MUHAMMAD AL-XORAZMIY
NOMIDAGI TATU FARG'ONA FILIALI

FERGANA BRANCH OF TUIT
NAMED AFTER MUHAMMAD AL-KHORAZMI

"AL-FARG'ONIY AVLODLARI"

ELEKTRON ILMIY JURNALI | ELECTRONIC SCIENTIFIC JOURNAL

TA'LIMDAGI ILMIY, OMMABOP VA ILMIY TADQIQOT ISHLARI

2-SON 1(2)
2023-YIL

TATU, FARG'ONA
O'ZBEKISTON

ISSN 2181-4252

O’ZBEKISTON RESPUBLIKASI RAQAMLI TEXNOLOGIYALAR VAZIRLIGI

MUHAMMAD AL-XORAZMIY NOMIDAGI TOSHKENT AXBOROT TEXNOLOGIYALARI UNIVERSITETI FARG’ONA FILIALI

Muassis: Muhammad al-Xorazmiy nomidagi Toshkent axborot texnologiyalari universiteti Farg‘ona filiali.
Chop etish tili: O‘zbek, ingliz, rus.
Jurnal texnika fanlariga ixtisoslashgan bo‘lib, barcha shu sohadagi matematika, fizika, axborot texnologiyalari yo‘nalishida maqolalar chop etib boradi.

Учредитель: Ферганский филиал Ташкентского университета информационных технологий имени Мухаммада ал-Хоразми.
Язык издания: узбекский, английский, русский.
Журнал специализируется на технических науках и публикует статьи в области математики, физики и информационных технологий.

Founder: Fergana branch of the Tashkent University of Information Technologies named after Muhammad al-Khorazmi.
Language of publication: Uzbek, English, Russian.
The magazine specializes in technical sciences and publishes articles in the field of mathematics, physics, and information technology.

2023 yil, Tom 1, №2
Vol.1, Iss.2, 2023 y

ELEKTRON ILMIY JURNALI **ELECTRONIC SCIENTIFIC JOURNAL**

«Al-Farg‘oniy avlodlari» («The descendants of al-Fargani», «Potomki al-Fergani») O’zbekiston Respublikasi Prezidenti administratsiyasi huzuridagi Axborot va ommaviy kommunikatsiyalar agentligida 2022-yil 21 dekabrda 054493-son bilan ro’yxatdan o’tgan.

Tahririyat manzili:
151100, Farg‘ona sh., Aeroport ko‘chasi 17-uy, 201A-xona
Tel: (+99899) 998-01-42
e-mail: info@al-fargoniy.uz
Qo‘lyozmalar taqrizlanmaydi va qaytarilmaydi.

FARG’ONA - 2023 YIL

TAHRIR HAY'ATI

Jurnal quyidagi bazalarda indekslanadi:

MUNDARIJA | ОГЛАВЛЕНИЕ | TABLE OF CONTENTS

ГЕНЕРАЦИЯ АВТОМАТИЧЕСКОЙ ДОКУМЕНТАЦИИ API В DJANGO REST FRAMEWORK С ПРИМЕНЕНИЕМ DRF SPECTACULAR

Солиев Бахромжон Набижонович,
Ассистент Ферганского филиала
Ташкентского университета информационных технологий
имени Мухаммада ал-Хоразми

Аннотация: Целью этой статьи является демонстрация возможностей drf-spectacular для документирования API и основного набора техник, которые покроют большую часть сценариев использования. Мы настроим генерацию документации и рассмотрим все основные способы гибкой настройки отображения.

Ключевые слова: API, Django, python, drf-spectacular, программирование, автоматической генерации документации, веб-API, Swagger, ReDoc, Django REST Framework, pip, extend_schema, doc-string, эндпоинт.

Введение. В этом введении будет рассказано о библиотеке drf-spectacular, которая предоставляет простой и элегантный способ автоматической генерации документации для Django-приложений, использующих Django REST Framework (DRF).

Drf-spectacular является инструментом, позволяющим создавать документацию для веб-API на основе DRF в автоматическом режиме. С его помощью разработчики могут генерировать понятные и подробные описания доступных эндпоинтов, параметров запросов и форматов данных, минимизируя усилия, которые необходимы для создания документации вручную.[1]

Благодаря drf-spectacular, возможно экспортировать документацию в различные форматы, такие как Swagger и ReDoc, что делает ее легко доступной для клиентов и разработчиков. Библиотека также поддерживает автоматическое распознавание URL-шаблонов, встроенную поддержку OpenAPI-спецификации и предоставляет дополнительные возможности конфигурации.

При использовании drf-spectacular разработчики Django-приложений могут упростить процесс создания и обслуживания документации для своего API. Она предоставляет мощные инструменты, которые помогут создать чистую, аккуратную и информативную документацию, сэкономив время и усилия разработчиков.

Теперь, когда мы ознакомились с общим представлением о drf-spectacular, давайте рассмотрим более подробно его функциональные возможности и то, как ими пользоваться.

drf-spectacular - единственный проект, который сейчас активно поддерживается. django-rest-swagger и drf-yasg устарели и не обновлялись уже пару-тройку лет.

Литературный обзор: В настоящей статье проведен обзор литературы, связанной с использованием библиотеки drf-spectacular в Django для автоматической генерации документации для веб-API. Были проанализированы научные публикации, официальная документация и сообщества разработчиков, чтобы изучить функциональные возможности и преимущества drf-spectacular.[2]

Библиотека drf-spectacular была представлена в качестве эффективного инструмента для создания документации в Django REST Framework (DRF) проектах. Она позволяет разработчикам автоматически генерировать понятную и полезную документацию на основе DRF представлений, сериализаторов и схем запросов. Также было выяснено, что drf-spectacular поддерживает экспорт документации в различные форматы, такие как Swagger и ReDoc, обеспечивая удобный доступ для клиентов и разработчиков.

Методология: Для достижения целей исследования, проведенного в данной статье, был использован следующий методологический подход. Во-первых, был проанализирован и изучен исходный код и документация drf-spectacular, чтобы получить полное представление о

https://doi.org/10.5281/zenodo.7979432

https://journals.indexcopernicus.com/search/article?articleId=3609138

функциональных возможностях и конфигурационных настройках библиотеки.

Во-вторых, были разработаны и выполнены эксперименты для оценки производительности и эффективности drf-spectacular в различных сценариях использования. Эксперименты включали создание простого Django-приложения с использованием DRF и drf-spectacular, а затем оценку качества сгенерированной документации, скорости генерации и общей надежности библиотеки.

В-третьих, были проведены сравнительные анализы с другими инструментами и библиотеками для генерации документации в Django, чтобы определить преимущества и недостатки drf-spectacular по сравнению с альтернативными решениями.

Наконец, на основе полученных результатов исследования были сделаны выводы о применимости и эффективности drf-spectacular в контексте разработки веб-API на основе Django и DRF.

Таким образом, предложенная методология исследования позволила достичь целей статьи и изучить возможности drf-spectacular, а также оценить его эффективность и преимущества в сравнении с другими инструментами для генерации документации в Django.

Как установить и добавить DRF Spectacular в проект? Сначала устанавливаем сам пакет через pip install drf-spectacular

После чего добавляем в INSTALLED_APPS:

```
INSTALLED_APPS = [
    # ВСЕ ВАШИ ПРИЛОЖЕНИЯ
    'drf_spectacular',
]
```

Далее необходимо добавить DEFAULT_SCHEMA_CLASS в настройки REST_FRAMEWORK в settings.py. Этот класс будет отвечать за генерацию самой схемы. В некоторых ситуациях мы можем прописать здесь свой класс для добавления дополнительной функциональности.[3]

```
REST_FRAMEWORK = {
    # ВАШИ НАСТРОЙКИ
    'DEFAULT_SCHEMA_CLASS':
'drf_spectacular.openapi.AutoSchema',
}
```

После чего добавляем в наш главный файл urls.py следующее:

```
urlpatterns = [
    # ВАШИ URL-АДРЕСА
    path('api/schema/',
SpectacularAPIView.as_view(),
name='schema'),
    path('api/docs/',
SpectacularSwaggerView.as_view(url_name='s
chema'), name='docs'),
]
```

По адресу api/schema/ будет генерироваться JSON-схема нашего API, из которой по адресу api/docs/ будет строиться Swagger-UI документация.[4]

После добавления этих настроек по адресу http://127.0.0.1:8000/api/docs/ можно будет увидеть список всех наших эндпоинтов вот в таком формате:

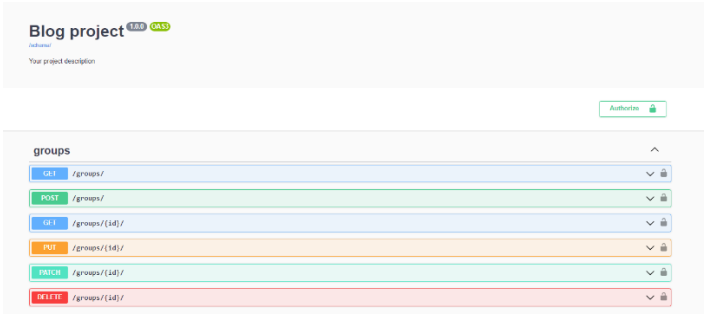

Рис 1. Общий вид сгенерированной Swagger-UI документации.

Базовая настройка отображения. Теперь поговорим о более гибкой настройке. Возьмем для примера /groups/

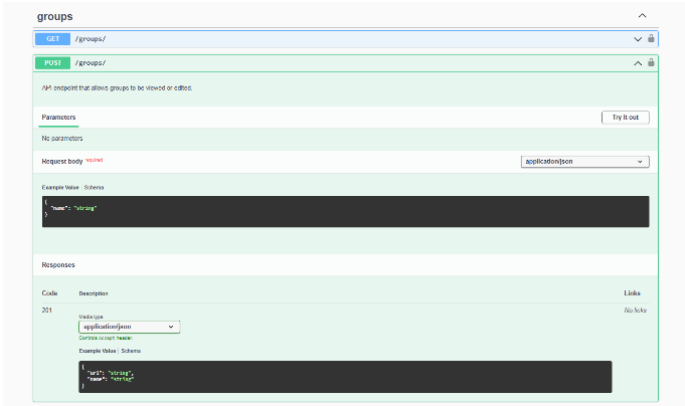

Рис 2. Базовая настройка отображения groups

https://doi.org/10.5281/zenodo.7979432

https://journals.indexcopernicus.com/search/article?articleId=3609138

Для начала более детально разберём интерфейс. Здесь есть четыре основных блока:

- описание, в котором может быть описано назначение эндпоинта и даны какие-либо подсказки
- принимаемые query-параметры
- пример запроса
- пример ответа для каждого статус-кода (200, 201, 400, 404 и так далее)[5]

Теперь попробуем изменить описание. Для этого есть три способа.

1. Использовать декоратор @extend_schema_view и явно указать описание для каждого метода.
2. Использовать @extend_schema для каждого метода отдельно
3. Указать описание в doc-string для каждого метода[6]

```
# первый пример
@extend_schema_view(
    list=extend_schema(
            summary="Получить список постов",
        ),
    update=extend_schema(
        summary="Изменение существующего поста",
    ),
    partial_update=extend_schema(
        description="""Lorem ipsum dolor sit amet, consectetur adipiscing elit.
        Cras euismod vehicula hendrerit. Integer placerat lobortis purus, molestie
        finibus nibh gravida sed. Etiam pretium gravida velit, et rutrum nisi posuere at.
        Proin hendrerit eros et enim placerat, in commodo lorem tristique. Donec hendrerit ultrices
        nulla, a maximus diam consectetur id. Cras suscipit ligula vitae sem vulputate vulputate.
        Duis enim turpis, mollis at maximus nec, tempus ac erat. Nullam at eleifend est, non lacinia erat.
        Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae;
        Nulla congue condimentum arcu, id sodales dolor aliquet nec. Nullam a consequat neque.""",
    ),
    create=extend_schema(
            summary="Создание нового поста",
        ),
)
class PostViewSet(viewsets.ModelViewSet):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
    # второй пример, так же отобразится в описании
    @extend_schema(summary="Test description of retrieve method")
    def retrieve(self, request, *args, **kwargs):

        # some logic

        return Response(status=status.HTTP_200_OK)

    def destroy(self, request, *args, **kwargs):
        # третий пример через doc-string

        """Test description of destroy method"""

        return Response(status=status.HTTP_200_OK)
```

Если описание короткое - используем summary, если длинное - используем description. Либо оба, если нужно и короткое и длинное описание. У summary есть большой плюс - он отображается напротив эндпоинта и помогает быстро сориентироваться.[7] Пример:

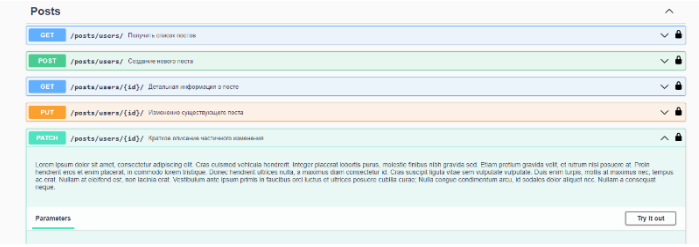

Рис 3. Отображение описаний в Swagger UI

https://doi.org/10.5281/zenodo.7979432

https://journals.indexcopernicus.com/search/article?articleId=3609138

https://al-fargoniy.uz/

Теперь можно добавить тег, для группировки эндпоинтов. Если у каждой группы эндпоинтов проставлен тег, это сильно упростит навигацию и поиск нужной информации.[8]

Теги добавляются с помощью @extend_schema над @extend_schema_view. Таким образом все эндпоинты, помеченные одним тегом будут сгруппированы в один блок.

```
@extend_schema(tags=["Posts"])
@extend_schema_view(
    retrieve=extend_schema(
        summary="Детальная информация о посте",
        ....
```

Теперь перейдем к описанию запросов и ответов. В аргумент responses нужно передать словарь вида {"код ответа": "сериализатор или объект OpenApiResponse"}

```
class DummyDetailSerializer(serializers.Serializer):
    status = serializers.IntegerField()


class DummyDetailAndStatusSerializer(serializers.Serializer):
    status = serializers.IntegerField()
    details = serializers.CharField()

@extend_schema(tags=["Posts"])
@extend_schema_view(
    retrieve=extend_schema(
        summary="Детальная информация о посте",
        responses={
            status.HTTP_200_OK: PostSerializer,
            status.HTTP_400_BAD_REQUEST: DummyDetailSerializer,
            status.HTTP_401_UNAUTHORIZED: DummyDetailSerializer,
            status.HTTP_403_FORBIDDEN: DummyDetailAndStatusSerializer,

status.HTTP_500_INTERNAL_SERVER_ERROR: OpenApiResponse(
                response=None,
                description='Описание 500 ответа'),
        },
    ),
```

Для описания стандартных ответов я использую заранее созданный DummyDetailSerializer/DummyDetailAndStatusSerializer.

Так будут выглядеть наши ответы:

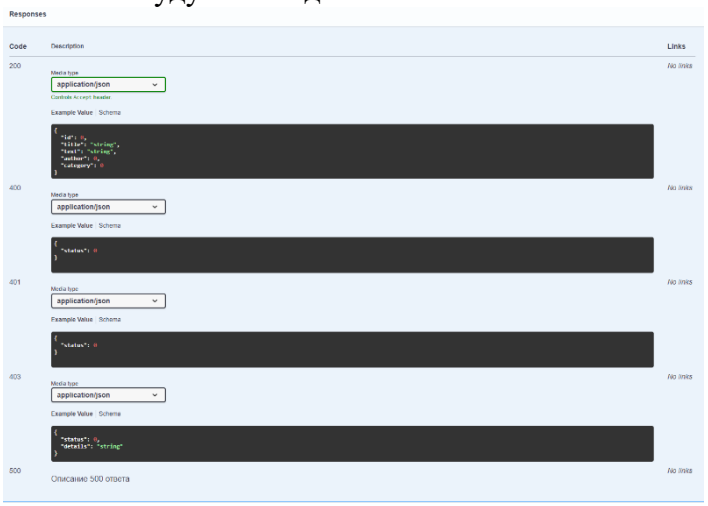

Рис 4. Пример отображения ответов от сервера в Swagger UI

Теперь перейдем к более детальной настройке входных параметров на примере update. С помощью аргумента parameters можно добавить новые параметры, либо дополнить существующие.[9]

```
update=extend_schema(
        summary="Изменение существующего поста",
        request=PostSerializer,
        responses={
            status.HTTP_200_OK: PostSerializer,
            status.HTTP_400_BAD_REQUEST: DummyDetailSerializer,
            status.HTTP_401_UNAUTHORIZED: DummyDetailSerializer,
            status.HTTP_403_FORBIDDEN: DummyDetailAndStatusSerializer,
        },
        parameters=[
            OpenApiParameter(
                name='some_new_parameter',
```


https://doi.org/10.5281/zenodo.7979432

https://journals.indexcopernicus.com/search/article?articleId=3609138

```
                location=OpenApiParameter.QUERY,
                description='some new
parameter for update post',
                required=False,
                type=int
            ),
        ]
    ),
```

Так это отобразится в интерфейсе:

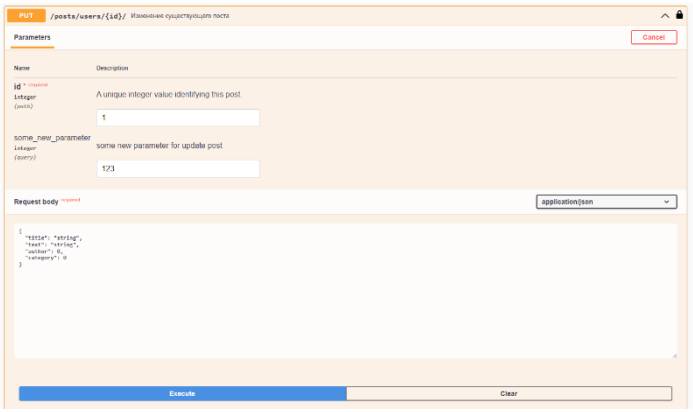

Рис 5. Пример отображения дополнительных параметров

Ключевой аргумент при создании дополнительного параметра - location. В нашем случае используем OpenApiParameter.QUERY. Это значит, что этот параметр будет передаваться как query параметр в формате ?some_new_parameter=123.[10]

Так же доступны:

- OpenApiParameter.PATH - доп. параметр в url
- OpenApiParameter.HEADER - доп. параметр в headers
- OpenApiParameter.COOKIE - доп. параметр в cookies

Теперь рассмотрим добавление примеров запроса. В аргумент examples мы должны передать список объектов OpenApiExample. Например:

```
create=extend_schema(
    summary="Создание нового поста",
    request=PostSerializer,
    responses={
        status.HTTP_200_OK:
PostSerializer,
        status.HTTP_400_BAD_REQUEST:
DummyDetailSerializer,
        status.HTTP_401_UNAUTHORIZED:
DummyDetailSerializer,
        status.HTTP_403_FORBIDDEN:
DummyDetailAndStatusSerializer,
    },
    examples=[
        OpenApiExample(
            "Post example",
            description="Test example for
the post",
            value=
            {
                "title": "testuser",
                "text": "Some text for
post",
                "category": 1,
                "author": 1,
            },

status_codes=[str(status.HTTP_200_OK)],
        ),
    ],
),
```

Наш пример отобразится в интерфейсе следующим образом:

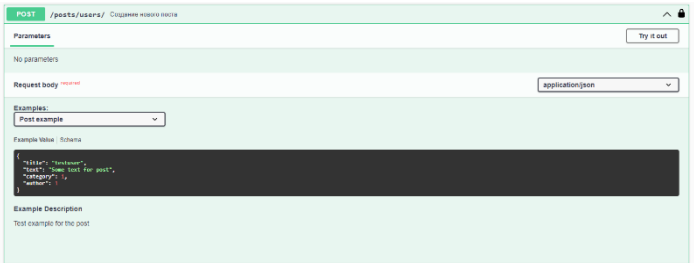

Рис 6. Отображение примеров в Swagger UI.

Общие настройки. Теперь осталось добавить общие настройки отображения. Это делается в файле settings.py следующим образом:

```
SPECTACULAR_SETTINGS = {
    "TITLE": "Post API", # название
проекта
    "VERSION": "0.0.1", # версия проекта
    "SERVE_INCLUDE_SCHEMA": False, #
исключить эндпоинт /schema
    "SWAGGER_UI_SETTINGS": {
        "filter": True, # включить поиск
по тегам
```


https://doi.org/10.5281/zenodo.7979432

https://journals.indexcopernicus.com/search/article?articleId=3609138

https://al-fargoniy.uz/

```
    },
    "COMPONENT_SPLIT_REQUEST": True
}
```

Заключение. В заключение могу уверенно сказать, что drf-spectacular - это гибкий и полезный инструмент, Хорошая документация API помогает на всех этапах разработки проекта, упрощает тестирование и поддержку кода.

Список использованной литературы

1. Солиев, Б. Н. Перспективы развития электронной торговли и онлайн-курсов в Узбекистане на основе системы LMS / Б. Н. Солиев // Исследования молодых ученых : Материалы XV Международной научной конференции, Казань, 20–23 декабря 2020 года / Под редакцией И.Г. Ахметова [и др.]. – Казань: Общество с ограниченной ответственностью "Издательство Молодой ученый", 2020. – С. 1-3. – EDN NFKTEB.

2. Солиев Б. Н. и др. ТЕНДЕНЦИИ РАЗВИТИЕ ИНТЕРНЕТ-ЛОГИСТИКИ В ЭЛЕКТРОННОЙ КОММЕРЦИИ //Журнал Технических исследований. – 2022. – Т. 5. – №. 1.

3. Автоматическая документация API Django Rest Framework с помощью DRF Spectacular, https://habr.com/ru/articles/733942/

4. Soliev B. N., kizi Abdurasulova D. B., Yakubov M. S. USING GINJA TEMPLATES TO CREATE E-COMMERCE PLATFORMS //Publishing House “Baltija Publishing”. – 2023.

5. Nabijonovich S. B., Mahamatovich R. A. Prospects for the Development of Electronic Trade Processes Based on Local Characteristics //International Journal on Orange Technologies. – 2021. – Т. 3. – №. 3. – С. 305-309.

6. Soliev B. N., Abdurasulova D., Yakubov M. S. USING THE DJANGO FRAMEWORK FOR E-COMMERCE PROCESSES //Journal of Integrated Education and Research. – 2022. – Т. 1. – №. 6. – С. 229-233.

7. Солиев Б. Н. и др. ИЗУЧИТЬ ОПЫТ ДРУГИХ СТРАН ПО РАЗВИТИЮ ЭЛЕКТРОННОЙ КОММЕРЦИИ В УЗБЕКИСТАНЕ //Журнал Технических исследований. – 2022. – Т. 5. – №. 1.

8. Musayev, X., & Soliev, B. (2023). PUBLIC, PROTECTED, PRIVATE MEMBERS IN PYTHON. Потомки Аль-Фаргани, 1(1), 43–46. извлечено от https://al-fargoniy.uz/index.php/journal/article/view/17

9. Zulunov, R., & Soliev, B. (2023). IMPORTANCE OF PYTHON LANGUAGE IN DEVELOPMENT OF ARTIFICIAL INTELLIGENCE. Потомки Аль-Фаргани, 1(1), 7–12. извлечено от https://al-fargoniy.uz/index.php/journal/article/view/3

10. Солиев Б. Н. Проблемы моделирования электронных торговых процессов на основе местных характеристик //Исследования молодых ученых. – 2020. – С. 8-11.

https://doi.org/10.5281/zenodo.7979432

https://journals.indexcopernicus.com/search/article?articleId=3609138