Culture GNU/Linux

Projet “Make our planet green again”
-- Django --

Master 2 ACDI

Auteurs

NDAYISHIMA Divin
SALIM Youssef

Professeur

TELETCHEA Stéphane

Table des matières

Introduction

Suite à la partie du cours sur le Framework Django et la virtualisation, nous avons pratiqué les notions apprises sur un projet qui consistait à développer un service web avec comme objectifs de :

- proposer un projet éco-responsable détaillé
- permettre de faire valider un projet par des utilisateurs qualifiés
- gérer les financements d'un projet

Dans ce rapport nous reviendrons sur le déroulement de cet exercice en passant par une présentation de quelques uns des technologies manipulés.

Vous pouvez récupérer le code source (documenté) du projet sur ce repository Github (https://github.com/dndayishima/mopga_django/tree/master).

1. Développement avec Django

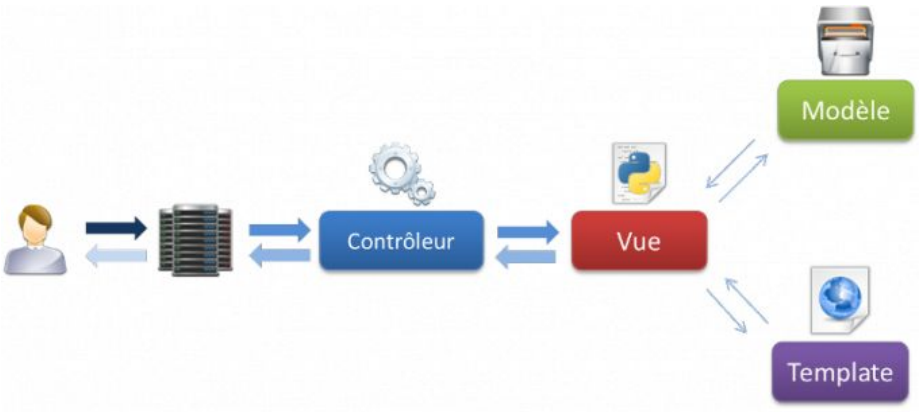
1.1. Qu'est-ce qu'un Framework

D'une manière assez simplifiée, un Framework en informatique est un ensemble de composants logiciels structurels, qui sert à définir les fondations et les grandes lignes d'un logiciel.
L'organisation d'un code suivant un Framework permet aux développeurs d'avoir une productivité maximale sur des projets simples et complexes. Le code produit aura pour avantage d'être facilement maintenable dans le temps.

1.2. Présentation du Framework Django

Django est un framework web open source développé en Python. Il rend le développement web simple et rapide. Le projet Django a pour slogan : "Le framework web pour les perfectionnistes avec des deadlines".
La communauté autour de ce framework est très active et de grands comptes développent leurs outils avec ce dernier, ce qui lui garantie une évolution dans le temps. Nous verrons au point suivant quelques exemples de comment les grands groupes utilisent Django dans leurs projets.

Django suit l'architecture de développement MVC - Modèle-Vue-Contrôleur (architecture que nous retrouvons souvent dans les framework modernes) ou plus particulièrement MTV (Modèle-Vue-Template).
À la différence de MVC, dans le pattern MTV c'est Django lui-même qui gère la partie *contrôleur*.



1.3. Quelques exemples de comment Django est utilisé

- Le module de gestion des images et de partage des vidéos sur **Instagram** est développé en Django. Ce module interagit avec de très gros volumes de données chaque seconde.

- La plateforme de streaming de musique **Spotify**, utilise quant à elle Django sur ses services back-end et pour l'apprentissage artificielle (machine learning).

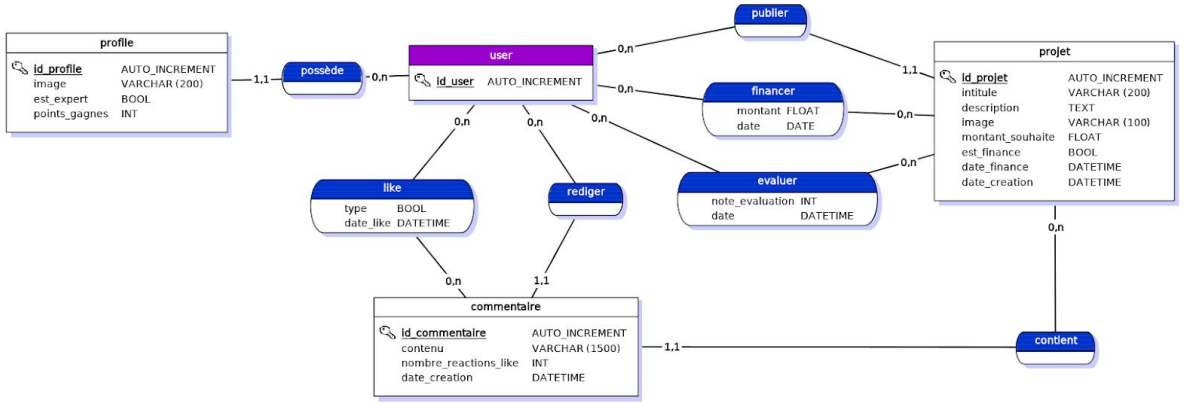
- L'emblématique plateforme de "partage vidéos" (video-sharing) **YouTube**, qui était auparavant développée en PHP a vu son code source être migré vers Django dans le but de la faire évoluer en y ajoutant beaucoup d'autres fonctionnalités tout en réduisant considérablement le temps de développement.

Pour n'en citer que ceux-là… Vous pouvez retrouver d'autres exemples en ligne, et l'on peut facilement se rendre compte qu'il y a plusieurs outils que nous utilisons dans notre quotidien, qui sont développés avec le framework Django.

Une liste d'autres exemple :
https://www.netguru.com/blog/top-10-django-apps-and-why-companies-are-betting-on-this-framework

2. Projet “Make our planet green again” (les modèles)

2.1. Schéma de la base de données



- **User et Profile**

La gestion des utilisateurs, pour ce projet, se base sur celle qui est proposée par Django. Celle-ci présente un certains nombre d’avantages car nous n’avons pas à réimplémenter les mécanismes d’authentification, les validations des formulaires utilisateurs et l’administration.

Modèle **User** de Django : https://docs.djangoproject.com/fr/3.0/ref/contrib/auth/

Quelques champs nécessaires pour stocker les informations d’un utilisateur sont déjà représentés dans ce modèle :
`username, first_name, last_name, email, password, groups, user_permissions, is_staff, is_active, is_superuser, last_login et date_joined.`

Pour pouvoir gérer d’autres informations, nous avons ajouté un autre modèle : **Profile**. Un utilisateur possède un profile et un profile appartient à un utilisateur (relation un vers un).

Dans le Profile, sont représentées les informations telles que :
- l’image de profile de l'utilisateur;
- le nombre de points gagnés (les conditions de gain de points seront expliquées dans une autre section);
- le statut de l’utilisateur (savoir s’il est **expert** ou pas).

Dans ce projet, il faut savoir qu’un expert représente un utilisateur très expérimenté, pouvant participer à l’attribution de notes sur des projets.

- **Projet**

Ce modèle représente les projets qui seront publiés par les utilisateurs. Nous sommes partis du principe que tous les utilisateurs peuvent être auteurs de projets.

Signification des champs :
- intitulé : intitulé du projet;
- description : texte décrivant en détails le projet;
- image : image associée au projet (logo du projet ou autre type d'image);
- montant_souhaité : cette information représente le montant nécessaire pour qu'un projet puisse démarrer;
- est_financé : valeur booléenne qui nous indiquera si le projet est déjà financé ou pas. Un projet est financé lorsque la somme des dons effectués par les utilisateurs est supérieure ou égale au montant souhaité pour que ledit projet puisse démarrer;
- date_finance : date à laquelle le montant nécessaire est atteint;
- date_création : date à laquelle un projet est publié.

Voici les opérations implémentées interagissant avec le modèle "Projet" :
- Un utilisateur peut publier un projet. Les enregistrements sur les relations entre les projets et leurs auteurs sont stockés dans la table "**publier**";
- un utilisateur peut financer un projet en faisant un don d'un montant donné. Les opérations de financement sont stockées dans la table "**financer**";
- un utilisateur, s'il est considéré comme étant **expert**, peut évaluer un projet, c'est-à-dire lui attribuer une note en fonction de sa pertinence. Sur ce point il n'y a pas de critères précis, les experts sont par définition capable de juger un projet.

- **Commentaire**

Un commentaire est rédigé par un utilisateur sur un projet. La table "**rédiger**" stocke les enregistrements du type : l'utilisateur d'identifiant **id_user** a rédigé un commentaire d'identifiant **id_commentaire**.
La table "**contient**" quand à elle décrit l'appartenance des commentaires aux projets.

Signification des champs :
- contenu : contenu textuel d'un commentaire;
- date_creation : date de publication d'un commentaire
- nombre_reactions_like : le nombre de mentions "**J'aime**" sur un commentaire. Cette information est utilisée pour le calcul du "**Karma**".

2.2. Le “Karma” (explication du calcul)

Dans notre modélisation, un utilisateur acquiert un statut d’expert (utilisateur qualifié) selon qu’il a gagné un certain nombre de points (Table “**Profile**” > champ **points_gagnes**).

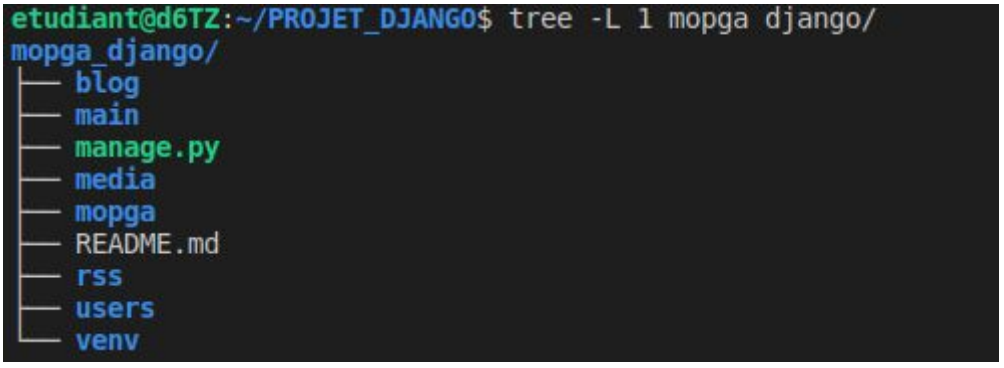
Pour un commentaire publié par rapport à un projet, chaque mention “J’aime” déclenche une opération d’augmentation du nombre de points pour l’utilisateur ayant publié ce commentaire.

3. Projet “Make our planet green again” (Structure Django)

Dans cette partie, nous aborderons des points techniques liés à l’utilisation de Django. Nous nous sommes appuyés sur 2 types de ressources (documentation et tutoriel) :

- Documentation officielle Django : https://docs.djangoproject.com/fr/2.2/
- Tutoriel YouTube (développement d’un blog) : https://www.youtube.com/playlist?list=PL-osiE80TeTtoQCKZ03TU5fNfx2UY6U4p

Un projet Django est un ensemble intégrant un certain nombre d’applications réutilisables, avec une configuration générale.



Le répertoire **mopga** contient les fichiers de configuration du projet.

Quelques fichiers du répertoire **mopga** :

- `settings.py` : C’est le fichier de configuration globale du projet. Il comprend plusieurs paramètres qui ont une influence direct sur les applications composant le projet.
- les fichiers `.pyc` : ce sont des scripts Python pré-compilé par l’interpréteur (fichiers binaires)

```
etudiant@d6TZ:~/PROJET_DJANGO/mopga_django$ tree -L 1 mopga/
mopga/
├── __init__.py
├── __pycache__
├── settings.py
├── urls.py
└── wsgi.py

1 directory, 4 files
```

3.1. Modifications apportées au fichier settings.py

- À chaque fois que nous démarrons une nouvelle application, il faut nous assurer qu'elle soit renseignée dans la variable "`INSALLED_APPS`" (un tableau - array).

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # CRISPY
    'crispy_forms',

    # My applications
    'main.apps.MainConfig',
    'users.apps.UsersConfig',
    'rss.apps.RssConfig'
]
```

- Par défaut, un projet Django fonctionne avec une base de données SQLITE se trouvant dans le fichier `db.sqlite3` (se trouvant à la racine du projet). Nous avons modifié la configuration "`DATABASES`" pour que nous puissions travailler avec une base de données MySQL.

```
# Database
# https://docs.djangoproject.com/en/2.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'db_mopga',
        'USER': 'root',
        'PASSWORD': 'admin',
        'HOST': '',
        'PORT': '',
        'default-character-set': 'utf8'
    }
}
```

- Modification de la langue d'une manière globale dans le projet

```
# Internationalization
# https://docs.djangoproject.com/en/2.2/topics/i18n/

LANGUAGE_CODE = 'fr-FR'
```

- Sur la capture d'écran qui suit, ce sont des paramètres rajoutés à la configuration de base :
  - `MEDIA_ROOT` et `MEDIA_URL` : répertoire où sont stockés (par exemple les images) des utilisateurs et URL pour y accéder;
  - `CRISPY_TEMPLATE_PACK` : template utilisé pour les formulaires d'authentification de l'utilisateur;
  - `LOGIN_REDIRECT_URL` : URL sur laquelle un utilisateur est redirigé après s'être connecté
  - `LOGIN_URL` : URL qui sera utilisé si l'utilisateur fait une action qui nécessite d'être authentifié, et qu'il n'est pas authentifié;
  - Les paramètres `EMAIL_*`, sont utilisés pour avoir un service d'envoi de mail. Ce service est utilisé dans notre projet pour envoyer des mails aux utilisateurs lorsqu'ils demandent la récupération du mot de passe (mot de passe oublié).

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/2.2/howto/static-files/

STATIC_URL = '/static/'

MEDIA_ROOT = os.path.join(BASE_DIR, "media")

MEDIA_URL = "/media/"

CRISPY_TEMPLATE_PACK = "bootstrap4"

LOGIN_REDIRECT_URL = "main-home"

LOGIN_URL = "login"

# L'envoi des e-mail
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = 'adressemail@gmail.com'
EMAIL_HOST_PASSWORD = 'ici_mettre_un_mot_de_passe'
```

3.2. Modèles, classes, templates et fichiers "static"

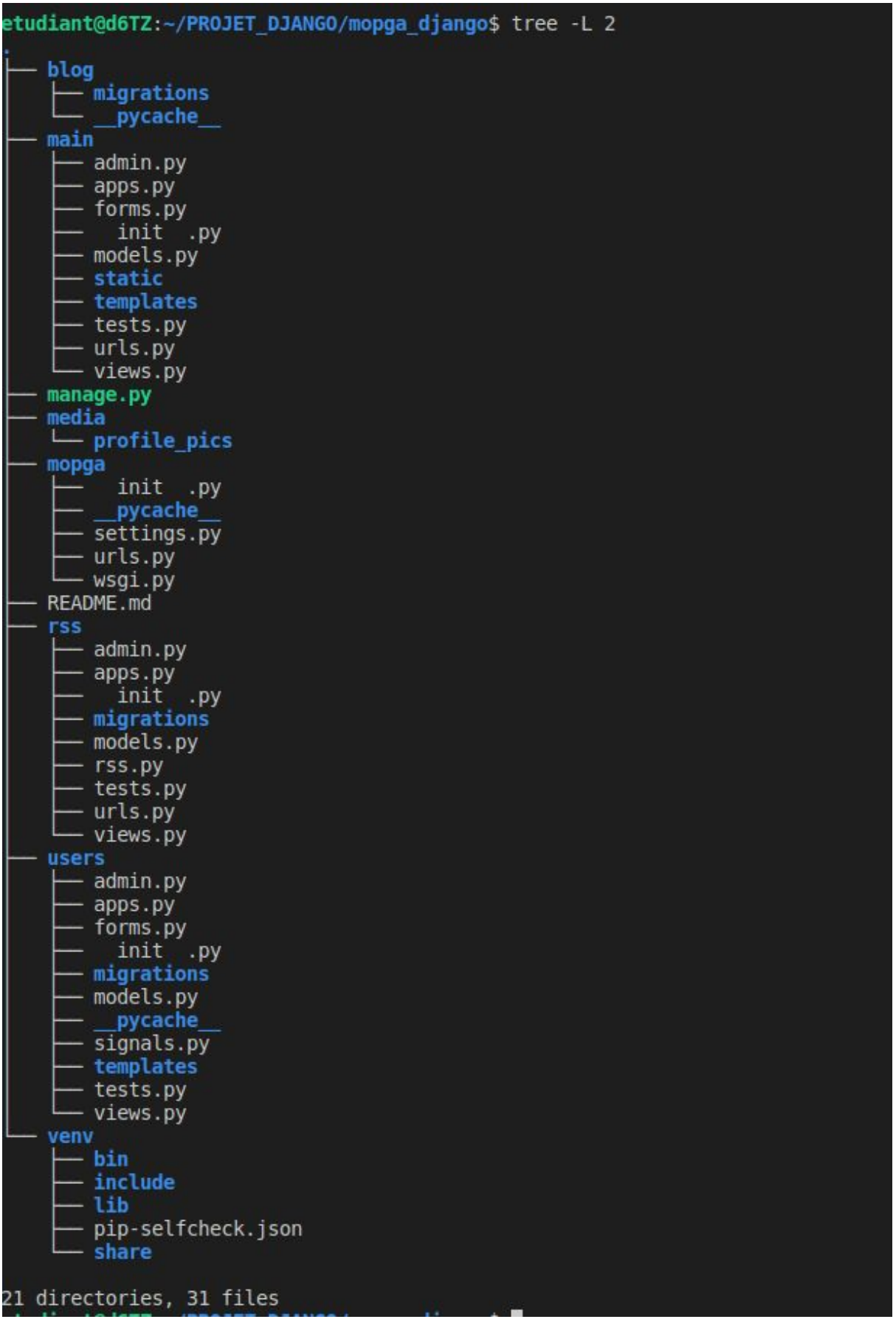
Tous les templates utilisés se trouvent dans le répertoire `main/templates`. Ils héritent tous du template `base.html`.

Voici la description de nos modèles :

- Profile : comme expliqué plus haut dans ce document (schéma de la base de données), c'est un modèle qui permet de représenter les informations supplémentaires d'un utilisateur;
- Commentaire : ce modèle permet de stocker les commentaires que les utilisateurs rédigent;
- Financer : représentation d'un financement d'un projet par un utilisateur
- Evaluer : ce modèle permet de donner une évaluation à un projet. C'est uniquement accessible par les utilisateurs considérés comme étant "experts".

La convention en Django est qu'il faut faut stocker les fichiers "static" dans un répertoire nommé `static`.
Dans ce répertoire nous y avons mis tous nos fichiers CSS utilisés, les scripts JS ainsi que la police de caractères utilisée sur l'ensemble du projet.

La capture d'écran ci-dessous montre l'organisation de notre projet avec les différentes applications réutilisables.


etudiant@d6TZ:~/PROJET_DJANGO/mopga_django$ tree -L 2
.
├── blog
│   ├── migrations
│   └── __pycache__
├── main
│   ├── admin.py
│   ├── apps.py
│   ├── forms.py
│   ├── __init__.py
│   ├── models.py
│   ├── static
│   ├── templates
│   ├── tests.py
│   ├── urls.py
│   └── views.py
├── manage.py
├── media
│   └── profile_pics
├── mopga
│   ├── __init__.py
│   ├── __pycache__
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── README.md
├── rss
│   ├── admin.py
│   ├── apps.py
│   ├── __init__.py
│   ├── migrations
│   ├── models.py
│   ├── rss.py
│   ├── tests.py
│   ├── urls.py
│   └── views.py
├── users
│   ├── admin.py
│   ├── apps.py
│   ├── forms.py
│   ├── __init__.py
│   ├── migrations
│   ├── models.py
│   ├── __pycache__
│   ├── signals.py
│   ├── templates
│   ├── tests.py
│   └── views.py
└── venv
    ├── bin
    ├── include
    ├── lib
    ├── pip-selfcheck.json
    └── share
21 directories, 31 files

3.3. Dépendances installées

```
(venv) etudiant@d6TZ:~/PROJET_DJANGO/mopga_django$ pip freeze
Django==2.2.9
django-crispy-forms==1.8.1
Pillow==7.0.0
pkg-resources==0.0.0
pytz==2019.3
sqlparse==0.3.0
```

- **Django version 2.2.9** : version de Django utilisé pour ce projet;
- **Django crispy forms** : templates des formulaires d'authentification de l'utilisateur;
- **Pillow** (version 7.0.0) : module Python qui nous permet de gérer les images;
- **pkg-resources** : fournit une API pour permettre aux bibliothèques Python d'accéder à leurs fichiers de ressources;
- **pytz** : module de définition et gestion du **Timezone**;
- **SQL parse** : module qui nous a permis de pour voir parse du SQL écrit en brut dans du code Python

4. Déploiement dans des conteneurs Docker

Nous déployons ce projet dans des conteneurs Docker :

- un conteneur Python : conteneur sur lequel tournera notre projet Django
- un conteneur MySQL : conteneur sur lequel tournera MySQL

Pour persister les données, nous montons des volumes sur le conteneur de la base de données, volumes qui sont associés au système de fichiers.

Pour automatiser les étapes de constructions d'images et de conteneurs, vous pouvez utiliser le script bash qui est fournit à la racine du projet : `install.sh`.

Ce script construira les images et lancera les 2 conteneurs.

Conclusion

Le travail réalisé dans le cadre de ce projet nous a permis de nous familiariser avec l'utilisation du Framework Django et c'est une expérience qui s'est avérée très intéressante.

Les connaissances acquises que ce soit par rapport au développement Django ou le déploiement dans des conteneurs Dockers nous sera sans doute utile dans plusieurs projets que nous serons amenés à réaliser.

La plupart des fonctionnalités demandées ont été implémentées, mais nous n'avons malheureusement pas pu tester le déploiement de notre projet sur notre noyau personnalisé.