Создание Django API используя Django Rest Framework часть 3

Это третья часть из серии статей про Django API. Первая часть находится [здесь](https://webdevblog.ru/sozdanie-django-api-ispolzuya-django-rest-framework-apiview/). Вторая [здесь](https://webdevblog.ru/sozdanie-django-api-ispolzuya-djangorestframework-chast-2/). В этой статье мы рассмотрим использование наборы представлений класс [viewsets](https://www.django-rest-framework.org/api-guide/viewsets/" \t "_blank) от Django REST Framework**.** Основное отличие **viewsets** от **GenericAPIView** и **APIView**, в том что это еще один дополнительный абстрагирующий класс, который с одной стороны позволяет уменьшить базовый код а с другой имеет большие возможности по кастомизации и расширению.

И так начнем, с изменений в файле представления. Используем базовый класс **viewsets.ViewSet** унаследуем от него наш класс ArticleView и определим методы получения всех статей (list) и одну выбранную статью (retrieve). Для этого внесем следующие изменения в **article/views.py**:

from django.shortcuts import get\_object\_or\_404

from rest\_framework import viewsets

from rest\_framework.response import Response

from .models import Article

from .serializers import ArticleSerializer

**class** ArticleView(viewsets.ViewSet):

"""

A simple ViewSet that for listing or retrieving users.

"""

**def** list(self, request):

queryset = Article.objects.all()

serializer = ArticleSerializer(queryset, many=**True**)

**return** Response(serializer.data)

**def** retrieve(self, request, pk=None):

queryset = Article.objects.all()

user = get\_object\_or\_404(queryset, pk=pk)

serializer = ArticleSerializer(user)

**return** Response(serializer.data)

Далее нам нужно связать наши методы list и retrieve с URL API. Для этого внесите изменения в **article/urls.py**:

from django.urls import path

from .views import ArticleView

app\_name = "articles"

# app\_name will help us do a reverse look-up latter.

urlpatterns = [

path('articles/', ArticleView.as\_view({'get': 'list'})),

path('articles/<int:pk>', ArticleView.as\_view({'get': 'retrieve'})),

]

Если теперь обратится по URL: http://127.0.0.1:8000/api/articles/ или http://127.0.0.1:8000/api/articles/1/ то должно все работать как раньше.

Теперь воспользуется новым для нас классом роутера DRF **DefaultRouter**. Этот класс позволяет значительно уменьшить код в URL. Зарегистрируем наш класс **ArticleView** в этом классе и позволяем ему теперь автоматически генерировать обработчики url. Удалите весь старый код из **article/urls.py** и внесите следующие изменения в этот файл:

**from** rest\_framework.routers **import** DefaultRouter

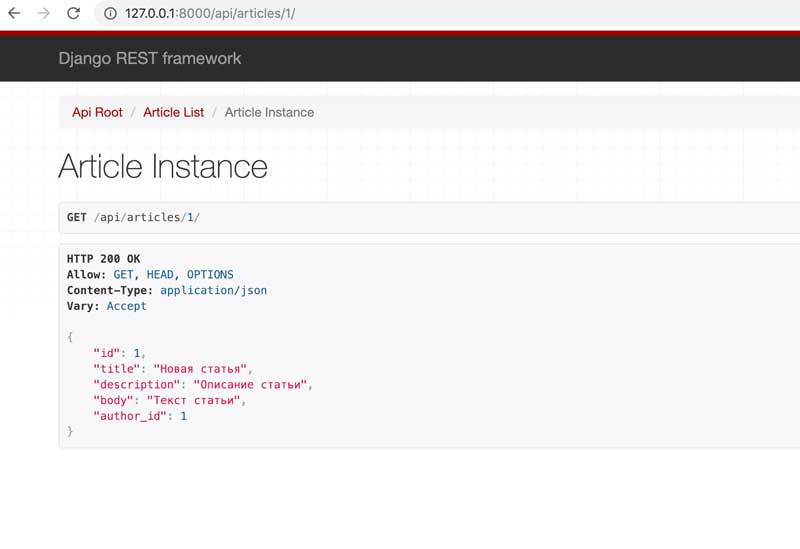
**from** *.views* **import** ArticleView

router = DefaultRouter()

router.register(r'articles', ArticleView, basename='user')

urlpatterns = router.urls

Теперь если обратится по URL: http://127.0.0.1:8000/api/articles/ или http://127.0.0.1:8000/api/articles/1/ то опять все должно работать как раньше. Но обратите внимание мы реализовали только два метода API получения списка статей и получение одной выбранной статье.



Что бы реализовать методы создания, редактирования и удаления мы можем определить эти методы вручную или воспользоваться классом viewsets.ModelViewSet в котором уже все сделано за нас. Внесите следующие изменения в файл **article/views.py**:

**from** rest\_framework **import** viewsets

**from** *.models* **import** Article

**from** *.serializers* **import** ArticleSerializer

**class** ArticleViewSet(viewsets.ModelViewSet):

serializer\_class = ArticleSerializer

queryset = Article.objects.all()

И соотвествующие изменения в файл **article/urls.py**

from rest\_framework.routers import DefaultRouter

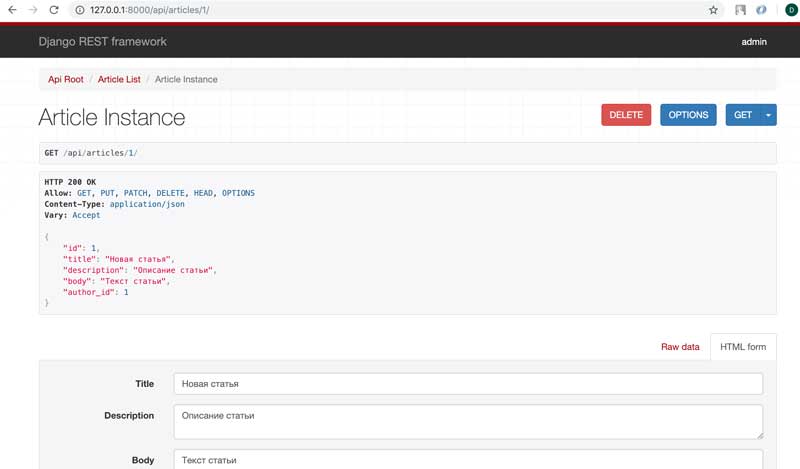
from .views import ArticleViewSet

router = DefaultRouter()

router.register(r'articles', ArticleViewSet, basename='user')

urlpatterns = router.urls

Теперь мы можем создавать, редактировать и удалять



Используя **viewsets** мы буквально в пару строк получили полностью рабочий CRUD для выбранной таблицы.  
На этом я пожалуй хотел завершить эту статью. В этой серии статей мы очень кратко рассмотрели и сравнили возможности классов DRF **APIView**, **GenericAPIView**, **ViewSets** для создания CRUD. На самом деле у этих классов гораздо больше возможностей и особенностей их применения и если вы захотите изучить их глубже обратитесь к официальной [документации DRF](https://www.django-rest-framework.org/).