Article
Dmitry Maslennikov · Jul 30 5m read

Open Exchange

Introduction to Django part 3

Continuing to observe the possibilities of Django, and usage with IRIS. The first we have looked how to define models and connect to tables already existing in IRIS, than we extended embedded Django Administration portal, with an ability to see what data we have in that models, with filters, editing and even pagination.

Time to go to real action, now we a going to create some REST API, on Django, based on the same data, we used before from the package posts-and-tags.

To do so, we will use Django REST Framework

Django REST framework is a powerful and flexible toolkit for building Web APIs.

Some reasons you might want to use REST framework:

- The Web browsable API is a huge usability win for your developers.
- Authentication policies including packages for OAuth1a and OAuth2.
- Serialization that supports both ORM and non-ORM data sources.
- Customizable all the way down - just use regular function-based views if you don't need the more powerful features.
- Extensive documentation, and great community support.
- Used and trusted by internationally recognised companies including Mozilla, Red Hat, Heroku, and Eventbrite.

First we need to update our requirements.txt file with dependencies

```
# Django itself
django>=4.0.0
```

```
# InterSystems IRIS driver for Django, and DB-API driver from InterSystems
django-iris=>0.1.13
https://raw.githubusercontent.com/intersystems-community/iris-driver-
distribution/main/DB-API/intersystems_irispython-3.2.0-py3-none-any.whl

# Django REST Framework and its optional dependencies
djangorestframework>=3.4.4
# Markdown support for the browsable API.
markdown>=3.0.0
# Filtering support
django-filter>=1.0.1
```

And install them

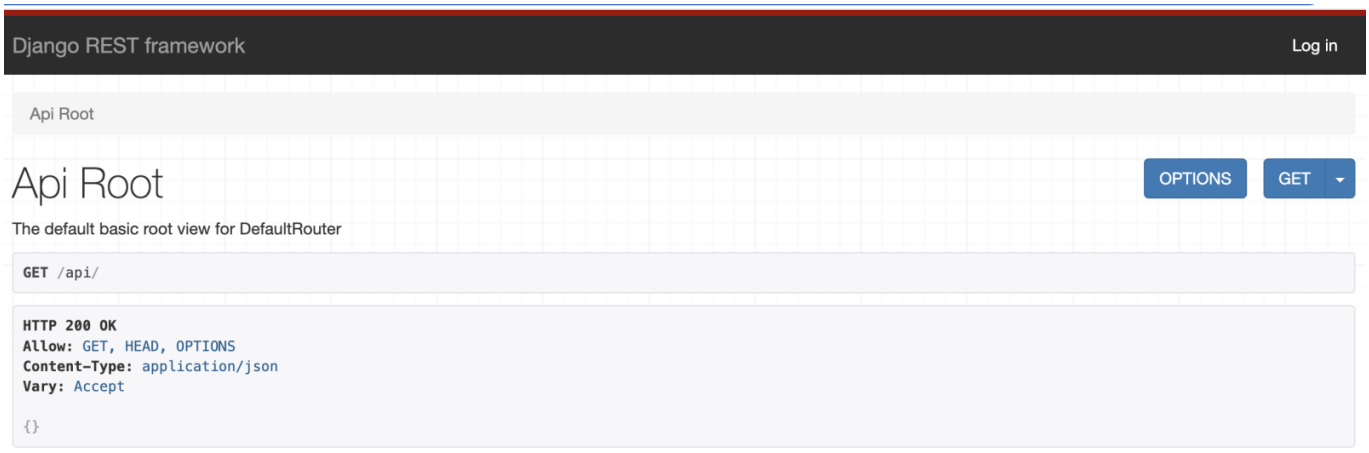
```
python install -r requirements.txt
```

First draft of API

Update *urls.py* file to this. Here we say, that from the root for our api as ***api/***, so any requests to http://localhost:8000/api/ will be used as root for our API requests

```
from django.contrib import admin
from django.urls import path, include
from rest_framework import routers

router = routers.DefaultRouter()

urlpatterns = [
    path('api/', include(router.urls)),
    path('admin/', admin.site.urls),
    path('api-auth/', include('rest_framework.urls')),
]
```

Django REST Framework, has embedded UI for API, when server running in development mode with DEBUG=True in settings.py. And we can open this URL



All working, even did not even define anything, just connected that framework to url. It does support authorization, for requests required authorization.

```
$ curl http://127.0.0.1:8000/api/
{}
```

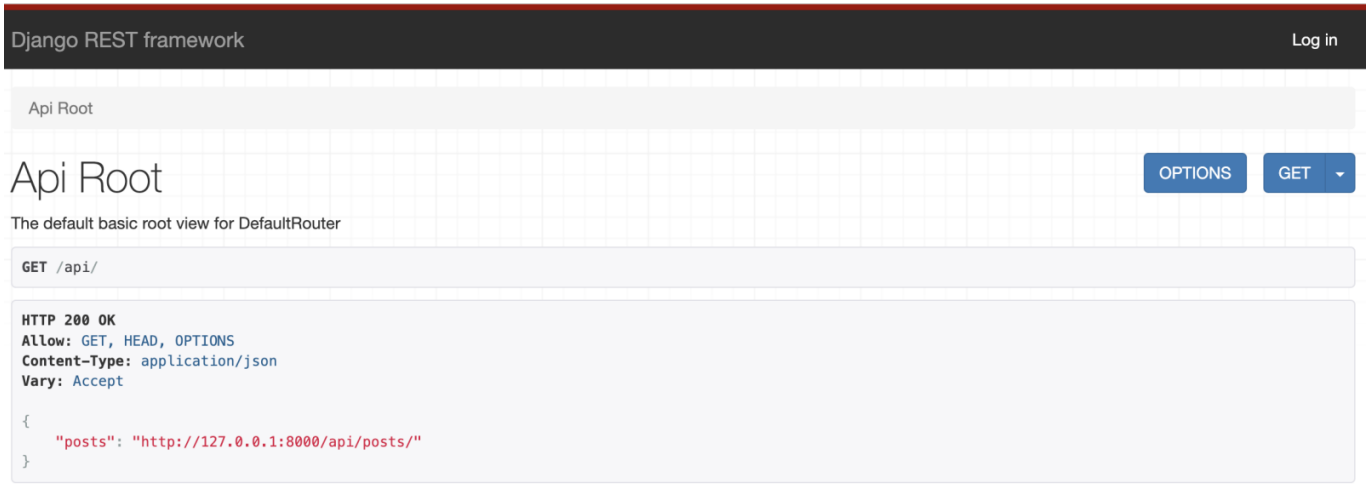
Do define an API for project, we will use a few features from REST Framework at minimum

- Serializers - allow complex data such as querysets and model instances to be converted to native Python datatypes that can then be easily rendered into JSON, XML or other content types. Serializers also provide deserialization, allowing parsed data to be converted back into complex types, after first validating the incoming data.
- ViewSet - allows you to combine the logic for a set of related views in a single class

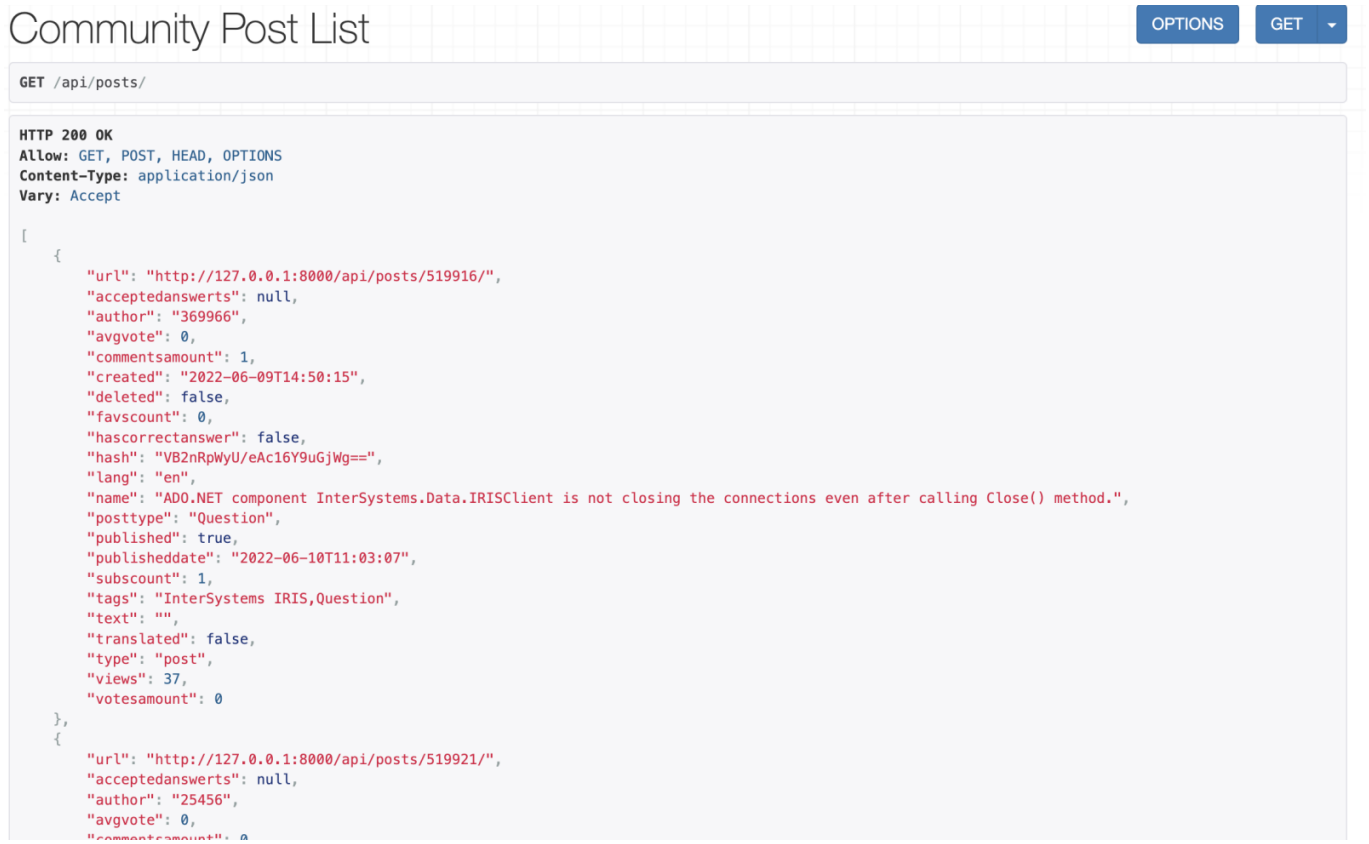
Let's add an endpoint for our posts. Very simple, yet. Look at the updated content of ***urls.py***

```
from django.contrib import admin
from django.urls import path, include
from rest_framework import routers, serializers, viewsets
from .models import CommunityPost

router = routers.DefaultRouter()

class CommunityPostSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        # class with model
        model = CommunityPost
        # list of fields to show, or just '__all__'
        fields = '__all__'
# ViewSets define the view behavior.
class CommunityPostViewSet(viewsets.ModelViewSet):
    queryset = CommunityPost.objects.all()
    serializer_class = CommunityPostSerializer

# connect it with API
router.register(r'posts', CommunityPostViewSet)

urlpatterns = [
    path('api/', include(router.urls)),
    path('admin/', admin.site.urls),
    path('api-auth/', include('rest_framework.urls')),
]
```

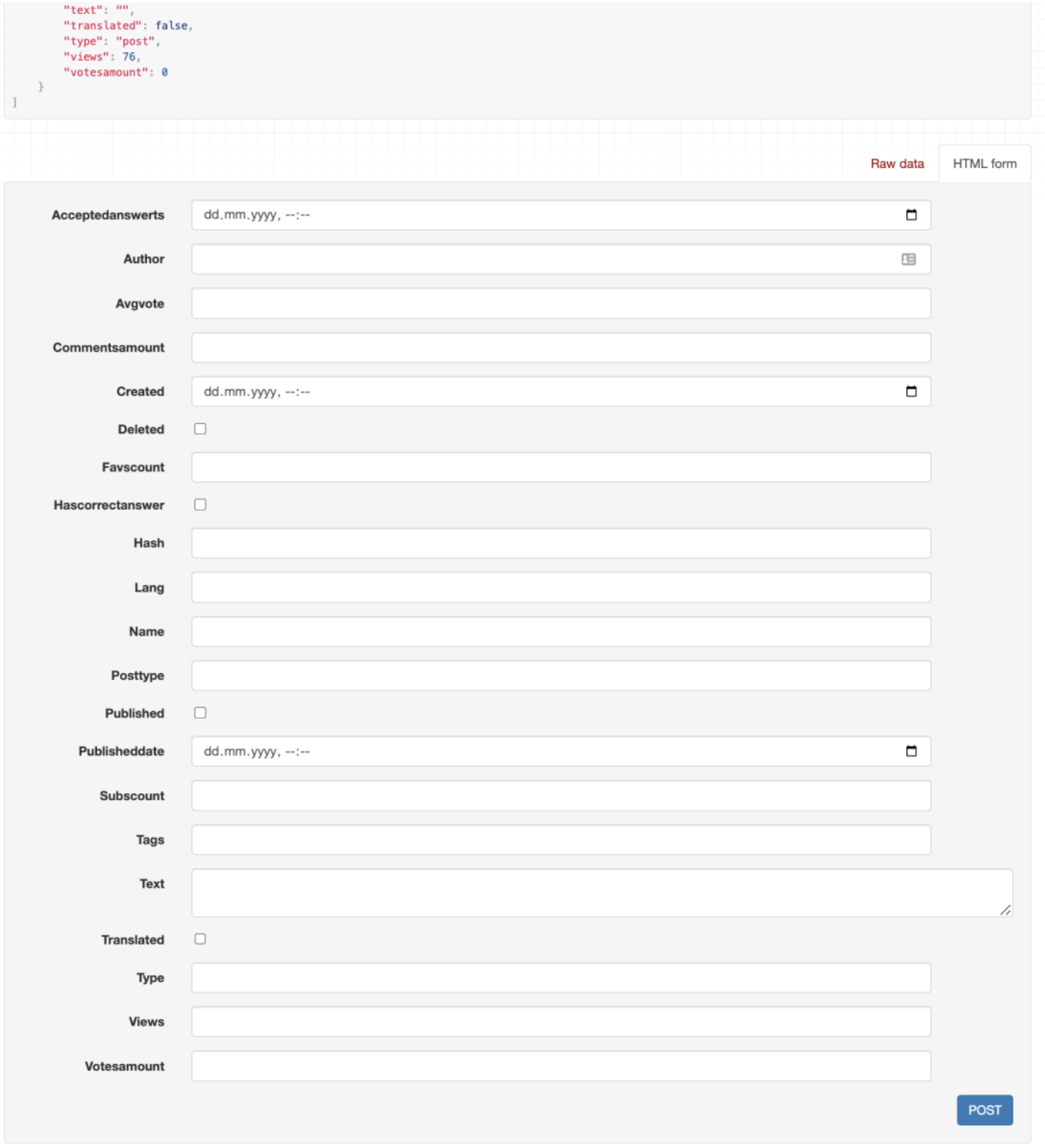
And it now appeared in the web UI



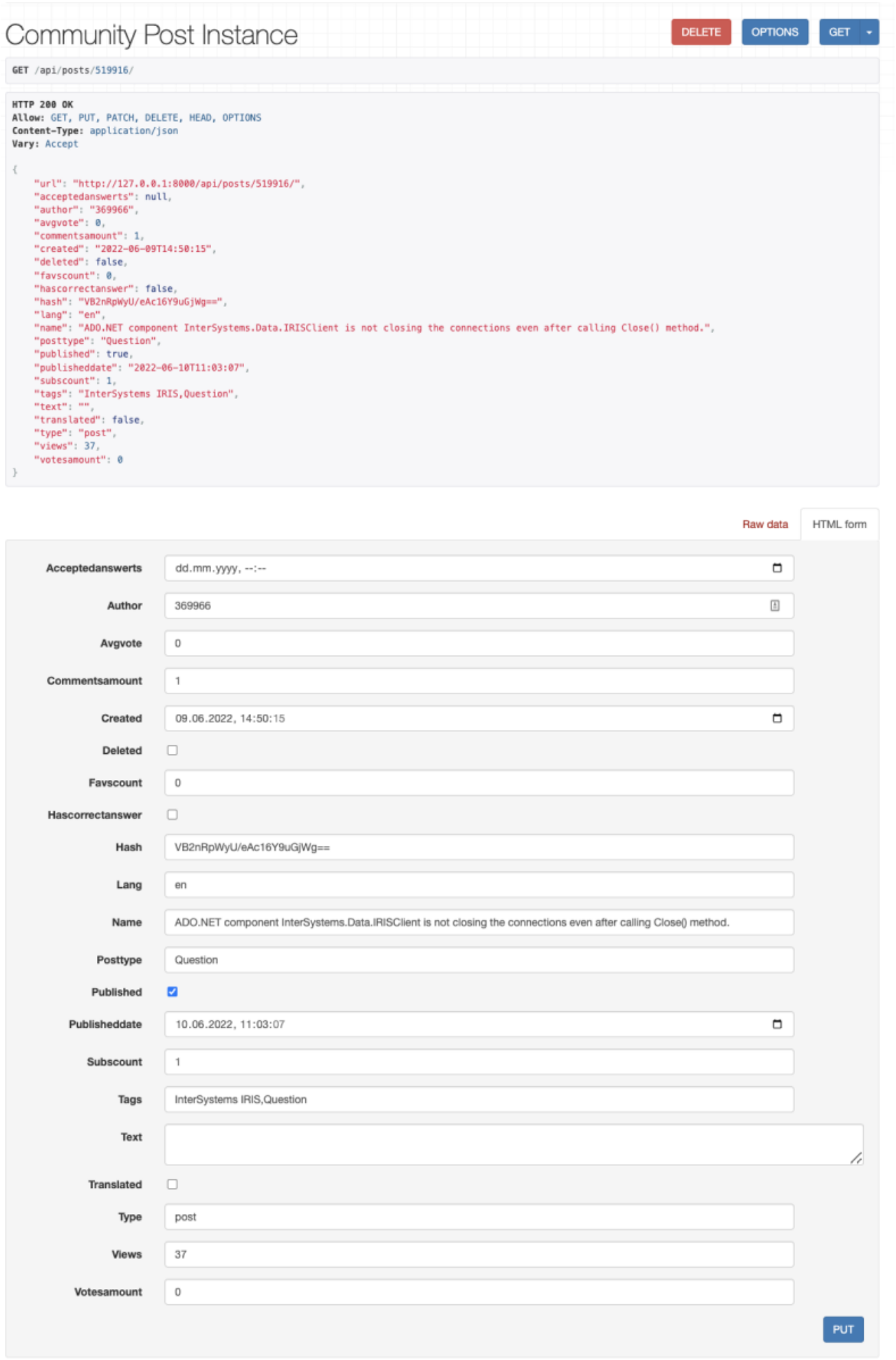
Just click on the link here, and you will the response for it



Scroll to the end, and you will find a generated form for new items, which can be added by POST request. All the fields are suitable for the type of properties



In the list of items, you may click on any item's URL, and see this. Just only this item in response, and an editing form with PUT request


Community Post Instance
DELETE
OPTIONS
GET
GET /api/posts/519916/
HTTP 200 OK
Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
{
"url": "http://127.0.0.1:8000/api/posts/519916/",
"acceptedanswerts": null,
"author": "369966",
"avgvote": 0,
"commentsamount": 1,
"created": "2022-06-09T14:50:15",
"deleted": false,
"favscount": 0,
"hascorrectanswer": false,
"hash": "VB2nRpWyU/eAc16Y9uGjWg==",
"lang": "en",
"name": "ADO.NET component InterSystems.Data.IRISClient is not closing the connections even after calling Close() method.",
"posttype": "Question",
"published": true,
"publisheddate": "2022-06-10T11:03:07",
"subscount": 1,
"tags": "InterSystems IRIS,Question",
"text": "",
"translated": false,
"type": "post",
"views": 37,
"votesamount": 0
}
Raw data
HTML form
Acceptedanswerts
dd.mm.yyyy, --:--
Author
369966
Avgvote
0
Commentsamount
1
Created
09.06.2022, 14:50:15
Deleted
Favscount
0
Hascorrectanswer
Hash
VB2nRpWyU/eAc16Y9uGjWg==
Lang
en
Name
ADO.NET component InterSystems.Data.IRISClient is not closing the connections even after calling Close() method.
Posttype
Question
Published
Publisheddate
10.06.2022, 11:03:07
Subscount
1
Tags
InterSystems IRIS,Question
Text
Translated
Type
post
Views
37
Votesamount
0
PUT

Authentication

And you already can do change the data, by PUT or POST. The need of authorization not activated, yet. REST Framework offers various combination of authentications for use, so, you can open some of the resources for anonymous access read-only. And authenticate to be able to do any changes. Or completely close any access. We just configure, read-only for anonymous, and require authentication for any changes. To do it, just add this lines to ***settings.py***

```
REST_FRAMEWORK = {
    # Use Django's standard `django.contrib.auth` permissions,
    # or allow read-only access for unauthenticated users.
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.DjangoModelPermissionsOrAnonReadOnly',
    ],
}
```

With this, you will not see form anymore until you'll login, with the username and password for instance created previously for Django Administration.

Pagination

By default, it does not have pagination, but we can easily add it to any list queries. Update **RESTFRAMEWORK** variable in ***settings.py***. Setup pagination class, and default page size

```
REST_FRAMEWORK = {
...
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.LimitOffsetPagination',
    'PAGE_SIZE': 10,
...
}
```

In result, it slightly changed the resulting JSON, added relevant items, such as, next and previous pages links, as well, as amount of all items. And with Web UI you can go through the pages.

Filtering and Searching

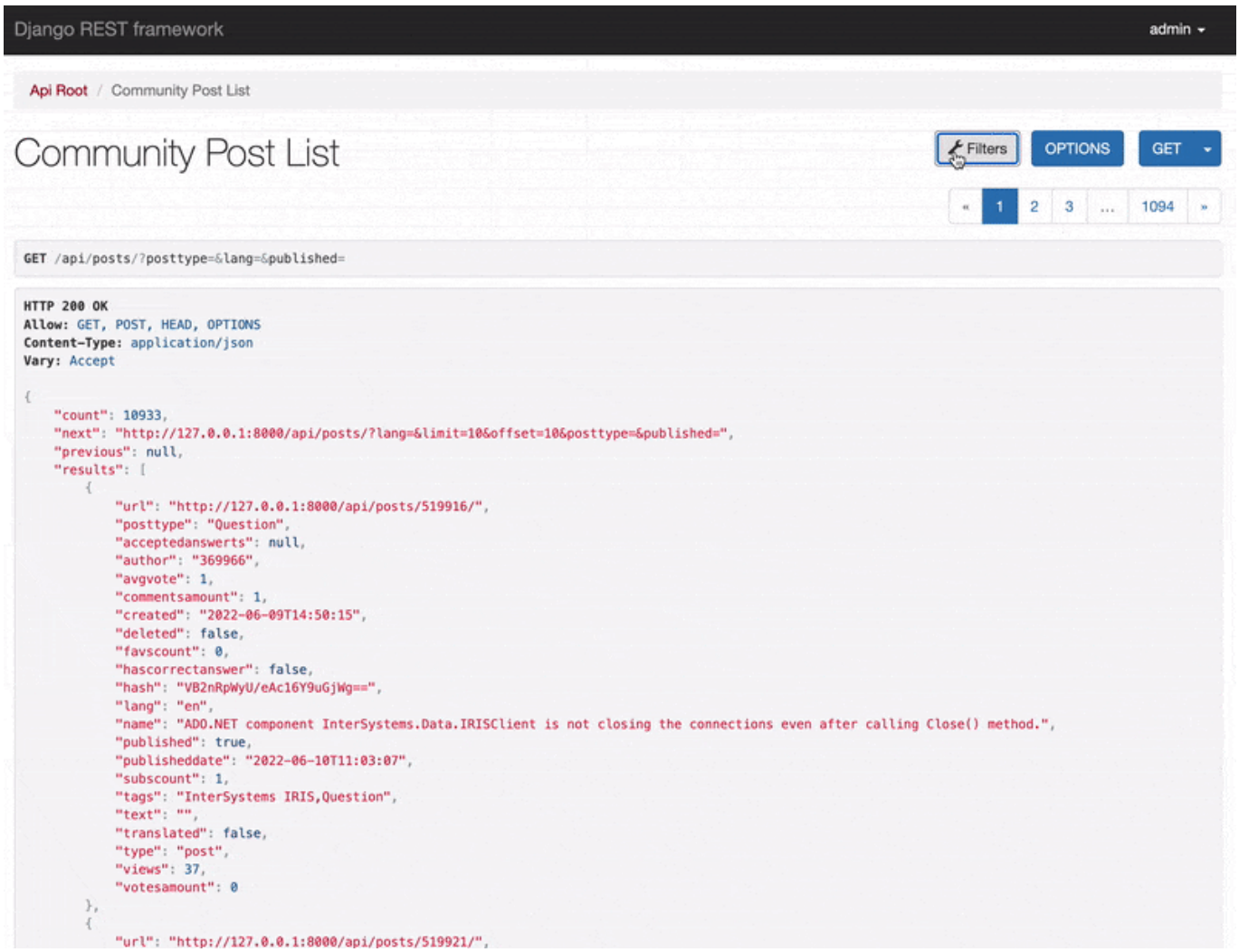
It is is also, quite simple to add filtering capabilities, and searching. Update **REST_FRAMEWORK** variable in ***settings.py***.

```
REST_FRAMEWORK = {
...
    'DEFAULT_FILTER_BACKENDS': [
        'django_filters.rest_framework.DjangoFilterBackend',
        'rest_framework.filters.SearchFilter',
    ],
...
]
```

And update CommunityPostViewSet with a list of fields for filtering and for search

```
class CommunityPostViewSet(viewsets.ModelViewSet):
    queryset = CommunityPost.objects.all()
    serializer_class = CommunityPostSerializer
    filterset_fields = ['posttype', 'lang', 'published']
    search_fields = ['name',]
```

And it's working right from the Web UI



And after all, we have fully functional REST API, just for this one resource, yet. Quite simple at the moment, but it's possible to make many customization, connect other resources and link them.

#Embedded Python #Python #InterSystems IRIS
Check the related application on InterSystems Open Exchange

Source URL:https://community.intersystems.com/post/introduction-django-part-3