Université d'Angers

M2-ACDI

Culture GNU / Linux

Développement d'un site web avec le Framework Django & Virtualisation sous Docker

---

Auteurs

DIOP Alassane

CISSE Sekou Aboubacar

TRAORE Hamidou

Enseignant

TÉLETCHÉA Stéphane

2018 – 2019

université angers

Table des matières

I. Structure du projet et de la base de données

I.1. Structure du projet

« **makeOurPlanetGreatAgain** » est le nom du projet.

Un projet django est une instance d'un certains nombre d'applications avec une configuration associée. La structure hierarchique du projet est donnée par la figure suivante : résultat de la commande **`$tree -d makeOurPlanetGreatAgain`**

```
(django) etudiant@d8CK:~/DjangoProject/makeOurPlanetGreatAgain$ tree -d
.
├── makeOurPlanetGreatAgain
│   ├── migrations
│   │   └── __pycache__
│   ├── __pycache__
│   ├── static
│   │   ├── css
│   │   └── img
│   └── templates
│       └── registration
├── media
│   └── img
├── profils
│   ├── migrations
│   │   └── __pycache__
│   ├── __pycache__
│   └── templates
└── projets
    ├── migrations
    │   └── __pycache__
    ├── __pycache__
    └── templates
```

Le répertoire **`makeOurPlanetGreatAgain`** contient les fichiers configuration du projet. C'est une application Django créée automatiquement à la création du projet. Ci-dessous une illustration de son contenu hiérarchique :

```
7 directories, 32 files
(django) etudiant@d8CK:~/DjangoProject/makeOurPlanetGreatAgain$ tree makeOurPlanetGreatAgain/
makeOurPlanetGreatAgain/
├── feeds.py
├── forms.py
├── __init__.py
├── migrations
│   ├── 0001_initial.py
│   ├── 0002_projet.py
│   ├── 0003_delete_projet.py
│   ├── 0004_delete_auteur.py
│   ├── __init__.py
│   └── __pycache__
│       ├── 0001_initial.cpython-35.pyc
│       ├── 0002_projet.cpython-35.pyc
│       ├── 0003_delete_projet.cpython-35.pyc
│       ├── 0004_delete_auteur.cpython-35.pyc
│       └── __init__.cpython-35.pyc
├── models.py
├── __pycache__
│   ├── feeds.cpython-35.pyc
│   ├── forms.cpython-35.pyc
│   ├── __init__.cpython-35.pyc
│   ├── models.cpython-35.pyc
│   ├── settings.cpython-35.pyc
│   ├── urls.cpython-35.pyc
│   ├── views.cpython-35.pyc
│   └── wsgi.cpython-35.pyc
├── settings.py
├── static
│   ├── css
│   │   └── general_style.css
│   └── img
├── templates
│   ├── about.html
│   ├── auteur_authentication.html
│   ├── base.html
│   ├── contact.html
│   └── privacy_policy.html
├── urls.py
├── views.py
└── wsgi.py

7 directories, 32 files
(django) etudiant@d8CK:~/DjangoProject/makeOurPlanetGreatAgain$
```

Certains fichiers dans cette hiérarchie sont générés à la création du projet : `settings.py`, `urls.py`...

- **settings.py** : il contient les réglages et configuration propres aux projet et aux différentes applications actives. Nos modifications sur ce fichier concerne le fuseau horaire, la langue d'affichage et l'ajout des applications, la spécification des dossiers de fichiers statiques.
- **Les fichiers .pyc** : ce sont des fichiers compilés qui seront régénérés. Pour la distribution du projet, ces fichiers peuvent être nettoyer du projet. Cela est réalisé en exécutant la commande **`$find . name '*.pyc' exec rm {} \;`**.
- ***Le dossier des fichier statiques*** *(static)* : c'est la conventions django qui recommande de nommé ce dossier ainsi. Il contient les fichiers statiques (ou fichiers externes) : **`css, image, Jquery.`**

Les autres fichiers sont présentés plus loin dans ce document (*point : applications actives*)

I.2. Les applications actives

Une fois le projet créé, on peut ajouter des applications avec la commande suivante :

- `$python manage.py startapp nom_application`

Nous avons trois applications Django dans notre projet (makeourPlanetGreatAgain) : makeourPlanetGreatAgain, profils et projets.

La structure hiérarchique d'une application se présentes comme suit : exemple de l'application projets

```
(django) etudiant@d8CK:~/DjangoProject/makeOurPlanetGreatAgain$ tree projets/
projets/
├── admin.py
├── apps.py
├── forms.py
├── __init__.py
├── migrations
│   ├── 0001_initial.py
│   ├── 0002_auto_20181206_2323.py
│   ├── 0003_auto_20181206_2325.py
│   ├── 0004_auto_20181207_1402.py
│   ├── 0005_auto_20190113_1932.py
│   ├── 0006_projet.py
│   ├── 0007_auto_20190113_2140.py
│   ├── 0008_auto_20190114_0936.py
│   ├── 0009_auto_20190114_1506.py
│   ├── 0010_auto_20190114_1510.py
│   ├── 0011_auto_20190114_1514.py
│   ├── 0012_auto_20190114_1515.py
│   ├── 0013_auto_20190115_1633.py
│   └── __init__.py
├── models.py
├── templates
│   ├── all_project.html
│   ├── evaluation_edit.html
│   ├── index_projet.html
│   ├── info_project.html
│   ├── new_projet.html
│   └── projets.html
├── tests.py
├── urls.py
└── views.py

2 directories, 28 files
(django) etudiant@d8CK:~/DjangoProject/makeOurPlanetGreatAgain$
```

Cette structure de fichiers est commune à toutes applications Django.

- **urls.py** : le lien entre les vus et le gabarits faisant appels à ces vues sont gérés par ce fichier. Pour passer de l'URL à la vue, Django utilise ce qu'on appelle des « URLconf ». Il contient

la déclarations des URL du projet et constitue une sorte de « *table des matières* » pour nos applications.

- **views.py** : c'est dans ce fichier que sont défini les vues de l'application. Une vue est un type de page web dans votre application Django qui sert généralement à une fonction précise et possède un gabarit spécifique.Nous avons définie les vues dans ce fichier et celui des autres applications actives du projet.

- **models.py** : il contient la définition du schémas de données. Un modèle est la source d'information unique et définitive pour les données. Il contient les champs essentiels et le comportement attendu des données stockées. Un extrait du contenu de ce fichiers est présenté ainsi qu'il suit :

```python
from django.db import models

from django.contrib.auth.models import User

from django.core.validators import MaxValueValidator

from django.dispatch import receiver

# Create your models here.
class Projet(models.Model):
    auteur_projet = models.ForeignKey(User, related_name='auteur', default=1, on_delete=models.CASCADE)
    titre_projet = models.CharField('Titre du projet ', max_length=40)
    finance_requis_projet = models.DecimalField("Financement réquis", default=0.00, max_digits=10, decimal_places=2)
    description_projet = models.TextField('Description ', max_length=4000, blank=True)
    date_ajout_projet = models.DateTimeField("Date d'ajout", auto_now_add=True, blank=True)
    financement_est_aquis = models.BooleanField("Financement déjà acquis", blank=False, default=False)
    date_acquis_finance = models.DateTimeField("Date aquisition financement", default=None, blank=True, null=True)

    def __str__(self):
        return self.titre_projet


class Financement(models.Model):
    """docstring for Financement"""
    projet = models.ForeignKey(Projet, default=1, on_delete=models.CASCADE)
    financeur = models.ForeignKey(User, default=1, on_delete=models.CASCADE)
    montant_finance = models.DecimalField("Montant ", default=0.00, max_digits=10, decimal_places=2, blank=True)
    date_finance = models.DateTimeField("Date du don", auto_now_add=True, blank=True)


class Evaluation(models.Model):
    """docstring for Evaluation"""
    projet = models.ForeignKey(Projet, default=1, on_delete=models.CASCADE)
    evaluateur = models.ForeignKey(User, default=1, on_delete=models.CASCADE)
    commentaire = models.TextField('Commentaire ', max_length=4000)
    note_evaluation = models.PositiveIntegerField("Note du projet", default=1, validators=[MaxValueValidator(5)])
    date_evaluation = models.DateTimeField("Date d'évaluation", auto_now_add=True, blank=True)
```

Ces classes de modèle (Model) sont mappé avec les tables de la base de données grâce à l'ORM Django. Il utilise le mécanisme de migration pour créer et mettre jour la structure relationnelle de la base données grâce à un mécanisme de migration. Ainsi, après chaque modifications du fichier models.py, il suffit juste d'exécuter les commandes suivantes :

  - `$python manage.py makemigrations nom_application :` cette commande crée un fichier de migration contenant les opérations à effectuer pour la mise à jour de la structure de la base

  - `$python manage.py migrate nom_application` : exécute les migrations décrites dans le fichier de migration.

  L'ensemble de migrations alors générés sont stockées dans un dossier « **`migrations`** »

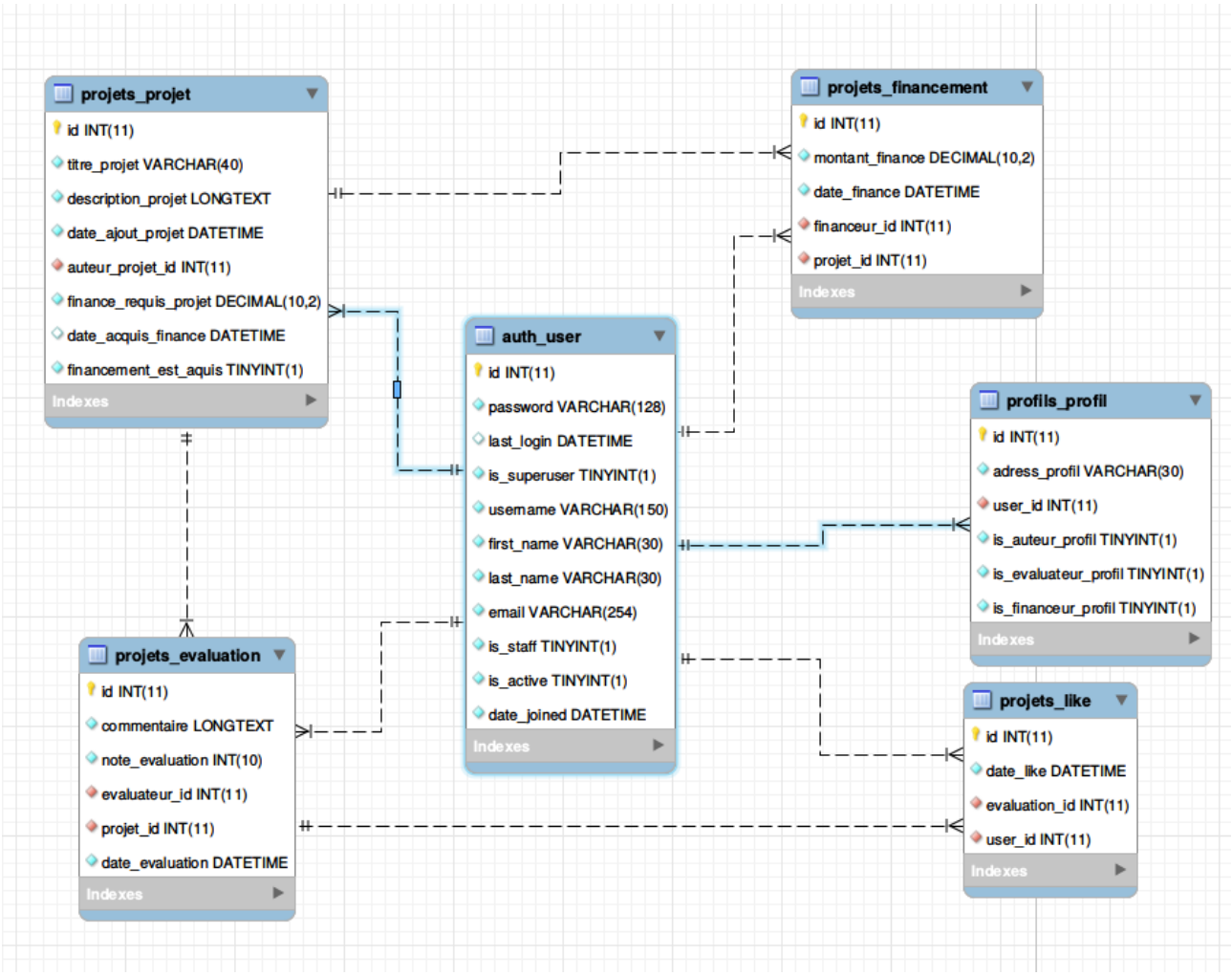
- **admin.py** : ce fichier contient les objets du modèle devant avoir une interface d'administration avec plus ou moins une spécification d'affichage.

- **tests.py** : ce fichier contient pour toute application Django les cas de tests du code. Nous n'avons pas défini de cas de test dans les applications de ce projet.
- **forms.py** : nous avons défini des formulaires pour la saisie des données utilisateurs. Django fournit un système de gestion des formulaire associé ou non aux models. Ces formulaires personnalisés sont à définir dans un fichier nommé forms.py (non généré).
- ***Le dossier des gabarits*** *(templates)* : aussi nommé ainsi par convention. Nous avons défini un gabarit de base (***`base.html`***) qui définit la structuration et les spécifications de bases d'une page web. Tous les autres gabarits du projet étendent ce gabarits de base : on peut noter la présence de « **`{% extends "base.html"%}`** » dans ces derniers gabarits en première ligne.
- **Fichiers externes** : nous avons utilisé du JavaScript pour la génération de graphique présentant le taux de financement d'un projet. Pour ce faire, cette ligne est incluse dans le fichier de template **`info_projet.html`** pour faire référence à une ressource externe.

```
6
7  {% block extrastyle %}
8      <script src="https://code.highcharts.com/highcharts.src.js"></script>
9  {% endblock %}
10
```

I.3. Structure de la base de données

Le digramme relationnelle de notre modèle de base données issues de classes django (Model) est illustré par la figure suivante :



Chaque entité ici excepté **aut_user** correspond à une table de la base de données du projet. Chacun est lié une classe correspondantes dans les fichier models.py par le mécanisme de mapping objet-relationnel.

- **aut_user :** table prédéfini par l'ORM Django pour la gestions des utilisateurs.
- **profils_profil** : mapping fait avec la classe `Profil` dans le fichier `models.py` du modules « profils ». Cette table référence la table aut_user et permet d'indiquer le fait qu'un utilisateur est enregistré comme « auteur », « évaluateur » ou « financeur » de projet. La classe model est défini comme suit :

```python
# Create your models here.
class Profil(models.Model):
    user = models.OneToOneField(User, related_name='utilisateur', on_delete=models.CASCADE)
    adress_profil = models.CharField(max_length=30, blank=True)
    is_auteur_profil = models.BooleanField("Est un auteur", blank=False, default=True)
    is_evaluateur_profil = models.BooleanField("Est un évaluateur", blank=False, default=False)
    is_financeur_profil = models.BooleanField("Est un financeur", blank=False, default=False)

    def __str__(self):
        return self.user.username
```

- Un *auteur* (de projet) est un utilisateur qui a publié au moins un projet.
- Un *évaluateur* est un auteur qui à donnés (posté) dont le nombre d'avis est supérieur à 1 et que la somme des notes positives (like) de ces avis est supérieur à 1.
- Un *financeur* est un utilisateur ayant participé au financement d'au moins un projet.

➢ **projets_projet** : mappé avec la classe `Projet` dans le fichier `models.py` du modules « projets ». Elle permet d'enregistrer un projet selon les propriétés de la tables. Ci-dessous la classe de model Django correspondant :

```python
# Create your models here.
class Projet(models.Model):
    auteur_projet = models.ForeignKey(User, related_name='auteur', default=1, on_delete=models.CASCADE)
    titre_projet = models.CharField('Titre du projet ', max_length=40)
    finance_requis_projet = models.DecimalField("Financement réquis", default=0.00, max_digits=10, decimal_places=2)
    description_projet = models.TextField('Description ', max_length=4000, blank=True)
    date_ajout_projet = models.DateTimeField("Date d'ajout", auto_now_add=True, blank=True)
    financement_est_aquis = models.BooleanField("Financement déjà acquis", blank=False, default=False)
    date_acquis_finance = models.DateTimeField("Date aquisition financement", default=None, blank=True, null=True)

    def __str__(self):
        return self.titre_projet
```

➢ **projets_financement** : mappé avec la classe `Financement` du fichier `models.py` dans le modules « projets ». Elle enregistre le financement d'un utilisateur pour un projet. Le model Django correspondant est :

```python

class Financement(models.Model):
    """docstring for Financement"""
    projet = models.ForeignKey(Projet, default=1, on_delete=models.CASCADE)
    financeur = models.ForeignKey(User, default=1, on_delete=models.CASCADE)
    montant_finance = models.DecimalField("Montant ", default=0.00, max_digits=10, decimal_places=2, blank=True)
    date_finance = models.DateTimeField("Date du don", auto_now_add=True, blank=True)

```

➢ **projets_evaluation** : permet d'enregistré les avis des utilisateurs sur les projets. Elle est mappé avec la classe `Evaluation` du fichier `models.py` dans le modules « projets ». Le model Django correspond ant est :

```python
class Evaluation(models.Model):
    """docstring for Evaluation"""
    projet = models.ForeignKey(Projet, default=1, on_delete=models.CASCADE)
    evaluateur = models.ForeignKey(User, default=1, on_delete=models.CASCADE)
    commentaire = models.TextField('Commentaire ', max_length=4000)
    note_evaluation = models.PositiveIntegerField("Note du projet", default=1, validators=[MaxValueValidator(5)])
    date_evaluation = models.DateTimeField("Date d'évaluation", auto_now_add=True, blank=True)

```

- projets_like : enregistre pour une « Evaluation » les avis positifs ou non (dans le sens de approuvé) des autres utilisateurs. Le model Django de cette table est :

```python

class Like(models.Model):
    """docstring for Like"""
    user = models.ForeignKey(User, default=1, on_delete=models.CASCADE)
    evaluation = models.ForeignKey(Evaluation, default=1, on_delete=models.CASCADE)
    date_like = models.DateTimeField("Date d'évaluation", auto_now_add=True, blank=True)

```

A cette structure spécifique à notre projet, s'ajoute les tables prédéfinies par l'ORM Django tel que les tables pour la gestion des profils d'utilisateur, les droits d'accès, …

Il y a aussi la structure des tables du module « ***django-registration-redux*** » .

Dans l'archive contenant ce document, il est joint un fichier de dump de la structure complète et des données de la base du projet.

I.4. Du model aux templates (page html) : les views et les urls

Dans la philosophie Django, la structure des données est définie par ce qu'on appelle les classe « Model » et l'affichage de ces données est réalisé par les templates (pages html). Comment donc notre modèle sauvegardé dans notre base de données,est rendu dans les templates ?

C'est le rôle que joue les vues (views). Il permettent de faire communiquer le modèle et les templates. C'est dans les vues que sont défini les types d'objets (données) à faire apparaître dans les pages.

II. Configuration de l'environnement de travail

La création et la gestion du projet et des applications Django nécessite pour cela un environnement de développement spécifique à mettre en place. Ci-dessous les dépendances installées et les fichiers externes ajoutés pour l'environnement nécessaire à la gestion de notre projet et ses applications

II.1. Modules de bases

La commande « pip list » permet de lister tous les modules (dépendances) installés, constituant l'environnement fonctionnel nécessaire à la gestion du projet. La figure suivante présentant ces module est le résultat de la commande $`pip list`.

```
(django) etudiant@d8CK:~/DjangoProject/makeOurPlanetGreatAgain$ pip list
Package                 Version
----------------------- -------
Django                  1.11
django-bootstrap3       11.0.0
django-mathfilters      0.4.0
django-registration-redux 2.5
mysqlclient             1.3.14
pip                     18.1
pkg-resources           0.0.0
pytz                    2018.7
setuptools              40.6.2
wheel                   0.32.3
(django) etudiant@d8CK:~/DjangoProject/makeOurPlanetGreatAgain$
```

Chacun de ces module répond à un besoin spécifique :

- ✔ **pip** : c'est un système de gestion de paquets utilisé pour l'installation et la gestion des librairies écrites en Python.
- ✔ **Django 1.11** : c'est tout simplement un paquet Python. Il constitue le module de base pour la creation d'applications python réutilisables. Son installation est fait avec $pip. Bon nombre de dépendances s'installent automatiquement lors de l'installation du module django.
- ✔ **setuptools** : c'est une bibliothèque qui facilite la gestion de paquets Python (téléchargement, construction ou compilation, installation/désinstallation, mise à jour)
- ✔ **pytz** et **wheel** sont en plus des deux précédents sont des dépendances automatiquement installées lors de l'installation des modules PIP ou Django
- ✔ **django-bootstrap3** version 11.0.0 : c'est une collection d'outils utile à la création du design (graphisme, animation et interactions avec la page dans le navigateur ... etc.) de sites et d'applications web. Après installation, ajouter 'bootstrap3' dans INSTALLED_APPS du fichier `setting.py`.

- ✔ **django-registration-redux 2.5** : fournit des fonctionnalités d'enregistrement et d'authentification des utilisateurs pour un site web Django. Ajouter le module « registration » dans INSTALLED_APPS du fichier `setting.py`.
- ✔ **django-mathfilters** : c'est module qui simplifie l'appel à des opérations mathématiques (arithmétiques) de base. Comme pour les modules ci-dessus, il doit être ajouté dans INSTALLED_APPS du fichier setting.py. Il est ensuite chargé dans les template comme suit : **`{ % load mathfilters %}`**.

II.2. Fichiers de configurations

Toute la configuration de base du projet est décrite dans le fichier **`setting.py`**. Ce fichier étant automatiquement généré lors de la création du projet, nous avons ajouté des certaines propriétés et apporté des modifications à certaines des propriétés existantes. Ci-dessous des portions du fichier où nous avons fais des modifications :

```
31  # Application definition
32
33  INSTALLED_APPS = [
34      'django.contrib.sites',
35      'registration',
36      'django.contrib.admin',
37      'django.contrib.auth',
38      'django.contrib.contenttypes',
39      'django.contrib.sessions',
40      'django.contrib.messages',
41      'django.contrib.staticfiles',
42
43      'bootstrap3',
44      'mathfilters',
45
46      'makeOurPlanetGreatAgain',
47      'projets',
48      'profils',
49  ]
50
51  # Database
52  # https://docs.djangoproject.com/en/1.11/ref/settings/#databases
53
54  DATABASES = {
55      'default': {
56          'ENGINE': 'django.db.backends.mysql',
57          'NAME': 'makeOurPlanetGreatAgain',
58          'USER': 'djangoadmin',
59          'PASSWORD': 'django',
60          'HOST': '',
61          'PORT': '',
62      }
63  }
64
65
66  ACCOUNT_ACTIVATION_DAYS = 1 # Délai d'activation d'un compte utilisateur
67  REGISTRATION_AUTO_LOGIN = True # Automatically log the user in.
68  SITE_ID=1
69
```

```
120
121 # Internationalization
122 # https://docs.djangoproject.com/en/1.11/topics/i18n/
123
124 LANGUAGE_CODE = 'fr-FR'
125
126 TIME_ZONE = 'Europe/Paris'
127
128 USE_I18N = True
129
130 USE_L10N = True
131
132 USE_TZ = True
133
134
135 # Static files (CSS, JavaScript, Images)
136 # https://docs.djangoproject.com/en/1.11/howto/static-files/
137
138 STATIC_URL = '/static/'
139
140
141 # Email server
142 EMAIL_BACKEND ='django.core.mail.backends.console.EmailBackend'
143 DEFAULT_FROM_EMAIL = 'hammidou.traore@etud.univ-angers.fr'
144 EMAIL_HOST_USER = 'localhost'
145 EMAIL_HOST_PASSWORD = ''
146 EMAIL_USE_TLS = False
147 EMAIL_PORT = 1025
```

III. Environnement de déploiement : DOCKER

Pour pouvoir déployer le projet Django sous forme d'image docker, nous avons créé les fichiers suivants :

III.1. Prérequis

Afin d'assurer la bonne exécution de ces containers, il faut au préalable créer une base de données nommée « **makeOurPlanetGreatAgain** » et un utilisateur « **root** » identifié par « **passer** » qui disposera de tous les droits sur cette base de données.

Il est possible d'utiliser un utilisateur autre que root ; cependant il faudra que cet utilisateur ait tous les droits sur la base de données. Il faudra aussi :

- modifier la partie DATABASE du fichier makeOurPlanetGreatAgain/settings.py
- modifier les variables d'environnement du service **db** du docker-compose
    - - MYSQL_ROOT_PASSWORD=motDePasseRootMySQL
    - - MYSQL_USER=User
    - - MYSQL_PASSWORD=UserPassword
    - - MYSQL_DATABASE=makeOurPlanetGreatAgain

III.2. Dockerfile

Il contient tous les packages dont l'application a besoin pour s'exécuter, toutes les ressources dont les containers auront besoin doivent être déclarées dans ce fichier.

III.3. Docker-compose

Ce fichier regroupe les services de l'application appelés encore containers :

- **db**: est le container responsable de la base de données. Il définit l'environnement de base de données c'est à dire les paramètres permettant de se connecter à la base de données. Pour restaurer les données de la base, on se sert du dossier d'initialisation de la base de données (c'est à dire le dossier `docker-entrypoint-initdb.d` qui contient les scripts lancés au démarrage), pour exécuter un script permettant de recréer la structure et les données de la base de données.
  Le dossier mysql-dump contient un script dump.sql qui permet de créer la structure et les données de la base de données. MySQL est la base de données traitée dans ce container.

- **Web** : ce container est responsable de l'exécution de l'application sous forme d'interface web. Il dépend du container `db` qui lui fournit les données de la base. Ce container démarre directement sur l'exécution du server, en effet comme les données de la base sont recrées dans le container db, aucune migration n'est nécessaire.

- **Phpmyadmin** responsable de la gestion de la base de données `MySQL`.

III.4. Exécution du docker

- exécuter le dockerfile :
  - **docker-compose build**

- démarrer le container db :
  - **docker-compose up db**

- importer les données de la base de données :
  - **docker exec -it mysql-container /bin/bash**
  - **mysql -u root -p makeOurPlanetGreatAgain < docker-entrypoint-initdb.d/dump.sql**

- sortir du terminal et démarrer le container web :
  - **docker-compose up web**

- se rendre sur **localhost:8000** pour voir le site