INITIATION À DJANGO

SOMMAIRE

- création projet + app
- Première vue et URLs
- Premier modèle
- Interface d'admin
- Django en ligne de commande
- Deuxième modèle et relations
- Outils de debug
- Écrire un test
- (Bonus) Première vue en Json

L'ENVIRONNEMENT DE DÉVELOPPEMENT

VERSIONS UTILISÉES

- Django 3.0+
- Python : 3.x
- Base de données : SQLite, PostgreSQL, MySQL

CONVENTIONS DE CODAGE

La documentation précise certaines conventions de codage spécifiques à Django.

La PEP 8 fait référence pour le reste.

https://docs.djangoproject.com/fr/2.0/internals/contribu
code/coding-style/

CÔTÉ PYTHON

Python parcourt sys.path pour chercher les modules à importer

- Par défaut ce path contient les répertoires systèmes tels que `/usr/lib/python`, `/usr/local/lib/python`, `~/.local/lib/python` ainsi que le répertoire courant en général
- Comme tout module python, il faut que Django soit accessible dans le path pour pouvoir l'utiliser

- **Virtualenv** permet de créer un environnement python en isolation du système, c'est la méthode préférable pour développer avec python
- Contrairement à PHP, il ne faut pas mettre le code python dans `/var/www/`. Vous pouvez le mettre dans `/home/Dev/` par exemple.

INSTALLER ET ACTIVER UN *VIRTUALENV*

```
$ sudo apt install python3-venv # si ce n'est pas déjà fait
$ python3 -m venv venv
```

INSTALLATION DE DJANGO

```
$ source venv/bin/activate
(venv) $ pip install django
```

CRÉER NOTRE PROJET

CRÉER LE PROJET

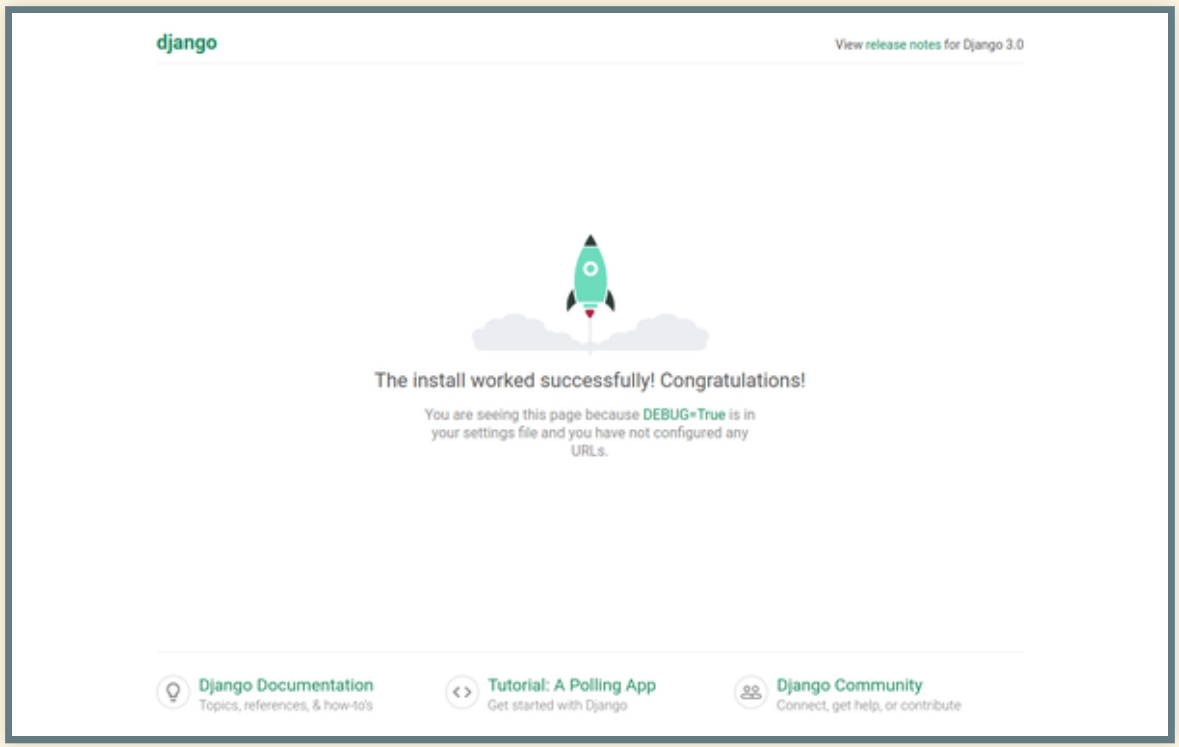
```
(venv) $ django-admin startproject monprojet
```

Rem: n'appelez pas votre projet "django" ou "test", ce sont des mots utilisés par Django.

LANCER LE SERVEUR

```
(venv) $ cd monprojet
(venv) $ ./manage.py runserver
```

IT WORKS !


django
View release notes for Django 3.0
The install worked successfully! Congratulations!
You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.
Django Documentation
Topics, references, & how-to's
Tutorial: A Polling App
Get started with Django
Django Community
Connect, get help, or contribute

CRÉER L'APPLICATION

```
(venv) $ ./manage.py startapp polls
```

STRUCTURE DU PROJET

```
monprojet
├── db.sqlite3
├── manage.py
├── monprojet
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── polls
    ├── admin.py
    ├── apps.py
    ├── migrations
    ├── models.py
    ├── tests.py
    └── views.py
```

FICHIERS DE L'APPLICATION

- `models.py` : déclaration des modèles de l'application
- `views.py` : écriture des vues de l'application
- `admin.py` : comportement de l'application dans l'interface d'administration
- `tests.py` : Il. Faut. Tester.
- `migrations`: modifications successives du schéma de la base de données

PROJET VS. APPLICATION

UNE APPLICATION

Constitue une fonctionnalité (système de blog, application de sondage)

UN PROJET

Contient les réglages et les applications pour un site Web particulier

Un projet est une combinaison d'applications

ACTIVER L'APPLICATION

```
# settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    ...
    'polls.apps.PollsConfig'
]
```

MA PREMIÈRE VUE

CRÉER UNE VUE PAGE D'ACCUEIL POUR L'APPLICATION POLLS

`index` reçoit la requête en paramètre, et renvoie une réponse.

```
# polls/views.py
from django.http import HttpResponse

def index(request):
    return HttpResponse("Bienvenue sur l'application Sondages !")
```

AJOUTER UNE URL

Déclarer un chemin et le relier à la vue

```
# polls/urls.py (il n'existe pas, vous devrez le créer)
from django.urls import path
from polls import views

urlpatterns = [
    path('', views.index),
    # …
]
```

INCLUSION D'*URLCONF*

La configuration des URLs du projet inclue celles de chaque application

```
# monprojet/urls.py
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('polls/', include('polls.urls')),
    path('admin/', admin.site.urls),
]
```

MON PREMIER MODÈLE

GÉNÉRER LA STRUCTURE DE LA BASE DE DONNÉES

Pour le moment, nous gardons le connecteur de base de données par défaut : SQlite3.

```
$ ./manage.py migrate
```

Remarque : par défaut Django utilise SQLite, en production PostgreSQL est plus indiqué.

LE MODÈLE QUESTION ET SES CHAMPS

- Question text → CharField
- Date de publication → DateField

Documentation sur les champs

todo : schéma de données ?

DÉCLARER LE MODÈLE

```
# polls/models.py
class Question(models.Model):
    """Model for Question"""
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')
```

TYPES DE CHAMPS

- Texte `CharField`, `EmailField`…
- Nombres : `IntegerField`, `FloatField`, `DecimalField`…
- Booléens : `BooleanField` et `NullBooleandField`
- Dates : `DateField`, `TimeField`, `DurationField`…
- Fichiers : `FileField`, `ImageField`…

OPTIONS POUR LES CHAMPS

Propriétés communes pour les champs

- `verbose_name`: label du champ
- `null` : valeur NULL autorisée ou non en base de données
- `default` : valeur par défaut pour une nouvelle instance
- `unique` ajoute une contrainte d'unicité
- `validators` permet d'ajouter des contraintes de validation au niveau du modèle
- ...

cf documentation sur les champs de modèle

MIGRATIONS DE LA BASE DE DONNÉES

- Générer le fichier de migration

```
$ ./manage.py makemigrations
```

- Appliquer les migrations à la base de données

```
$ ./manage.py migrate
```

Rem: Commande pour visualiser les modifications apportées à la base de donnée

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

LES MIGRATIONS

Django permet de faire évoluer les modèles sans effacer les données, en génèrant des « diffs » appelés migrations, qu'il applique ensuite à la base de données

GÉNÉRATION DES MIGRATIONS

```
./manage.py makemigrations
```

1. Compare la dernière migration aux modèles déclarés
2. Génère un nouveau fichier de migration

APPLICATION DES MIGRATIONS

```
./manage.py migrate
```

1. Convertit en SQL la ou les migrations qui n'ont pas encore été appliquées
2. Exécute le code SQL sur la base de donnée

REMARQUES

La liste des migrations déjà faites est stockée dans la table `django_migrations` en base de donnée.

Les fichiers de migrations sont numérotés et rangés dans `migrations/`.

Le SQL généré dépend de la base de donnée utilisée.

cf documentation sur les migrations

LE SHELL DJANGO

MANIPULER LES OBJETS

```
$ ./manage.py shell
```

LISTER LES OBJETS D'UN MODÈLE

```
>>> from polls.models import Question
>>> Question.objects.all()
<QuerySet []>
```

CRÉER UN OBJET

```
>>> from django.utils import timezone
>>> q = Question(question_text="Comment ça va ?", pub_date=timezo
>>> q
<Question: Question object (None)>
```

Sauvegarder l'objet dans la base de données

```
>>> q.save()
>>> q.id
1
>>> q
<Question: Question object (1)>
```

MANIPULER L'OBJET

```
>>> q.question_text = "Ça roule ?"
>>> q.save()
```

PERSONNALISER LA REPRÉSENTATION DE L'OBJET

```
class Question(models.Model):
    # ...
    def __str__(self):
        return self.question_text
```

```
>>> from polls.models import Question
>>> Question.objects.all()
<QuerySet [<Question: Ça roule ?>]>
```

L'ADMINISTRATION DJANGO

CRÉER UN SUPER UTILISATEUR

```
$ ./manage.py createsuperuser
```

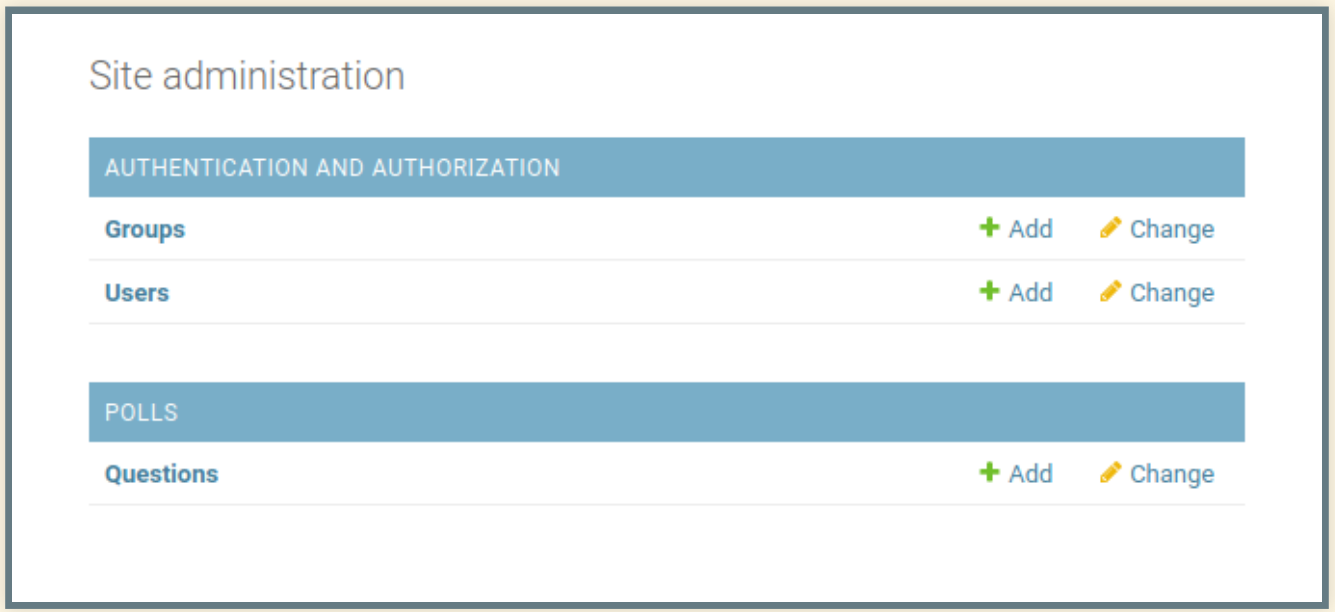
Puis se connecter sur http://127.0.0.1:8000/admin/

DÉCLARER LE MODÈLE DANS L'ADMIN

```
### polls/admin.py
from django.contrib import admin
from polls.models import Question

admin.site.register(Question)
```

MODÈLE DANS L'INTERFACE ADMIN



"BACK-OFFICE AUTOMATIQUE" DE DJANGO

Liste les instances et par introspection des modèles, crée les formulaires de création/modification correspondants.

Personnalisable, permet de modifier :

- les filtres et l'ordre des listes
- l'affichage des listes
- les formulaires et l'ordre des champs
- ajouter des actions en masse sur les listes

cf documentation sur l'interface d'administration

RELATIONS ENTRE LES MODÈLES

LES CHAMPS DE RELATIONS

Champs spécifiques pour représenter les relations entre modèles.

- `models.ForeignKey` : relation **1-N**
- `models.ManyToManyField` : relation **N-N**
- `models.OneToOneField` : relation **1-1**

Nous allons expérimenter le premier, et verrons les deux autres plus tard.

RELATION 1-N :

`models.ForeignKey`

- Premier argument le modèle auquel il fait référence
- `related_name`: nomme la relation inverse
- En base de donnée : contrainte de type clé étrangère

Par exemple : 1 question ← → N choix.

Créer un deuxième modèle `Choice`

```
# polls/models.py
class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCA
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
```

ET ENSUITE

- Django permet de réaliser une appli web complète
- Système de vues, templates, formulaires…
- Dans notre cours : API utilisable par une autre appli web