Django

L'objectif de ce TP est d'utiliser un logiciel populaire, Django, qui fait ORM mais permet aussi de faire des sites complets.

1 Premier site

1.1 Installation

On peut installer django via pip (avec ou sans environnement virtuel).

```
# pour créer un environnement virtuel nommé mon_env
virtualvenv mon_env

# pour utiliser un environnement virtuel nommé mon_env_existant
source mon_env_existant

# pour installer django
pip install django django_extensions
```

1.2 Création de site

Une fois que Django est installé vous pouvez créer un site nommé `TPdjango` avec :

```
django-admin startproject TPdjango
```

1.3 Lancement de django

On peut lancer le site que l'on vient de créer avec :

```
cd TPdjango
python manage.py runserver
```

et consulter à l'URL `http://127.0.0.1:8000/` que django marche.

1.4 Dernière étape de l'installation

En Django, un site est composé d'une ou plusieurs applications. Par défaut, Django créé automatiquement une application du même nom que le site (c'est pourquoi il y a un dossier `TPdjango` dans le dossier `TPdjango`) mais ce n'est pas idéal de l'utiliser directement. La dernière étape consiste donc à créer une application que l'on modifiera ensuite.
Comme notre application manipulera des films on va l'appeler simplement base_film ce que l'on peut faire de la façon suivante :

```
django-admin startapp base_film
```

Si vous avez suivi toutes mes indications, le dossier TPdjango devrait contenir le fichier `manage.py` et les dossiers `base_film` et `TPdjango`.
Attention, il reste une dernière étape nécessaire pour que la suite fonctionne correctement, l'application `base_film` du site `TPdjango` a bien été créée mais il faut prévenir django que nous allons l'utiliser, pour cela il faut modifier le fichier `TPdjango/settings.py` et ajouter à la liste `INSTALLED_APPS` notre application en rajoutant `'base_film.apps.BaseFilmConfig'`.

2 Créer une page web simpliste

Pour créer une page web dans Django il faut à la fois créer une fonction qui renvoie le contenu de la page web et indiquer quelle URL pointe vers quelle fonction. Pour la fonction, on la met dans le fichier `base_film/views.py`. Voici un exemple fonction très simple qui affiche juste "Hello, world.".

```
from django.http import HttpResponse

def helloworld(request):
    return HttpResponse("Hello, world.")
```

Pour que Django sache quelle URL pointe vers quelle fonction, on doit lui indiquer. Pour faire cela simplement on va modifier le fichier `TPdjango/urls.py` en lui quelle url pointe vers quelle fonction. Pour dire que l'url "/hello" pointe vers la fonction, "helloworld" de views dans le fichier `base_film/urls.py`, on peut juste ajouter au tableau `url_pattern` :

```
    path('hello', base_film.views.helloworld),
```

mais pour que cela marche il faut aussi importer le fichier views, ce que l'on fait en ajoutant :

```
import base_film.views
```

3 Création d'une classe d'entité films

3.1 Définition de l'entité

On va maintenant créer une classe d'entité pour représenter des films. Voici un exemple d'entité tiré du tutoriel django[1]

```
class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField("date published")
```

En mettant ce code dans le fichier `base_film/models.py`, on indique que l'on veut créer une entité `Question` avec deux attributs `question_text` et `pub_date`. Adapter ce code pour faire un entité `Film` qui a des attributs représentants le titre du film et son année de sortie. Une liste de types Django peut se trouver ici`https://www.geeksforgeeks.org/django-model-data-types-and-fields-list/?ref=lbp`.

3.2 Répercutions des modifications des entités sur la base de données

La répercussion d'une modification des classes entités que vous avez créées dans `models.py` sur le schéma de la base de données n'est pas automatique, il faut demander à django de créer puis d'appliquer les *migrations*. Une migration est un petit script qui modifie le schéma d'une base de données et applique aux données le traitement nécessaire. On peut fabriquer les migrations avec la commande suivante :

```
python manage.py makemigrations
```

Si vous avez correctement fait les choses, cette commande devrait créer un fichier `base_film/migrations/0001_initial.py`. On peut ensuite appliquer cette migration avec la commande suivante :

```
python manage.py migrate
```

[1] `https://docs.djangoproject.com/en/4.2/intro/tutorial02/#creating-models`

Cette commande va appliquer votre migration ainsi que d'autres, créées pour les besoins de django. Quelles sont les tables créées dans la base de données `db.sqlite3` ? On rappelle que l'utilitaire par défaut pour consulter les bases de données SQLite3 est la commande sqlite3 et que l'on peut voir le schéma avec la commande `.fullschema`. Selon vous, quelle est la ou les tables utilisées pour l'entité Film ? Quels champs ont cette/ces tables ?

3.3 Ajouter des films via l'interface admin

Vous n'avez a priori pas (encore !) les connaissances pour faire des formulaires web qui permettent d'ajouter des films facilement. On va utiliser une fonctionnalité de Django qui permet d'avoir très facilement une interface fonctionnelle pour éditer des entités avec le mode admin. Pour utiliser ce mode admin (pour administration) il faut un compte admin que l'on peut créer avec la commande suivante :

```
# Noter que le site que l'on créé est un site jouet, peu importe les
# logins, mot de passe et email que vous mettez.
python manage.py createsuperuser
```

Il est maintenant possible d'accéder à l'interface admin qui se trouve à l'adresse `http://127.0.0.1:8000/admin` (il faut évidemment que le serveur soit lancé). Pour l'instant, cette interface est très vide, car nous n'avons pas dit à Django de pouvoir éditer l'entité `Film` depuis l'interface admin. Pour corriger cela, il faut éditer le fichier `base_film/admin.py` et rajouter le code suivant :

```
from .models import Film

admin.site.register(Film)
```

Ajouter quelques films dans la base de données. Constater que le nom des films dans l'interface administrateur n'est pas très joli puis changer la façon dont django calcule ce nom en définissant une fonction `def __str__(self):` dans la classe `Film`. On pourra prendre la convention d'afficher un film par son titre suivi de l'année de sortie entre parenthèse.

4 Manipulation des données en python, objets et Querysets

Dans cette section nous allons manipuler les objets django depuis python. Pour cela il sera pratique de travailler avec un shell python qui arrive à interagir avec django. Pour cela, on peut lancer un shell avec la commande `python manage.py shell`. Une fois dans le shell, on importera la classe Film avec la commande python suivante `from base_film.models import Film`.

4.1 Création d'objets depuis python

En django, il est possible de créer des objets directement depuis python. Pour cela on peut faire `totoro = Film()` qui créé un nouvel objet `Film` en mémoire temporaire (pas dans la base d edonnées) et dont l'on peut modifier les champs en faisant `totoro.titre = "Totoro"` ou `totoro.annee_sortie = 1988`. Si l'on veut que cet objet soit stocké de façon pérenne, il faut l'enregistrer dans la base de donnée en faisant `totoro.save()`. Créer et enregistrer quelques films avec le shell python.

4.2 Requêtage simple

En Django la manière de requêter la données passe par les QuerySet, on évite en général d'écrire directement du SQL, on laisse le système calculer la requête à faire au système. Par exemple, pour avoir la liste des films on peut faire

```
Film.objects.all()
```

Pour récupérer seulement les films fait en 1977 on peut écrire `Film.objects.filter(annee_sortie=1997)`.
Tous ces queryset retournent une liste mais il est aussi possible de récupérer un objet directement en utilisant une clef primaire, par exemple en faisant `Film.objects.get(id=1)`.
Essayer de manipuler vos données.

4.3 Requêtage avancé

La syntaxe des queryset devant être compatible avec python elle est parfois un peu étrange, par exemple pour avoir la liste des films sortis avant l'an 2000 il faut écrire `Film.objects.filter(annee_sortie__lt=2000)`, Django transformant le `__lt` en comparaison. La documentation sur les queryset se trouve à cette adresse `https://docs.djangoproject.com/en/4.2/ref/models/querysets/`.

5 Amélioration de notre schéma

Nous voulons maintenant un schéma de données bien plus développé, il faut pouvoir représenter des films, des personnes, le fait que telle personne a joué telle rôle dans tel film, que telle liste de personnes a réalisé le film, avoir des utilisateurs qui notent des films, etc.
Implémenter les changements nécessaires pour représenter ces données puis remplir la base de données pour que chaque entité ait quelques exemples.

6 Des vues qui calculent

Faire de nouvelles vues qui calculent les meilleurs films ou les films dans lesquels tel ou telle actrice a joué.
Note : il est possible en django de créer des vues qui prennent des paramètres. Par exemple, si l'on met l'entrée `path("person/<int:personid>", views.person_details)`, dans `url_pattern`, cela appellera la fonction `person_details` avec l'argument personid=42 quand le navigateur pointe sur l'url "film/42". Faire une vue qui liste les films dans lesquels une personne a joué sous la forme d'une succession de phrases "<nom acteur ou actrice> a joué dans le film <nom du film>".
Voir la documentation sur les URLs django `https://docs.djangoproject.com/fr/4.2/topics/http/urls/`