Application Django

I. L'interface d'administration de django

1- Interface admin de django

L'un des aspects les plus importants de **Django** est qu'il contient une **interface de contrôleur** d'**administration** automatisée par défaut. Le site de l'administrateur Django lit les métadonnées de vos modèles pour fournir un lien rapide et modulaire pour gérer le contenu de votre site.

L'administrateur est activé sur le modèle de projet par défaut créer via la commande **startproject**. L'interface d'administration est basée sur le **module Django contrib**. Pour continuer, vous devez vous assurer que les autres modules sont importés dans les tuples **INSTALLED_APPS** et **MIDDLEWARE_CLASSES** du fichier **/settings.py.**

Pour les applications installées, assurrez vous d'avoir:

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

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

2- Migration de la base de données Django

2.1 Définition de la migration

La **migration** est un moyen d'appliquer les modifications que nous avons apportées à un **modèle**, dans le schéma de la **base de données**. Django crée un fichier de migration dans le dossier de migration pour chaque modèle afin de créer le schéma de table, et chaque table est mappée au modèle dont la migration est créée.

Django fournit les différentes commandes utilisées pour effectuer les tâches liées à la migration. Après avoir créé un modèle, nous pouvons utiliser ces commandes.

1. **makemigrations :** Il est utilisé pour créer un fichier de migration qui contient du code pour le schéma en table d'un modèle.
2. **migrate :** Il crée une table selon le schéma défini dans le fichier de migration.
3. **sqlmigrate :** Il est utilisé pour afficher une requête SQL brute de la migration appliquée.
4. **showmigrations :** Il répertorie toutes les migrations et leur statut.

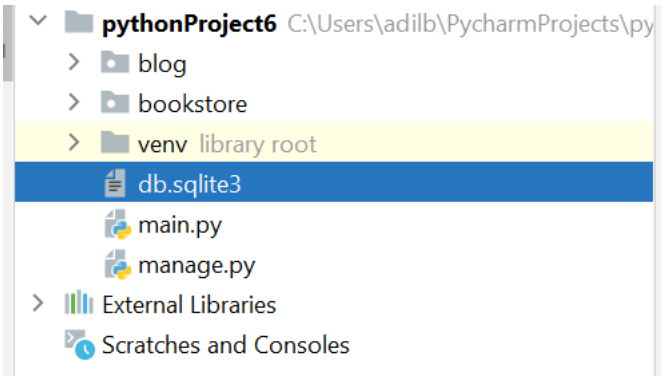
2.2 Création des tables sql via une migration de la base de donnée

Comme cité ci-dessus, la migration ne peut être effectuée qu'après avoir créer un modèle ! Mais ici on va effectuer une migration basée sur modèle existant et livré par défaut par django dès la création d'un projet, c'est le **modèle users**.

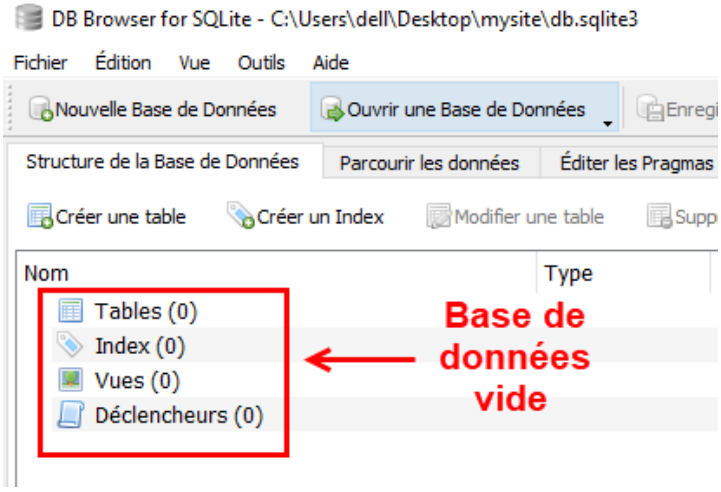
Si vous créer un nouveau projet django et vous le lancer via la commande :

```
python manage.py runserver
```

Vous allez constater la création d'une base de données SQLite3 nommée **"db.sqlite3"** qui se crée automatiquement à la racine du répertoire de votre projet:

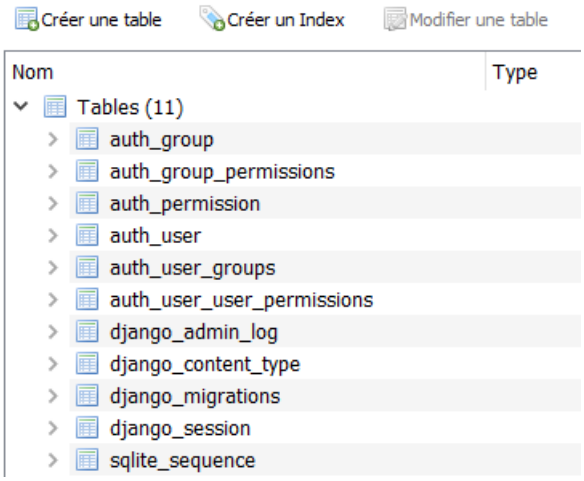


Il s'agit d'un type de fichier qui ne peut être lu qu'avec des programmes spécifiques comme **DB Browser for SQLite** par exemple. En ouvrant ce dernier avec ce logiciel, vous allez voir que la base de données est vide ne contenant **aucune table.**



Mais en effectuant la **migration**, un ensemble de **tables SQL**, nécessaire au fonctionnement du projet django se crée automatiquement:

```
python manage.py migrate
```

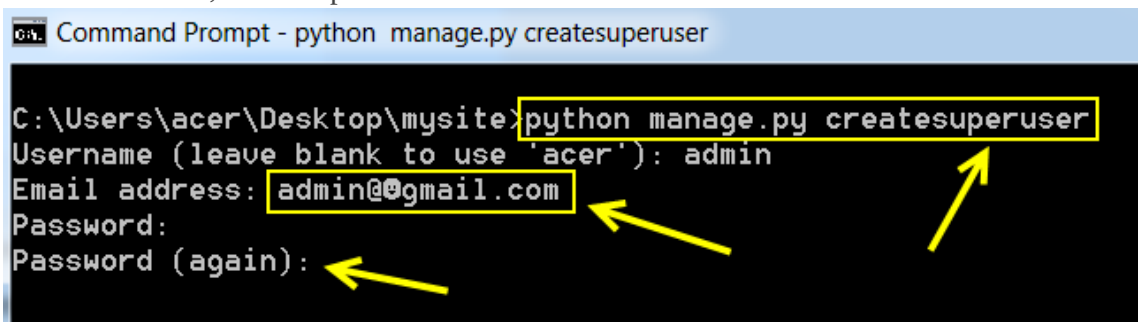


3- Création d'un compte super admin

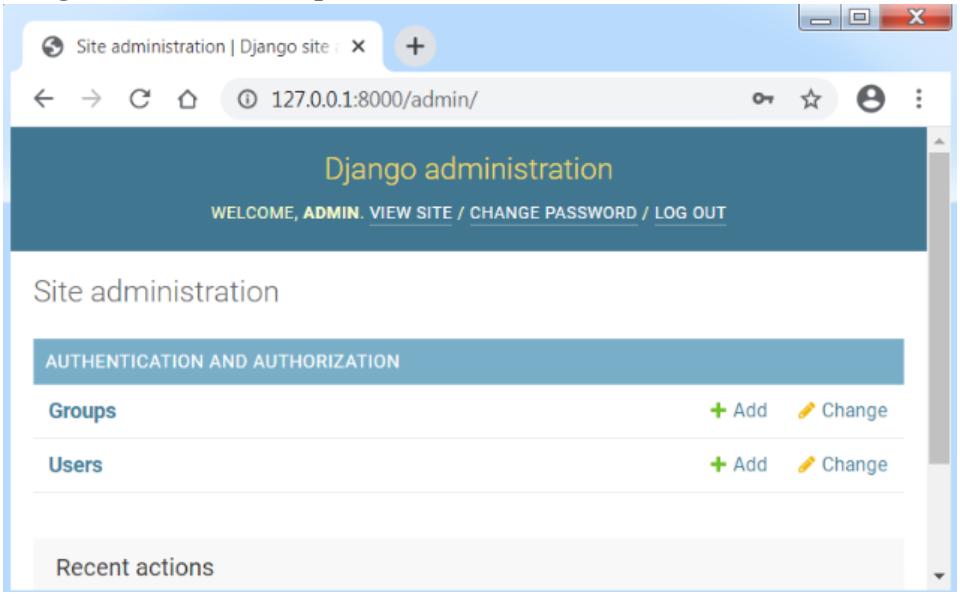
Maintenant après avoir effectué la migration, le terrain est bien préparé pour créer un **super administrateur !** A cet effet, il suffit d'exécuter la commande:

```
python manage.py createsuperuser
```

A ce moment là l'invite de commande vous demande de saisir vos informations de connexion: nom d'utilisateur, email et password:



Maintenant pour accéder à la zone admin, démarrez votre serveur et tapez dans votre navigateur, l'adresse : **http://127.0.0.1:8000/admin/** et saisir vos identifiants de connexion:



II. Les modèles de django

1- Définition des modèles Django

Afin d'**organiser** les **données** du **stockage** de votre **app, django** vous offre l'outil nommé **modèle** (**django model**).

1. **Un modèle abrite** sous forme des **classes** les **noms** des **tables** et des **champs** (attributs) des données du stockage.
2. **Chaque modèle** correspond à une seule **table** de **base de données**.
3. Chaque **modèle** est une **classe Python** qui hérite de **django.db.models.Model.**
4. Chaque **attribut** du **modèle** représente un **champ** de base de données.

2- Les différents types de champs dans un modèle django

Django modèle fourni une variété de types de champs:

1. **AutoField :** designe un champ du type entier qui s'incrémente automatiquement.
2. **BigAutoField** : il s'agit d'un entier 64 bits, un peu comme un AutoField, sauf qu'il est garanti qu'il peut contenir des nombres de 1 à 9223372036854775807.
3. **BigIntegerField :** il s'agit d'un entier 64 bits, un peu comme un IntegerField, sauf qu'il est garanti qu'il s'adapte aux nombres de -9223372036854775808 à 9223372036854775807.
4. **BinaryField :** un champ pour stocker des données binaires brutes.
5. **BooleanField :** un champ vrai / faux. Le widget de formulaire par défaut pour ce champ est un CheckboxInput.
6. **CharField :** il s'agit d'un champ de texte destiné au stockage des petites chaines de caractères.
7. **DateTimeField :** il s'agit d'une date, représentée en Python par une instance datetime.date.
8. **DecimalField :** il s'agit d'un nombre décimal à précision fixe, représenté en Python par une instance Decimal.
9. **DurationField :** un champ pour stocker des périodes de temps.
10. **EmailField :** il s'agit d'un CharField qui vérifie que la valeur est une adresse e-mail valide.
11. **FileField :** il s'agit d'un champ de téléchargement de fichiers.
12. **FloatField :** il s'agit d'un nombre à virgule flottante représenté en Python par une instance flottante.
13. **ImageField :** il hérite de tous les attributs et méthodes de FileField, mais exige également que l'objet téléchargé soit une image valide.
14. **IntegerField :** il s'agit d'un champ entier. Les valeurs de -2147483648 à 2147483647 sont sûres dans toutes les bases de données prises en charge par Django.
15. **NullBooleanField :** comme un BooleanField, mais autorise NULL comme l'une des options.
16. **PositiveIntegerField :** comme un IntegerField, mais doit être positif ou nul (0). Les valeurs de 0 à 2147483647 sont sûres dans toutes les bases de données prises en charge par Django.
17. **SmallIntegerField :** comme un IntegerField, mais n'autorise que les valeurs sous un certain point (dépendant de la base de données).
18. **TextField :** un large champ de texte. Le widget de formulaire par défaut pour ce champ est une zone de texte.

19. **TimeField :** un champ qui représente l'heure, représentée en Python par une instance datetime.time.

3- Création d'un modèle Django

Avant de passer à la création d'un modèle, il faut au préalable enregistrer votre app:

Ajoutez votre app au fichier **mysite/settings.py:**

```
from django.db import models


class Students(models.Model):
    name = models.CharField(max_length=25)
    email = models.EmailField(max_length=40)
    phone = models.IntegerField(max_length=40)
    section = models.CharField(max_length=25)
```

Ceux-ci créera une **table SQLite** formée des champs : **firstName, lastName, email** et **adress,** mais pas avant que les **migrations** nécessaires soient crées:
A ce effet, exécuter en ligne de commande:

```
python manage.py makemigrations monapp
```

et ensuite :

```
python manage.py migration
```

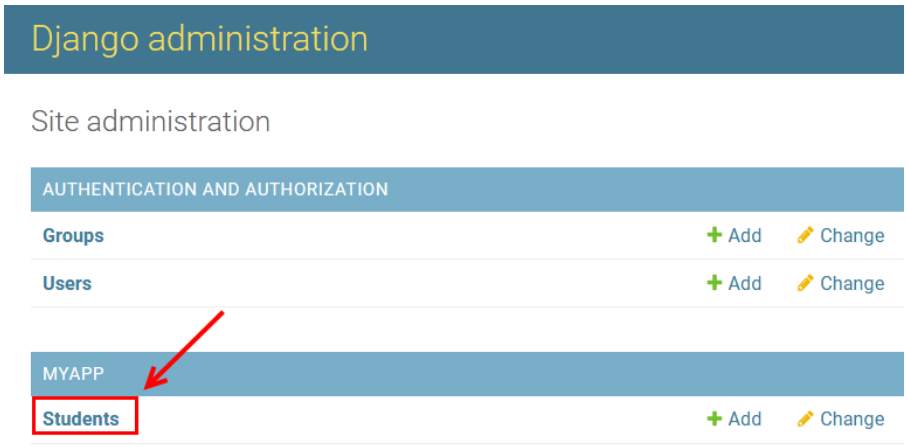
4- Ajouter le modèle à la zone admin

Afin de pouvoir gérer le modèle qu'on vient de créer il est donc nécessaire de le charger dans la zone admin du site. Pour ce faire, il suffit d'éditer le fichier myapp/admin.py en ajoutant les lignes de codes:

admin.py

```
1  from django.contrib import admin
2  from .models import Students
3  admin.site.register(Students)
```

Si vous accéder maintenant à la **zone admin** du site, vous trouver la **rubrique Students**, qui vous donne la possibilité d'édition et de modification:



5- Amélioration de l'affichage au niveau de la zone admin

Nous pouvons maintenant améliorer l'affichage des enregistrements au niveau de la zone admin en affichant la liste des noms, des emails, des sections... A cet effet nous devons importer le **module admin** depuis **django.contrib** sur le fichier **models.py**

On crée ensuite une **classe** au sein du fichier **models.py** qui **hérite** de la classe **admin.ModelAdmin** qui permet d'indiquer la **liste des attributs** à **afficher** et les attributs selon lesquels s'effectue le **filtrage** des enregistrements:

models.py

```
1  from django.contrib import admin
2
3
4  class Students(models.Model):
5      name = models.CharField(max_length=25)
6      email = models.EmailField(max_length=40)
7      phone = models.IntegerField(max_length=40)
8      section = models.CharField(max_length=25)
9
10
11 class StudentsAdmin(admin.ModelAdmin):
12     list_display = ('name', 'email', 'section')
13     list_filter = ('name',)
```

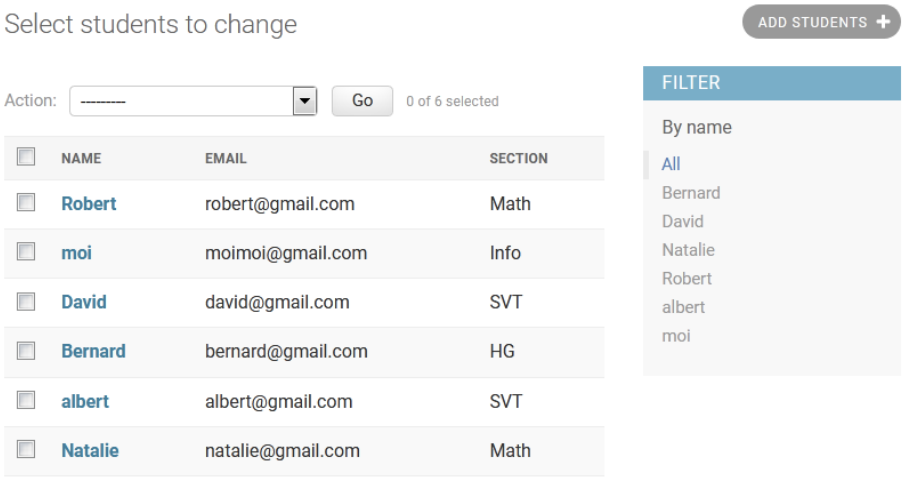
Il reste maintenant quelques modifications au niveau du fichiers **admin.py** au sein duquel on doit **importer** et **enregistrer** la **classe StudentsAdmin** qu'on vient de créer:

models.py | admin.py

```
from django.contrib import admin
from studentsApp.models import Students , StudentsAdmin

admin.site.register(Students , StudentsAdmin)
```

Et finalement en accédant à la zone admin on s'aperçoit qu'il y a effectivement amélioration de l'affichage des résultats faisant apparaitre la liste des noms, des eamils, des sections...*



Les clés étrangères

models.py

```
class Commande(models.Model):

    product  = models.ForeignKey('Product', verbose_name="Produit")
```

Le champ Many-To-Many

backoffice/models.py

```
class ProductTag(models.Model):

    attributes  = models.ManyToManyField("ProductAttributeValue", related_name="product_item")
```