Créer un site Web avec Django pour Python

Documentation inspirée de la formation « Découvrez le framework Django » du site Openclassrooms :

https://openclassrooms.com/fr/courses/4425076-decouvrez-le-framework-django

1. Création de l'environnement de travail

1.1. Installation du module Django

Installez le module Django avec **pip** (voir https://docs.djangoproject.com/en/3.2/topics/install/#installing-official-release).
Ouvrir une commande shell sur windows (cmd) et tapez :

```
python -m pip install Django
```

Il sera peut être nécessaire de mettre à jour **pip** :

```
python -m pip install --upgrade pip
```

Vérifiez ensuite que Django est bien installé et accessible depuis Python :

```
python
```

```
>>> import django
>>> print(django.get_version())
3.0.3
```

Il est parfois nécessaire de modifier les variables d'environnement Windows pour permettre l'accès à Python.

1.2. Création du projet et de l'application

Il est prévu dans le module Django de développer plusieurs applications à partir d'un projet.
Ouvrir une commande shell sur windows (cmd) .
Se mettre dans le répertoire de travail désiré.
Créer un projet :

```
django-admin startproject mon_projet
```

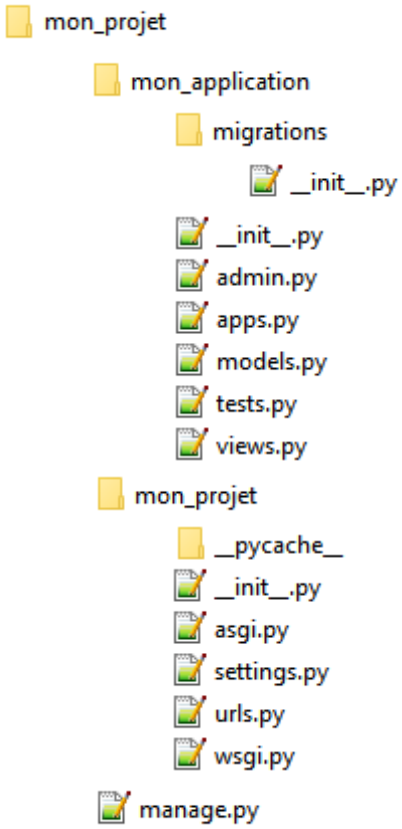
Se déplacer dans le répertoire du projet ainsi créé :

```
cd mon_projet
```

Créer une application dans le projet :

```
python manage.py startapp mon_application
```

L'arborescence des répertoires est alors :

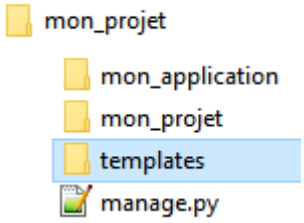


1.3. Mise à jour des paramètres généraux et des répertoires

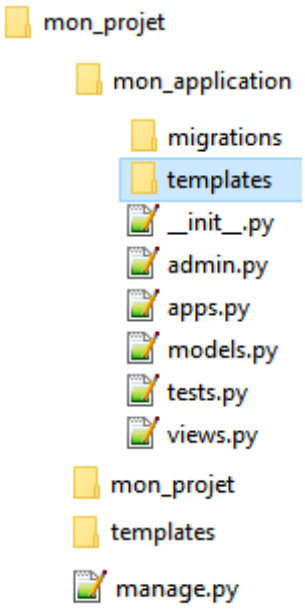
Ouvrir le fichier **settings.py** et rajouter votre application à la liste des applications installées :

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

Ajouter un répertoire **templates** à la racine de votre projets pour les templates propres à votre projet (erreurs404, squelette général de votre design, ...) :



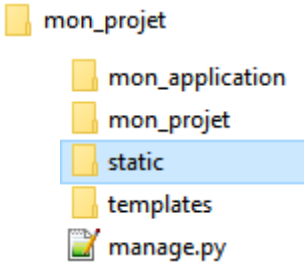
Ajouter un répertoire **templates** dans votre répertoire **mon_application** pour les templates propres à votre application :



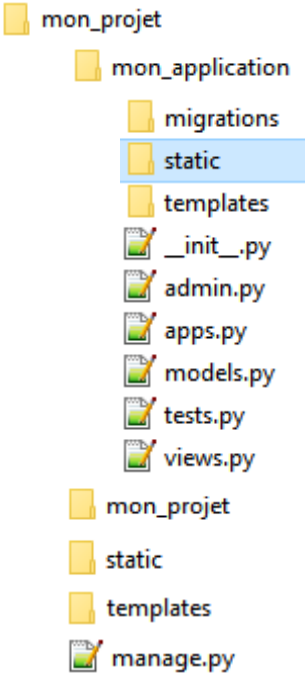
Ajouter le répertoire **templates** du projet dans le fichier **settings.py** en modifiant la valeur de la clé **DIRS** de la liste **TEMPLATES** :

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [
            os.path.join(BASE_DIR, 'templates'),
            ],
        'APP_DIRS': True,
```

Ajouter un répertoire **static** à la racine de votre projets pour les fichiers statiques généraux du projet (images, feuilles de style CSS, fichiers javascript) :



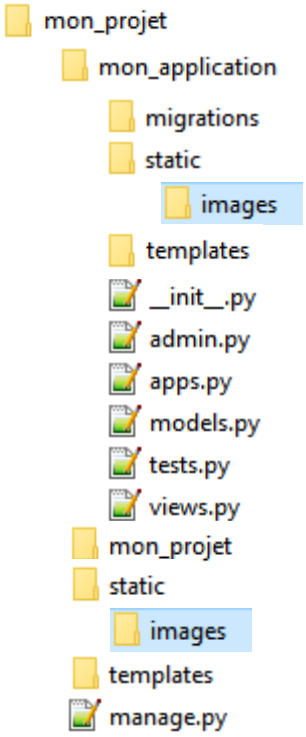
Ajouter un répertoire **static** dans votre répertoire **mon_application** pour les fichiers statiques propres à votre application :



Ajouter le répertoire **static**, à la fin fichier **settings.py**, en tapant les lignes suiantes:

```
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, "static"),
)
```

Créer un répertoire **images** dans les deux répertoires **static** pour y stocker les images de votre site (celles de votre projet et celle de votre application) :



Créer un répertoire **javascript** dans les deux répertoires **static** si besoin pour ranger les fichiers javascript.

La constante **DEBUG** du fichier **settings.py** est à mêttre à **TRUE** pendant la phase de développement et à **FALSE** en phase de production.
Changer la langue en modifiant **LANGUAGE_CODE** du fichier **settings.py** :

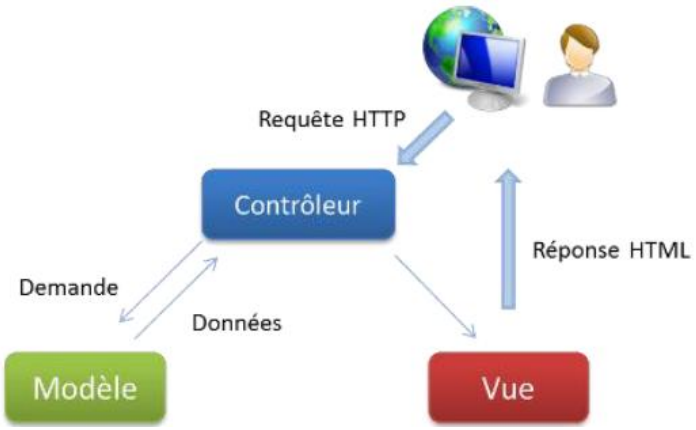
```
LANGUAGE_CODE = 'fr-FR'
```

Il est possible de choisir le système de gestion de la base de donnée en modifiant le paramètre **DATABASES** (voir documentation Django pour plus de détails).

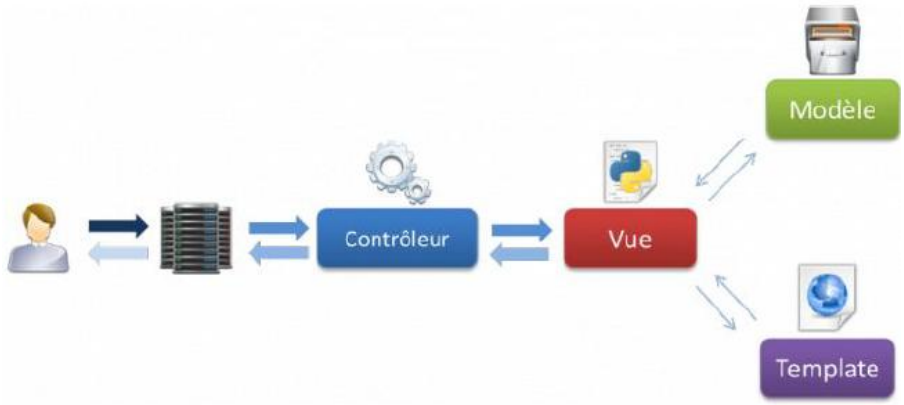
2. Architecture MVT de Django

La plupart des frameworks utilisés pour réaliser des sites Web se basent sur une architecture Modèle-Vue-Contrôleur (MVC). Cette architecture est composée de trois entités distinctes :

- Le Modèle est une interface qui permet de gérer les informations manipulées par le site Web et qui sont stockées en base de données,
- La Vue est la représentation visuelle de ces informations. C'est ce qui est présenté aux *client*s dans un navigateur Internet,
- Le Contrôleur est le chef d'orchestre du système. En fonction des actions de l'utilisateur, il récupère les données utiles via le Modèle et renvoie une réponse via la Vue.



Dans le cas du framework Django, l'architecture est légèrement différente : Django gère lui-même la partie contrôleur. On parle alors d'architecture Modèle-Vue-Template (MVT).
Le Template est un ensemble de fichiers HTML, appelé « gabarits », qui servent à la Vue pour afficher ce que le *client* voit sur son navigateur. Django permet d'utiliser des structures conditionnelles afin de rendre l'affichage dynamique en fonction des actions de l'utilisateur et de l'état courant du site.



Lorsque le *client* appelle une page du site, Django se charge, via un aiguillage (routage) d'adresses de page (URL), d'exécuter la Vue correspondante à sa demande. Cette Vue récupère les données du Modèle et génère la réponse HTML à partir du Template et de ces données.

Le développement d'un site avec Django demande donc de réaliser les quatre parties suivantes :

- Le routage des requêtes du *client* en fonction de l'URL demandée,
- La représentation des données manipulées par le site (Modèle),
- L'affichage de ces données et autres informations au format HTML/CSS (Template),
- La façon dont les données sont insérées dans la page demandée (Vue).

3. Création d'une page Web « accueil.hml »

3.1. Création des templates

Chaque page de votre site correspond à un template (« squelette ») au format HTML qui est sauvegardé dans le répertoire **mon_projet/mon_application/templates**.

Pour créer la page d'accueil de votre site, créer un fichier **accueil.html** dans ce répertoire et y écrire :

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional/EN">
2  <html>
3      <head>
4          <meta http-equiv="content-type" content="text/html" charset="UTF-8">
5          <title>Mon site</title>
6      </head>
7      <body>
8          <p>
9              Mon premier site web avec Django
10         <p>
11     </body>
12 </html>
```

Si vous souhaitez utiliser un feuille de style (fichier **mon_application.css)**, il faut la sauvegarder dans le répertoire **mon_projet/mon_application/static** et ajouter les lignes 1 et 6 suivantes :

```
1  {% load static %}
2  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional/EN">
3  <html>
4      <head>
5          <meta http-equiv="content-type" content="text/html" charset="UTF-8">
6          <link rel="stylesheet" href="{% static 'mon_application.css' %}">
7          <title>Mon site</title>
8      </head>
9      <body>
10         <p>
11             Mon premier site web avec Django
12         <p>
13     </body>
14 </html>
```

Si vous souhaitez ajouter une images (**NSI.jpg**) sauvegardée dans le répertoire **mon_projet/mon_application/static/images/**, ajouter la ligne 12 :

```
{% load static %}
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional/EN">
<html>
    <head>
        <meta http-equiv="content-type" content="text/html" charset="UTF-8">
        <link rel="stylesheet" href="{% static 'mon_application.css' %}">
        <title>Mon site</title>
    </head>
    <body>
        <p>
            Mon premier site web avec Django <br/>
            <img title="NSI" src="{% static '/images/NSI.jpg' %}" alt="Spécialité NSI"/>
        <p>
    </body>
</html>
```

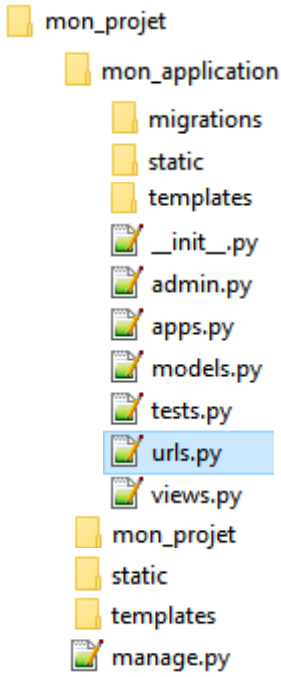
3.2. Création des URLs

Pour afficher une page de votre site il faut faire le lien entre l'URL qui sera appelée dans un navigateur Web et la fonction que vous allez écrire pour l'afficher. Cela se fait dans le fichier **urls.py**.

Ouvrir le fichier **urls.py** du répertoire **mon_projet/mon_projet**, importer la fonction **include** (ligne 17) et modifier la variable **urlpatterns** pour y ajouter le fichier **urls** de votre application (ligne 21) :

```
16  from django.contrib import admin
17  from django.urls import path, include
18
19  urlpatterns = [
20      path('admin/', admin.site.urls),
21      path('mon_application/', include('mon_application.urls')),
22  ]
```

Créer un fichier **urls.py** dans votre répertoire **mon_projet/mon_application** :



Ouvrir votre fichier **urls.py** de votre application (répertoire **mon_projet/mon_application**) et y taper le code suivant afin d'y ajouter votre page d'accueil :

```
1  from django.urls import path
2  from . import views
3
4  urlpatterns = [
5      path('', views.accueil, name="accueil"),
6      path('accueil.html', views.accueil, name="accueil"),
7  ]
```

Vous noterez que l'URL de la page d'accueil est écrite à la ligne 7.
Lorsque vous ajouterez des pages à votre site il faudra duppliquer la ligne 7 et la modifier en fonction de l'URL visée.

3.3. Création des views

À chaque fois que vous appelez une page de votre site, une fonction du fichier **views.py** sera à écrire afin de gérer l'affichage de votre page.
Le nom de la fonction à écrire est défini dans le fichier **urls.py** de votre application.
Pour la page d'accueil de votre site on code la fonction **accueil** (paramètre **views.accueil** du ficher **urls.py**) dans le fichier **views.py** :

```
1   from django.shortcuts import render
2
3   # Create your views here.
4
5   def accueil(request):
6       """
7       Fonction appelée pour afficher la page d'accueil du site
8       """
9       return render(request, 'accueil.html')
```

La fonction **render** gère le chargement de votre fichier **accueil.html** et son affichage.

Si vous souhaitez afficher des données dans une page de votre site (partie dynamique du site), il faut :

- modifier la fonction correspondante du fichier **views.py** pour « transmettre » vos données à la page HTML sous la forme d'un dictionnaire.
- modifier le template HTML de votre page pour afficher ces données,

Exemple :
Supposons qu'on est stockée votre nom dans la variable **mon_nom** et qu'on veuille l'afficher dans la page **accueil.html** de votre site.

On modifie la fonction **accueil** du fichier **views.py** :

```
5   def accueil(request):
6       """
7       Fonction appelée pour afficher la page d'accueil du site
8       """
9       mon_nom = "Xavier"
10      return render(request, 'accueil.html',
11                                {
12                                'mon_nom':mon_nom,
13                                })
```

Et on modifie **accueil.html** de la façon suivante :

```
13              <br/>
14              Bonjour {{ mon_nom }}
```

On peut ajouter des structures de contrôle dans les templates :

```
# IF
{% if age > 25 %}
    ...
{% elif age > 16 %}
    ...
{% else %}
    ...
{% endif %}

#FOR avec une liste: couleurs = ['rouge', 'orange', 'jaune']
{% for couleur in couleurs %}
    <li>{{ couleur }}</li>
{% endfor %}

#FOR avec un dictionnaire: couleurs = {'FF0000':'rouge', 'ED7F10':'orange'}
{% for code, nom in couleurs.items %}
    <li {{ code }}, {{ nom }}</li>
{% empty %}
    < message par defaut >
{% endfor %}
```

Il est possible de gérer une session dans laquelle enregistrer des données pour les transmettre de page en page.
Il suffit pour cela de rajouter un élément dans l'attribut **session**, de type dictionnaire, de l'objet **request** :

```
request.session['nom'] = mon_nom
```

L'effacement de toutes les données de la session se fait ainsi :

```
request.session.flush()
```

Les paramètres transmis dans l'URL se récupèrent dans le dictionnaire **POST** de l'objet **request** :

```
addresse = request.POST['adresse']
```

Remarque : attention à encapsuler ces appels aux attibuts de l'objet **request** dans un **try** car si l'attribut n'existe pas cela va lever une exception.

3.4. Créations des modèles

La base de données est définie dans le fichier **models.py** du répertoire **mon_projet/mon_application**.
Pour chaque table que l'on souhaite ajouter dans la base de données il faut créer une classe qui hérite de **models.Model**.
On définit ensuite dans cette classe les champs que l'on veut grâce aux types définis dans la classe models :

```
# champs texte avec limite de taille 'max_length'
Nom_champs_1 = models.CharField(max_length=Text_Limit_1)
# champs texte sans limite de taille, avec possibilité qu'il ne soit pas renseigné 'null'
Nom_champs_2 = models.TextField(null=True)
# champs date avec date par défaut 'default' et nom à afficher 'verbose_name'
Nom_champs_3 = models.DateTimeField(default=Date_par_defaut_1, verbose_name="Nom_champs_1")
# champs entier positif
Nom_champs_3 = models.IntegerField()
# champs slug
Nom_champs_3 = models.SlugField(max_length=Text_Limit_1)


# champs lié à une autre table : Nom_Table_2 peut contenir plusieurs entrée de Nom_Table_1
Nom_champs_4 = models.ForeignKey('Nom_Table_2', on_delete=models.CASCADE)
Nom_champs_4 = models.ForeignKey('Nom_Table_2', on_delete=models.SET_NULL)
Nom_champs_4 = models.ForeignKey('Nom_Table_2', on_delete=models.PROTECT)

# champs lié à une autre table : Nom_Table_2 peut contenir une seule entrée de Nom_Table_1
Nom_champs_4 = models.OneToOneField('Nom_Table_2', on_delete=models.CASCADE)

# champs multiple :
    # avec modèle gérant la liaison 'through'       sans relation inverse (related_name='+')
Nom_champs_4 = models.ManyToManyField(Nom_Table_2, through='Nom_Table_3', related_name='+')
    # sans modèle de liaison        avec changement de nom de la relation inverse 'related_name'
Nom_champs_5 = models.ManyToManyField(Nom_Table_2, related_name="Nom_Table_1")
```

Il faut aussi définir une sous-classe **Meta** afin de spécifier le nom qui sera affiché dans l'outil d'administration et dans quel ordre afficher les données.
Et enfin définir une fonction **__str__** pour spécifier la façon d'afficher les données des tables dans l'outil d'administration.

Exemple :
Supposons que l'on veuille créer une table **NOMS** qui contienne un champs **nom** de type **chaîne de caratères** et de 20 caractères maximum, voici le code correspondant du fichier **models.py** :

```
1   from django.db import models
2
3   # Create your models here.
4
5   class NOMS(models.Model):
6       nom = models.CharField(max_length=20)
7
8       class Meta:
9           # précision du nom de la table pour outil d'administration
10          verbose_name = "NOMS"
11          # ordre par défaut de tri
12          ordering = ['id']
13
14      def __str__(self):
15          return str(self.id)+": "+self.nom
```

Remarque : chaque table est automatiquement indexée par un numéro **id** que l'on peut récupérer dans le code.

Après avoir ajouter nos tables dans le fichier **models.py**, il faut enregistrer ces tables dans Django. Pour cela il faut modifier le fichier **admin.py** du répertoire **mon_projet/mon_application** :

```
1  from django.contrib import admin
2
3  # Register your models here.
4  from .models import NOMS
5
6  admin.site.register(NOMS)
```

3.5. Intéractions avec la BDD

Pour pouvoir intéragir avec une table de la BDD il faut tout d'abord faire une importation du modèle correspondant, puis utiliser les fonctions suivantes pour ajouter ou supprimer un enregistrement :

```
# ajout d'une entrée
New_Entree_1 = Nom_Table_1(Nom_champs_1="Contenu_Entree_1_champs_1", Nom_champs_2="Contenu_Entree_1_champs_2")
New_Entree_1.Nom_champs_3 = "Contenu_Entree_1_champs_3"
# crée directement l'objet et l'enregistre:
New_Entree_1 = Nom_Table_1.objects.create((Nom_champs_1="Contenu_Entree_1_champs_1")
# avec liaison
New_Entree_1.Nom_champs_3.add(objet_1,...)

# enregistrement de l'entrée en BDD
New_Entree_1.save()
# retrait d'une entrée
New_Entree_1.delete()
# avec liaison
New_Entree_1.Nom_champs_3.remove(objet_1)
# retrait de toutes les liaisons
New_Entree_1.Nom_champs_3.clear()
```

Exemple :

On veut ajouter lors de l'appel de la page d'accueil le nom contenu dans la variable **mon_nom** dans la table **NOMS** :

```
 9       from .models import NOMS
10
11       mon_nom = "Xavier"
12
13       nouveau_nom = NOMS(nom=mon_nom)
14       nouveau_nom.save()
```

Il est important d'appeler la méthode **save()** pour effectuer l'ajout dans la BDD ou utiliser l'autre méthode d'ajout directe **.objets.create()**.

La lecture dans la BDD se fait à l'aide de QuerySet :

```
# réponse sous forme de QuerySet
# liste de toutes les entrées
New_Table_1.objects.all()
# liste de toutes les entrées avec filtrage inclusif
Nom_Table_1.objects.filter(Nom_champs_1="Contenu_Entree_1_champs_1")
# liste de toutes les entrées avec filtrage exclusif
Nom_Table_1.objects.exclude(Nom_champs_1="Contenu_Entree_1_champs_1")
# liste de toutes les entrées avec filtrage inclusif partiel
Nom_Table_1.objects.exclude(Nom_champs_1__contains="Contenu_Entree_1_champs_1")
Nom_Table_1.objects.exclude(Nom_champs_1__lt=max_limit)
Nom_Table_1.objects.exclude(Nom_champs_1__gt=min_limit)
# liste de toutes les entrées triées croissant
Nom_Table_1.objects.order_by(Nom_champs_1)
# liste de toutes les entrées triées décroissant
Nom_Table_1.objects.order_by(-Nom_champs_1)
# inversion de la liste
Nom_Table_1.objects.objects.order_by(Nom_champs_1).reverse()

# réponse sous forme d'un seul objet (erreur si objet non existant)
Nom_Table_1.objects.get(Nom_champs_1="Contenu_Entree_1_champs_1").Nom_champs_2
# réponse sous forme d'un seul objet (création si objet non existant) + booléen (Objet créé?)
Nom_Table_1.objects.get(Nom_champs_1="Contenu_Entree_1_champs_1").Nom_champs_2

# accès à la table liée en sens inverse
# après un new_table_2 = Nom_Table_2(Nom_champs_1="Nom_champs_1")
new_table_2.nom_Table_1_set.all()
# si relation OneToOneField: accès direct sans '_set'
new_table_2.new_Table_1
```

Exemple :

On veut récupérer tous les noms de la tabel **NOMS** et les afficher dans le shell :

```
16      noms = NOMS.objects.all()
17
18      for nom in noms:
19          print(nom.nom)
```

3.6. Lancement du serveur du site

Avant de pouvoir lancer votre site, il faut tout d'abord que la base de données soit créée à partir de votre fichier **models.py**. Dans votre shell, placez vous dans le répertoire de votre projet et tapez les deux commandes suivantes :

```
python manage.py makemigrations
python manage.py migrate
```

Cela va créer un nouveau fichier dans le répertoire **mon_projet/mon_application/migrations**.

Exemple :

Avec la table **NOMS** définie précédemment, voici ce que donne ces deux commandes :

```
C:\MyProject\mon_projet>python manage.py makemigrations
Migrations for 'mon_application':
  mon_application\migrations\0001_initial.py
    - Create model NOMS

C:\MyProject\mon_projet>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, mon_application, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying mon_application.0001_initial... OK
  Applying sessions.0001_initial... OK
```

Si vous changez la définition d'une table dans votre fichier **models.py** il faudra relancer ces deux commandes pour mettre à jour la base de données.

Le fichier de migrations créé contient alors :

```python
# Generated by Django 3.0.3 on 2021-09-18 17:27

from django.db import migrations, models


class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='NOMS',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('nom', models.CharField(max_length=20)),
            ],
            options={
                'verbose_name': 'NOMS',
                'ordering': ['id'],
            },
        ),
    ]
```

On retrouve bien le nom de notre table **NOMS**, le champs **nom**, et les options d'affichage dans l'outil d'administration. On voit aussi le champs automatique **id** qui sert de clé primaire à votre table.

Vous pouvez maintenant lancer le serveur de votre site. Placez vous dans le répertoire de votre projet et tapez la commande suivante :

```
python manage.py runserver
```

Si tout se passe bien vous devriez avoir l'affichage suivant :

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

System check identified no issues (0 silenced).
September 18, 2021 - 19:38:38
Django version 3.0.3, using settings 'mon_projet.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Vous pouvez stopper le serveur à tout moment en faisant Ctrl+C.

Pour accèder à votre page d'accueil de votre site, il faut ouvrir un navigateur Web et y taper l'URL suivante :

`http://localhost:8000/mon_application/` ou

`http://localhost:8000/mon_application/accueil`

Remarque : attention au cache sur votre navigateur Web. Si vous modifier une page de votre site et la recharger, ils se peut que vous ne voyiez pas le changement car votre navigateur va utiliser la page qu'il a sauvegardé dans son cache au lieu de charger la nouvelle version. Il faut donc forcer le rechargement de vos pages lorsque vous en modifier le code (shift+F5 sur google Chrome).

3.7. Outil d'administration de la BDD

Pour accèder à l'outil d'administration de la BDD, il faut d'abord créer un « superuser » avec la commande suivante dans votre shell :

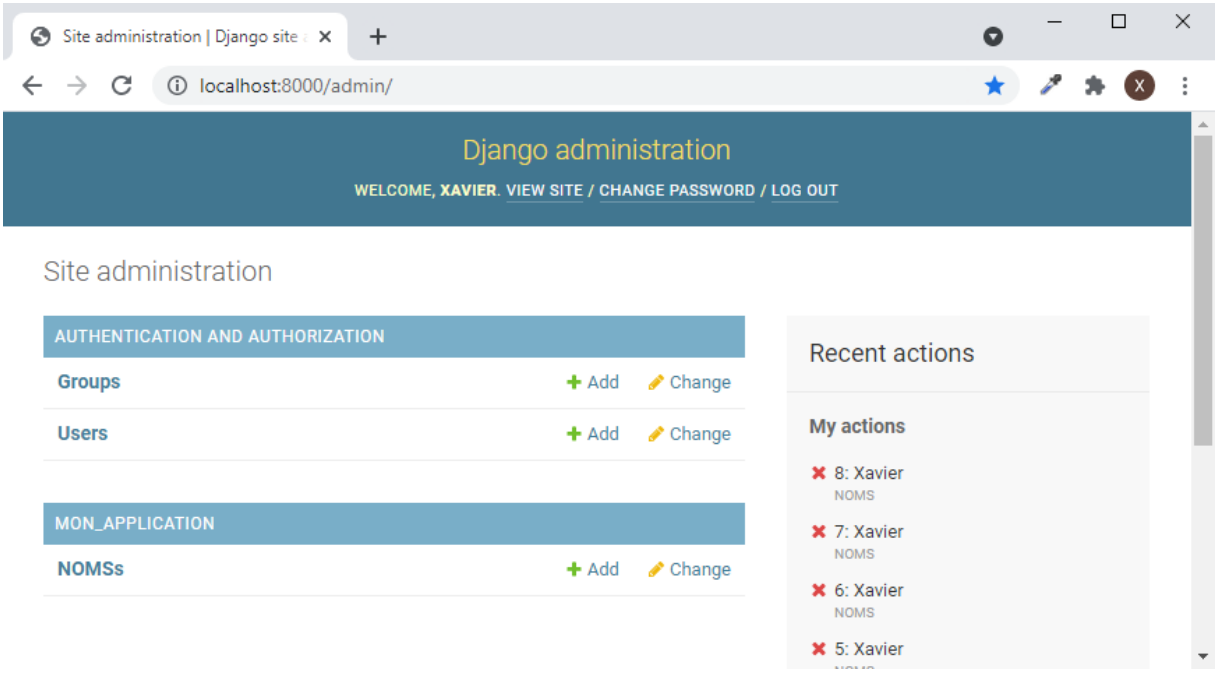
```
python manage.py createsuperuser
```

Il faudra choisir un « **username** » et un « **password** ».

Lancez votre serveur et tapez l'URL suivante dans votre navigateur Web :

```
http://localhost:8000/admin
```

Une page d'identification vous est présentée : entrez votre **username** et **password** définis précédemment et vous accéderez à vos tables :



Si vous cliquez sur le nom de votre table **NOMS**, vous accédez aux enregistrements :