INITIATION À DJANGO REST FRAMEWORK

SOMMAIRE

- Ajouter DRF à Django
- Création API sur 1 modèle
- Première vue
- URLs de l'API
- Utilisation avec curl
- Deuxième modèle

AJOUTER DRF AU PROJET

AJOUT DANS LES PAQUETS REQUIS

```
pip install djangorestframework
```

Dans `requirements.txt`

```
pip freeze > requirements.txt
```

AJOUT DE L'APP DANS LE PROJET

Dans `settings.py`

```
INSTALLED_APPS = (
    ...
    'rest_framework',
)
```

PREMIERS PAS

SERIALIZER

Les **serializers** convertissent des instances de modèles et de querysets en structures de données Python natives

```
# polls/serializers.py
from rest_framework import serializers
from polls.models import Question

class QuestionSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Question
        fields = ['url', 'question_text', 'pub_date']
```

VIEWSET

Un ViewSet permet de grouper les vues liste et détail

```
# polls/views.py
from rest_framework import viewsets
from polls.models import Question
from polls.serializers import QuestionSerializer

class QuestionViewSet(viewsets.ModelViewSet):
    queryset = Question.objects.all().order_by("question_text")
    serializer_class = QuestionSerializer
```

CONFIGURATION DES URLS

Les routeurs génèrent une liste d'URLs à partir des viewsets enregistrés

```
# polls/urls.py
from django.urls import include
from rest_framework import routers

router = routers.DefaultRouter()
router.register(r'question', views.QuestionViewSet)

urlpatterns = [
    # ...
    path('api/v1/', include(router.urls)),
]
```

POURQUOI `v1` ?

Gérer les évolutions de l'API sans que les applications clientes ne soient impactés.

Pour permettre une rétro-compatibilité en cas d'évolution majeure et impactante
→ créer une v2 de l'API, et garder la v1 accessible.

PAGINATION

```
# monprojet/settings.py
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNu
    'PAGE_SIZE': 10
}
```

'DEFAULT_PAGINATION_CLASS' : nombre d'objets et liens page précédente / suivante

TESTER

Utiliser l'interface web :
http://localhost:8000/polls/api/v1/

Ou utiliser `curl`

REQUÊTE GET

Obtenir la liste des questions

```
$ curl -H 'Accept: application/json; indent=4' \
    http://localhost:8000/polls/api/v1/question/
```

REQUÊTE POST

```
curl -H 'Accept: application/json; indent=4' \
    -d "question_text=Youpi&pub_date=2020-01-11T11:00" \
    http://localhost:8000/polls/api/v1/question/
```

Renvoie

```
{
    "url": "http://localhost:8000/polls/api/v1/question/4/",
    "question_text": "Youpi",
    "pub_date": "2020-01-11T11:00:00Z"
}
```

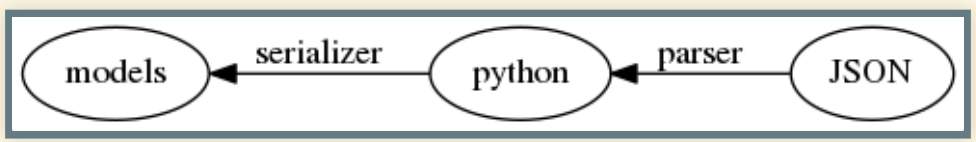
LES SERIALIZERS

SÉRIALISATION

- les **serializers** convertissent des instances de modèles et de querysets en structures de données Python natives
- ces structures de données Python peuvent ensuite être converties en JSON, XML, etc. par des **renderers**

DÉSÉRIALISATION

- les **parsers** convertissent les formats textuels en structures Python
- les **serializers** créent des instances de modèles à partir de ces structures



DEUXIÈME MODÈLE

SERIALIZER

```
# serializers.py
class ChoiceSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Choice
        fields = ['url', 'question', 'choice_text', 'votes']
```

VIEWSET

```
# views.py
class ChoiceViewSet(viewsets.ModelViewSet):
    queryset = Choice.objects.all().order_by('choice_text')
    serializer_class = ChoiceSerializer
```

URL

```
# urls.py
router.register(r'choice', views.ChoiceViewSet)
```

CONCLUSION

Avec peu de code, nous avons construit une API proposant 2 "end-points".