TP 1 : Technologies Web II [côté serveur]

Premiers pas avec Django

Objectifs du TP

Le but de ce TP est d'installer puis de vous familiariser avec le framework Django pour la programmation d'une première application web (côté serveur).

Consigne

Lisez bien l'énoncé, il contient beaucoup d'informations et chaque mot est utile!
N'hésitez pas à chercher des informations sur les fonctions sur Internet, un bon informaticien doit savoir chercher des informations en ligne. Vous retrouverez notamment la documentation Django ici : `https://docs.djangoproject.com/en/4.1/`

Attention aux "copier-coller" du PDF vers votre code, des différences d'espace et d'indentation peuvent avoir lieu.
Il est donc préférable d'écrire vous même les commandes et le code demandés (c'est relativement court).

Installation

Anaconda + Python

Comme vous allez travailler en langage Python, une bonne pratique est d'installer la distribution Anaconda qui inclut Python, mais aussi beaucoup d'autres outils qui vous seront très utiles pour vos projets futures (IDE, Prompt, Gestion de paquets, etc.).
Rendez-vous sur le site `https://www.anaconda.com` et téléchargez l'installateur graphique correspondant à votre système d'exploitation.
Suivez- les instructions pour installer Anaconda.

Une fois Anaconda installé :

1. Vous pouvez lancer un terminal (**Anaconda prompt sous Windows**)
2. Vérifiez la bonne installation de Python en ouvrant l'interpréteur Python via la commande :

   ```
   $ python
   ```

   Vérifiez que vous utilisez une version 3 de Python
3. Vous pouvez fermer l'interpréteur Python via la commande :

   ```
   >> exit()
   ```

4. Vérifiez la bonne installation d'Anaconda en tapant la commande suivante dans le terminal :

   ```
   $ conda --version
   ```

   Si l'installation s'est bien déroulée, la version correspondante d'Anaconda s'affiche.
5. Un des avantages d'Anaconda est la possibilité d'installer des environnements virtuels. Les environnement virtuels permettent de travailler dans un environnement dédié et d'installer des paquets spécifiques sans risque pour le système d'exploitation. Créez un environnement virtuel avec la commande suivante :

```
$ conda create -n djangoenv
```

Cette commande crée un environnement s'appelant "djangoenv" (vous pouvez bien entendu changer ce nom si vous le souhaitez).

6. Une fois l'environnement créé, il faut l'activer via la commande :

```
$ conda activate djangoenv
```

Vous devriez alors voir à gauche de l'interpréteur de commande le nom de l'environnement entre parenthèses : `(djangoenv)`.

Django

1. Vérifiez que l'environnement virtuel est bien activé, puis installez la Django avec la commande :

```
$ conda install -c anaconda django
```

2. Si dans la liste des paquets installés, la version de Django n'est pas la 4.1, alors mettez à jour Django avec la commande :

```
$ conda update django
```

3. Vérifiez la bonne installation en ouvrant un interpréteur Python via la commande :

```
$ python
```

Dans l'interpréteur, ajoutez le code python suivant pour importer django :

```
>> import django
>> print(django.get_version())
```

La version 4.1 de Django doit s'afficher. Vous pouvez alors fermer l'interpréteur avec la commande :

```
>> exit()
```

4. Il est également possible d'obtenir directement la version de Django via la commande :

```
$ python -m django --version
```

1 Premier pas avec Django

1.1 Création d'un projet Django

Pour créer un projet, il est nécessaire d'exécuter du code configurant un ensemble de réglages particuliers comprenant la configuration d'une base de données ainsi que des options spécifiques à Django ou à une application.
Pour cela, depuis le terminal, déplacez-vous dans votre répertoire de travail à l'aide de la commande `cd`, puis exécutez la commande suivante :

```
$ django-admin startproject ensisa_project
```

`ensisa_project` correspond au nom de votre projet (vous bien entendu en choisir un autre).
Vous devez éviter de nommer vos projets en utilisant des noms réservés de Python ou des noms de composants de Django. Cela signifie en particulier que vous devez éviter d'utiliser des noms comme django (qui entrerait en conflit avec Django lui-même) ou test (qui entrerait en conflit avec un composant intégré de Python).

La commande précédente a créé la hiérarchie de répertoires et fichiers suivante :

```
ensisa_project/
  manage.py
  ensisa_project/
    __init__.py
    settings.py
    urls.py
    asgi.py
    wsgi.py
```

Ces fichiers et repertoires sont décrits ci-après :

— **ensisa_project/** : le répertoire racine contenant votre projet.
— **manage.py** : un utilitaire en ligne de commande permettant d'interagir avec vote projet.
— Le sous-repertoire **ensisa_project/** correspond au paquet Python de votre projet. Il sera à utiliser lorsque vous souhaiterez importer ce qu'il contient.
— **ensisa_project/__init.py__** : un fichier qui indique à Python que ce répertoire doit être considéré comme un paquet (permettant notamment de structurer plusieurs modules en un seul paquet).
— **ensisa_project/settings.py** : Le script Python permettant le réglage et la configuration de votre projet Django.
— **ensisa_project/urls.py** : Le fichier Python permettant de déclarer les différentes URL de votre projet. C'est en quelque sorte la "table des matières" de votre projet.
— **ensisa_project/asgi.py** et **ensisa_project/wsgi.py** : Deux fichiers importants pour déployer votre projet selon deux types de serveurs web. Nous ne les utiliserons pas dans ce TD.

1.2 Serveur Web de développement

Avec Django, un serveur Web léger entièrement écrit en Python est disponible pour une exécution locale de votre projet. Il permet notamment de développer et tester rapidement votre projet sans avoir à vous occuper de la configuration d'un serveur de production (comme Apache), tant que vous restez en développement local de votre projet.
Ce serveur est fait uniquement pour tester votre travail durant le développement. N'utilisez pas ce serveur pour la mise en production finale de votre application Web.

Pour lancer le serveur de développement et vérifier que votre projet Django fonctionne, déplacez-vous dans votre répertoire **ensisa_project**, puis lancez la commande suivante :

```
$ python manage.py runserver
```

Vous verrez alors ce message s'afficher, indiquant le lancement du serveur Web :

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply
the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
November 01, 2022 - 14:42:51
Django version 4.1, using settings 'poll_website.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Vous pouvez pour le moment ignorer l'avertissement concernant les migrations de la base de données.

Maintenant que le serveur tourne en local, vous pouvez ouvrir un navigateur et aller à cette adresse : `http://127.0.0.1:8000` (ou `http://localhost:8000`). Vous verrez alors la page suivante :

The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

Pour arrêter le serveur, appuyez sur **CTRL+C**.

Par défaut, la commande **runserver** démarre le serveur de développement sur l'IP locale sur le port 8000. Vous pouvez modifier ce port (par exemple 8080) avec la commande suivante :

```
$ python manage.py runserver 8080
```

Testez à présent la modification en accédant à la page.

Pour information, le serveur de développement recharge automatiquement le code Python lors de chaque requête, si nécessaire. Il n'est alors pas nécéssaire de redémarrer le serveur pour que des changements de code soient pris en compte. Néanmoins, si de nouveaux fichiers sont ajoutés à votre projet, ceux-ci ne seront pris en compte qu'après un redémarrage du serveur.

1.3 Création d'une application

Maintenant que le projet est en place, vous allez pouvoir créer votre première application.

Attention, ne confondez pas Projet et Application ! Une application est une application Web qui fait quelque chose (par exemple une application de sondage). Un projet est un ensemble de réglages et d'applications pour un site Web particulier. Un projet peut contenir plusieurs applications. Une application peut apparaître dans plusieurs projets. On ne peut pas utiliser une application sans projet.

La première application que vous allez créer est une une application de sondage. Assurez-vous d'être dans votre répertoire **ensisa_project/** (là ou le script Python `manage.py` est présent) et exécutez la commande suivante :

```
$ python manage.py startapp polls
```

Cela a créé le dossier **polls/** qi est structuré de la manière suivante :

```
polls/
  __init__.py
  admin.py
  apps.py
  migrations/
    __init__.py
  models.py
  tests.py
  views.py
```

En particulier, on retrouve :

- **models.py** : fichier dans lequel vous écrirez, comme son nom l'indique, les modèles de votre application (au sens MVC).
- **views.py** : fichier dans lequel vous écrirez, **comme son nom ne l'indique pas**, les contrôleurs de votre application (au sens MVC).
- **tests.py** : fichier dans lequel vous écrirez, comme son nom l'indique, vos tests (unitaires et d'intégration) de votre application (nous verrons cela dans un prochain TD).

1.4 Ecriture d'une première vue (au sens contrôleur)

Comme indiqué juste au dessus, dans Django, les vues sont en fait les contrôleur au sens du modèle MVC. Elles agissent comme un chef d'orchestre qui définit le comportement souhaité de l'application.

Pour écrire votre première vue, ouvrez le fichier **polls/views.py** et placez-y le code suivant :

```
from django.http import HttpResponse

def welcome(request):
  return HttpResponse("Bienvenue dans l'application de sondage !")
```

Voyons ce que contient cette première vue :

— La première ligne permet d'importer le module HttpResponse pour pouvoir répondre à une requête.

— La suite est une fonction **welcome** qui sera appelée. Elle retourne une réponse http avec le texte choisi.

Pour appeler cette vue, il faut l'associer à une URL. La configuration des URL se fait à l'aide d'un fichier dédié communément appelé un *URLconf*.

Placez-vous dans le répertoire **polls/** et créez un nouveau fichier nommé **urls.py**. A présent la hiérarchie de votre répertoire devrait ressembler à ceci :

```
polls/
  __init__.py
  admin.py
  apps.py
  migrations/
    __init__.py
  models.py
  tests.py
  urls.py
  views.py
```

Ouvrez le fichier **urls.py** et insérez le code suivant :

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

urlpatterns = [
  path('', views.welcome, name='welcome-view'),
]
```

Vous avez donc mis en place les règles de routage (définition des routes) au sein de l'application de sondage.

L'étape suivante consiste à en faire de même au niveau du projet. Chaque projet Django contient également un fichier **urls.py**. Cependant, plutôt que de définir les mêmes routes depuis le projet, Django permet simplement de faire en sorte que le projet délègue une partie de la gestion des règles de routage à différentes applications. Ainsi, chaque application sera responsable du routage interne des URL la concernant.

Pour faire cela, ouvrez le fichiers **ensisa_project/urls.py**. Celui contient déjà des règles de routage concernant l'administration que nous verrons plus tard. Modifiez le fichier comme ceci (vous pouvez laisser les premières lignes de commentaire d'aide) :

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
  path('admin/', admin.site.urls),
  path('polls/', include('polls.urls')),
]
```

La fonction **include()** permet ainsi de référencer d'autres configurations d'URL (par exemple celle de l'application de sondage).

Cette étape importante permet également de rendre les applications très modulaires et facilement réutilisables dans d'autres projets.

Vous avez donc à présent relié une vue **welcome** dans la configuration d'URL. Vérifiez que la vue fonctionne en lançant votre serveur :

```
$ python manage.py runserver
```

Rendez-vous maintenant à l'adresse : `http://localhost:8000/polls/` dans votre navigateur et observez le résultat.

Si vous obtenez un message d'erreur 'Page not found', vérifiez que vous avez bien ajouté **polls/** à votre adresse dans le navigateur.

1.4.1 La fonction path()

Regardons de plus prêt la fonction **path(route,view,kwargs=None,name=None)** dont la documentation se trouve ici : `https://docs.djangoproject.com/fr/4.1/ref/urls/#django.urls.path`. La fonction **path()** reçoit quatre paramètres, dont deux sont obligatoires (**route** et **view**), et deux optionnels (**kwargs** et **name**) :

- **route** est une chaine de caractères contenant un motif d'URL. Lorsque Django traite une requête, il analyse tous les motifs présents dans la liste **urlpatterns** jusqu'à ce qu'il en trouve un qui correspond à la requête demandée.
- **view** permet d'indiquer la fonction de vue à exécuter lorsque le motif correspondant.
- **kwargs** est optionnel et correspond à des paramètres supplémentaires qui peuvent être transmis dans un dictionnaire vers la vue cible.
- **name** permet de référencer l'URL correspondante avec un nom précis pour faciliter sa réutilisation.

1.4.2 Une deuxième vue ?

Comme vous l'avez certainement remarqué, dans la fonction **path()** de votre exemple, le paramètre **route** est une chaine de caractère vide : **''**. Cela correspond à la route initiale de l'application de sondage : `http://localhost:8000/polls/`. Il est possible bien entendu de définir d'autres routes supplémentaires dans notre application pour définir d'autres comportements.

Bien que nous verrons cela plus en détail, dans un prochain TD, regardons en avant première comment ajouter une deuxième route et donc une deuxième vue. Commencez par modifier le fichier **polls/urls.py** comme ceci :

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

urlpatterns = [
  path('', views.welcome, name='welcome-view'),
  path('hello', views.hello, name='hello-view'),
]
```

Comme vous le voyez, une nouvelle route **'hello'** a été définie. Elle exécutera le comportement **views.hello** qu'il faut à présent définir en modifiant le fichier **polls/views.py** comme ceci :

```
from django.http import HttpResponse

def welcome(request):
  return HttpResponse("Bienvenue dans l'application de sondage !")

def hello(request):
  return HttpResponse("Bonjour")
```

Si vous vous rendez à présent à l'adresse `http://localhost:8000/polls/hello` dans votre navigateur, le texte "Bonjour" sera maintenant affiché.

Tant que nous y sommes, essayons d'aller encore un peu plus loin (toujours en avant première!) en ajoutant une nouvelle route pour afficher "Bonjour" + le prénom d'une personne renseigné dans l'url. Commencez par modifier le fichier **polls/urls.py** comme ceci :

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

urlpatterns = [
  path('', views.welcome, name='welcome-view'),
  path('hello', views.hello, name='hello-view'),
  path('hello/<firstname>', views.hello, name='hello_name-view'),
]
```

Dans cette nouvelle route, un paramètre supplémentaire **firstname** entre < > a été ajouté. En conséquence, il est à présent attendu que la fonction de vue correspondante (**hello()**) reçoive également un paramètre supplémentaire : **firstname**. Modifiez la fonction **hello()** du fichier **polls/views.py** en conséquence :

```
def hello(request, firstname=None):
  if firstname:
    return HttpResponse("Bonjour " + firstname + " !")
  else:
    return HttpResponse("Bonjour")
```

Dans votre navigateur, tapez l'url `http://localhost:8000/polls/hello/maxime` et observez le texte "Bonjour maxime!" à l'écran.

Notez que nous avons gardé la vue précédente pour afficher le message d'origine si aucun prénom n'est entré en paramètre.