

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»
(БГТУ им. В.Г.Шухова)**

Лабораторная работа по теме
«REST API»
дисциплина:
Технологии web-программирования

Выполнил: ст. группы ВТ-41
Фаракшин Н. Р.
Проверил: ст. пр. Картамышев С.В.

Белгород 2020

Лабораторная работа № 5

REST API

Цель работы: изучить основы разработки API для web-приложений. Разработать REST API для своего проекта.

Задание к лабораторной работе:

1. Изучить структуру формата представления данных JSON.
2. Изучить типы запросов к API: HEAD, GET, POST, PUT, DELETE.
3. Спроектировать и реализовать собственное REST API (Получение, создание, изменение и удаление каких-либо объектов).
4. В отчёт необходимо предоставить документацию к использованию методов. (Либо словесным описанием, либо через Swagger)

Выполнение работы

Для реализации REST API была использована библиотека Django REST Framework. В качестве вспомогательных библиотек использовались библиотеки Django-filters для обеспечения фильтрации запросов согласно параметрам и drf-simple-jwt для предоставления возможности аутентификации с помощью JWT токенов.

В ходе выполнения описаны классы сериализаторов моделей, использующие в качестве основы стандартный класс ModelSerializer, а также отображения (views), основанные на стандартных классах ModelViewSet. Поля моделей, используемые в API, ограничены с помощью полей fields и excluded данных классов.

Документация к API сгенерирована с помощью библиотеки drf-yasg.

Результатом работы является описание API в формате Swagger.

Получившееся описание представлено с помощью веб-клиента Swagger UI.

Описание API

 **Swagger**
framework

http://localhost:8000/api/swagger/?format=openapi

Explore

Blog API v1
[Base URL: localhost:8000/api]
http://localhost:8000/api/swagger/?format=openapi

Blog REST API description
[Contact the developer](#)
[BSD License](#)

Schemes
HTTP

Django Login

Authorize

Filter by tag

comments 

GET

/comments/


comments_list



POST

/comments/

comments_create



GET

/comments/{id}/

comments_read



PUT

/comments/{id}/

comments_update



PATCH

/comments/{id}/

comments_partial_update



DELETE

/comments/{id}/

comments_delete



contents 

GET

/contents/

contents_list



POST

/contents/

contents_create



GET

/contents/{id}/

contents_read



PUT

/contents/{id}/

contents_update



PATCH

/contents/{id}/

contents_partial_update



DELETE

/contents/{id}/

contents_delete



my_posts 

GET

/my_posts/

my_posts_list



GET

/my_posts/{id}/

my_posts_read



posts 

GET

/posts/

posts_list



POST

/posts/

posts_create



GET

/posts/{id}/

posts_read



PUT

/posts/{id}/

posts_update



PATCH

/posts/{id}/

posts_partial_update



DELETE

/posts/{id}/

posts_delete



token 

POST

/token/

token_create



POST

/token/refresh/

token_refresh_create



users 

GET

/users/

users_list



POST

/users/

users_create



GET

/users/{id}/

users_read



PUT

/users/{id}/

users_update



PATCH

/users/{id}/

users_partial_update



DELETE

/users/{id}/

users_delete



Models 

Comment >

Content >

Post >

TokenObtainPair >

TokenRefresh >

BlogUser >

Описание моделей, используемых в API

Models	
Comment ▾ {	
id	integer title: ID readOnly: true
creation_time*	string(\$date-time) title: Creation time
content*	string title: Content maxLength: 1000 minLength: 1
user*	string title: User
post*	integer title: Post
}	
Content ▾ {	
id	integer title: ID readOnly: true
description*	string title: Description maxLength: 200 minLength: 1
image_data	string(\$url) title: Image data readOnly: true
text_data	x-mutable: true string title: Text data maxLength: 1000 x-mutable: true
code_data	string title: Code data maxLength: 1000 x-mutable: true
}	
Post ▾ {	
id	integer title: ID readOnly: true
creation_time*	string(\$date-time) title: Creation time
title*	string title: Title maxLength: 100 minLength: 1
text*	string title: Text maxLength: 1000 minLength: 1
header_image	string(\$url) title: Header image readOnly: true
user*	string title: User
}	
TokenObtainPair ▾ {	
username*	string title: Username minLength: 1
password*	string title: Password minLength: 1
}	
TokenRefresh ▾ {	
refresh*	string title: Refresh minLength: 1
}	
BlogUser ▾ {	
id	integer title: ID readOnly: true
username*	string title: Username pattern: ^[w.@+]+\$ maxLength: 150 minLength: 1 Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.
email	string(\$email) title: Email address maxLength: 254
profile_pic	string(\$url) title: Profile pic readOnly: true x-mutable: true
}	

Приложение

Исходный код файлов serializers.py и views.py, реализующих сериализаторы и отображения, соответственно. Исходный код файла urls.py, управляющего структурой URL-адресов в проекте.

serializers.py

```
from rest_framework import serializers

from logic.models import Post, BlogUser, Comment, PostPiece

class PostSerializer(serializers.ModelSerializer):
    creation_time = serializers.DateTimeField(format='%D.%M.%Y')

    class Meta:
        model = Post
        fields = "__all__"

class ContentSerializer(serializers.ModelSerializer):
    class Meta:
        model = PostPiece
        exclude = ['order', 'parent']

class BlogUserSerializer(serializers.ModelSerializer):
    class Meta:
        model = BlogUser
        fields = ['id', 'username', 'email', 'profile_pic']

class CommentSerializer(serializers.ModelSerializer):
    creation_time = serializers.DateTimeField(format='%D.%M.%Y')

    class Meta:
        model = Comment
        fields = "__all__"
```

views.py

```
import json

from django.http import HttpResponse
from django_filters.rest_framework import DjangoFilterBackend
from rest_framework import viewsets
from rest_framework.permissions import IsAuthenticated

from api.serializers import BlogUserSerializer, CommentSerializer, PostSerializer, ContentSerializer
from logic.models import Post, BlogUser, Comment, PostPiece
```

```
class MyPostsROViewSet(viewsets.ReadOnlyModelViewSet):
    serializer_class = PostSerializer
    permission_classes = [IsAuthenticated]

    def get_queryset(self):
        return Post.objects.filter(user=self.request.user)
```

```
class ContentViewSet(viewsets.ModelViewSet):
    serializer_class = ContentSerializer
    queryset = PostPiece.objects.order_by('order')
    filterset_fields = ['parent']
```

```
class PostViewSet(viewsets.ModelViewSet):
    queryset = Post.objects.order_by('creation_time')
    serializer_class = PostSerializer
    filterset_fields = ['user']
```

```
class BlogUserViewSet(viewsets.ModelViewSet):
    queryset = BlogUser.objects.all()
    serializer_class = BlogUserSerializer
```

```
class CommentViewSet(viewsets.ModelViewSet):
    queryset = Comment.objects.all()
    serializer_class = CommentSerializer
    filterset_fields = ['post', 'user']
```

urls.py

```
from django.conf.urls import url
from django.urls import path
from rest_framework import routers, permissions
from drf_yasg.views import get_schema_view
from drf_yasg import openapi
from rest_framework_simplejwt.views import TokenObtainPairView, TokenRefreshView
from api.views import BlogUserViewSet, CommentViewSet, PostViewSet, ContentViewSet, get_user_id, MyPostsROViewSet

schema_view = get_schema_view(
    openapi.Info(
        title="Blog API",
        default_version='v1',
        description="Blog REST API description",
        contact=openapi.Contact(email="contact@snippets.local"),
        license=openapi.License(name="BSD License"),
    ),
    public=True,
    permission_classes=[permissions.AllowAny],
```

```

)

router = routers.SimpleRouter()
router.register(r'contents', ContentViewSet, basename='contents')
router.register(r'posts', PostViewSet, basename='posts')
router.register(r'users', BlogUserViewSet, basename='users')
router.register(r'comments', CommentViewSet, basename='comments')
router.register(r'my_posts', MyPostsROViewSet, basename="my_posts")

urlpatterns = [
    path('token/',
        TokenObtainPairView.as_view(),
        name='token_obtain_pair'),
    path('token/refresh/',
        TokenRefreshView.as_view(),
        name='token_refresh'),
    url(r'^swagger(?P<format>\.json|\.yaml)$', schema_view.without_ui(cache_timeout=0), name='schema-json'),
    url(r'^swagger/$', schema_view.with_ui('swagger', cache_timeout=0), name='schema-swagger-ui'),
    url(r'^redoc/$', schema_view.with_ui('redoc', cache_timeout=0), name='schema-redoc'),
]

urlpatterns += router.urls

```