MEMO REST API DJANGO

Source :

http://www.django-rest-framework.org/tutorial/3-class-based-views/#tutorial-3-class-based-views

https://godjango.com/41-start-your-api-django-rest-framework-part-1/

https://scotch.io/tutorials/build-a-rest-api-with-django-a-test-driven-approach-part-2

Etape I

api/urls.py

Création d'un fichier urls.py dans l'app ici 'api'

```
from django.conf.urls import patterns, url

urlpatterns = patterns(
    'api.views',
    url(r'^tasks/$', 'task_list', name='task_list'),
    url(r'^tasks/(?P<pk>[0-9]+)$', 'task_detail', name='task_detail'),
)
```

models.py

Création du models pour la REST API

```
from django.db import models

class Task(models.Model):
    completed = models.BooleanField(default=False)
    title = models.CharField(max_length=100)
    description = models.TextField()
```

serializers.py

Création d'un fichier serializer.py

```
from rest_framework import serializers

from task.models import Task


class TaskSerializer(serializers.ModelSerializer):

    class Meta:
        model = Task
        fields = ('title', 'description', 'completed')
```

settings.py

Paramétrage dans le settings.py du projet

```
INSTALLED_APPS = (
    'task',
    'rest_framework',
    'api',
)
```

Etape II

todo/urls.py

Paramétrage url dans le fichier urls.py du projet

```
from django.conf.urls import patterns, include, url

from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns(
    '',
    url(r'^api/', include('api.urls')),
    url(r'^admin/', include(admin.site.urls)),
)
```

Etape III

views.py

Mise en place des views sans le mode générique des méthodes GET & POST

```
from rest_framework import status
from rest_framework.decorators import api_view
from rest_framework.response import Response

from task.models import Task
from api.serializers import TaskSerializer


@api_view(['GET', 'POST'])
def task_list(request):
    """
    List all tasks, or create a new task.
    """
    if request.method == 'GET':
        tasks = Task.objects.all()
        serializer = TaskSerializer(tasks, many=True)
        return Response(serializer.data)

    elif request.method == 'POST':
        serializer = TaskSerializer(data=request.DATA)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        else:
            return Response(
                serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Etape III (suite …)

Suite des views pour l'API le mode générique des méthodes GET & POST& PUT

```
@api_view(['GET', 'PUT', 'DELETE'])
def task_detail(request, pk):
    """
    Get, udpate, or delete a specific task
    """
    try:
        task = Task.objects.get(pk=pk)
    except Task.DoesNotExist:
        return Response(status=status.HTTP_404_NOT_FOUND)

    if request.method == 'GET':
        serializer = TaskSerializer(task)
        return Response(serializer.data)

    elif request.method == 'PUT':
        serializer = TaskSerializer(task, data=request.DATA)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        else:
            return Response(
                serilizer.errors, status=status.HTTP_400_BAD_REQUEST)

    elif request.method == 'DELETE':
        task.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)
```

Rewriting notre API en utilisant CLASS-BASED VIEWS

```
from django.conf.urls import url
from rest_framework.urlpatterns import format_suffix_patterns
from snippets import views

urlpatterns = [
    url(r'^snippets/$', views.SnippetList.as_view()),
    url(r'^snippets/(?P<pk>[0-9]+)/$', views.SnippetDetail.as_view()),
]

urlpatterns = format_suffix_patterns(urlpatterns)
```

On récrit le fichier urls.py de notre app.

On efface les fonctions dans notre fichier *views.py* basé sur 2 classes :

- Première classe : Générer la liste de notre model de tous nos objets
- Deuxième classe : Afficher le détail de chaque objet et de faire les opérations : update,delete,get

Première classe :

```python
from snippets.models import Snippet
from snippets.serializers import SnippetSerializer
from django.http import Http404
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status


class SnippetList(APIView):
    """
    List all snippets, or create a new snippet.
    """
    def get(self, request, format=None):
        snippets = Snippet.objects.all()
        serializer = SnippetSerializer(snippets, many=True)
        return Response(serializer.data)

    def post(self, request, format=None):
        serializer = SnippetSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Deuxième classe :

```python
class SnippetDetail(APIView):
    """
    Retrieve, update or delete a snippet instance.
    """
    def get_object(self, pk):
        try:
            return Snippet.objects.get(pk=pk)
        except Snippet.DoesNotExist:
            raise Http404

    def get(self, request, pk, format=None):
        snippet = self.get_object(pk)
        serializer = SnippetSerializer(snippet)
        return Response(serializer.data)

    def put(self, request, pk, format=None):
        snippet = self.get_object(pk)
        serializer = SnippetSerializer(snippet, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    def delete(self, request, pk, format=None):
        snippet = self.get_object(pk)
        snippet.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)
```