

## TP 2 : Technologies Web II [côté serveur]

Base de données, premiers modèles et site d'administration

---

### Objectifs du TP

Le but principal de ce TP est de découvrir la gestion de base de données dans Django et l'utilisation des modèles.

---

### Consigne

Lisez bien l'énoncé, il contient beaucoup d'informations et chaque mot est utile!

N'hésitez pas à chercher des informations sur les fonctions sur Internet, un bon informaticien doit savoir chercher des informations en ligne. Vous retrouverez notamment la documentation Django ici : <https://docs.djangoproject.com/en/4.1/>

**Attention aux "copier-coller" du PDF vers votre code, des différences d'espace et d'indentation peuvent avoir lieu.**

Il est donc préférable d'écrire vous même les commandes et le code demandés (c'est relativement court).

---

## 1 Le jeu du nombre mystère

Avant d'aller plus loin dans la prise en main, vérifions que ce que nous avons vu dans le TP1 sur la création d'une application, les routes et les vues est bien compris. Pour cela, dans cette première partie, il est demandé de créer dans votre projet une nouvelle application du jeu du nombre mystère. Dans ce jeu, un nombre entier entre 0 et 100 doit être deviné. Ce nombre sera défini "en dur" dans votre code. Par l'intermédiaire de requêtes du type `http://127.0.0.1:8000/magicnumber/10`, l'utilisateur propose un nombre (ici 10 par exemple).

- Si le nombre proposé est plus grand que le nombre mystère, alors le message suivant doit s'afficher : "Le nombre mystère est plus petit".
- Si le nombre proposé est plus petit que le nombre mystère, alors le message suivant doit s'afficher : "Le nombre mystère est plus grand".
- Si le nombre proposé est égal au nombre mystère, alors le message suivant doit s'afficher : "Bravo, le nombre mystère est bien 42".

L'objectif est bien entendu de deviner le nombre mystère en moins de tentatives possibles.

En reprenant ce que nous avons vu dans le précédent TP, créez l'application du jeu du nombre mystère. Vous aurez probablement besoin :

1. De créer une nouvelle application correspondant à votre jeu (dans votre projet `ensisa_project`).
2. De configurer les routes nécessaires permettant à un utilisateur de deviner le nombre mystère (pense à la configuration au niveau de l'application mais aussi au niveau du projet).
3. D'écrire une fonction de vue permettant de définir le comportement souhaité du jeu en fonction du nombre entré dans l'URL.
4. De transformer une requête de type *chaîne de caractère* en *entier*. Pour cela il existe en Python la méthode `int('chaîne')`. Notez également que Django offre une autre solution pour éviter d'avoir à convertir les requêtes. Il est en effet possible de définir le format de l'url attendue dans l'URLconf. Par exemple, il est possible de terminer le modèle d'url par `<int :number>` pour indiquer qu'une valeur entière est attendue.

Le jeu du nombre magique dans sa version actuelle est très limité. Il pourrait être intéressant de ne pas à avoir à indiquer "en dur" dans le code le nombre à deviner, ou bien d'avoir la possibilité de compter le nombre d'essais utilisé par un utilisateur, ou encore d'établir un classement de plusieurs utilisateurs, etc.

Pour conserver toutes ces informations, une base de données va être nécessaire, comme pour toute application Web. Nous allons donc voir comment utiliser et gérer les bases de données dans la section suivante.

## 2 Modèles et base de données

### 2.1 Configuration et activation de la base de données

La configuration de la base de données se fait dans le fichier **ensisa\_project/settings.py**. En ouvrant ce fichiers, vous remarquerez que plusieurs réglages par défaut sont déjà établis à la création du projet. En particulier, le paramètre **DATABASES** contient un dictionnaire '**default**' avec deux clés '**ENGINE**' et '**NAME**'. La première clé correspond au moteur de base de données (par défaut SQLite). La second clé correspond au chemin absolu vers le fichier de base de données. Pour notre projet, ces valeurs par défaut conviennent très bien. Sachez que si vous souhaitez modifier le moteur de base de données, c'est ici que cela se passera.

**Vous avez probablement vu que le fichier de base de données par défaut est bel et bien présent dans votre projet : **db.sqlite3**. Celui est encore vide et prend 0 Ko d'espace disque**

Un deuxième paramètre important dans le fichier **settings.py** est **INSTALLED\_APPS**. Comme son nom l'indique, il correspond à différentes applications installées par défaut dans les projets Django (en particulier, nous verrons plus tard l'application **admin**). Certaines de ces applications utilisent au moins une table de la base de données. Ces tables doivent donc être créées au préalable pour que ces applications puissent être utilisées correctement. Pour cela, exécutez la commande suivante dans le terminal :

```
$ python manage.py migrate
```

Cela crée les tables nécessaires dans la base de données.

**Si vous regardez la taille de votre fichier **db.sqlite3**, il n'est plus vide ( > 0 Ko).**

**Pour information, si vous relancez à présent votre serveur, vous vous apercevrez que les messages de "warning" que vous obteniez précédemment ne s'affichent plus. C'est normal, ils indiquaient justement que certaines applications avaient besoin de tables dans la base de données mais celles-ci n'étaient pas encore créées.**

### 2.2 Création des modèles

Une fois que nous avons vu la configuration des bases de données (même si nous n'avons pas fait grand chose), nous pouvons passer à la création des modèles. Pour rappel, Les modèles correspondent à la description des données souhaitées.

Reprenons l'application de sondage **polls** créée dans le TP1. Nous allons créer deux modèles : **Question** et **Choice**. Une question contient le texte de la question et une date de mise en ligne. Un choix contient le texte correspondant au choix et le nombre de votes pour ce choix. De plus, un choix est associé une question.

1. Pour décrire ces données, modifiez le fichier **polls/models.py** de la manière suivante :

```
from django.db import models

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    publication_date = models.DateTimeField('date published')

class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
```

Chaque classe correspond à un modèle et contient plusieurs champs de différents types.

2. Le modèle **Choice** n'est pas complet. Ajoutez les champs correspondant au texte du choix (de type **CharField**) et au nombre de vote pour ce choix (de type **IntegerField**). Vous trouverez plus d'informations sur les types des champs ici : <https://docs.djangoproject.com/fr/4.1/ref/models/fields/#field-types>

## 2.3 Activation des modèles

Une fois les modèles créés, il faut maintenant les activer. Django va donc pouvoir créer, dans la base de données, les tables correspondantes aux modèles.

1. La première étape consiste à indiquer à Django que l'application **polls** doit être incluse. Nous l'avons vu plus haut, cela se passe dans le fichier **ensisa\_project/settings.py**, dans le réglage **INSTALLED\_APPS**. Pour inclure l'application, une référence à sa classe de configuration doit être ajoutée. Cette classe de configuration se trouve dans le fichier **polls/apps.py** et s'appelle **PollsConfig**. Le chemin correspondant est donc **'polls.apps.PollsConfig'**. Ajoutez ce chemin au réglage **INSTALLED\_APPS** comme ceci :

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

2. A présent, exécutez la commande suivante et observez les messages obtenus :

```
$ python manage.py makemigrations polls
```

La commande **makemigrations** indique à Django que des modifications des modèles ont été effectuées et que ces changements doivent être stockés sous forme de migration. En l'occurrence, comme indiqué dans le message obtenu, ces changements sont stockés dans le fichier **polls/migrations/0001\_initial.py**. Ouvrez ce fichier pour observer et pour vérifier.

3. Il est possible de voir quelles instructions SQL seront effectuées dans la base de données pour une migration donnée. Par exemple, pour la migration précédente, tapez la commande suivante :

```
$ python manage.py sqlmigrate polls 0001
```

Vous devriez alors voir les instructions SQL suivantes (pour information) :

```
BEGIN;
--
-- Create model Question
--
CREATE TABLE "polls_question" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
"question_text" varchar(200) NOT NULL, "publication_date" datetime NOT NULL);
--
-- Create model Choice
--
CREATE TABLE "polls_choice" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
"question_id" bigint NOT NULL REFERENCES "polls_question" ("id") DEFERRABLE INITIALLY DEFERRED);
CREATE INDEX "polls_choice_question_id_c5b4b260" ON "polls_choice" ("question_id");
COMMIT;
```

4. Ce que vous avez fait aux questions 2 et 3 n'effectue pas réellement les migrations mais indique uniquement les instructions à effectuer. Pour réellement appliquer la migration, comme nous l'avons vu plus haut, exécutez la commande suivante :

```
$ python manage.py migrate
```

**makemigration vs. migrate** : ce sont bien deux opérations différentes. La première identifie et prépare les changements à effectuer. La seconde effectue réellement les changements. Pour information, c'est la même idée qui est utilisée dans les systèmes de contrôle de versions (Github, SVN, etc.) avec les 'commits'.

## 2.4 Gestion des données

Jusqu'à présent, nous avons uniquement préparé le schéma des données et créé les tables correspondantes dans la base de données. Nous allons donc voir la partie la plus importante, la gestion des données.

Pour cela, nous allons utiliser le shell interactif Python mis à disposition par Django (identique au shell Python dont vous avez l'habitude). Tapez la commande suivante pour ouvrir le shell :

```
$ python manage.py shell
```

Commençons donc à explorer notre base de données :

1. Importez les modèles créés plus haut :

```
>>> from polls.models import Choice, Question
```

2. Affichez la liste des questions :

```
>>> Question.objects.all()
```

Cela retourne l'ensemble vide <QuerySet []>, ce qui est normal car nous n'avons encore créé aucune question.

3. Nous allons donc créer une première question avec les champs nécessaires de la manière suivante :

```
>>> from django.utils import timezone
>>> q = Question(question_text="Comment allez-vous ?", publication_date=timezone
                  .now())
```

Il faut ensuite sauvegarder le nouvel objet dans la base de données :

```
>>> q.save()
```

4. Vérifiez que la question a bien été créée en affichant la liste des questions. Cela doit à présent retourner : <QuerySet [<Question: Question object (1)>]>.
5. Il est possible d'accéder aux différents champs de l'objet créé comme par exemple son identifiant ou ses champs. Essayez les instructions suivantes :

```
>>> q.id
>>> q.question_text
>>> q.publication_date
```

6. De la même manière, il est également possible d'appeler des méthodes (fonctions) associées aux objets si ils en ont. Par exemple, commençons par créer une méthode dans notre modèle de Question. Cette fonction renvoie **True** si la question a été créée il y a moins de 7 jours. Pour cela ajoutez une fonction **publication\_this\_week** à la classe **Question** dans **polls/models.py** comme ceci (cela nécessite l'ajout des modules **datetime** de Python et **timezone** de Django) :

```

from django.db import models
from django.utils import timezone
import datetime

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    publication_date = models.DateTimeField('date published')

    def published_this_week(self):
        if self.publication_date >= (timezone.now() - datetime.timedelta(days=7)):
            return True
        else:
            return False
# ...

```

Vérifiez cela en ouvrant à nouveau le shell python et appelez la méthode sur la question. Pour faire une requête sur une question, la méthode par "clé primaire" est la plus utilisée (équivalent à l'id) :

```

>>> from polls.models import Choice, Question
>>> q = Question.objects.get(pk=1)
>>> q.published_this_week()

```

Cela retourne True.

- Il est bien entendu possible de modifier les champs d'un objet. Par exemple, modifiez le texte de la question et vérifiez l'affichage, comme ceci :

```

>>> q.question_text = 'Comment ca va ?'
>>> q.save()
>>> q.question_text

```

- Modifiez la date de publication de la question à une date plus ancienne d'au moins une semaine et vérifiez que la fonction retourne bien False. Pour modifier la date, il est nécessaire d'utiliser en plus la méthode **make\_aware()** pour indiquer le fuseau horaire utilisé dans Django pour les dates :

```

>>> import datetime
>>> from django.utils import timezone
>>> d = datetime.datetime(1989, 11, 9)
>>> d = timezone.make_aware(d)
>>> q.publication_date = d
>>> q.published_this_week()

```

- Une méthode qu'il est souvent intéressant d'ajouter aux classes est la fonction d'affichage d'un objet d'une classe. Par exemple, lorsque vous avez affiché la liste des questions, le résultat a été le suivant : <QuerySet [<Question: Question object (1)>]>. Cela n'est pas très informatif et ne permet pas de différencier plusieurs questions potentielles par exemples. Pour remédier à cela, une bonne pratique consiste à ajouter aux classes la méthode **\_\_str\_\_()** qui redéfinit la façon dont les objets sont affichés. Ajoutez ces méthodes aux classes **Question** et **Choice** comme ceci :

```

class Question(models.Model):
    #...

    def __str__(self):
        return self.question_text

    #...

class Choice(models.Model):
    #...

    def __str__(self):
        return self.choice_text

```

Relancez le shell Python et exécutez les commandes suivantes pour voir le nouveau résultat qui doit être : <QuerySet [ <Question: Comment ça va >]>?

```
>>> from polls.models import Choice, Question
>>> Question.objects.all()
```

10. Créez une deuxième question avec le texte : "Participerez-vous à la nuit de l'Info cette année?". N'oubliez pas de sauvegarder pour l'ajouter dans la base de données.

11. Affichez la liste des choix possibles pour cette nouvelle question :

```
>>> q2.choice_set.all()
```

Cela retourne logiquement : <QuerySet []>.

12. Ajoutez un choix possible à cette question comme ceci :

```
>>> q2.choice_set.create(choice_text='Oui', nb_votes=0)
```

Cela retourne le choix ajouté (grâce à la méthode `__str__` écrite plus haut).

13. Il est donc possible lors de la création de récupérer ce choix créé dans une variable comme cela :

```
>>> c = q2.choice_set.create(choice_text='Je ne sais pas encore', nb_votes=0)
```

Cela permet notamment de récupérer la question liée à ce choix via la commande :

```
>>> c.question
```

qui retourne <Question: Participerez-vous à la nuit de l'Info cette année >?.

14. Créez deux autres choix possibles pour cette question et affichez la liste des choix pour obtenir quelque chose comme cela :

```
<QuerySet [ <Choice: Oui évidemment>, <Choice: Je ne sais pas encore>,
<Choice: Non, je veux dormir>, <Choice: C'est quoi la nuit de l'Info>]>
```

15. Il est également possible de récupérer le nombre de choix possible par cette commande :

```
>>> q2.choice_set.count()
```

qui doit retourner 4.

16. Django contient une API de recherche puissante pour filtrer des requêtes dans une base de données, grâce à la méthode `filter()`. Par exemple pour récupérer toutes les question dont l'id est 1 (il n'y en a qu'une seule), tapez la commande suivante :

```
>>> Question.objects.filter(id=1)
```

Pour récupérer toutes les questions qui comment par "Comment", tapez la commande suivante :

```
>>> Question.objects.filter(question_text__startswith='Comment')
```

La recherche prend en compte les relations entre les classes, ainsi que les relations entre les éléments d'un champs (par exemple, le champ `publication_date` est un type qui contient les éléments `day`, `month`, `year`, etc.). Les relations sont représentées par `__` (double "underscore") pour les différencier du simple utilisé dans les noms de variables. Par exemple pour retrouver les choix dont la question a été publiée cette année, tapez les commandes suivantes :

```
>>> current_year = timezone.now().year
>>> Choice.objects.filter(question__publication_date__year=current_year)
```

17. Pour supprimer un élément, la commande `delete()` est utilisé. Par exemple :

```
>>> c = q2.choice_set.filter(choice_text__startswith="C'est")
>>> c.delete()
>>> q2.choice_set.all()
```

Vous avez à présent vu comment créer des modèles et les tables correspondantes dans la base de données, ainsi que la gestion des données (création, modification, suppression).

## 3 Compléments sur les modèles dans Django

Pour aller légèrement plus loin, nous allons explorer quelques modèles de relations classiques de Django :

- La relation ManyToOne
- La relation ManyToMany
- La relation OneToOne

Pour explorer ces relations vous pouvez si vous le souhaitez créer une nouvelle application.

### 3.1 La relation ManyToOne

La relation ManyToOne relie plusieurs objets à un seul objet. Nous l'avons en fait déjà vu dans les questions précédentes. Dans Django, elle se met en place par l'intermédiaire du type **ForeignKey**. Par exemple, si on souhaite créer un objet **Promotion** représentant une promotion de l'ENSISA et qui a plusieurs **Students**, on créera les modèles suivants :

```
class Promotion(models.Model):
    speciality = models.CharField(max_length=150)
    year = models.IntegerField(default=0)

    def __str__(self):
        return self.username

class Student(models.Model):
    firstname = models.CharField(max_length=250)
    lastname = models.CharField(max_length=250)

    promo = models.ForeignKey(Promotion, on_delete=models.CASCADE, related_name="
                               students")

    def __str__(self):
        return self.firstname + self.lastname
```

Notez ici, un élément nouveau par rapport à ce que nous avons vu auparavant : l'ajout du paramètre **related\_name** dans la **ForeignKey**. Ce paramètre permettra notamment de retrouver facilement la liste des étudiants d'une promotion par exemple avec la commande suivante appelée sur un objet **promo** de type **Promotion** :

```
>>> promo.students.all()
```

### 3.2 La relation ManyToMany

La relation ManyToMany relie plusieurs objets à plusieurs objets. Par exemple, si on souhaite représenter les différents clubs de l'ENSISA qui sont composés de plusieurs étudiants sachant qu'un étudiant peut également faire partie de plusieurs clubs, nous pourrions créer les modèles suivants :

```
class Club(models.Model):
    clubname = models.CharField(max_length=50)
    member = models.ManyToManyField(Email, related_name='members')

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

### 3.3 La relation OneToOne

À l'inverse, la relation OneToOne représente la relation où un objet n'est lié qu'à un seul objet et inversement. Par exemple, un pays a exactement une seule capitale et une capitale n'est la capitale que d'un seul pays. Pour cette relation, nous aurons les modèles suivants :

```

class Country(models.Model):
    countryname = models.CharField(max_length=50)

    def __str__(self):
        return self.countryname

class Capital(models.Model):
    capitalname = models.CharField(max_length=50)
    country = models.OneToOneField(Country, on_delete=models.CASCADE)

    def __str__(self):
        return self.capitalname

```

### 3.4 Application de gestion de projet

Pour vous entraîner à la création de modèle et la gestion des données, créez une nouvelle application de gestion de projet. Dans cette application, il y aura plusieurs **User** auxquels il sera possible d'assigner une ou plusieurs **Task**. Chaque **User** aura un unique **Email**. Afin de permettre au chef du projet de notifier les utilisateurs sur des sujets particuliers, il pourra créer des **DiffusionListe** pour conserver une liste des **Email**. Un utilisateur pouvant être concerné par plusieurs **DiffusionListe**, son **Email** peut être dans plusieurs **DiffusionListe**.

1. Créez les modèles correspondants en utilisant les 3 types de relation
2. Pensez à prévoir les attributs **related\_name** et les méthodes **\_\_str()** qui sont des bonnes pratiques à retenir pour faciliter la gestion des données
3. Via l'interface Python de Django, créez 4 utilisateurs différents et 4 emails associés à ces 4 utilisateurs. Créez une dizaine de tâches et assignez-les aux utilisateurs. Ensuite, créez 3 listes de diffusion différentes et ajoutez-y des adresses emails parmi les 4 disponibles (faites en sorte qu'au moins un email se retrouve dans au moins deux listes). Pour ajouter un email à la liste de diffusion, il faut d'abord que la liste soit dans la base (méthode **save()**), puis vous pouvez appeler la méthode **add(<object>)** sur le champ **email** de votre objet **DiffusionList**.
4. Affichez la liste des tâches assignées aux 4 utilisateurs par des commandes similaire à :

```
>>> user1.tasks.all()
```

5. Affichez, pour chaque email, les listes de diffusion auquel il est abonné par des commandes similaires à :

```
>>> email1.listes.all()
```

6. Affichez pour un email choisi, l'utilisateur correspondant
7. Affichez pour un utilisateur choisi, son email
8. Filtres : affichez les tâches assignées aux utilisateurs dont l'email commence par 'us'
9. Affichez la liste des utilisateurs et la liste des emails. Supprimez un des utilisateurs puis affichez à nouveau la liste des utilisateur et la liste des emails. Que remarquez-vous? C'est l'effet du paramètre **on\_delete=models.CASCADE** qui supprime également les objets qui font référence à l'objet supprimé.

## 4 Retour au jeu du nombre mystère

Utilisez ce que vous venez de voir pour modifier votre application du jeu du nombre mystère. L'objectif est de pouvoir stocker deux informations : le nombre à deviner et le nombre d'essais de l'utilisateur en cours. Il sera alors possible de modifier dynamiquement la valeur du nombre à deviner dans la base de données sans avoir à le modifier "en dur" dans le code, ainsi que d'incrémenter le nombre d'essais en cours de partie. Lorsque le nombre est deviné, le compteur est remis à zéro.



- Pour simplifier, on considèrera qu'un seul objet de chaque classe (nombre à deviner et nombre d'essais) est créé.
- Dans un premier temps, créer un objet de type 'nombre à deviner' et un objet de type 'nombre d'essais' via le shell Python pour les sauvegarder dans la base.
- Une fois la base de données initialisées, utilisez ces données dans vos vues pour notamment les mettre à jour.
- Pour chaque requête, ajoutez au texte existant, le nombre d'essais
- Pensez à réinitialiser le nombre d'essais à zéro lorsque le nombre a été trouvé.

Certains navigateurs, comme Chrome ou Safari, peuvent envoyer plusieurs fois les requêtes et sans forcément valider par 'Entrer'. Cela peut augmenter plus rapidement que prévu le nombre d'essais... Utilisez Firefox pour ne pas avoir ce problème

## 5 Le site d'administration de Django

Vous l'avez probablement remarqué, ajouter, modifier ou supprimer du contenu via le shell est légèrement pénible. Dans une application Web, il est souvent très intéressant d'avoir une interface pour faciliter cette gestion des données. Django offre cette possibilité avec son site d'administration.

L'interface d'administration n'est pas destinée à être consultée ou utilisée par les visiteurs du site Web. Elle est mise à disposition des gestionnaires du site Web pour faciliter la gestion du contenu.

### 5.1 Premières configurations

Toujours dans la même philosophie, l'interface d'administration dans Django est une application à part entière (admin). Vous avez probablement remarqué qu'elle se trouve par défaut dans la liste des applications installées dans le fichier **settings.py**. Il n'y a donc pas besoin de faire quoique ce soit pour l'activer.

### 5.2 Création d'un utilisateur administrateur

Comme l'interface administrateur n'a pas vocation à être utilisée par tout le monde, il est nécessaire de créer des utilisateurs qui pourra se connecter à l'interface : un **superuser**. Pour créer un **superuser**, lancez la commande suivante :

```
$ python manage.py createsuperuser
```

Saisissez ensuite le nom d'utilisateur et appuyer sur 'Entrer' :

```
Username (leave blank to use 'mdevanne'): admin
```

Indiquez ensuite une adresse email (pas nécessairement réelle) 'Entrer' :

```
Email address: admin@gmail.com
```

Pour finaliser, indiquez un mot de passe d'au moins 8 caractères :

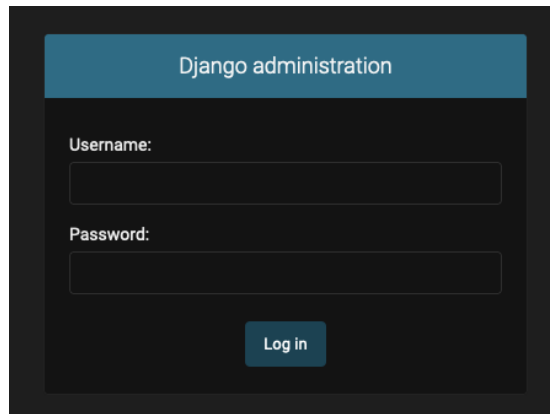
```
Password:
Password (again):
Superuser created successfully.
```

Le superuser **admin** (ou autre) est maintenant créé.

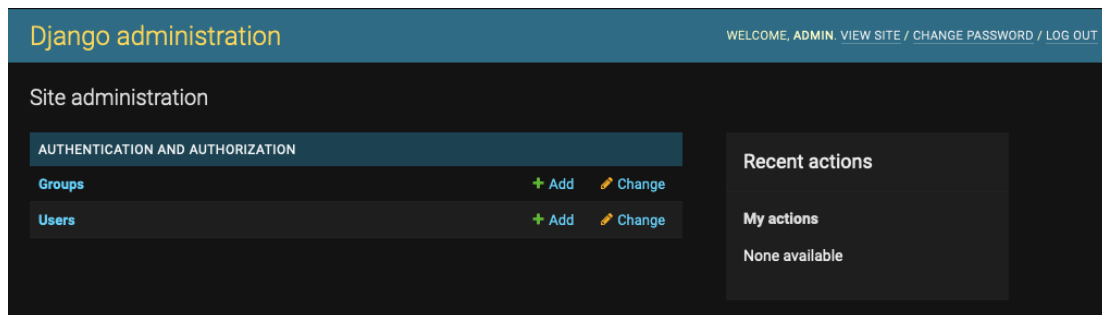
### 5.3 Entrée dans l'interface d'administration

Comme nous l'avons vu, l'application admin est activée par défaut. Vous pouvez donc lancer le serveur de développement et vous rendre à l'url : `http://127.0.0.1:8000/admin`

Vous verrez alors l'interface de connexion comme ceci :

The image shows the Django administration login interface. It has a dark blue header with the text "Django administration". Below the header, there are two input fields: "Username:" and "Password:". At the bottom, there is a "Log in" button.

Connectez-vous avec les identifiants créés précédemment, vous devriez voir cette interface :

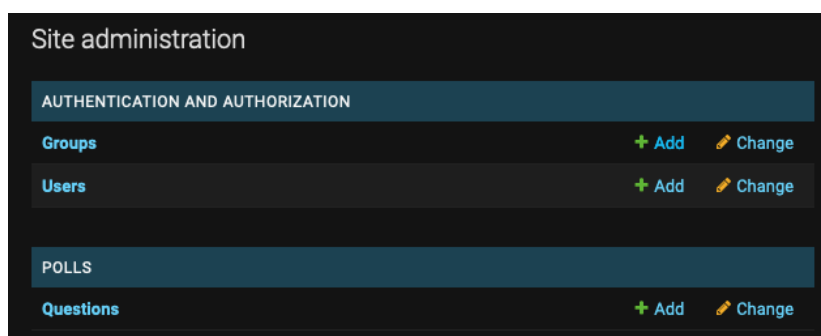
The image shows the Django administration site overview. It has a dark blue header with the text "Django administration" and a welcome message "WELCOME, ADMIN" with links for "VIEW SITE", "CHANGE PASSWORD", and "LOG OUT". Below the header, there is a "Site administration" section. Under "AUTHENTICATION AND AUTHORIZATION", there are links for "Groups" and "Users", each with "Add" and "Change" buttons. To the right, there is a "Recent actions" section with "My actions" and "None available".

Comme vous pouvez le voir, par défaut, seules les données de l'application d'authentification des utilisateurs sont visibles. Pour rendre visibles d'autres données, il est nécessaire d'enregistrer les modèles correspondant. Pour cela, par exemple pour le modèle **Question** de l'application de sondages, ouvrez le fichier **polls/admin.py** et enregistrer le modèle comme ceci :

```
from django.contrib import admin
from .models import Question

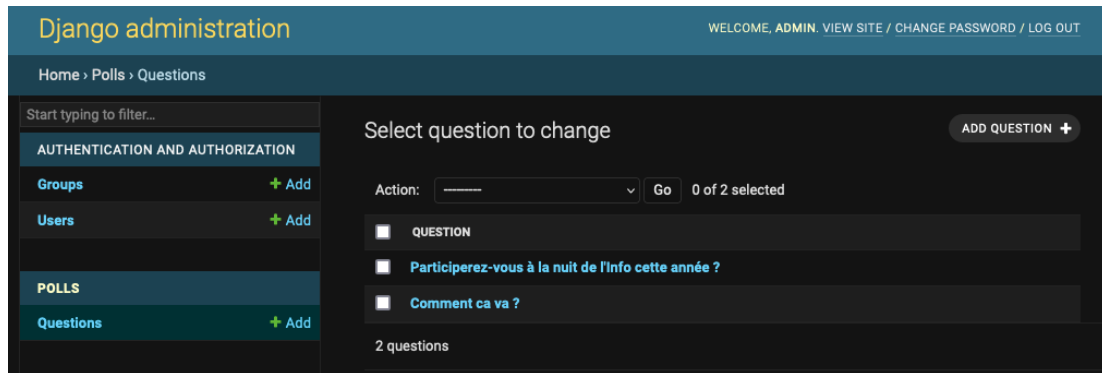
admin.site.register(Question)
```

Rafraîchissez la page et vous verrez alors apparaître le modèle Question :

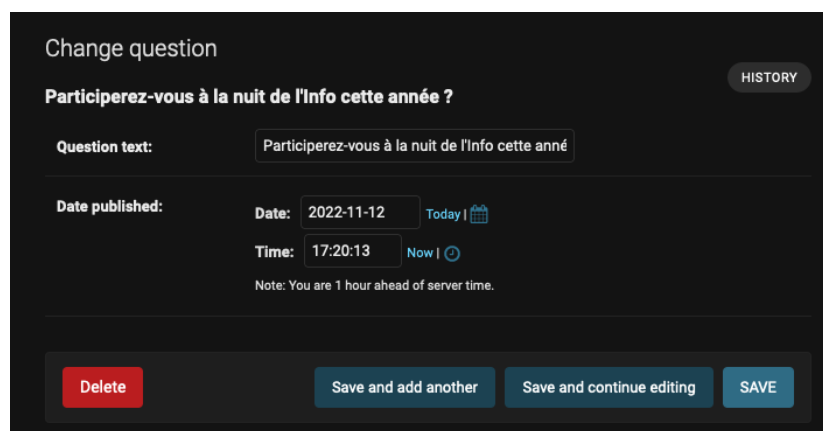
The image shows the Django administration site overview after refreshing. It has a dark blue header with the text "Site administration". Below the header, there is a "Site administration" section. Under "AUTHENTICATION AND AUTHORIZATION", there are links for "Groups" and "Users", each with "Add" and "Change" buttons. Below this, there is a new section "POLL" with a link for "Questions" and "Add" and "Change" buttons.

## 5.4 Gestion des données

En cliquant sur **Questions**, vous verrez alors la liste des questions précédemment créées :

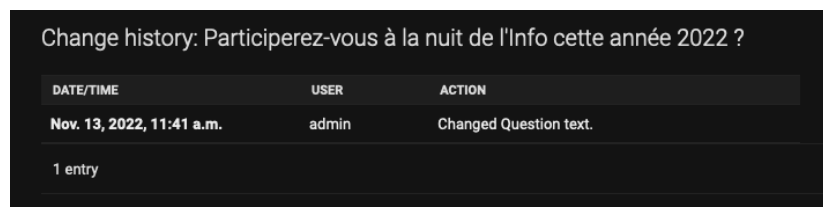


En cliquant sur une question, vous pourrez alors modifier les champs de la question en cliquant sur un des boutons de sauvegarde :

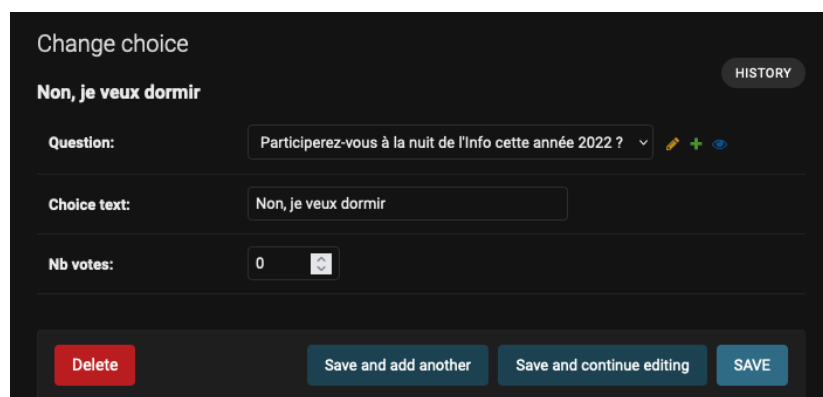


Vous pouvez supprimer une question en cliquant sur **Delete**.

Si vous cliquez sur **History**, vous verrez alors les différents changements apportés à la question :



Enregistrez également le modèle **Choice** pour qu'il apparaisse dans l'administration. En cliquant sur un choix, vous verrez alors les différents champs du choix, dont la question à laquelle il est lié :



Vous pouvez notamment modifier la question à laquelle le choix est lié en choisissant une autre question dans le menu déroulant. L'historique des changements de la question sera alors :

Change history: Non, je veux dormir

DATE/TIME	USER	ACTION
Nov. 13, 2022, 11:48 a.m.	admin	Changed Question.

1 entry

### 5.4.1 A vous de jouer

Enregistrez tous les autres modèles des autres applications créées auparavant et familiarisez-vous avec l'interface pour manipuler les données. Beaucoup plus simple qu'en utilisant le shell Python, pas vrai?

Site administration

AUTHENTICATION AND AUTHORIZATION	
Groups	<a href="#">+ Add</a> <a href="#">Change</a>
Users	<a href="#">+ Add</a> <a href="#">Change</a>

MAGICNUMBER	
Magic numbers	<a href="#">+ Add</a> <a href="#">Change</a>
Nb trialss	<a href="#">+ Add</a> <a href="#">Change</a>

POLLS	
Choices	<a href="#">+ Add</a> <a href="#">Change</a>
Questions	<a href="#">+ Add</a> <a href="#">Change</a>

PROJECTMANAGER	
Diffusion lists	<a href="#">+ Add</a> <a href="#">Change</a>
Emails	<a href="#">+ Add</a> <a href="#">Change</a>
Tasks	<a href="#">+ Add</a> <a href="#">Change</a>
Users	<a href="#">+ Add</a> <a href="#">Change</a>