Cute cats web app (Django + VueJs) — Create a REST API with DRF

Jrmy Follow
Jun 21, 2018 · 8 min read

Summary

Part I—Settings

Part II—Create VueJs views

Part III—Create Rest API with DRF

Part IV—Get datas from API with VueJs

Introduction

In previous parts we created a new Django project and see how to use VueJs as frontend instead of Django templates. Then, we saw how to customize VueJs views developping several components. That's cool but in this state, our cute cat application is not really usefull. In fact, we just take a picture from Internet and display it in a view.

By now, we are going to create an API endpoint to be able to

- Save datas about cute cats in a database
- Render a list of those cats
- Show a ID Card when clicking on a cat

Are you ready to go ?

Django Rest Framework

Because we are working with Python and Django, I choose to work with Django Rest Framework (DRF). It's a powerful and flexible toolkit for building Web APIs. We can check datas in browsers, define some models and serializers to completly customize the way datas are rendered

First, we have to install DRF and tell it to Django.

```
# get DRF
pip install django-rest-framework


# settings.py

INSTALLED_APPS = [
    'django.contrib.admin',
    ...
    'rest_framework',
]
```

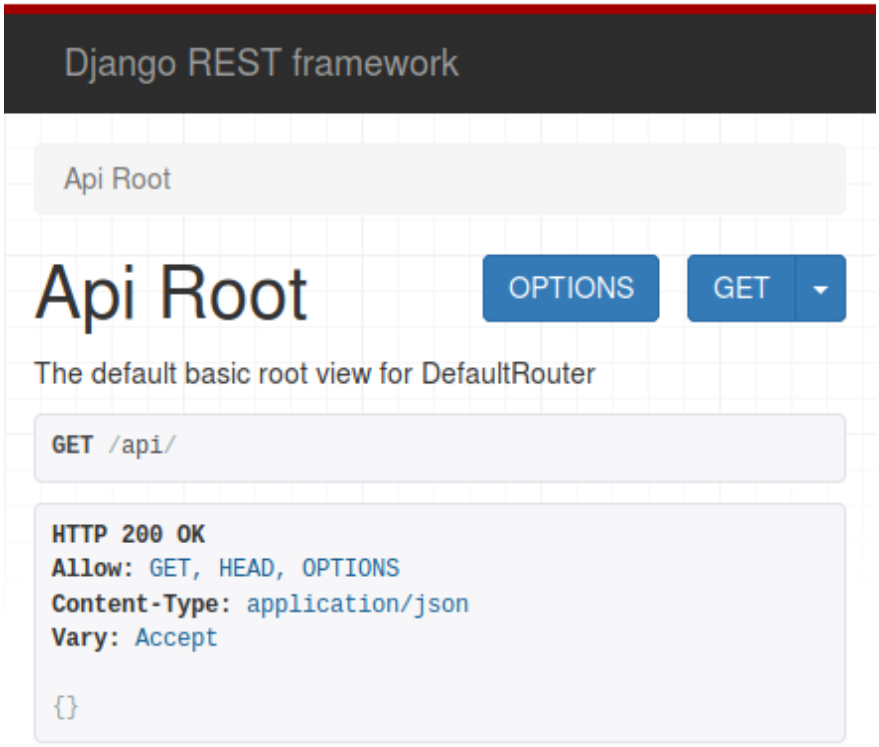
Then, we have to add the url where we can reach the API endpoint

```
# django_project/urls.py
```

```
from django.conf.urls import include, url
from django.contrib.staticfiles.urls import
staticfiles_urlpatterns
from rest_framework import routers


router = routers.DefaultRouter()
```

```
app_name = "django_project"
urlpatterns = [
    url(r'^$',
TemplateView.as_view(template_name='index.html'),
name='uHome'),
    url(r'^api/',include(router.urls)),
]

urlpatterns += staticfiles_urlpatterns()
```

Now, in our browser we are requesting *localhost:8000/api* . We should see the following screen



Welcome in your API root. As you can see, there is nothing here but later you will be able to check every datas available in database.

DRF is ready to use by now. We have to feed him with some cats. First, we have to create a new django app, let's say a cat app.

Cat app

To create a new django application we are using *django-admin startapp* command

```
# Create cat app in apps folder.
```

```
django-admin startapp cat
```

```
# Treeview after cat app creation
```

```
-django_project
|_ dist
|_ django_project
   |_ settings.py
   |_ urls.py
   |_ views.py
   |_ apps
      |_ cat
```

```
|_ src
|_ index.html
```

```
|_ ...
```

Once cat application is created, we must define it in django_projects settings.

```
# settings.py

INSTALLED_APPS = [
    'django.contrib.admin',
    ...
    'rest_framework',
    'cat'
]
```

Let's see cat folder content

```
-django_project
|_ dist
|_ django_project
   ...
   |_ apps
      |_ cat
         |_ __init__.py
         |_ admin.py
         |_ apps.py
         |_ models.py
         |_ tests.py
         |_ views.py
```

- *admin.py* is to describe how to render datas in django admin template
- *models.py* is to define a model, or a schema, about cat information
- *tests.py* to write some test about the app
- *views.py* is to set a server response depending of the request

In addition to those files, we have to create the following :

- *renderers.py* to define how datas will be rendered
- *serializers.py* to choose which information must be shown or not
- *urls.py* to define where we can find information

Well, what do we want to know about cats ? We can answer to this in *models.py file.*

Cat model

A cat can have

- owner *[char value]*
- name *[short char value]*
- colors *[selected value]*
- age *[int value]*
- photo *[image value]*
- favorite toy *[short char value]*
- mood *[selected value]*

Those seven informations will appaer in our cat model.

```
# models.py

from django.db import models
# from django.contrib.auth.models import (
```

```
#     AbstractBaseUser, BaseUserManager,
PermissionsMixin
# )
```

```
BLACK = 'BLK'
WHITE = 'WHE'
GREY = 'GRY'
BROWN = "BRW"
RED = "RED"
```

```
COLORS = (
    (BLACK, 'black'),
    (WHITE, 'white'),
    (GREY, 'grey'),
    (BROWN, 'brown'),
    (RED, 'red')
)
```

```
HAPPY = "HAY",
GRUMPY = "GRY",
BAD = "BAD",
MILKHOLIC = "MIC"


MOODS = (
    (HAPPY, "happy"),
    (GRUMPY, "grumpy"),
    (BAD, "bad"),
    (MILKHOLIC, "milkholic")
)
class Cat(models.Model):
    owner = models.CharField(max_length = 150, blank =
True)
    name = models.CharField(max_length = 50, blank =
True)
    colors = models.CharField(max_length = 3, choices
= COLORS, default = BLACK)
    age = models.IntegerField(blank = True)
    photo = models.URLField(blank=True)
    toy = models.CharField(max_length = 50, blank =
True)
    mood = models.CharField(max_length = 3, choices =
MOODS, default = HAPPY)
```

```
    created_at =
models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

```
    class Meta:
      ordering = ('created_at',)
```

A model is a class inheriting from models.Model. We are creating a Cat class and adding several fields.

- CharField will contain some text, it always have a max_length attribute
- IntergerField will contain a number
- URLField will containt a link. To make it simple, we are going to save a link from a cat image.
- created_at field will save date and time at the creation of the new entry
- updated_at field will be save date and time each time the entry change.

All those fields have a blank attribute set to True because we don't care if information are missing.

Well, each time we change this model we have to update the database by using makemigrations and migrate manage.py method in a command windows.

```
# terminal

python manage.py makemigrations

python manage.py migrate
```

Cat Serializer

Now that the database is updated with the Cat model, we will define a CatSerializer.

```
# apps/cat/serializers.py

from rest_framework import serializers
from .models import Cat

class CatListSerializer(serializers.ModelSerializer):
    owner = serializers.CharField(max_length = 150,
allow_blank = True, required = False)
    name = serializers.CharField(max_length = 50,
allow_blank = True, required = False)
    age = serializers.IntegerField(required = False)
    photo = serializers.URLField(required = False)
```

```
class Meta:
        model = Cat
        fields = ('id', 'owner', 'name', 'age',
'photo')
```

```
class CatSerializer(serializers.ModelSerializer):
    owner = serializers.CharField(max_length = 150,
allow_blank = True, required = False)
    name = serializers.CharField(max_length = 50,
allow_blank = True, required = False)
    colors = serializers.CharField(max_length = 3,
required = False)
    age = serializers.IntegerField(required = False)
    toy = serializers.CharField(max_length = 50,
allow_blank = True, required = False)
    mood = serializers.CharField(max_length = 3,
required = False)
    photo = serializers.URLField(required = False)
```

```
class Meta:
        model = Cat
        fields = ('id', 'owner', 'name', 'colors',
'age',
                  'toy', 'mood', 'photo')
```

Serializers are pretty similar to models. It is a class where we define the type of each field and wich one will be render by the server. We can define different serializers for the same models. As below, CatListSerializer and CatSerializer are defined. CatListSerializer will return less fields than CatSerializer. In the list of cat we just need a summary about each, so we don't care about colors or mood. Instead of CatSerializer where we want to know everything about our favorite cat ❤.

Cat renderer

Let's say that we want to get cats datas in a JSON object as the following.

```
{
 'cats': {... datas ...}
}
```

To define it, go to renderers.py

```
import json
from rest_framework.renderers import JSONRenderer
```

```
class CatJSONRenderer(JSONRenderer):
    charset = 'utf-8'
```

```
    def render(self, data, media_type=None,
renderer_context=None):
        errors = data.get('errors', None)

        if errors is not None:
            return super(CatJSONRenderer,
self).render(data)
```

```
    return json.dumps({
            'cats': data
        })
```

Here, we are checking if there is some errors, thanks to JSONRenderer, if not we return a JSON object.

Well, cats datas are packed in a ready to use JSON object

It's time to define what we are going to do with those datas for each request (GET, POST, PUT, DELETE, etc…). To do that go to the *views.py* file.

Cat views

The application should display a list of all the cute cat present in the database and when the user click on a cat picture, he will be redirect to the cat ID Card. So, we need to define to different views. ListView and RetrieveView.

ListView will return, as you can guess, a complete known cat list.

```
# views.py
```

```
from rest_framework import status
from rest_framework.generics import ListAPIView,
RetrieveAPIView
from rest_framework.permissions import AllowAny
from rest_framework.response import Response
```

```
from .models import Cat
from .renderers import CatJSONRenderer
from .serializers import CatSerializer,
CatListSerializer
```

```
class CatListApiView(ListAPIView):
    model = Cat
    queryset = Cat.objects.all()
    permissions_classes = (AllowAny, )
    renderer_classes = (CatJSONRenderer, )
    serializer_class = CatListSerializer
```

With DRF it's really simple to retrieve a list. Just import ListAPIView from rest_framework.generics and create a CatListApiView whose inherit from.

Three attributes are define :

- permissions_classes set to AllowAny, it's mean that everybody can acces to this list
- renderers_classes is equal to CatRenderer we defined before
- serializer_class is set to CatSerializer

To get more details about a specific cat, we need to define a retrieve view. Let's add some code to views.py

```
class CatRetrieveApiView(RetrieveAPIView)
    permission_classes = (AllowAny, )
    renderer_classes = (CatJSONRenderer, )
    serializer_class = CatSerializer
```

```
    def retrieve(self, request, cat, *args, **kwargs):
      cat = Cat.objects.get(id = cat.id)
      serializer = self.serializer_class(cat)
```

```
      return Response(serializer.data, status =
status.HTTP_200_OK)
```

To access to those views we will define urls as entries points.

Cat urls

```
# cat/urls.py
```

```
from django.conf.urls import url
from .views import CatListApiView, CatRetrieveApiView
```

```
app_name = 'cat'
urlpatterns = [
    url(r'^cats/$', CatListApiView.as_view())
    url(r'^cats/(?P<cat_id>\w+)/?$',
CatRetrieveApiView.as_view()),
]
```

```
# ___ django_project/urls.py ___
```

```
from django.conf.urls import include, url
from django.contrib import admin
from django.views.generic import TemplateView
from rest_framework import routers

router = routers.DefaultRouter()
```

```
app_name = "databoks"
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$',
TemplateView.as_view(template_name='index.html'),
name='uHome'),
    url(r'^api/',
include('django_project.apps.cat.urls',
namespace='cats')),
    url(r'^api/',include(router.urls)),
]
```

We import views defined earlier and use them as views. Don’t forget to update django_project urls by including cats urls.

- *./api/cats* will return a full list of cats
- *./api/cats/cat_id* will return details about a specific cat

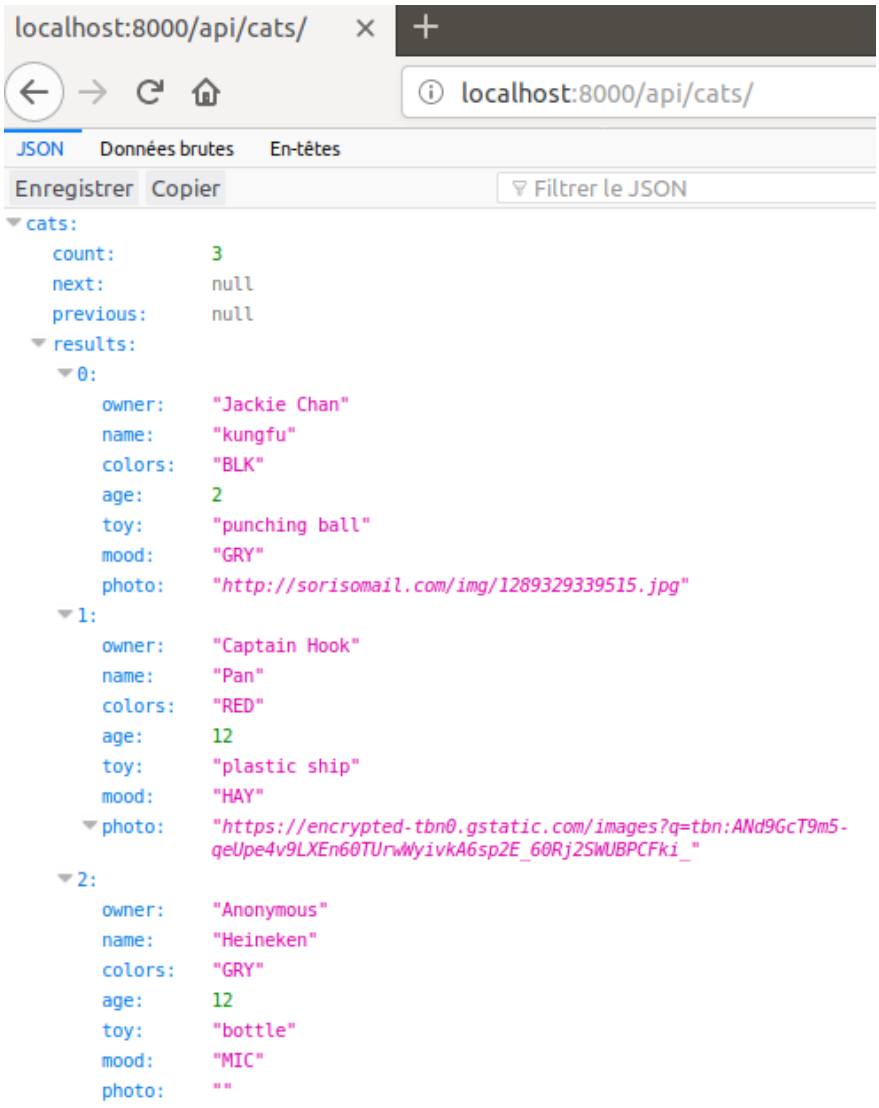
Guess what !? We are done ! Everything is ready to use our new Rest API.

Testing with DRF browsable API

First, we need to populate the database with some cats. Make it simple, I just use shell_plus to save datas in Cat table. At the end the Cat object contain three entries.

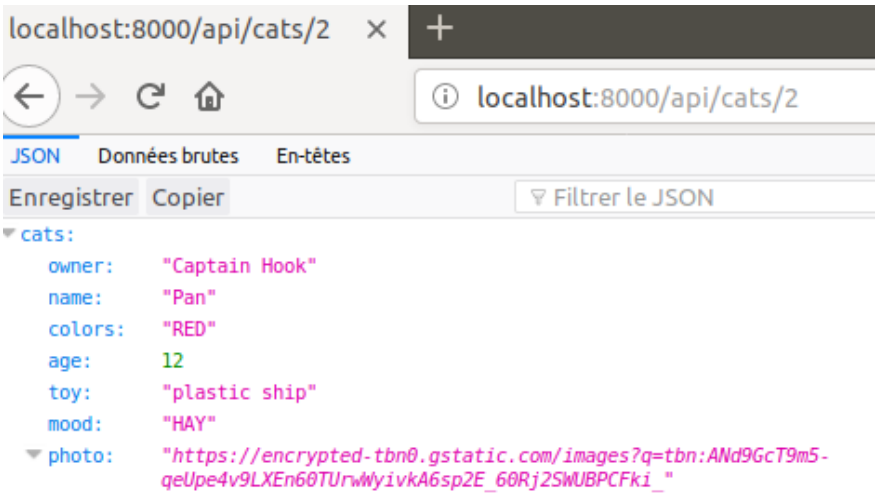
```
en, Exists, OuterRef, Subquery
from django.utils import timezone
from django.urls import reverse
Python 3.6.3 (default, Oct  3 2017, 21:45:48)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.4.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: Cat.objects.create(owner="Jackie Chan", name="kungfu", colors="BLK", age
   ...: =2, toy="punching ball", mood="GRY",photo="http://sorisomail.com/img/128
   ...: 9329339515.jpg")
Out[1]: <Cat: Cat object (1)>

In [2]: Cat.objects.create(owner="Captain Hook", name="Pan", colors="RED", age=1
   ...: 2, toy="plastic ship", mood="HAY",photo="https://encrypted-tbn0.gstatic.
   ...: com/images?q=tbn:ANd9GcT9m5-qeUpe4v9LXEn60TUrwWyivkA6sp2E_60Rj2SWUBPCFki
   ...: _")
Out[2]: <Cat: Cat object (2)>

In [3]: Cat.objects.create(owner="Anonymous", name="Heineken", colors="GRY", age
   ...: =12, toy="bottle", mood="MIC",photo="")
Out[3]: <Cat: Cat object (3)>

In [4]: quit()
```

Open your favorite browser, run python server and make a request to …api/cats. It should display a JSON object who is containing all cats



Then, if we request a specific cat, with his ID number, browser must display a JSON object who is containing only information about that cat.

Go to api/cats/2



Done !

Conclusion

With part I and II we were able to display VueJS template with Python server, but the content was static. To fix that we want to save datas in a database and get them when we want.

The solution here, is to use DRF. This one allow us to create data models and make request, like GET, POST, PUT, easily. By defining a serializer, we select needed fields. By defining a renderer, we choose how datas are render to us. By defining urls, we choose how to request data that we need.

By now, We can access to all datas saved in database whenever we want. Next step will be about how to allow user to get and see those datas. See you next time ;)