
TP initiation Django

Installation

Pour Windows:

`"python -m venv djangolnit"`

Votre répertoire "djangolnit" doit ressembler par la suite à cela:

```
26/11/2022 15:14 <DIR> .
26/11/2022 15:14 <DIR> ..
26/11/2022 15:14 <DIR> Include
26/11/2022 15:14 <DIR> Lib
26/11/2022 15:14      165 pyenv.cfg
26/11/2022 15:14 <DIR> Scripts
                1 fichier(s)      165 octets
                5 Rép(s) 122 349 940 736 octets libres
```

Ensuite à la racine lancer la commande pour activer votre venv:

`".\Scripts\activate"`

Dès que l'environnement est activé, vous le constaterez sur votre terminal:

```
(djangoInit) C:\Users\User\Desktop\INFORMATIQUE\M2MIAGE\ARI2\djangoInit>
```

Installer Django en lançant la commande :

`"py -m pip install Django"`

ou

`"python -m pip install Django"`

Le résultat devra être de la forme:

```
Collecting tzdata
  Downloading tzdata-2022.6-py2.py3-none-any.whl (338 kB)
    338.8/338.8 kB 3.5 MB/s eta 0:00:00
Collecting asgiref<4,>=3.5.2
  Downloading asgiref-3.5.2-py3-none-any.whl (22 kB)
Installing collected packages: tzdata, sqlparse, asgiref, Django
Successfully installed Django-4.1.3 asgiref-3.5.2 sqlparse-0.4.3 tzdata-2022.6

[notice] A new release of pip available: 22.2.2 -> 22.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip

(djangoInit) C:\Users\User\Desktop\INFORMATIQUE\M2MIAGE\ARI2\djangoInit>
```

Désormais, votre installation est terminée !
vous pouvez vérifier votre version django en lançant :
"django-admin --version"

normalement la version est >= 4.0

Vous devez par la suite activer l'environnement virtuel à chaque fois que vous souhaitez travailler sur votre projet

Pour Linux/MacOs:

Nb: la structure devra ressembler à celle du Windows

"python3 --version"

"python3 -m venv djangoInit"

Accéder à djangoInit puis lancer :

"source ./bin/activate"

ensuite :

"python -m pip install Django"

```
Collecting sqlparse>=0.2.2
  Downloading sqlparse-0.4.3-py3-none-any.whl (42 kB)
    42.8/42.8 KB 38.9 MB/s eta 0:00:00
Installing collected packages: sqlparse, asgiref, Django
Successfully installed Django-4.1.3 asgiref-3.5.2 sqlparse-0.4.3
○ (djangoInit) zakaria.hairane.etu@b13p1:~/Desktop/django/djangoInit$
```

Création du projet

Par la suite, nous allons créer un projet.

Vous pouvez rester sur le répertoire djangoInit et lancer la commande suivante:

`"django-admin startproject myDjangoInitPorject"`

```

26/11/2022 15:29 <DIR> .
26/11/2022 15:29 <DIR> ..
26/11/2022 15:29          697 manage.py
26/11/2022 15:29 <DIR> myDjangoInitPorject
                1 fichier(s)          697 octets
                3 Rép(s) 122 286 649 344 octets libres

(djangoInit) C:\Users\User\Desktop\INFORMATIQUE\M2MIAGE\ARI2\djangoInit\myDjangoInitPorject>

26/11/2022 15:29 <DIR> .
26/11/2022 15:29 <DIR> ..
26/11/2022 15:29          431 asgi.py
26/11/2022 15:29          3 383 settings.py
26/11/2022 15:29          782 urls.py
26/11/2022 15:29          431 wsgi.py
26/11/2022 15:29           0 __init__.py
                5 fichier(s)          5 027 octets
                2 Rép(s) 122 280 038 400 octets libres

(djangoInit) C:\Users\User\Desktop\INFORMATIQUE\M2MIAGE\ARI2\djangoInit\myDjangoInitPorject\myDjangoInitPorject>

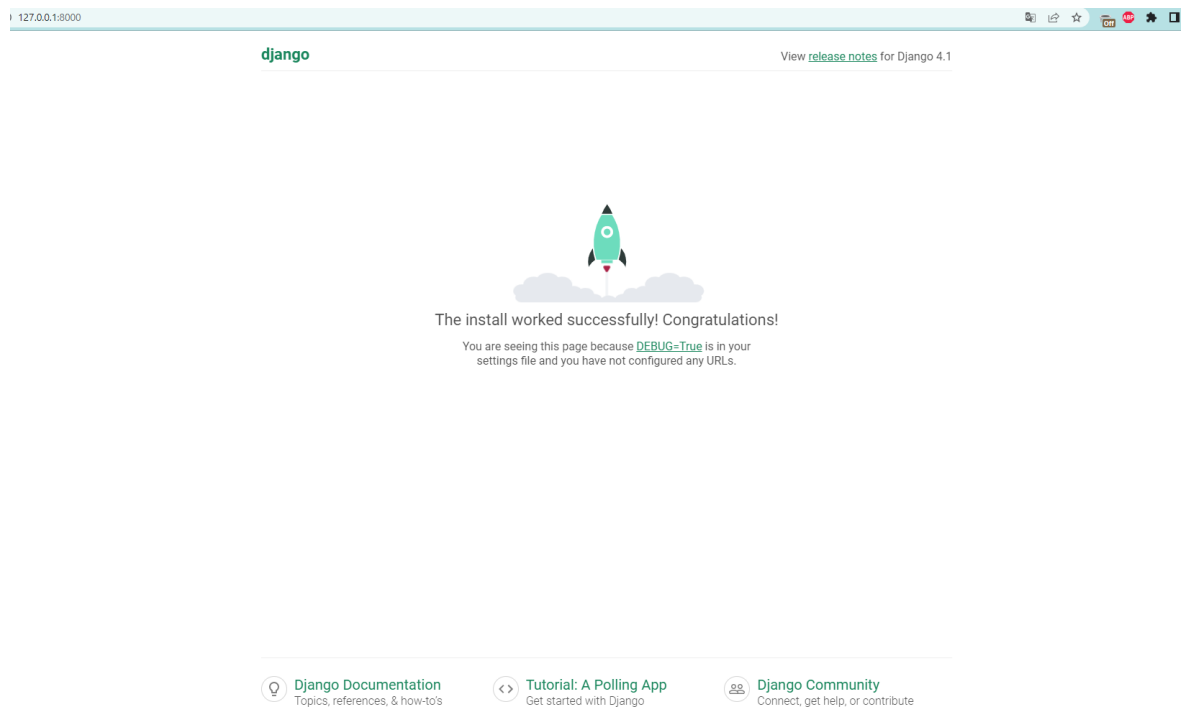
```

Vous remarquerez que Django s'est occupé de créer les répertoires et fichiers nécessaires. Ce premier répertoire myDjangoInitProject est la localisation de votre projet et c'est là où vous commencerez à créer vos applications.

Revenez à la racine et lancer la commande suivante pour lancer votre projet et voir à quoi cela ressemble sur votre navigateur :

`"python manage.py runserver"`

naviguez ensuite sur : <http://127.0.0.1:8000/>



Si tout est ok, vous devez avoir une interface pareil :)

Création d'une application

C'est quoi une application:

Une application est une application web qui a une signification spécifique dans votre projet, comme une page d'accueil, un formulaire de contact, ou une base de données de membres.

Dans ce tp, nous allons créer une app qui nous permet de lister et d'enregistrer des salariés dans une base de données.

Mais d'abord, créons une simple application Django qui affiche "Hello World !".

Restez sur la racine avec le manage.py et lancer la commande permettant de créer une application :

```
"manage.py startapp employees"
```

Votre répertoire employees devra ressembler à cela :

```
26/11/2022 16:04 <DIR> ..
26/11/2022 16:04      66 admin.py
26/11/2022 16:04     156 apps.py
26/11/2022 16:04 <DIR> migrations
26/11/2022 16:04      60 models.py
26/11/2022 16:04      63 tests.py
26/11/2022 16:04      66 views.py
26/11/2022 16:04       0 __init__.py
      6 fichier(s)      411 octets
      3 Rép(s) 126 712 930 304 octets libres

(djangoInit) C:\Users\User\Desktop\INFORMATIQUE\VM2MIMAGE\ARI2\djangoInit\myDjangoInitPorject\employees>
```

Ce sont tous des fichiers et des dossiers ayant une signification spécifique. Vous apprendrez à connaître la plupart d'entre eux plus tard dans ce tp.

Tout d'abord, jetez un coup d'œil au fichier views.py.

C'est là que nous rassemblons les informations dont nous avons besoin pour renvoyer une réponse correcte.

Les vues Django sont des fonctions Python qui prennent des requêtes http et renvoient des réponses http, comme des documents HTML.

Une page web qui utilise Django est pleine de vues avec différentes tâches et missions.

Remplacez le contenu de votre vue par celui-ci:

```
'''
from django.shortcuts import render
from django.http import HttpResponse

def index(request):

    return HttpResponse("Hello world!")
'''
```

Mais comment pouvons-nous exécuter la vue ? Eh bien, nous devons appeler la vue via une URL.

Créez un fichier nommé `urls.py` dans le même dossier que le fichier `views.py`, et tapez ce code dedans :

```
"""
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

Le fichier `urls.py` que vous venez de créer est spécifique à l'application des membres. Nous devons également faire un peu de routage dans le répertoire racine `myDjangoInnitProject`

Dans `urls.py` de votre `myDjangoInnitProject` ajouter l'import du module

```
"from django.urls import include"
```

Dans la liste `urlpatterns` ajouter le path pour `employees` comme ceci:

```
"path('employees/', include('employees.urls')),"
```

relancer le serveur puis naviguer à l'url : <http://127.0.0.1:8000/employees/>

Vous êtes désormais capable de voir votre helloworld dans sur la page d'accueil de votre application

Le résultat d'une vue doit être en HTML, et il doit être créé dans un modèle, alors faisons-le.

Créez un dossier `templates` dans le dossier `employees`, et créez un fichier HTML nommé `myfirsttemplate.html`

copier ce code sur votre template :

```

"""
<!DOCTYPE html>
<html>
<body>

<h1>Hello World!</h1>
<p>Welcome to my first Django project!</p>

</body>
</html>

```

Ensuite modifions la vue en remplaçant le contenu d'index par celui dessous:

```

"""
from django.http import HttpResponse
from django.template import loader

def index(request):
    template = loader.get_template('myfirsttemplate.html')

    return HttpResponse(template.render())

```

```

"""

```

Ensuite nous devons informer le projet qu'une nouvelle application a été créée pour qu'il le prenne en compte.

Dans le fichier settings.py de votre projet (et non pas de l'application) on ajoute la référence à l'application dans la liste INSTALLED_APPS

```

"""
'employees.apps.EmployeesConfig'

```

Ensuite lançons cette commande pour la migration:

```

"""
python manage.py migrate

```

Ensuite lançons le serveur



Hello World!

Welcome to my first Django project!

Cool, désormais, vous comprenez bien la relation entre une vue et une template, vous avez compris les deux derniers lettres du "MVT"

Un contrôle+s suffit pour reload votre projet par la suite ;)

Lorsque nous avons créé le projet Django, nous avons obtenu une base de données SQLite vide. Elle a été créée dans le dossier racine de myworld.

Nous allons utiliser cette base de données dans ce tp.

Les modèles

Dans models.py de votre application, nous allons créer un modèle d'employé.

```

"""
from django.db import models

class Employee(models.Model):
    nomComplet = models.CharField(max_length=255)
    departement = models.CharField(max_length=255)
    telephone = models.CharField(max_length=255)
"""

```

Maintenant, revenons à la racine du projet et lançons une migrations pour ce modèle

```

"""
python manage.py makemigrations employee
"""

```

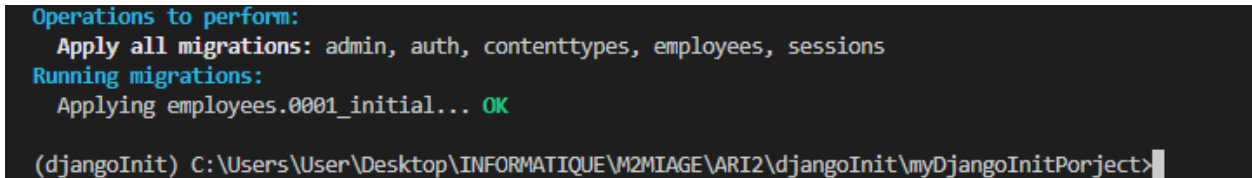
Django vous informera qu'il a effectué une migration pour employee modèle

Django crée un fichier avec toutes les nouvelles modifications et stocke le fichier dans le dossier /migrations/.

La prochaine fois que vous exécuterez `python manage.py migrate`, Django créera et exécutera une instruction SQL, basée sur le contenu du nouveau fichier dans le dossier migrations.

Exécutez la commande migrate :

```
python manage.py migrate
```



A terminal window with a dark background showing the output of the Django migrate command. The text is as follows:

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, employees, sessions
Running migrations:
  Applying employees.0001_initial... OK

(djangoInit) C:\Users\User\Desktop\INFORMATIQUE\MIAGE\ARI2\djangoInit\myDjangoInitProject>
```

Maintenant vous avez une table Employee dans votre base de données.

Il existe plusieurs moyens pour alimenter votre table, néanmoins nous choisirons celle qui montre le plus la force de django en utilisant sa plateforme d'administration intégré 😊

Mais avant, nous devons créer un SuperUser qui aura tous les rôles sur la plateforme d'administration :

```
python manage.py createsuperuser
```

Ensuite lancer le serveur et naviguer sur l'url : <http://127.0.0.1:8000/admin/> puis vous connecter.

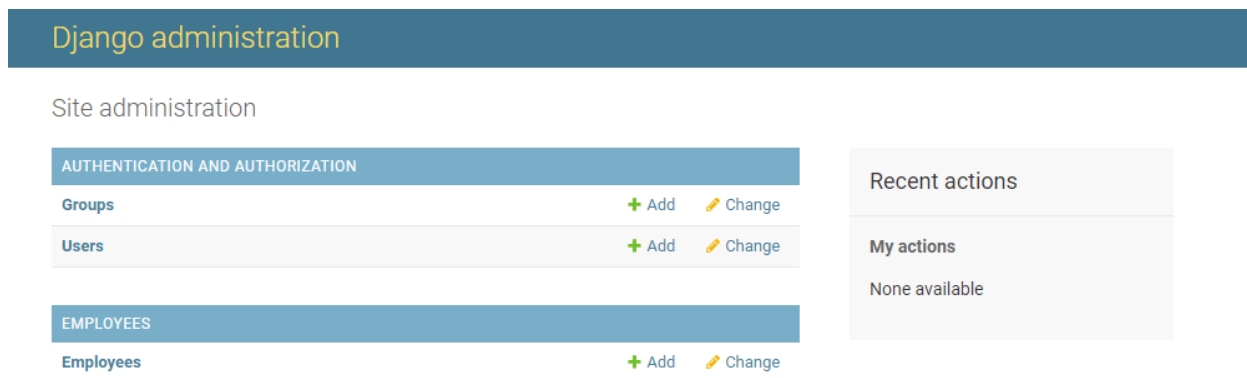
Site administration

AUTHENTICATION AND AUTHORIZATION			Recent actions	
Groups	+ Add	Change	My actions	
Users	+ Add	Change	None available	

Vous remarquerez l'absence de votre table, c'est totalement normal, nous devons enregistrer les models sur admin.py de votre application pour qu'il en prenne compte de leur existence.

- Tout d'abord, importer votre modèle :
 - `from .models import Employee`
- Ensuite coller le code suivant pour l'enregistrer :
 - `admin.site.register(Employee)`

Cette fois-ci en lançant votre serveur vous allez pouvoir voir votre table.



Rien de plus simple par la suite, cliquez sur add puis ajoutez 3 à 4 employée.

A ce stade, nous avons compris la signification des trois lettres "MVT", nous allons par la suite voir comment nous pouvons utiliser tous ces éléments pour créer un tableau simplifié de gestion des employées.

Tout d'abord ajoutons Bootstrap pour un peu de style, deux manières de faire:

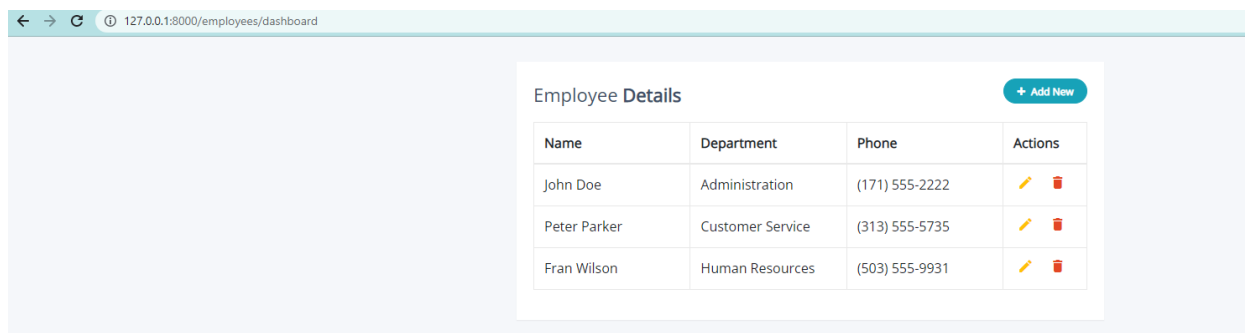
- En utilisant directement CDN (on utilisera cette manière pour le tp)
- Ou en utilisant les staticFiles de bootstrap, doc : <https://www.javatpoint.com/django-and-bootstrap>

Récupérez le template le code employeeTemplate.html du fichier source et créer dans votre dossier templates une nouvelle template employeeTemplate.html avec le code récupéré.

Vous savez certainement ce qui vous reste à faire pour que vous puissiez voir votre nouvelle template:

- Créer une nouvelle fonction de vue dans votre views.py qui load votre template

- Référencer votre template avec une url dans urls.py :
 - J'ai nommé ma fonction dans views dashboard, le path sera donc comme dessous
 - `path('dashboard', views.dashboard, name='dashboard'),`



Par la suite, nous allons faire en sorte d'afficher nos employées, en ajouter, modifier, supprimer ^^

Les templates tags :

`{% %}` : sont utilisés pour les tags (for/if/while ...)

`{{ }}` : sont utilisés pour les variables.

un exemple de création de table avec les tags :

'''

```
<table border="1">
{% for x in range(5)%} ou {% for x in employees%}
<tr>
<td>{{ x.nomComplet }}</td>
<td>{{ x.departement }}</td>
<td>{{ x.telephone }}</td>
</tr>
{% endfor %}

</table>
```

'''

Modifions maintenant la vue pour récupérer nos employees.







on doit :

- Importer le modele: `'from .models import Employee'`
- Appeler les employees dans notre fonction de vue : `'myEmployees = Employee.objects.all().values()'`

- Créer un context:
 - `'context = { 'myEmployees': myEmployees,}'`
- passer le context et la request dans le template.render de notre réponse :
 - `'return HttpResponse(template.render(context, request))'`

Après, vous devez vous inspirer de l'exemple de la table ci-dessus pour rendre votre `employeeTemplate.html` dynamique et afficher les employés de notre base de données.

Vous constaterez par la suite si tout est bien fait que le tableau comportera les données de votre base de données:

Employee Details				+ Add New
Name	Department	Phone	Actions	
Zakaria HAIRANE	Informatique	0700000000		
Omar AHMANI	Informatique	0600000000		
Sifax RAHMOUNE	Data	0700000000		

Par la suite nous allons configurer le reste des opérations pour ajouter, éditer et supprimer.

Ajouter un employee:

Commençons par créer un template `add.html`, vous trouverez le code également dans vos ressources.

Remarque : Django exige cette ligne dans le formulaire :

```
{% csrf_token %}
```

pour gérer les Cross sites dans les formulaires où la méthode est POST.

Ensuite nous allons ajouter une fonction dans notre vue nommé "add", cette fonction comme les autres aura un template qui est add.html, puis retournera pour l'instant un context vide et la "request"

Dans urls.py de votre application, ajouter le path pour le template add :

```
'path('add/', views.add, name='add'),'
```

Comme vous le constaterez l'action du formulaire est addRecord, cette action doit aussi figurer dans urls.py de votre application:

```
'path('add/addrecord/', views.addrecord, name='addrecord'),'
```

qui dit nouveau url, dit nouvelle fonction dans notre vue, nous allons créer la fonction addrecord dans notre vue, pour cela :

- Importez HttpResponseRedirect et reverse :

```
'from django.http import HttpResponseRedirect'
```

```
'from django.urls import reverse'
```

- il faut récupérer les POST du formulaire :
- `x = request.POST['nomCompleet']` (de même pour les deux paramètres restants)
- créer un objet Employee avec les paramètres récupéré :

```
'employee = Employee(nomCompleet= x, departement = y, telephone= z)'
```

- enregistrer l'objet dans la base de données: `employee.save()`
- revenir à votre dashboard :

```
'return HttpResponseRedirect(reverse('index'))'
```

Supprimer un employé:

Pour pouvoir supprimer un employee, nous avons besoin de son Id (django s'en occupe implicitement pas besoin que votre modèle comporte un champ ID).

Dans ce cas, il faut ajouter à la balise `<a>` qui sert de delete un href permettant de passer l'id de l'employeur en url, ex : `"delete/<id employé>"`

Les deux reflex qui devront suivre c'est de modifier ainsi votre `employee/urls.py` et votre `employee/views.py` pour gérer la suppression.

Attention : pour passer un paramètre dans un path on utilise `"votreChemin/< <type param> : <nom param> >"`

Dans `employee/views.py`, votre fonction `delete` prendra en paramètre en plus de la requête un id cette fois-ci.

Pour avoir un employée par son id : `'employee = Employee.objects.get(id=id)'`

Pour le supprimer on utilise la fonction `.delete()`

Ensuite on retourne une `HttpResponseRedirect` pour repartir sur notre dashboard des employés.

Modifier un employé:

Pour modifier un employé, il faudra s'inspirer à la fois de l'ajout et de la suppression d'un joueur.

Tips:

- On récupère son id au moment de la demande d'update.
- On crée une fonction `update` au niveau de notre `views.py` qui prend en paramètre une requête et un id
- Cette fonction :
 - Devra récupérer l'objet de la base à travers son id;
 - Créer une variable template en se basant sur une template qu'on va créer (`update.html`)
 - renvoie le context et la requête comme le cas pour notre Dashboard
- Ensuite nous ajouterons dans notre `employee/urls.py` le path pour l'update avec l'id en paramètre
- Dans notre `update.html` l'action de notre formulaire devra être un POST qui a comme action `'fonction/<id employé>'`
- Vous l'avez compris, oui, il faut créer une fonction `updaterecord` dans votre `views.py` et ajouter un path adéquat dans votre `urls.py`

Django Rest FrameWork

La logique :

Django rest-framework met à votre disposition des librairies qui permettent de créer un API de A à Z. Le rest-framework utilise une certaine logique qu'il vous faudra comprendre si vous désirez créer un API

Pour ce TP pour la simplicité nous allons créer une api en se basant sur VIEWSET pour avoir un CRUD de Posts par exemple

Installation:

```
"python -m pip install djangorestframework"
```

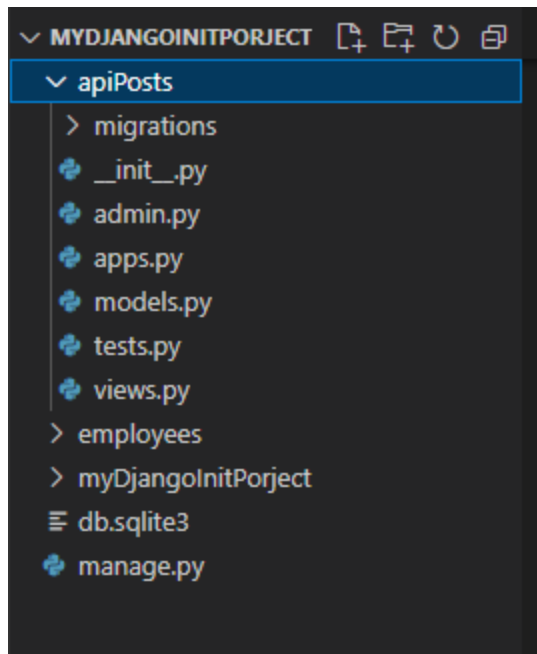
Création de l'api:

```
"""
```

```
python manage.py startapp apiPosts
```

```
"""
```

votre arborescence ressemblera à cela:



Dans INSTALLED_APPS de votre myDjangoInitProject.settings.py vous devez ajouter les trois références suivantes:

```
"""
```

```
'rest_framework',
```

```
'rest_framework.authtoken',
```

```
'apiPosts',
```

```
"""
```

- rest_framework: nous permettra de créer notre API
- rest_framework.authtoken: nous permettra d'ajouter une sécurité à notre API
- posts indique à Django d'utiliser notre application posts

Création du modèle Post:

Votre modèle contiendra un : title (CharField(max_length=255), body (TextField), create_on (DateTimeField(auto_now_add=True)) et une référence à un user : 'user = models.ForeignKey(User, on_delete=models.CASCADE)'

Nb: il faut importer le user avec 'from django.contrib.auth import get_user_model' et l'instancier avant votre modèle avec : 'User = get_user_model()'

Enregistrez Post dans votre apiPosts/admnis.py :

```
"""
```

```
from .models import Post
```

```
admin.site.register(Post)
```

```
"""
```

Ensuite nous lançons encore une fois une migration pour mettre à jour notre makemigrations :

```
"""
```

```
python manage.py makemigrations
```

```
python manage.py migrate
```

```
"""
```

Votre adminPannel prendra cette forme si tout va bien :

Home › Employees › Employees

Start typing to filter...

APIPOSTS

Posts [+ Add](#)

AUTH TOKEN

Tokens [+ Add](#)

AUTHENTICATION AND AUTHORIZATION

Groups [+ Add](#)

Users [+ Add](#)

EMPLOYEES

Employees [+ Add](#)

«

Select employee

Action:

☐ EMPLOYEE

☐ Employee object (

☐ Employee object (

☐ Employee object (

3 employees

Format d'échange de données:

L'échange de donnée entre le backend et le frontend se fait sous format texte structuré au format JSON.

Le rest-framework prévoit donc une conversion entre le format Django et un format compatible JSON. Cette conversion utilise la classe Serializer.

Créer un Serializer:

La première étape est donc de créer un serializer. Pour ce faire créer un fichier nommé serializers.py sous le dossier de l'application posts et ajouter ces lignes de code :

```
from rest_framework import serializers
from .models import Post
```

```
class PostSerializer(serializers.ModelSerializer):
```

```
    class Meta:
```

```
        model = Post
```

```
        fields = '__all__'
```

```
'''
```

Voilà, ce n'est pas plus compliquer ! En faite, le gros du travail est fait par la librairie rest-framework. Ici, il suffit de mentionner quel modèle et quels champs nous désirons utiliser pour l'API.

À noter qu'à la place de 'all' vous auriez pu spécifier le nom de chaque champ voulu
 fields = ['title', 'body']

Créer une vue:

Remplacez le contenu de votre views.py par celui dessous :

```
'''
```

```
from rest_framework import viewsets
```

```
from .models import Post
```

```
from .serializers import PostSerializer
```

```
class PostViewSet(viewsets.ModelViewSet):
```

```
    serializer_class = PostSerializer
```

```
    queryset = Post.objects.all()
```

```
'''
```

Encore une fois, vous avez une démonstration de la force de rest-framework. Il suffit de créer une view basé sur le 'ModelViewSet' et ensuite spécifier quel est le serializer et d'où proviendront les données.

Avec ce simple code vous aurez accès aux 6 actions toutes héritées du ModelViewSets et de ses 6 méthodes équivalentes : list, create, retrieve, update, partial_update et destroy

Créer le chemin de l'url:

Comme avant nous allons créer un urls.py dans le dossier de notre apiPosts

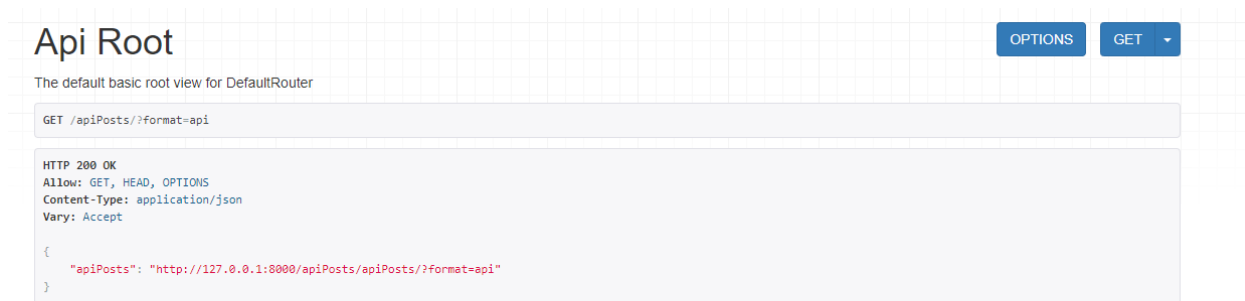
```
"""
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import PostViewSet
router = DefaultRouter()
router.register('apiPosts', PostViewSet, 'apiPosts')
urlpatterns = [
    path("", include(router.urls)),
]
```

Il vous reste désormais simplement l'action de relier le chemin de votre api dans votre urls.py du projet config myDjangoInitPorject

Tests:

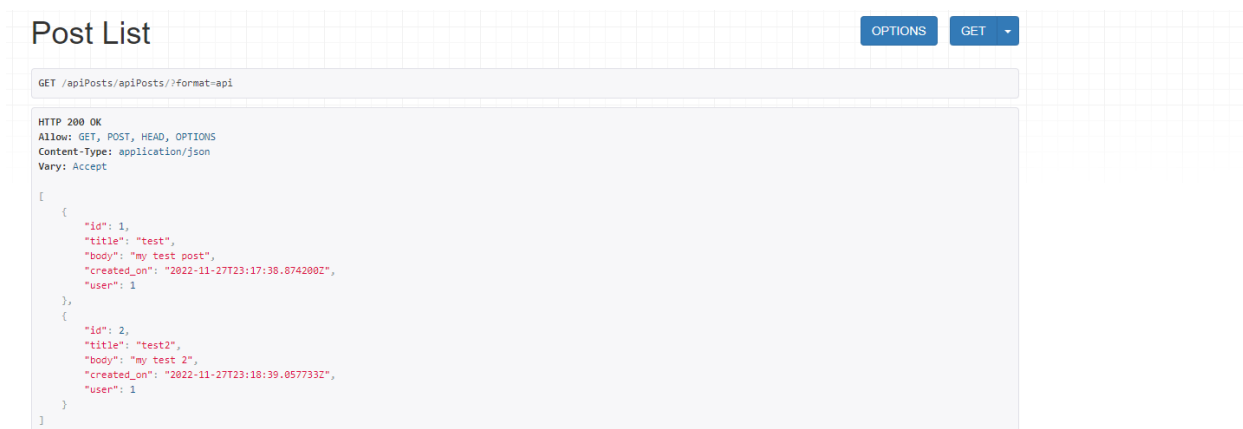
Visiter ensuite : <http://127.0.0.1:8000/apiPosts/>

Vous aurez une interface comme celle là :



Ensuite vous cliquer sur le lien à côté d'apiPosts pour tomber sur cette interface qui vous permettra par la suite de faire vos Get et Post

Exemple de get :



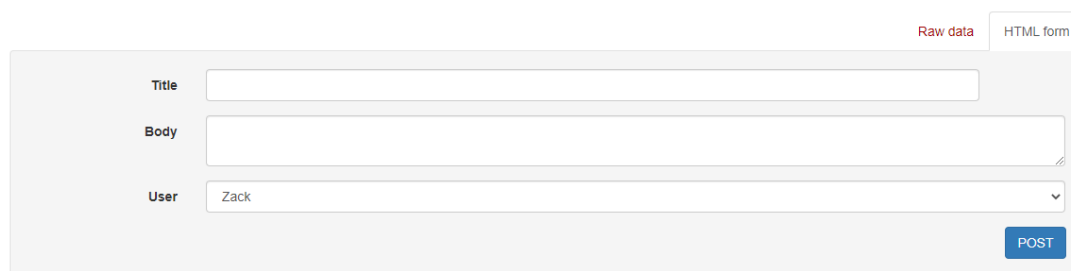
Post List

GET /apiPosts/apiPosts/?format=api

HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

```
[
  {
    "id": 1,
    "title": "test",
    "body": "my test post",
    "created_on": "2022-11-27T23:17:38.874200Z",
    "user": 1
  },
  {
    "id": 2,
    "title": "test2",
    "body": "my test 2",
    "created_on": "2022-11-27T23:18:39.057733Z",
    "user": 1
  }
]
```

Exemple de post:



Raw data HTML form

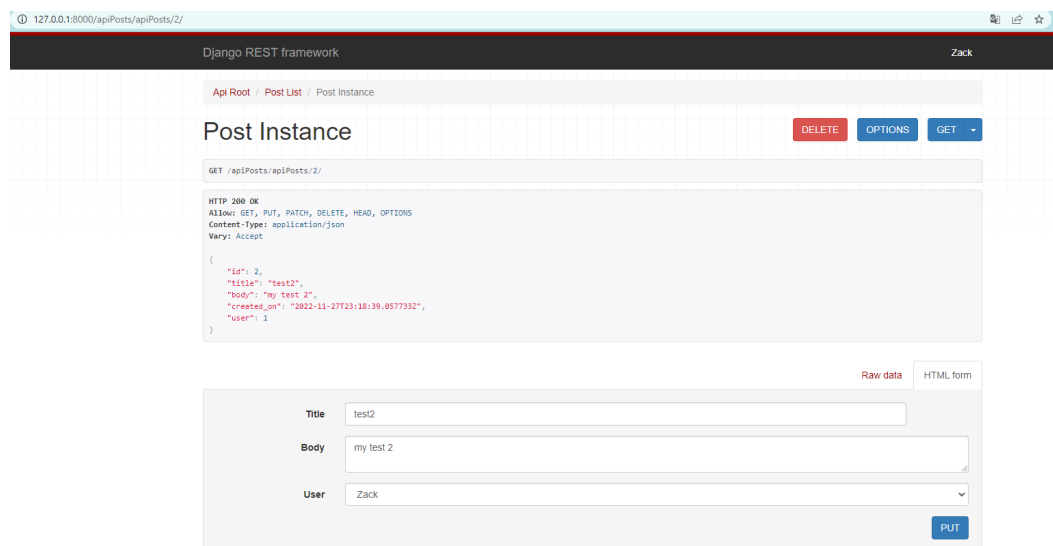
Title

Body

User

POST

Ensuite vous pouvez aller sur un id précis apiPosts/2 par exemple :



127.0.0.1:8000/apiPosts/apiPosts/2

Django REST framework

Api Root / Post List / Post Instance

Post Instance

DELETE OPTIONS GET

GET /apiPosts/apiPosts/2/

HTTP 200 OK
Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

```
{
  "id": 2,
  "title": "test2",
  "body": "my test 2",
  "created_on": "2022-11-27T23:18:39.057733Z",
  "user": 1
}
```

Raw data HTML form

Title

Body

User

PUT

Vous pouvez par la suite soit update le record soit le supprimer 😊

Aller plus loin:

Vous pouvez refaire l'api avec un ApiView au lieu d'un ViewSet ;) nous resterons là en cas de besoin !