5. Les modèles dans Django

Maintenant, nous aimerions créer quelque chose qui permet de stocker les articles de notre blog. Mais avant de pouvoir faire ça, il faut raisonner sur des objets.

Les objets

Grâce à la programmation orientée objets, on modélise les choses ainsi que la manière dont elles interagissent entre elles.

Du coup, c'est quoi un objet ? C'est une collection de propriétés et d'actions. Ça a l'air bizarre dit comme ça. Un exemple devrait vous permettre d'y voir un peu plus clair.

Pour ça, il faut répondre à la question : qu'est-ce qu'un article de blog ? Quelles propriétés devrait-il avoir ?

Pour commencer, notre billet de blog doit avoir du texte : il a bien du contenu et un titre, n'est-ce pas ? Et puis, ce serait bien de savoir aussi qui l'a écrit. On a donc besoin d'un auteur. Enfin, on aimerait aussi savoir quand l'article a été écrit et publié.

```
Billet
--------
titre
texte
auteur
creation_date
publication_date
```

Quel genre d'actions pourrions-nous faire sur un article de blog ? Un bon début serait d'avoir une méthode qui permet de publier le billet.

On va donc avoir besoin d'une méthode "publier".

Voilà, nous avons une idée de ce que nous avons besoin. Allons modéliser tout ça dans Django!

Les modèles dans Django

Maintenant que nous savons ce qu'est un objet, nous allons pouvoir créer un modèle Django pour notre billet de blog.

Un modèle Django est un type particulier d'objet : il est sauvegardé dans la base de données. Une base de données est une collection de données. C'est à cet endroit que l'on stocke toutes les informations au sujet des utilisateurs, des billets de blog, etc. Pour stocker nos données, nous allons utiliser une base de données SQLite. C'est la base de données par défaut dans Django. Elle sera largement suffisante pour ce que nous voulons faire.

Pour vous aider à visualiser ce qu'est une base de données, pensez à un tableur avec des colonnes (champs) et des lignes (données).

Créer une application

Pour éviter le désordre, nous allons créer une application séparée à l'intérieur de notre projet. Prenez l'habitude de bien tout organiser dès le début. Afin de créer une application, nous avons besoin d'exécuter la commande suivante dans notre console (prenez garde à bien être dans le dossier "djangology" où se trouve le fichier "manage.py") :

Mac OS X and Linux:

```
~/djangology$ python manage.py startapp blog
```

Windows:

```
C:\Users\Name\djangology> python manage.py startapp blog
```

Vous pouvez voir qu'un nouveau dossier "blog" a été créé et qu'il contient différents fichiers. Les dossiers et fichiers liés à votre projet doivent maintenant être organisés selon cette structure :

```
  djangology
  |-- blog
  |   |-- admin.py
  |   |-- apps.py
  |   |-- __init__.py
  |   |-- migrations
  |   |   |-- __init__.py
  |   |-- models.py
  |   |-- tests.py
  |   `-- views.py
  |-- db.sqlite3
  |-- manage.py
  |-- mysite
  |   |-- __init__.py
  |   |-- settings.py
  |   |-- urls.py
  |   `-- wsgi.py
  `-- requirements.txt
```

Après avoir créé une nouvelle application, vous devez dire à Django de l'utiliser. Nous faisons cela via le fichier "mysite/settings.py". Ouvrez-le dans votre éditeur. Trouvez la section "INSTALLED_APPS" et ajoutez une ligne "'blog.apps.BlogConfig'," juste avant "]". La section doit maintenant ressembler à ceci :

mysite/settings.py

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

```
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'blog.apps.BlogConfig',
]
```

Créer un modèle de blog post

Le fichier "blog/models.py" permet de définir les objets que nous appelons des "modèles". C'est à cet endroit que nous allons définir ce que c'est qu'un billet de blog.

Ouvrez le fichier "blog/models.py" dans votre éditeur, supprimez tout ce qui s'y trouve et copiez-y le morceau de code suivant :

blog/models.py

```
from django.conf import settings
from django.db import models
from django.utils import timezone

class Billet(models.Model):
    author = models.ForeignKey(settings.AUTH_USER_MODEL,
on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    text = models.TextField()
    created_date = models.DateTimeField(default=timezone.now)
    published_date = models.DateTimeField(blank=True, null=True)

    def publish(self):
        self.published_date = timezone.now()
        self.save()

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

Vérifiez que vous avez bien utilisé deux tirets bas (_) autour de "str". C'est une convention fréquemment utilisée en Python qui porte même un petit nom en anglais : "dunder", pour "double-underscore".

Ce gros morceau de code a l'air effrayant, mais, ne vous inquiétez pas : nous allons vous expliquer ce que signifie chacune de ces lignes!

Toutes les lignes qui commencent par "from" ou "import" sont des lignes qui permettent d'importer des morceaux d'autres fich

iers. Concrètement, au lieu de recopier ou de copier-coller la même chose dans différents fichiers, nous pouvons tout simplement faire référence à certains morceaux d'autres fichiers à l'aide de "from ... import ...".

class Billet(models.Model): - C'est cette ligne qui permet de définir notre modèle (ce qui est un "object").

- Le mot clef spécial "class" permet d'indiquer que nous sommes en train de définir un objet.

- "Billet" est le nom de notre modèle. Vous pouvez lui donner un autre nom (mais vous ne pouvez pas utiliser des caractères spéciaux ou accentués et insérer des espaces). Le nom d'une classe commence toujours par une majuscule.

- "models.Model" signifie que "Billet" est un modèle Django. Comme ça, Django sait qu'il doit l'enregistrer dans la base de données.

Maintenant, nous allons pouvoir définir les propriétés dont nous parlions au début de ce chapitre : "title (titre)", "text (texte)", "created_date (date de création)", "published_date (date de publication)" et "author (auteur)". Pour cela, nous allons avoir besoin de définir le type de chaque champ (Est-ce que c'est du texte? Un nombre ? Une date ? Une relation à un autre objet, comme un objet utilisateur par exemple ?)

- models.CharField - Cela nous permet de définir un champ texte avec un nombre limité de caractères.

- models.TextField - Cela nous permet de définir un champ texte sans limite de caractères. Parfait pour le contenu d'un billet de blog, non ?

- models.DateTimeField - Détinit que le champ en question est une date ou une heure.

- models.ForeignKey - C'est un lien vers un autre modèle.

Si vous voulez en savoir plus sur les modèles Django, n'hésitez pas à consulter la documentation officielle de Django (https://docs.djangoproject.com/en/2.2/ref/models/fields/#field-types).

Et sinon, c'est quoi "def publish(self):" ? Il s'agit de notre méthode "publish" dont nous parlions tout à l'heure. "def" signifie que nous créons une fonction/méthode qui porte le nom "publish". Vous pouvez changer le nom de la méthode si vous le souhaitez. La règle de nommage est d'utiliser des minuscules et des tirets bas à la place des espaces. Par exemple, une méthode qui calcule le prix moyen d'un produit pourrait s'appeler "calcul_prix_moyen".

Les méthodes renvoient ("return") souvent quelque chose. C'est le cas de la méthode "str". Dans notre tutoriel, lorsque nous appellerons la méthode "str()", nous allons obtenir du texte (string) avec un titre de Billet.

Créer des tables pour votre modèle dans votre base de données

La dernière étape pour cette section est d'ajouter notre nouveau modèle à notre base de données. Tout d'abord, nous devons signaler à Django que nous venons de créer notre modèle (nous venons de le terminer !). Allez sur votre terminal et tapez "python manage.py makemigrations blog". Le résultat devrait ressembler à ça :

command-line

```shell ~/djangology$ python manage.py makemigrations blog Migrations for 'blog':

```
blog/migrations/0001_initial.py:
  - Create model Billet
```

```

Remarque : N'oubliez pas de sauvegarder les fichiers que vous modifiez. Dans le cas contraire, votre ordinateur exécute la version précédente, ce qui pourrait vous donner des messages d'erreur inattendus.

Django vient de nous préparer un fichier de migration que nous allons pouvoir appliquer dès maintenant à notre base de données. Pour cela, tapez "python manage.py migrate blog". Normalement, vous devrez voir ceci s'afficher dans votre console :

command-line

```shell ~/djangology$ python manage.py migrate blog Operations to perform:

```
Apply all migrations: blog
```

Running migrations:

```
Applying blog.0001_initial... OK
```

```

Youpi ! Notre modèle Billet est maintenant intégré à la base de données. Ce serait cool de voir à quoi il ressemble réellement ! Pour ça, il va falloir attaquer la section suivante ! Au boulot !

6. Administration

From:
https://wiki.centrale-med.fr/informatique/ - **WiKi informatique**

Permanent link:
**https://wiki.centrale-med.fr/informatique/public:appro-s7:td_web:modeles?rev=1697622934**

Last update: **2023/10/18 11:55**