Objectifs de la feuille

- Création environnement virtuel
- Installation DJANGO sous Linux
- Création d'un projet de base, lancement du serveur, création base de données
- Création d'une application et d'une première vue
- Un petit bonus pour digérer

Création environnement virtuel et installation sous Linux

Dans ce TD/TP, vous allez vous initier au framework DJANGO basé sur python. Suivez les étapes :

- création : `virtualenv ~/venv`
- initialisation : `source ~/venv/bin/activate`
- installation de DJANGO : `pip install django`
- vérification de la version : `django-admin --version`

Le package Django est installé, ainsi qu'un certain nombre de dépendances : des packages dont Django a besoin pour fonctionner. Gardons la trace de tous ces packages dans un fichier requirements.txt. Il existe un raccourci très pratique pour faire cela : `pip freeze > requirements.txt`

Une fois Django installé, nous sommes prêts à commencer !

Création d'un projet

Avant de nous plonger dans la construction, une poignée de fichiers du projet doit être mise en place. Il faut générer notre code de base DJANGO : `django-admin startproject TutoDjango`

Plus précisément, la génération automatique de code met en place un ensemble de réglages particuliers à une instance de Django, qui comprend la configuration de la base de données, des options spécifiques à Django et d'autres propres à l'application.

Observez la hiérarchie des fichiers créés : `tree`

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

Ces fichiers sont (pour les plus importants):

- Le premier répertoire racine `TutoDjango/` est un contenant pour votre projet. Son nom n'a pas d'importance pour Django ; vous pouvez le renommer comme vous voulez.
- `manage.py` : un utilitaire en ligne de commande qui vous permet d'interagir avec ce projet Django de différentes façons.
- Le sous-répertoire `TutoDjango/` correspond au paquet Python effectif de votre projet. C'est le nom du paquet Python que vous devrez utiliser pour importer ce qu'il contient (ex. TutoDjango.urls).
- `TutoDjango/__init__.py` : un fichier vide qui indique à Python que ce répertoire doit être considéré comme un paquet.
- `TutoDjango/settings.py` : réglages et configuration de ce projet Django. Comme souvent en python le fichier de configuration n'est pas un simple fichier texte mais un script qui déclare des variables d'environnement. C'est ici que vous pourrez indiquer quelle base de données utiliser ou quelles applications ajouter au projet, où se situent les fichier statiques, etc.
- `TutoDjango/urls.py` : les déclarations des URL de ce projet Django, une sorte de «table des matières» de votre site Django. Pour chaque URL définie on peut l'envoyer vers une fonction choisie.

Maintenant, lorsque nous utiliserons l'utilitaire de ligne de commande de Django, nous l'appellerons via `manage.py` au lieu de django-admin comme nous le faisions auparavant. En fait, `manage.py` est conçu pour fonctionner spécifiquement avec notre projet, alors que django-admin est une version plus générique de l'utilitaire. Avec le code de base en place, nous avons tout ce dont nous avons besoin pour lancer notre site pour la première fois.

Lancement du serveur de développement

En prenant soin de se mettre dans le bon répertoire, appelons à nouveau l'utilitaire de ligne de commande, cette fois via manage.py, et lançons la sous-commande runserver :

```
python manage.py runserver
```

L'utilitaire nous indique que le serveur de développement a démarré à l'adresse http://127.0.0.1:8000/. Jetons un coup d'œil à cette adresse dans le navigateur. Normalement l'écran d'accueil de DJANGO doit apparaître. Ça marche ! Vous avez démarré le serveur de développement de Django, un serveur Web léger entièrement écrit en Python.

Prenez un moment pour admirer la page par défaut de Django, puis jetez un autre coup d'œil à votre terminal. Pendant que vous parcourez les pages de votre application web, vous verrez des messages de journal apparaître dans le terminal. Ils peuvent s'avérer utiles pour le débogage.

En effet, la première chose à connaître c'est qu'il existe une variable qui gère le rendu votre projet , c'est à dire si votre projet se trouve dans un environnement de développement ou de production le rendu ne sera pas le même: dans un environnement de développement on affiche les erreurs / debug, ce qui est fortement déconseillé dans un environnement de production.

Voici la variable (dans le fichier settings.py): DEBUG = True

De même, remplacez la valeur de la variable TIME_ZONE par le lieu géographique de votre serveur: TIME_ZONE = "Europe/Paris"

Pour modifier les paramètres de la langue: LANGUAGE_CODE = "fr-fr".

Pour modifier les paramètres de la langue: DATE_INPUT_FORMATS = ('%d/%m/%Y','%Y -%m -%d')

Nous indiquons ici où sont stockés les fichiers statiques et médias et comment les atteindre :

```
STATIC_URL = '/static/'
STATICFILES_DIRS = (BASE_DIR / 'static/',)

MEDIA_URL = '/media/'
MEDIA_ROOT = BASE_DIR /  'media/'
```

Création de la base de données du projet

Maintenant, ouvrez TutoDjango/settings.py. C'est un module Python tout à fait normal, avec des variables de module qui représentent des réglages de DJANGO. La configuration DATABASES par défaut utilise SQLite. Si vous débutez avec les bases de données ou que vous voulez juste essayer DJANGO, il s'agit du choix le plus simple. SQLite est inclus dans Python, vous n'aurez donc rien d'autre à installer pour utiliser ce type de base de données. Lorsque vous démarrez votre premier projet réel, cependant, vous pouvez utiliser une base de données plus résistante à la charge comme PostgreSQL, afin d'éviter les maux de tête consécutifs au changement perpétuel d'une base de données à l'autre.

Notez également le réglage INSTALLED_APPS au début du fichier. Cette variable contient le nom des applications DJANGO qui sont actives dans cette instance de DJANGO. Les applications peuvent être utilisées dans des projets différents, et vous pouvez empaqueter et distribuer les vôtres pour que d'autres les utilisent dans leurs projets. Par défaut, INSTALLED_APPS contient les applications suivantes, qui sont toutes contenues dans DJANGO:

- django.contrib.admin – Le site d'administration. Vous l'utiliserez très bientôt.
- django.contrib.auth – Un système d'authentification.
- django.contrib.contenttypes – Une structure pour les types de contenu (content types).
- django.contrib.sessions – Un cadre pour les sessions.
- django.contrib.messages – Un cadre pour l'envoi de messages.
- django.contrib.staticfiles – Une structure pour la prise en charge des fichiers statiques.

Ces applications sont incluses par défaut par commodité parce que ce sont les plus communément utilisées. Certaines de ces applications utilisent toutefois au moins une table de la base de données, donc il nous faut créer les tables dans la base avant de pouvoir les utiliser. Pour ce faire, lancez la commande suivante : python manage.py migrate

La commande examine le réglage INSTALLED_APPS et crée les tables de base de données nécessaires en fonction des réglages de base de données dans votre fichier TutoDjango/settings.py et des migrations de base de données contenues dans l'application (nous les aborderons plus tard). Vous verrez apparaître un message pour chaque migration appliquée. Nous apprendrons beaucoup de choses sur les migrations tout au long de ce module, mais pour l'instant, nous devons simplement comprendre que les migrations représentent un moyen de configurer la base de données de notre application.

Maintenant, listez le contenu du répertoire et vous verrez qu'un fichier de base de données a été créé, appelé db.sqlite3. Il s'agit de la base de données qui contiendra toutes les données de notre application, c'est donc une bonne chose de l'avoir créée dès maintenant.

Création d'une application

Maintenant que votre environnement (un projet) est en place, vous êtes prêt à commencer à travailler. Chaque application que vous écrivez avec Django est en fait un paquet Python qui respecte certaines conventions. Django est livré avec un utilitaire qui génère automatiquement la structure des répertoires de base d'une application, ce qui vous permet de vous concentrer sur l'écriture du code, plutôt que sur la création de répertoires.

Dans Django, une application est une sous-section de votre projet entier. Django nous encourage à compartimenter notre projet entier Django en applications, pour deux raisons principales :

- Cela permet de garder notre projet organisé et gérable au fur et à mesure qu'il se développe ;
- Cela signifie qu'une application peut éventuellement être réutilisée dans plusieurs projets.

Comme il s'agit de notre tout premier projet Django, il sera suffisamment petit pour que notre code puisse s'intégrer sans peine dans une seule application. Chaque application doit avoir un nom approprié qui représente le concept dont l'application est responsable.

Quelle est la différence entre un projet et une application ? Une application est une application Web qui fait quelque chose – par exemple un système de blog, une base de données publique ou une petite 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.

Créons maintenant cette application, en utilisant la sous-commande `startapp` dans l'utilitaire de ligne de commande dans le répertoire `TutoDjango` (racine du projet) : `./manage.py startapp monApp`

Nous avons un nouveau répertoire appelé `monApp`, contenant plusieurs fichiers de code de base. C'est notre répertoire d'applications, et il contiendra tout le code relatif aux groupes.

Notez que `monApp`, votre répertoire d'applications, se trouve à côté de `TutoDjango`, votre répertoire de projet. Remarquez aussi que le répertoire du projet a le même nom (`TutoDjango`) que le répertoire au-dessus de lui par défaut. C'est la convention dans la structure d'un projet Django.

Chaque répertoire d'applications est spécifique à une application. Mais votre répertoire de projet contient des fichiers de configuration pour l'ensemble du projet : c'est un peu la «tour de contrôle» de votre projet Django. La dernière étape de l'ajout de notre application `monApp` à notre projet `TutoDjango` consiste à installer l'application dans le projet.

Lorsque nous avons généré le code de base de notre projet, l'un des fichiers créés s'appelait settings.py. Ouvrez maintenant ce fichier et trouvez une liste Python appelée INSTALLED_APPS. En bas de cette liste, ajoutez la chaîne de caractères 'monApp':

```
INSTALLED_APPS = [
'django.contrib.admin',
...
'django.contrib.staticfiles',
'monApp',
]
```

Par défaut, le code de base de Django comprend l'installation d'un certain nombre d'applications utiles, que la plupart des projets utiliseront probablement à un moment ou à un autre. Celles-ci incluent django.contrib.admin que nous utiliserons plus tard. Ainsi, même si le code que nous écrivons durant ce module tient dans une seule application, nous serons en mesure de constater la puissance de l'intégration de plusieurs applications réutilisables dans notre projet.

Contrairement à notre application monApp, ces applications supplémentaires ne se trouvent pas dans notre code source, nous ne voyons pas de répertoire dans notre code appelé django.contrib.admin, par exemple. Ces applications sont, en fait, importées depuis le package de Django que nous avons installé au début de ce chapitre avec pip. La meilleure pratique consiste à ajouter notre application en bas de la liste afin qu'elle soit la dernière à se charger.

Ajout d'une première page

Ce sera une page très basique, certes, mais ce sera tout de même une page !
Écrivons la première vue. Ouvrez le fichier monApp/views.py et placez-y le code Python suivant :

```
from django.http import HttpResponse

def home(request):
   return HttpResponse("<h1>Hello Django!</h1>")
```

Nous venons de créer une vue, l'un des blocs constitutifs de l'architecture MVT. Une vue a pour fonction de répondre à la visite d'un utilisateur sur le site en renvoyant une page que l'utilisateur peut voir. Une vue est une fonction qui accepte un objet HttpRequest comme paramètre et retourne un objet HttpResponse . Dans notre exemple de vue, nous renvoyons une réponse HTTP avec un contenu HTML simple : un titre H1 disant «Hello Django !».

Il s'agit ici de la vue la plus basique possible avec Django. Pou y accéder au travers d'un navigateur, il est nécessaire de la faire correspondre à une URL, et pour cela, il faut définir une configuration d'URL, appelée aussi «URLconf» pour faire court. Ces configurations d'URL sont définies dans chaque application Django sous la forme de fichiers Python nommés urls.py. Pour définir une URLconf pour l'application monApp, créez un fichier monApp/urls.py avec le contenu suivant :

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

urlpatterns = [
   path("", views.home, name="home"),
]
```

L'étape suivante est de configurer l'URLconf globale du projet TutoDjango pour y inclure l'URLconf définie dans monApp.urls. Pour cela, ajoutez une importation django.urls.include dans TutoDjango/urls.py et insérez un appel à include() dans la liste urlpatterns, ce qui donnera :

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

urlpatterns = [
   path("monApp/", include("monApp.urls")),
   path("admin/", admin.site.urls),
]
```

La fonction include() permet de référencer d'autres configurations d'URL. Quand Django rencontre un include(), il tronque le bout d'URL qui correspondait jusque là et passe la chaîne de caractères restante à la configuration d'URL incluse pour continuer le traitement. L'idée derrière include() est de faciliter la connexion d'URL. Comme l'application monApp possède son propre URLconf (monApp/urls.py), ses URL

peuvent être injectés sous «/monApp/», sous « /fun_monApp/ » ou sous « /content/monApp/ » ou tout autre chemin racine sans que cela change quoi que ce soit au fonctionnement de l'application.
Vous avez maintenant relié une vue home dans la configuration d'URL. Vérifiez qu'elle fonctionne avec la commande suivante : manage.py runserver

Ouvrez http://localhost:8000/monApp/ dans votre navigateur et vous devriez voir le texte «Hello Django!» qui a été défini dans la vue home.

Si vous n'arrivez pas à afficher ce rendu , c'est que soit vous avez des très gros doigts et que vous avez fait une faute de frappe quelque part, soit DJANGO venge de tout à l'heure et qu'il vous affiche un ancien rendu. Forcez DJANGO à vous fournir le bon rendu en appuyant simultanément et frénétiquement sur les touches Ctrl + F5 .

La fonction path() reçoit quatre paramètres :

- dont deux sont obligatoires : route et view,
- et deux facultatifs : kwargs et name.

À ce stade, il est intéressant d'examiner le rôle de ces paramètres (les plus importants):

- route est une chaîne contenant un motif d'URL. Lorsqu'il traite une requête, Django commence par le premier motif dans urlpatterns puis continue de parcourir la liste en comparant l'URL reçue avec chaque motif jusqu'à ce qu'il en trouve un qui correspond.
- view : Lorsque Django trouve un motif correspondant, il appelle la fonction de vue spécifiée, avec un objet HttpRequest comme premier paramètre et toutes les valeurs «capturées» par la route sous forme de paramètres nommés.
- name : Le nommage des URL permet de les référencer de manière non ambiguë depuis d'autres portions de code Django, en particulier depuis les gabarits. Cette fonctionnalité puissante permet d'effectuer des changements globaux dans les modèles d'URL de votre projet en ne modifiant qu'un seul fichier.

Petit challenge

C'est à vous ! Ajoutez des pages à votre site avec des modèles d'URL et des vues. Je vous ai montré comment ajouter de nouvelles pages à votre site en utilisant des modèles d'URL et des vues. Maintenant, je veux que vous utilisiez ce que vous avez appris pour ajouter deux nouvelles pages à votre application :

- Une page «contact us» où nous créerons un formulaire de contact plus tard
- Une page «about us»

Pour le moment, ces pages peuvent être très simples : juste une balise <h1> et une balise <p>. Nous construirons le contenu complet plus tard dans le cours. Vous devez visiter ces pages dans le navigateur pour vérifier qu'elles s'affichent correctement.

Si vous avez besoin de conseils, pour chaque page, vous aurez besoin de :

- Une vue dans monApp/views.py ;
- Un modèle d'URL dans monApp/urls.py. Chaque modèle d'URL nécessite 2 arguments :
  - une chaîne qui correspond au chemin d'accès à l'URL de la page,
  - une vue (n'oubliez pas de l'importer en haut du fichier !)

Pour le bouquet final, compliquons légèrement les choses pour pouvoir router des urls du type:

http:/localhost:8000/monApp/home/Cricri.

Modifions tout d'abord le fichier urls.py de monApp de la façon suivante :

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

Ceci signifie que les urls du type http:/localhost:8000/monApp/home/Cricri seront routées vers notre vue. Modifier la fonction home() avec le profil suivant (à vous de la compléter, j'aimerais bien que le site me dise «bonjour Cricri !») :

```
from django .http import HttpResponse

def home(request , param ):
return HtppResponse (…)
```

Puis tester différentes urls en modifiant le paramètre fourni.
Comment faire en sorte que l'URL http://localhost:8000/monApp/home/ soit également routée ?