LP Cours Python pour le Web 2020–2021

Introduction à Django

tutoriel Django

Exercice 1. Création environnement virtuel et installation sous linux Dans ce TD, vous allez vous initier au framework django basé sur python. Suivez les étapes :

1. Creation d'un environnement virtuel venv3 dans votre home :
   - `virtualenv -p python3 ~/venv3`
   - Si vous utilisez la distribution Python anaconda : `conda create -n myenv python=3.7 python`
   - l'initialiser : `source ~/venv3/bin/activate` ou avac conda : `conda activate myenv`
   - Installation de django. Ne pas oublier de configurer les variables d'environnement du proxy : `pip install django`
   - vérification de la version de django installée : `django-admin.py --version`
   - Création d'un nouveau projet nommé GestionTaches : `django-admin.py startproject GestionTaches`
   - Observez la hiérachie des fichiers créés avec la commande unix `tree`
2. Test du bon fonctionnement en lançant le serveur web de django :

```
cd GestionTaches
./manage.py migrate
./manage.py runserver
```

3. vous pouvez ajouter à cette commande un numéro de port, par défaut celui-ci est 8000
4. lancer votre navigateur favori avec l'url `http://localhost:8000` Normalement l'écran d'accueil de django doit apparaître.

Exercice 2. Création d'une application nommée les Taches

- Les Taches est une simple application de gestion des taches que vous avez à réaliser avec une date de début et une date de fin souhaitée.
  Placez-vous dans le répertoire GestionTaches et créez l'application par la commande :

```
./manage.py startapp lesTaches
```

Vérifier ensuite la structure du répertoire `GestionTaches` à l'aide la commande tree et éditer le fichier `settings.py`, configurer l'internationalisation et ajouter l'application lesTaches aux applications installées
En particulier modifier ou ajouter les lignes suivantes si besoin :

```
LANGUAGE_CODE = 'fr-fr'
TIME_ZONE = 'Europe/Paris'
USE_I18N = True
USE_L10N = True
USE_TZ = True
DATE_INPUT_FORMATS = ('%d/%m/%Y','%Y-%m-%d')
```

Exercice 3. les URL et les Vues

1. Editer le fichier `GestionTaches/urls.py` et ajouter la ligne suivante dans urlpatterns :

   ```
   path('lesTaches/',include('lesTaches.urls'))
   ```

   (Vous aurez un petit `import` à ajouter ...)
2. anciennenment, on utilisait :

   ```
   url(r'^lesTaches/',include('lesTaches.urls'))
   ```

   mais la fonction url va être dépréciée et remplacée par : `django.urls.re_path()` pour la gestion d'url plus complexes à l'aide d'expressions régulières.
3. Créer ensuite dans le répertoire `GestionTaches/lesTaches` le fichier `urls.py` de votre application

   ```
   from django.urls import path
   from . import  views
   urlpatterns=[
     path('home',views.home,name='home'),
   ]
   ```

4. Dans le répertoire `GestionTaches/lesTaches` il va falloir ajouter dans le fichier `views.py` la fonction home() :

   ```
   from django.http import HttpResponse
   def home(request):
     return HttpResponse('bonjour à tous')
   ```

5. Relancer le serveur (si vous l'avez stoppé) et dans le navigateur, pointez l'URL `http://localhost:8000/lesTaches/home`. Le message 'Bonjour à tous' devrait apparaître.

6. Compliquons légèrement les choses pour pouvoir router des urls du type `http:/localhost:8000/lesTaches/home/toto`.
   Modifions tout d'abord le fichier `urls.py` de lesTaches de la façon suivante :

```
1    path('home/<param>',views.home,name='home'),
```

   Ceci signifie que les urls du type `http:/localhost:8000/lesTaches/home/toto` seront routées vers notre vue.
7. Modifier la fonction home() avec le profil suivant (à vous de la compléter) :

```
1 from django.http import HttpResponse
2 def home(request,name):
3   return HtppResponse(...)
```

   du coup les urls du type
   `http:/localhost:8000/home/lesTaches/toto` seront routées vers notre vue
8. Puis tester différentes urls en modifiant le paramètre fourni à path.
9. Comment faire en sorte que l'URL `http://localhost:8000/lesTaches/home/` soit également routée ?

Exercice 4. Le modele de «lesTaches» la class Task

1. Une tâche sera définie par :
   - un nom
   - une description
   - une date de création
   - la date à laquelle la tache devra être terminée (due_date)
   - la date à laquelle vous aimeriez vous mettre à travailler sur cette tache (schedule_date)
   - un booleen closed qui indique si la tâche est terminée ou pas
2. Pour l'instant, dans le répertoire de votre application, complétez le fichier `models.py` dans lequel vous définirez la classe Task héritant de la classe models.Model de django.db

```
1 from django.db import models
2
3 class Task(models.Model):
4   name = models.CharField(max_length=250)
5   description = models.TextField()
6   created_date = models.DateField(auto_now_add=True)
7   closed = models.BooleanField(default=false)
```

```
 8
 9
10    def __str__(self):
11      return self.name
12
13    # classe à compléter
```

3. ajoutez `lesTaches` à la liste `INSTALLED_APPS` dans le fichier `settings.py`
4. Puis utilisez la commande manage.py pour prendre en compte ce nouveau modèle et observer le SQL correspondant :

```
    python manage.py makemigrations lesTaches
    python manage.py sqlmigrate lesTaches 0001
    python manage.py migrate
```

5. Enfin, placez-vous dans le shell de django :

```
  python manage.py shell
```

   Et effectuez-y les commandes suivantes :

```
1 >>> from lesTaches.models import Task # importer la
    class Task
2 >>> Task.objects.all() # lister la liste des taches cré
    ées
3 []
```

   pour l'instant il n'y a aucune tache enregistrée

6. Créons à présent une tâche :

```
1 >>> tache=Task()
2 >>> tache.name='une premiere tache'
3 >>> tache.description='la première mais pas la derniere
    '
4 >>> #  on enregistre
5 >>> tache.save()
```

7. Vérifions qu'elle est bien enregistrée

```
1 >>> tache=Task.objects.all()[0]
```

8. Vérifiez que les champs que vous avez renseignés sont bien présents.
9. Créez une tache supplémentaire et testez d'autres filtres que all() : filter, exclude, get , order_by,etc..
10. Ces filtres correspondent à des requêtes SQL. Par exemple Task.objects.all() génère :

```
1 select * from Task ;
```

Task.objects.filter(due_date_lte='2020-01-01') génère la requête :

```
1 select * from Task WHERE due_date<='2020-01-01';
```

Exercice 5. Utilisation de l'interface d'administration

1. L'interface administration de django est une application au même titre que celle que vous êtes en train de créer. Vous allez créer d'abord un compte d'administrateur avec un mot de passe qui vous permettra de rentrer dans le mode administrateur par la commande

   ```
   ./manage.py createsuperuser
   ```

   Il vous sera demandé un nom de login une adresse electronique et un mot de passe qu'il faudra confirmer.

2. vérifier que `django.contrib.admin` est présent dans la partie `INSTALLED_APPS` de `settings.py` et vérifier que la ligne

   ```
   1 from django.contrib import admin
   ```

   est présente dans `GestionTaches/urls.py`
   et que `urlpatterns` contient bien

   ```
   1  urlpatterns = [
   2  url(r'^ admin',include(admin.site.urls))
   3  // ou
   4  path('admin', adim.site.urls)
   5  ../..
   6  ]
   ```

   et vous pourrez avec l'URL `http://localhost:8000/admin` accéder à l'administration de la base de données.

3. Pour ajouter le modele Task à l'administration il faut modifier le fichier `admin.py` de l'application de la manière suivante :

   ```
   1 from django.contrib import admin
   2 from lesTaches.models import Task
   3 admin.site.register(Task)
   ```

4. Complétez le modèle de la classe Task si nécéssaire. Que faut-il faire pour que django prenne en compte nos modifications ? (migrations, ...)
5. L'ennui est que les taches ne présentent que leur nom dans admin. On souhaiterait voir figurer également la due_date et la schedule_date. Pour ce faire, on modifie le fichier `admin.py` en lui ajoutant une classe héritant de admin.modelAdmin

```python
from django.contrib import admin
from lesTaches.models import Task

class TaskAdmin(admin.ModelAdmin):
  list_display=('name','schedule_date','due_date')

admin.site.register(Task,TaskAdmin)
```

6. Modifier l'affichage du champ due_date en fonction de la date du jour :
   — vert si la tache est à réaliser dans plus d'une semaine
   — orange si la tache est à réaliser dans la semaine e
   — rouge sinon

   Pour cela, vous ajouterez la méthode suivante dans la class Task

```python
def colored_due_date(self):
     due_date=django_date(self.due_date,"d F Y")
    # definition de la variable color à faire en
       fonction de la due_date
  return format_html("<span style=color:%s>%s</span>"
     %(color_due_date))
```

7. Ajoutez l'affichage de `colored_due_date` dans l'interface admin.

Exercice 6. les bases de l'interface client

1. Pour commencer, présentons une simple liste des taches en utilisant les templates dans le fichier `lesTaches/views.py` ajoutons :

```python
from lesTaches.models import Task # import de la class
   Task

def task_listing(request):
  from django.template import Template,Context
  objets=Task.objects.all().order_by('due_date')
  template=Template('{%for elem in objets %} {{elem}} <
     br />{%endfor%}')
  print(str(template))
  context=Context({'objets':objets})
  print(str(template.render(context)))
  return HttpResponse(template.render(context))
```

2. Pour que vous puissiez visualiser votre liste il faut modifier le fichier `lesTaches/urls.py`

```
1    path('listing', views.task_listing,name="listing"),
```

3. Tester l'URL
`http://localhost:8000/lesTaches/listing` si votre serveur est toujours actif.
4. Cette solution, si elle est operationnelle est un peu frustre nous allons l'améliorer en utilisant les templates. Commençons par créer un répertoire contenant les fichiers templates de notre application dans le répertoire.

```
cd lesTaches
mkdir  templates
```

5. Maintenant, il ne vous reste plus qu'à écrire le fichier template `list.html` dans le répertoire `lesTaches/templates/` Pour utiliser le fichier `list.html`, on modifiera la vue en utilisant la fonction render() comme suit :

```
1 def task_listing(request):
2   objets=Task.objects.all().order_by('due_date')
3   return render(request,template_name='list.html',
       context={'objets':objets})
```

avec list.html :

```
1   {% block content %}
2   <h1>Liste des tâches</h1>
3   <ul>
4     {% for elem in objects %}
5       <li>{{elem}}</li>
6     {% endfor %}
7   </ul>
8   {% endblock %}
```

6. Tester et améliorer la présentation de votre liste en utilisant un fichier css ou mieux en utilisant Twitter BootStrap.