewSet)

urlpatterns = (
    # Подключаем к роутеру списки задач
    path('api/', include(router.urls)),
)

```

(Рис. 28)

В файле urls.py главного приложения DR добавляем маршруты:

```

# URL адреса приложения Blog
path('blog/', include('Blog.urls')),
# URL адреса приложения QuotesCentralBank
path('quotes/', include('QuotesCentralBank.urls')),
# URL адреса приложения ToDo
path('todo/', include('ToDo.urls')),

```

(Рис. 29)

### 3.5 Создание разрешений.

В файле permissions.py приложений ToDo и Blog создать класс разрешения:

```

from rest_framework import permissions

4 usages
class IsOwnerOrAdminOrAuthenticatedReadAndCreate(permissions.BasePermission):

    # Разрешение на просмотр и создание дано только пользователям прошедшим проверку подлинности
    def has_permission(self, request, view):
        return request.user.is_authenticated

    # Разрешение на обновление дано только создателю, а удаление администратору и создателю
    def has_object_permission(self, request, view, obj):
        if view.action == "update" or view.action == "partial_update":
            return obj.owner == request.user

        return bool((request.user and request.user.is_staff) or (obj.owner == request.user))

```

(Рис. 30)

Система разрешений позволяет проверять учетные данные пользователей и определять права доступа для пользователей.

### 3.6 Скрейпинг и парсинг котировок ЦБ.

Скрейпинг — автоматизированный сбор данных.

Парсинг — процесс, на котором из скачанных данных извлекается нужная информация и превращается в нужный нам читаемый формат. Проще говоря, второй этап веб-скрейпинга.

В `services.py` приложения `QuotesCentralBank` создаём функцию для сбора котировок ЦБ.

```
import requests
import xml.etree.ElementTree as Etree
from datetime import datetime
from DR.settings import EXTERNAL_URLS

# Получаем котировки ЦБ для заполнения БД
2 usages
def get_quotes() -> list[dict[str, str | int | float]] | None:
    # Ассоциативный словарь для столбцов БД
    keys: dict[str, str | int | float] = {"Name": 'name',
                                         'Nominal': 'nominal',
                                         'NumCode': 'num_code',
                                         'CharCode': 'char_code',
                                         'VunitRate': 'vunit_rate'
                                         }

    # Получаем актуальные данные
    today = datetime.now().strftime('%d/%m/%Y')
    response = requests.get(f'{EXTERNAL_URLS["quotes_cbfr"]}{today}')

    try:
        if response.status_code != 200:
            raise requests.RequestException()
    except requests.RequestException:
        print(f'Ошибка инициализации котировок. Сайт Центробанка недоступен. Статус код: {response.status_code}')
        return None

    root = Etree.fromstring(response.text)

    quotes: list[dict[str, str | int | float]] = []
```

(Рис. 31)

В `init_quotes.py` приложения `QuotesCentralBank` создаём командный класс для добавления котировок ЦБ в БД.

```
from django.core.management.base import BaseCommand
from QuotesCentralBank.models import Quote
from QuotesCentralBank.services import get_quotes

# Заполняем базу данных котировками
class Command(BaseCommand):
    help: str = "Инициализация котировок"

    def handle(self, *args, **options):
        quotes = get_quotes()
        if quotes is None:
            self.stdout.write("Инициализация котировок не удалась")
        else:
            Quote.objects.bulk_create(
                [Quote(**quote) for quote in quotes]
            )
            self.stdout.write("Инициализация котировок прошла успешно")
```

(Рис. 32)

### 3.7 Добавление авторизации.

Авторизация в Django REST Framework (DRF) — это методы и механизмы, которые используются для проверки личности пользователей, обращающихся к веб-API, построенному с помощью Django REST Framework.

Авторизация обеспечивает доступ только авторизованным пользователям к API.

В settings.py приложения DR добавляем словарь:

```
2 REST_FRAMEWORK = {  
3     'DEFAULT_PERMISSION_CLASSES': (  
4         'rest_framework.permissions.IsAuthenticated',  
5     ),  
6  
7     'DEFAULT_AUTHENTICATION_CLASSES': (  
8         'rest_framework_simplejwt.authentication.JWTAuthentication',  
9         'rest_framework.authentication.TokenAuthentication',  
10        'rest_framework.authentication.SessionAuthentication',  
11    ),  
12  
13    'DEFAULT_THROTTLE_CLASSES': [  
14        'rest_framework.throttling.UserRateThrottle',  
15    ],  
16  
17    'DEFAULT_THROTTLE_RATES': {  
18        'user': '60/min',  
19    },  
20 }
```

(Рис. 33)

### 3.7.1 Авторизация по сессиям.

В главном приложении DR добавляем маршрут в файл urls.py в кортеж urlpatterns:

```
# URL адреса Аунтефикации  
path('api/baseauth/', include('rest_framework.urls')),
```

(Рис. 34)

### 3.7.2 Авторизация по токенам.

В главном приложении DR добавляем маршруты в файл urls.py в кортеж



urlpatterns:

```
re_path(r'^api/auth/', include('djoser.urls')),  
re_path(r'^api/authorization/', include('djoser.urls.authtoken')),
```

(Рис. 35)

### 3.7.3 Авторизация по JWT токенам.

В главном приложении DR добавляем маршруты в файл urls.py в кортеж urlpatterns:

```
path('api/JWT/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),  
path('api/JWT/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),  
path('api/JWT/token/verify/', TokenVerifyView.as_view(), name='token_verify'),
```

(Рис. 36)

В settings.py приложения DR добавляем словарь:

```
2 REST_FRAMEWORK = {  
3     'DEFAULT_PERMISSION_CLASSES': (  
4         'rest_framework.permissions.IsAuthenticated',  
5     ),  
6  
7     'DEFAULT_AUTHENTICATION_CLASSES': (  
8         'rest_framework_simplejwt.authentication.JWTAuthentication',  
9         'rest_framework.authentication.TokenAuthentication',  
10        'rest_framework.authentication.SessionAuthentication',  
11    ),  
12  
13    'DEFAULT_THROTTLE_CLASSES': [  
14        'rest_framework.throttling.UserRateThrottle',  
15    ],  
16  
17    'DEFAULT_THROTTLE_RATES': {  
18        'user': '60/min',  
19    },  
20 }  
21
```

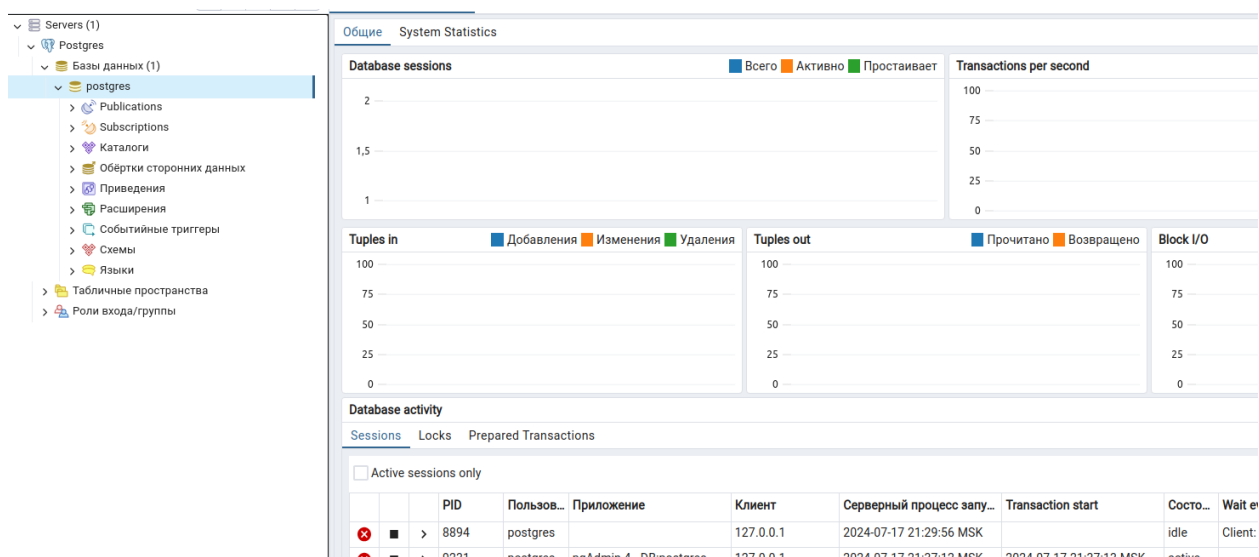
(Рис. 37)

### 3.7.8 Работа с PostgreSQL

Для работы с PostgreSQL необходима библиотека psycopg2.

Заходим в pgAdmin4 создаём сервер или используем уже существующий.

Далее создаём базу данных



(Рис. 38)

Для подключения к базе данных, измените в Django:

```
serializers.py  Blog/serializers.py  ToDo/permissions.py  ToDo/serializer

13
14     'DEFAULT_SCHEMA_CLASS': 'drf_spectacular.openapi.AutoSchema'
15 }
16
17
18 # Database
19 # https://docs.djangoproject.com/en/5.0/ref/settings/#databases
20
21 DATABASES = {
22     'default': {
23         'ENGINE': 'django.db.backends.postgresql',
24         'NAME': 'postgres',
25         'USER': 'postgres',
26         'PASSWORD': 'postgres',
27         'HOST': 'localhost',
28         'PORT': 5432,
29     }
30 }
31
32
```

(Рис. 39)

Создаём миграции **python manage.py makemigrations**

Мигрируем **python manage.py migrate**

Запускаем сервер `python manage.py runserver`

## 4. Исследование библиотек автодокументаций.

### 4.1 Библиотека drf-yasg.

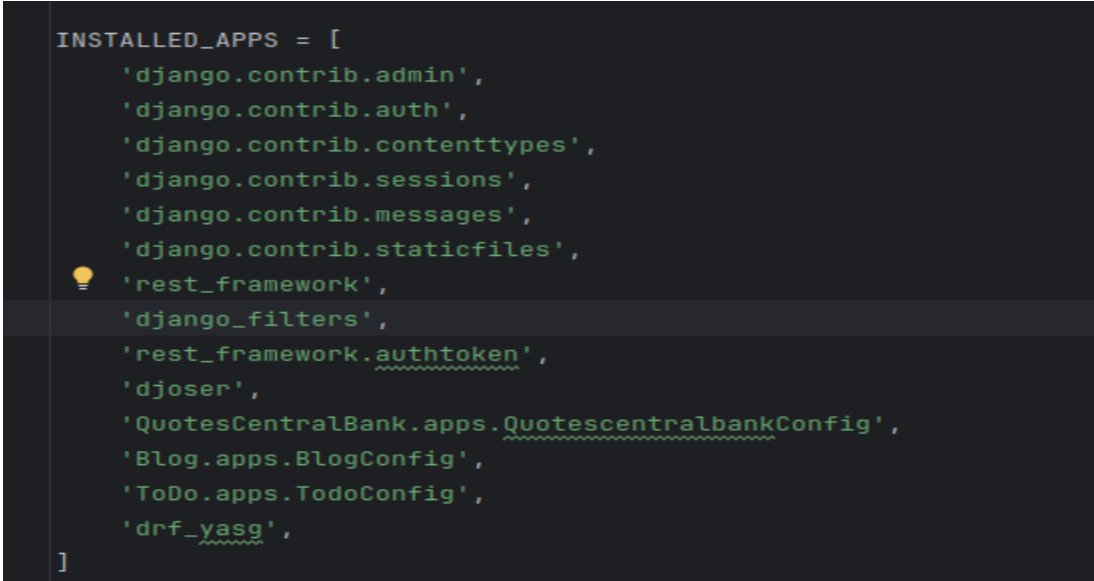
Установим библиотеку:

`pip install drf-yasg`

#### 4.1.1 Скорость разработки.

Подключение автодокументации происходит в 3 шага:

Добавляем в settings.py приложения DR проложение 'drf\_yasg'



```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'rest_framework',  
    'django_filters',  
    'rest_framework.authtoken',  
    'djoser',  
    'QuotesCentralBank.apps.QuotescentralbankConfig',  
    'Blog.apps.BlogConfig',  
    'ToDo.apps.ToDoConfig',  
    'drf_yasg',  
]
```

(Рис. 40)

В приложении DR создаём файл yasg.py содержащий код из документации:

В файле url.py приложения DR подключаем список urlpatterns, как doc\_urls из yasg.py и прибавляем его к urlpatterns:

```

import rest_framework
from django.contrib import admin
from django.urls import path, include, re_path
from rest_framework_simplejwt.views import TokenObtainPairView, TokenRefreshView, TokenVerifyView
from .yasg import urlpatterns as doc_urls

urlpatterns = [
    # Админ панель
    path('admin/', admin.site.urls),
    # URL адреса Аутентификации
    path('api/baseauth/', include('rest_framework.urls')),
    re_path(r'^api/auth/', include('djoser.urls')),
    re_path(r'^api/authorization/', include('djoser.urls.auth_token')),
    path('api/JWT/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/JWT/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
    path('api/JWT/token/verify/', TokenVerifyView.as_view(), name='token_verify'),
    # URL адреса приложения Blog
    path('blog/', include('Blog.urls')),
    # URL адреса приложения QuotesCentralBank
    path('quotes/', include('QuotesCentralBank.urls')),
    # URL адреса приложения ToDo
    path('todo/', include('ToDo.urls')),
]

urlpatterns += doc_urls

```

(Рис. 41)

## 4.1.2 Безопасность.

### Конфигурация

Get\_schema\_view параметры:

info - Объект Swagger API Info; если не указан, по умолчанию используется DEFAULT\_INFO

url - Базовый URL-адрес API; если оставить пустым, он будет выведен из местоположения, в котором обслуживается представление

patterns - передано в SchemaGenerator

urlconf - передано в SchemaGenerator

public - если False, включает только конечные точки, к которым у текущего пользователя есть доступ

validators - список имен валидаторов для применения к сгенерированной схеме; ssvv настоящее время поддерживается только

`generator_class` - используемый класс генератора схем; должен быть подклассом `OpenAPISchemaGenerator`

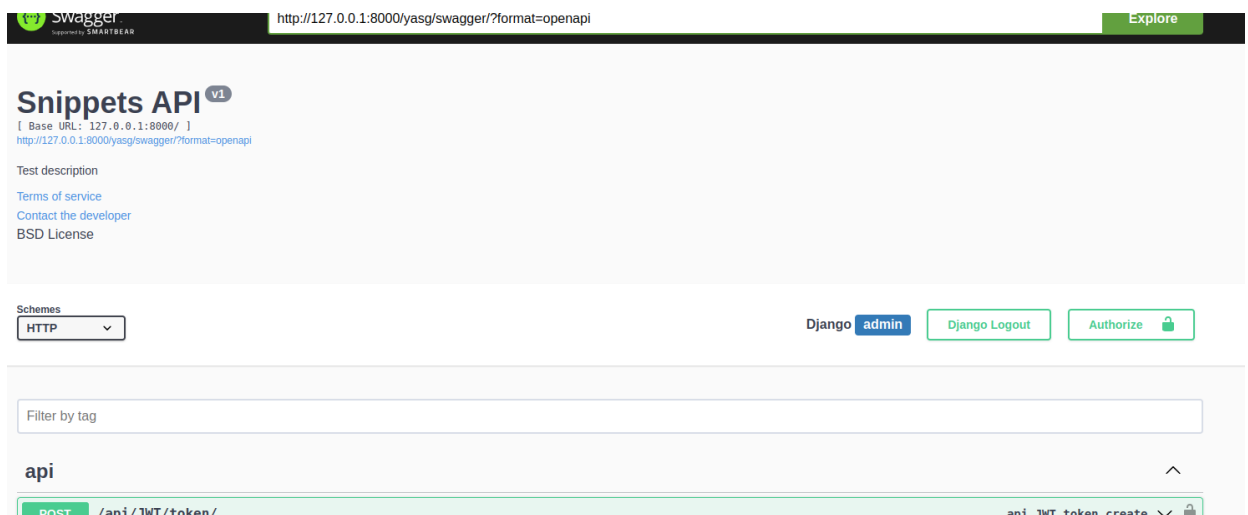
`authentication_classes` - классы аутентификации для самого представления схемы

`permission_classes` - классы разрешений для самого представления схем

Параметр `permission_classes` позволяет ограничить доступ к документации достаточно гибко, прописав собственный класс разрешений или воспользовавшись существующими.

### 4.1.3 Удобство интерфейса

Указан хост сервера, доступен поисковик, есть фильтр по тегам



(Рис. 42)

Запросы разнесены по приложениям, модели отделены от запросов

blog ^		
GET	/blog/api/categories/	blog_api_categories_list
POST	/blog/api/categories/	blog_api_categories_create
GET	/blog/api/categories/{id}/	blog_api_categories_read
PUT	/blog/api/categories/{id}/	blog_api_categories_update
PATCH	/blog/api/categories/{id}/	blog_api_categories_partial_update
DELETE	/blog/api/categories/{id}/	blog_api_categories_delete
GET	/blog/api/comments/	blog_api_comments_list
POST	/blog/api/comments/	blog_api_comments_create
GET	/blog/api/comments/{id}/	blog_api_comments_read
PUT	/blog/api/comments/{id}/	blog_api_comments_update
PATCH	/blog/api/comments/{id}/	blog_api_comments_partial_update
DELETE	/blog/api/comments/{id}/	blog_api_comments_delete

(Рис. 43)

DELETE	/blog/api/posts/{id}/	blog_api_posts_delete
quotes ^		
GET	/quotes/api/	quotes_api_list
GET	/quotes/api/{id}/	quotes_api_read
todo ^		
GET	/todo/api/	todo_api_list
POST	/todo/api/	todo_api_create
GET	/todo/api/{id}/	todo_api_read
PUT	/todo/api/{id}/	todo_api_update
PATCH	/todo/api/{id}/	todo_api_partial_update
DELETE	/todo/api/{id}/	todo_api_delete

(Рис. 44)

Доступна redoc документация

127.0.0.1:8000/yasg/redoc/

Q Search...

api >

blog >

quotes >

todo >

Snippets API (v1)

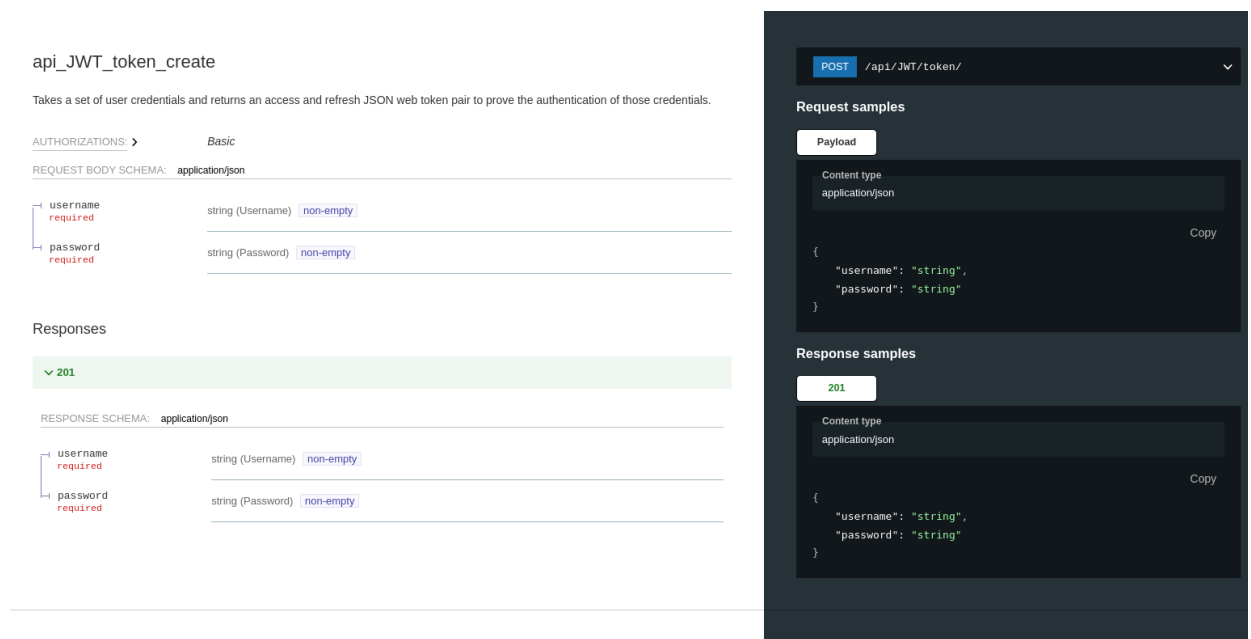
Download OpenAPI specification: [Download](#)

E-mail: [contact@snippets.local](mailto:contact@snippets.local) | License: [BSD License](#) | [Terms of Service](#)

Test description

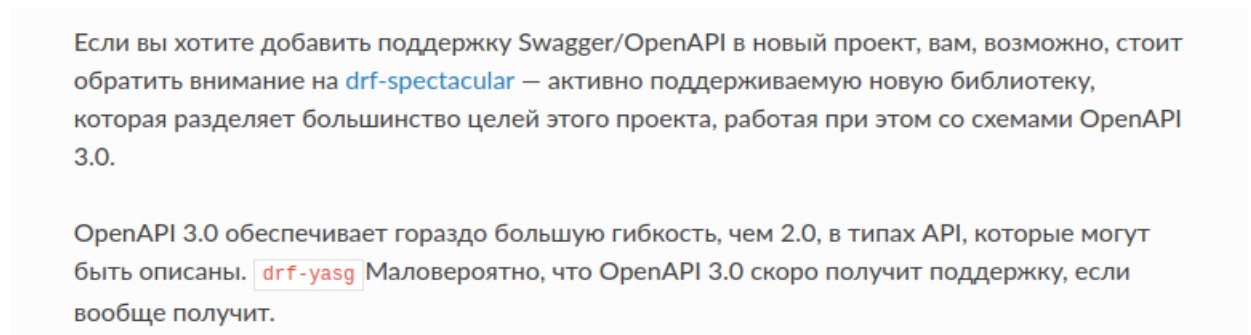
(Рис. 45)

30

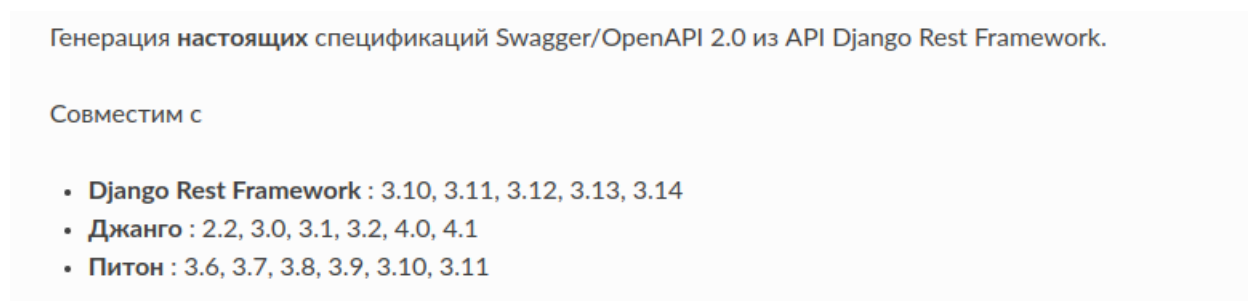


(Рис. 46)

drf-yasg поддерживает OpenAPI 2.0, разработчики предлагают перейти на drf-spectacular поддерживающий OpenAPI 3.0.



(Рис. 47)



(Рис. 48)

Функции:

- полная поддержка вложенных сериализаторов и схем
- Схемы и описания ответов
- определения моделей, совместимые с инструментами кодогенерации
- возможности настройки на всех этапах процесса генерации спецификаций
- Формат JSON и YAML для спецификации
- объединяет последнюю версию swagger-ui и redoc для просмотра сгенерированной документации
- Представление схемы кэшируется из коробки

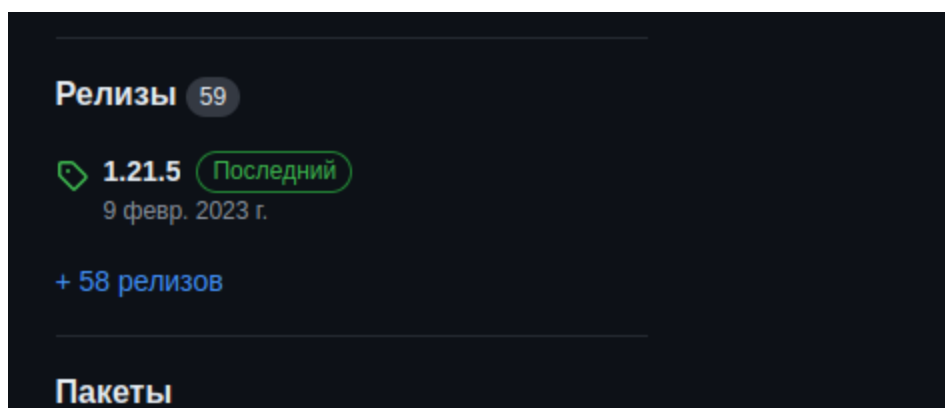
Сгенерированная схема Swagger может быть автоматически проверена с помощью swagger-spec-validator

поддерживает управление версиями API Django REST Framework с помощью URLPathVersioning и NamespaceVersioning; другие DRF или пользовательские схемы управления версиями в настоящее время не поддерживаются

#### 4.1.4 Репозиторий

3.4k пользователей следят за drf-yasg

(Рис. 49)



(Рис. 50)



Активность

Все филиалы | Вся деятельность | Все пользователи | Все время | Сначала показаны самые последние

Разрешить Swagger использовать пользовательские настройки CSRF и читать файлы cookie CSRF (#660) <a href="#">Слияние запросов на вытягивание</a>	...
Джоэл Лефковиц продвинул 1 коммит в 1.21.x • c1c25ee...5f889de • 21 июля 2023 г.	
Продвинутый мастер для выпуска 1.21.7 (#815) <a href="#">Слияние запросов на вытягивание</a>	...
Джоэл Лефковиц продвинул 1 коммит в владелец • 6557517...7803110 • 20 июля 2023 г.	
Добавить python3.6 в матрицу действий, но исключить его из тестов	...
Джоэл Лефковиц продвинул 1 коммит в 1.21.x • db42d35...c1c25ee • 20 июля 2023 г.	
Удаленный тег	...
ДжоэлЛефковиц удален ссылки/теги/v1.21.6 • 20 июля 2023 г.	
Обновление журнала изменений	...
Джоэл Лефковиц продвинул 1 коммит в 1.21.x • 2548298...db42d35 • 20 июля 2023 г.	
Тесты: Улучшение тестового покрытия 95,91% -> 98,30%. (#862) <a href="#">Слияние запросов на вытягивание</a>	...
Джоэл Лефковиц продвинул 1 коммит в 1.21.x • 5e239a8...2548298 • 20 июля 2023 г.	
[исправлено] Исправлено отображение источника карты (#859) <a href="#">Слияние запросов на вытягивание</a>	...
Джоэл Лефковиц продвинул 1 коммит в 1.21.x • 106d7b2...5e239a8 • 13 июля 2023 г.	
Особенность: Добавить drf_yasg.inspectors.query.DrfAPICompatInspector. (#857) <a href="#">Слияние запросов на вытягивание</a>	...
Джоэл Лефковиц продвинул 1 коммит в 1.21.x • 24c4c1b...106d7b2 • 5 июля 2023 г.	
Удаленный тег	...
ДжоэлЛефковиц удален ссылки/теги/v1.21.6 • 16 июня 2023 г.	
Обновление журнала изменений	...
Джоэл Лефковиц продвинул 1 коммит в 1.21.x • a836604...24c4c1b • 16 июня 2023 г.	
Исправление: Удалить требуемую зависимость coreapi. (#854) <a href="#">Слияние запросов на вытягивание</a>	...
Джоэл Лефковиц продвинул 1 коммит в 1.21.x • afea1bc...a836604 • 16 июня 2023 г.	
Функция: переход на PyYAML для генератора yaml. (#845) <a href="#">Слияние запросов на вытягивание</a>	...
Джоэл Лефковиц продвинул 1 коммит в 1.21.x • 353f071...afea1bc • 17 мая 2023 г.	

(Рис. 51)

203 открытых проблемы и 406 закрытых:

Проблемы 223 | Запросы на извлечение 35 | Действия | Безопасность | Инсайты

Хотите внести свой вклад в axnsan12/drf-yasg? [Увольнять](#)

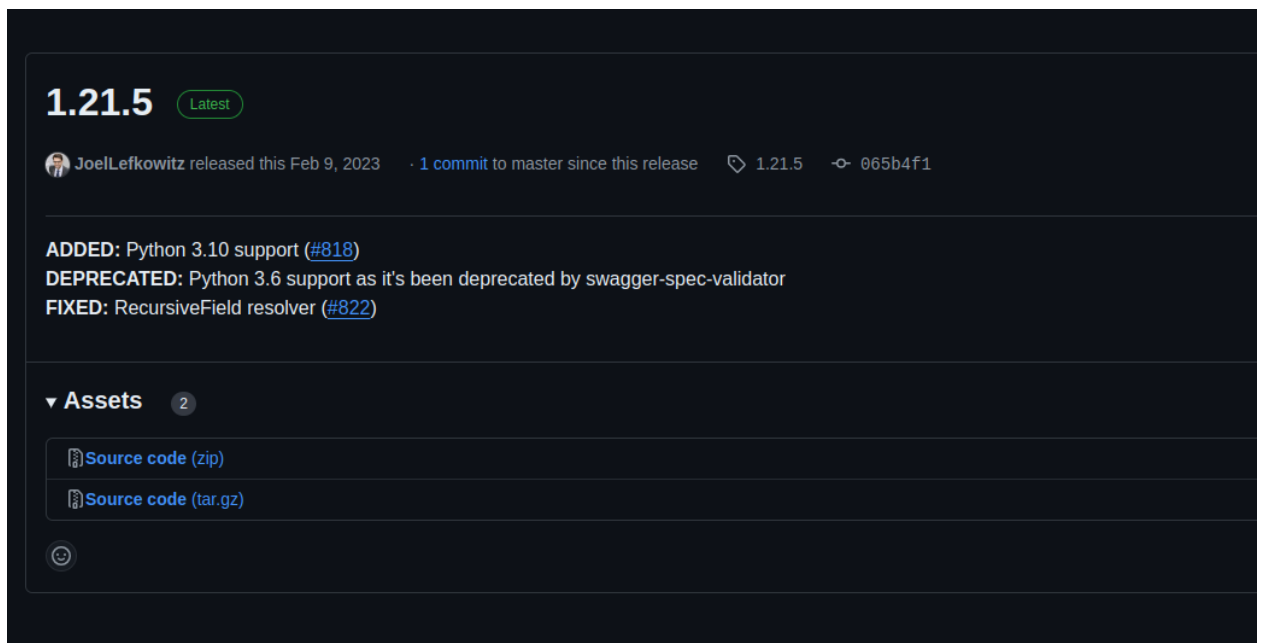
Если у вас есть сообщение об ошибке или идея, прочтите [правила внесения вклада](#), прежде чем открывать тему. Если вы готовы взяться за решение некоторых открытых вопросов, мы собрали для вас несколько хороших [первых вопросов](#).

Фильтры | Q is:issue is:open | Метки 16 | Вехи 9 | [Новый выпуск](#)

223 Открыто	406 Закрыто	Автор	Этикетка	Проекты	Вехи	Правопреемник	Сортировать
Запрос о генерации пользовательской схемы в drf-yasg							
#896 открытпрошлый месяцот AhnSeongHyun							
Войти с помощью JWT							
#882 открыт15 январюот towfish							
django_filters несовместим							
#881 открыт20 декабря 2023 г.от Jorgeamayaarabon							
Как удалить URL-адреса по умолчанию, сгенерированные base_url маршрутизатора из пользовательского интерфейса Swagger?							
#880 открыт18 декабря 2023 г.от fazialjnd							

(Рис. 52)

Последний релиз был в феврале 2023 года:



(Рис. 53)

## 4.2 Библиотека drf-spectacular.

Установим библиотеку:

**`pip install drf-spectacular[sidecar]`**

### 4.2.1 Скорость разработки

Подключение к проекту происходит в несколько шагов:

Добавляем в файл `settings.py` приложения DR:

```
INSTALLED_APPS = [  
    # ALL YOUR APPS  
    'drf_spectacular',  
    'drf_spectacular_sidecar', # required for Django collectstatic discovery  
]  
SPECTACULAR_SETTINGS = {  
    'SWAGGER_UI_DIST': 'SIDECAR', # shorthand to use the sidecar instead  
    'SWAGGER_UI_FAVICON_HREF': 'SIDECAR',  
    'REDOC_DIST': 'SIDECAR',  
    # OTHER SETTINGS  
}
```

(Рис. 53)

Добавляем в файл `urls.py` приложения DR URL адреса и импортируем

классы представлений:

```
path('todo/', include('todo.urls')),
# YOUR PATTERNS
path('spectacular/api/schema/', SpectacularAPIView.as_view(), name='schema'),
# Optional UI:
path('spectacular/api/schema/swagger-ui/', SpectacularSwaggerView.as_view(url_name='schema'), name='swagger-ui'),
path('spectacular/api/schema/redoc/', SpectacularRedocView.as_view(url_name='schema'), name='redoc'),
]
```

(Рис. 54)

### 4.2.2 Безопасность.

В файле `setting.py` приложения DR добавляем в словарь

`SPECTACULAR_SETTINGS` поле `SERVE_PERMISSIONS` с заданным вами классом разрешений, что позволяет ограничить доступ на ваше усмотрение.

```
SPECTACULAR_SETTINGS = {
    'SWAGGER_UI_DIST': 'SIDECAR', # shorthand to use the sidecar instead
    'SWAGGER_UI_FAVICON_HREF': 'SIDECAR',
    'REDOC_DIST': 'SIDECAR',
    'SERVE_PERMISSIONS': ['rest_framework.permissions.IsAdminUser'],
    # OTHER SETTINGS
}
```

(Рис. 55)

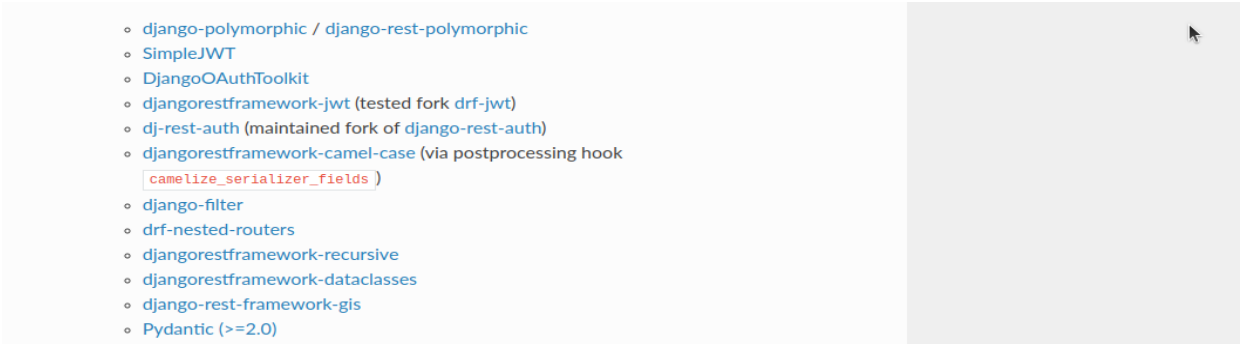
### 4.2.3 Удобство интерфейса

Функции

- Сериализаторы смоделированы как компоненты. (поддерживается произвольная вложенность и рекурсия)
- Декоратор `@extend_schema` для настройки `APIView`, `ViewSet`,

представлений на основе функций и `@action`

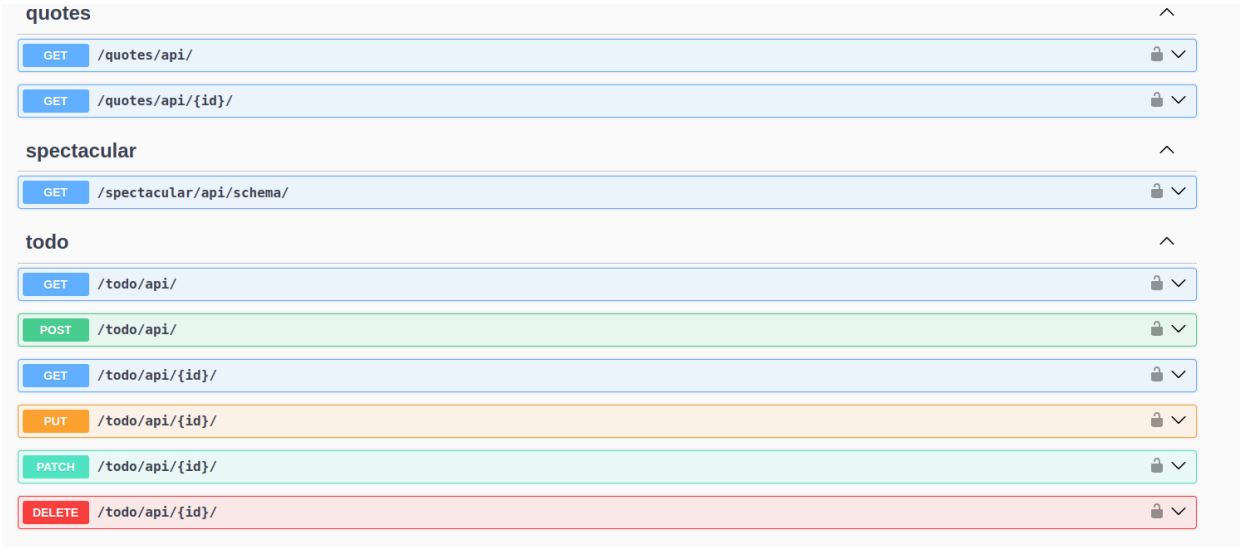
- дополнительные параметры
- Переопределение сериализатора запроса/ответа (с кодами состояния)
- полиморфные ответы вручную с помощью `PolymorphicProxySerializer` помощника или через `rest_polymorphicPolymorphicSerializer`
- ... и больше возможностей настройки
- Поддержка аутентификации (включая собственные DRF-функции, легко расширяемые)
- Поддержка класса пользовательского сериализатора (легко расширяемая)
- `SerializerMethodField()` тип через подсказку типа или `@extend_schema_field`
- поддержка `118n`
- Извлечение тегов
- Примеры запросов/ответов/параметров
- Описание извлечено из docstrings
- Расширения спецификации поставщика (x-\*) в информации, операциях, параметрах, компонентах и схемах безопасности
- Разумные запасные варианты
- Разумное `operation_id` наименование (на основе пути)
- Обслуживание схемы `SpectacularAPIView` (также доступны представления `Redoc` и `Swagger-UI`)
- Разделение компонентов опционального сериализатора ввода/вывода
- Операции обратного вызова
- Поддержка OpenAPI 3.1 (через настройку `OAS_VERSION`)
- Включена поддержка:



(Рис. 56)



(Рис. 57)



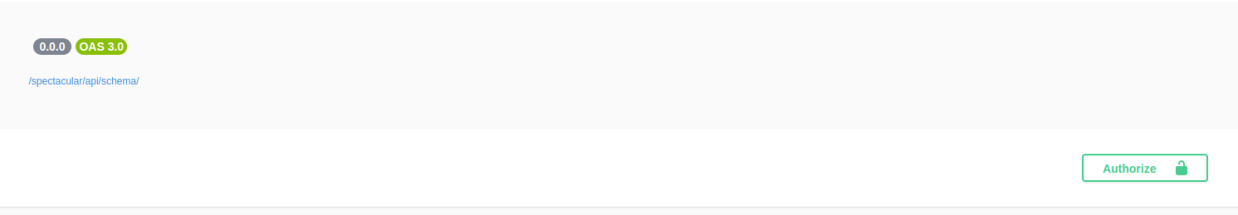
(Рис. 58)

blog		
GET	/blog/api/categories/	🔒
POST	/blog/api/categories/	🔒
GET	/blog/api/categories/{id}/	🔒
PUT	/blog/api/categories/{id}/	🔒
PATCH	/blog/api/categories/{id}/	🔒
DELETE	/blog/api/categories/{id}/	🔒
GET	/blog/api/comments/	🔒
POST	/blog/api/comments/	🔒
GET	/blog/api/comments/{id}/	🔒
PUT	/blog/api/comments/{id}/	🔒

(Рис. 59)

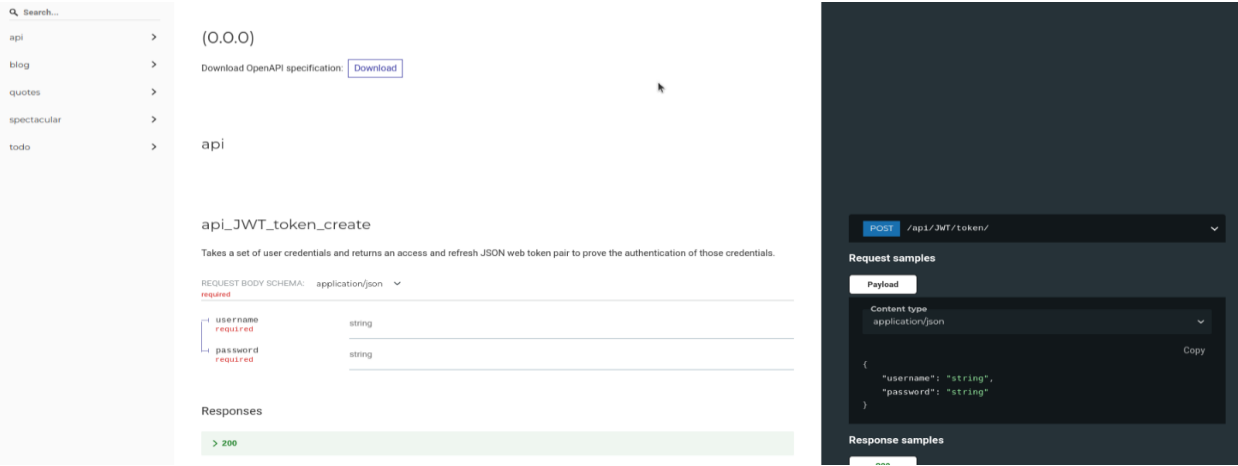
Изначально фильтрация по тегам отсутствует, локальный хост не указан.

В процессе разработки есть возможность добавления и кастомизации.



(Рис. 58)

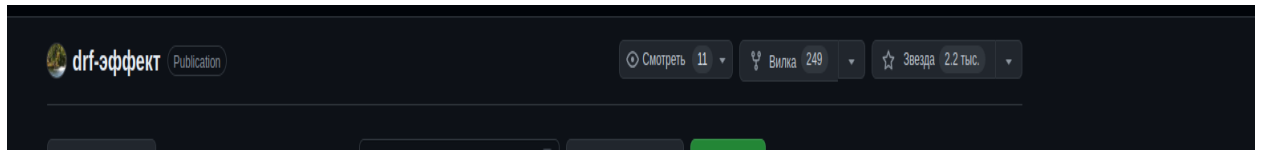
Присутствует Redoc



(Рис. 59)

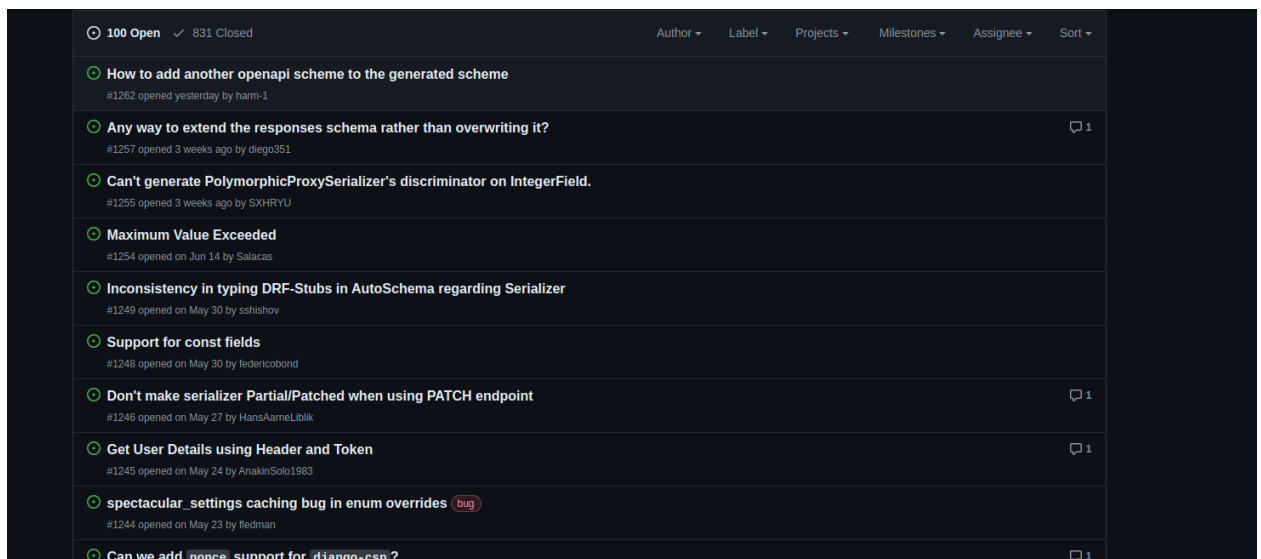
## 4.2.4 Репозиторий

Репозиторий имеет 2.2 тысячи звёзд:



(Рис. 60)

Более 100 открытых проблем и 831 закрытую:



(Рис. 61)

За последний год произошло несколько релизов:

Apr 1

tfranzel

0.27.2

b1a34b0

Compare

0.27.2 Latest

Important notes

- Some bugfixes and some functionality gaps closed.

PRs

- Update link to redoc settings by [@OttoAndrey](#) in [#1154](#)
- Fix ordered dict by [@tfranzel](#) in [#1160](#)
- Add auth extension for django-rest-knox by [@callumgare](#) in [#1163](#)
- remove official 3.6 support due to upstream breakage. by [@tfranzel](#) in [#1170](#)
- Bump django from 4.2.7 to 4.2.10 in /requirements by [@dependabot](#) in [#1169](#)
- Add tags support to OpenApiWebhook by [@federicobond](#) in [#1146](#)
- Document `extend_schema_view` support for `@action` by [@johnthagen](#) in [#1178](#)
- Add support for direct usage of higher order hints [#1174](#) by [@tfranzel](#) in [#1181](#)
- fix custom `http_method_names` for actions [#1184](#) by [@tfranzel](#) in [#1185](#)
- Update SWAGGER\_UI\_DIST settings.rst by [@Azelpur](#) in [#1187](#)
- Add a specific Action Wrapper for OAuth Authorization requests [#1190](#) by [@BramEsposito](#) in [#1191](#)
- DRF 3.15 by [@tfranzel](#) in [#1197](#)
- Fix builtin Swagger-UI OAuth login by [@ftsell](#) in [#1142](#)
- Add support for TypeAliasType by [@igorgaming](#) in [#1214](#)
- higher order hints for `@extend_schema_field` (case 2) [#1174](#) [#1212](#) by [@tfranzel](#) in [#1216](#)
- fix non-translated enum override hash [#1198](#) by [@tfranzel](#) in [#1217](#)

New Contributors

- [@OttoAndrey](#) made their first contribution in [#1154](#)
- [@callumgare](#) made their first contribution in [#1163](#)
- [@dependabot](#) made their first contribution in [#1169](#)
- [@Azelpur](#) made their first contribution in [#1187](#)
- [@BramEsposito](#) made their first contribution in [#1191](#)
- [@igorgaming](#) made their first contribution in [#1214](#)

(Рис. 62)

Jan 18

tfranzel

0.27.1

db414d6

Compare

0.27.1

Important notes

- Fixed some OAS 3.1 corner cases and added support for `Webhooks` (3.1) in addition to existing `Callbacks`

PRs

- coerse Decimal to float format explicitly [#1134](#) by [@tfranzel](#) in [#1136](#)
- Provide all fixed field names for the components object in ResolvedComponent.\* constants by [@sergei-maertens](#) in [#1138](#)
- Optional setting for enum suffixes by [@jrbeaumont](#) in [#1140](#)
- Fix handling of metadata when using OAS 3.1 by [@Viicos](#) in [#1139](#)
- Add OpenAPI 3.1 webhook support by [@federicobond](#) in [#1135](#)

New Contributors

- [@jrbeaumont](#) made their first contribution in [#1140](#)
- [@Viicos](#) made their first contribution in [#1139](#)
- [@federicobond](#) made their first contribution in [#1135](#)

Full Changelog: [0.27.0...0.27.1](#)

Contributors

federicobond

sergei-maertens

and 3 other contributors

Assets

2

3 people reacted

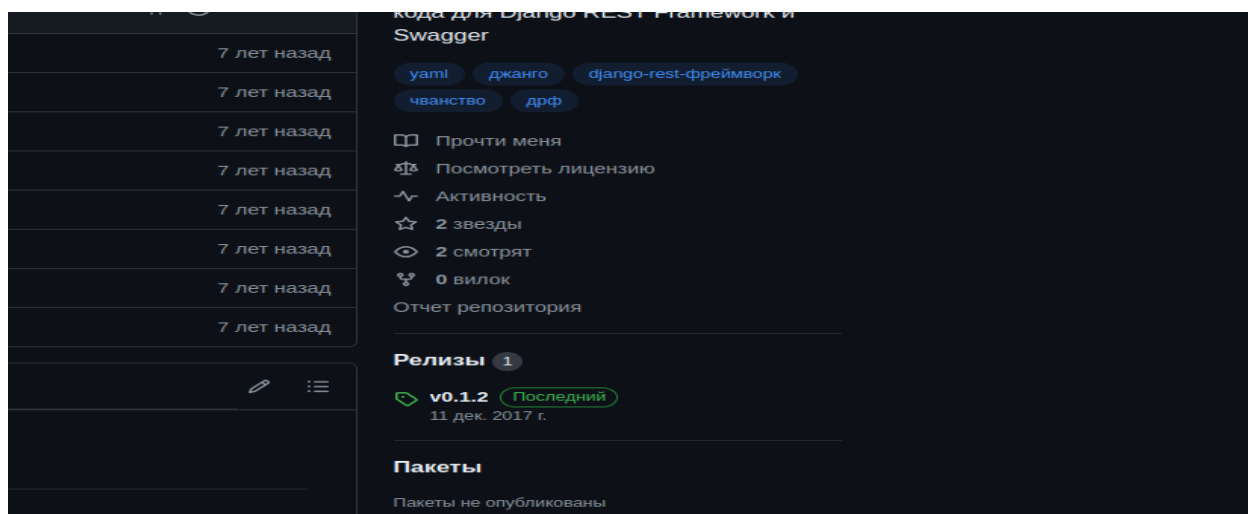
(Рис. 63)

## 4.3 Библиотека drf-swagger

40



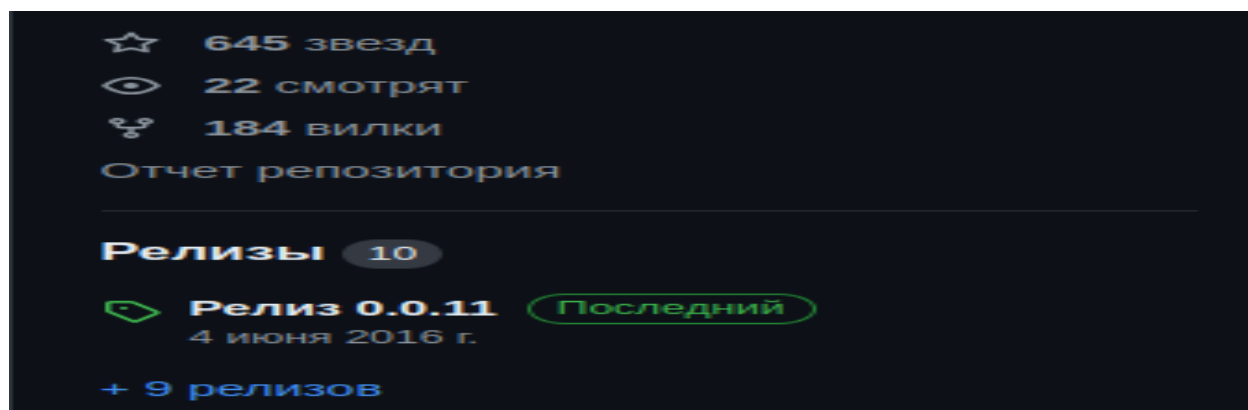
На данный момент библиотека является заброшенной, последний релиз был 11 декабря 2017 года:



(Рис. 64)

## 4.4 Библиотека drf-docs

Библиотека drf-docs больше не поддерживается, последний релиз был 4 июня 2016 года



(Рис. 65)

## 5. Заключение.

С учетом последних обновлений и требований к документации API, я пришел к выводу, что библиотека drf-yasg устарела и поддерживает версии Django Rest Framework только до 3.14, а также OpenAPI 2.0. Кроме того, библиотеки drf-swagger и drf-docs также устарели и больше не поддерживаются.

На мой взгляд, лучшим вариантом в настоящее время является библиотека drf-spectacular. Она поддерживает Django Rest Framework версии 3.15 и OpenAPI 3.0, что соответствует современным требованиям. Кроме того, она предлагает более простой способ подключения и обеспечения защиты API

## **6. Список литературы.**

- <https://drf-spectacular.readthedocs.io/en/latest/readme.html#>
- <https://drf-yasg.readthedocs.io/en/stable/readme.html>
- <https://django-rest-swagger.readthedocs.io/en/latest/>
- <https://www.drfdocs.com/>
- <https://github.com/tfranzel/drf-spectacular>
- [https://github.com/koyouhun/drf\\_swagger](https://github.com/koyouhun/drf_swagger)
- <https://github.com/MarkTools/DRF-Docs>
- <https://github.com/axnsan12/drf-yasg>
- <https://ilyachch.gitbook.io/django-rest-framework-russian-documentation>
- <https://djangodoc.ru/3.2/>
- <https://postgres-py.readthedocs.io/en/latest/>
- [https://github.com/koyouhun/drf\\_swagger](https://github.com/koyouhun/drf_swagger)
- <https://github.com/manosim/django-rest-framework-docs>

## **7. Ссылка на GitHub с исследованием.**

- <https://github.com/NikitaMarnykh>